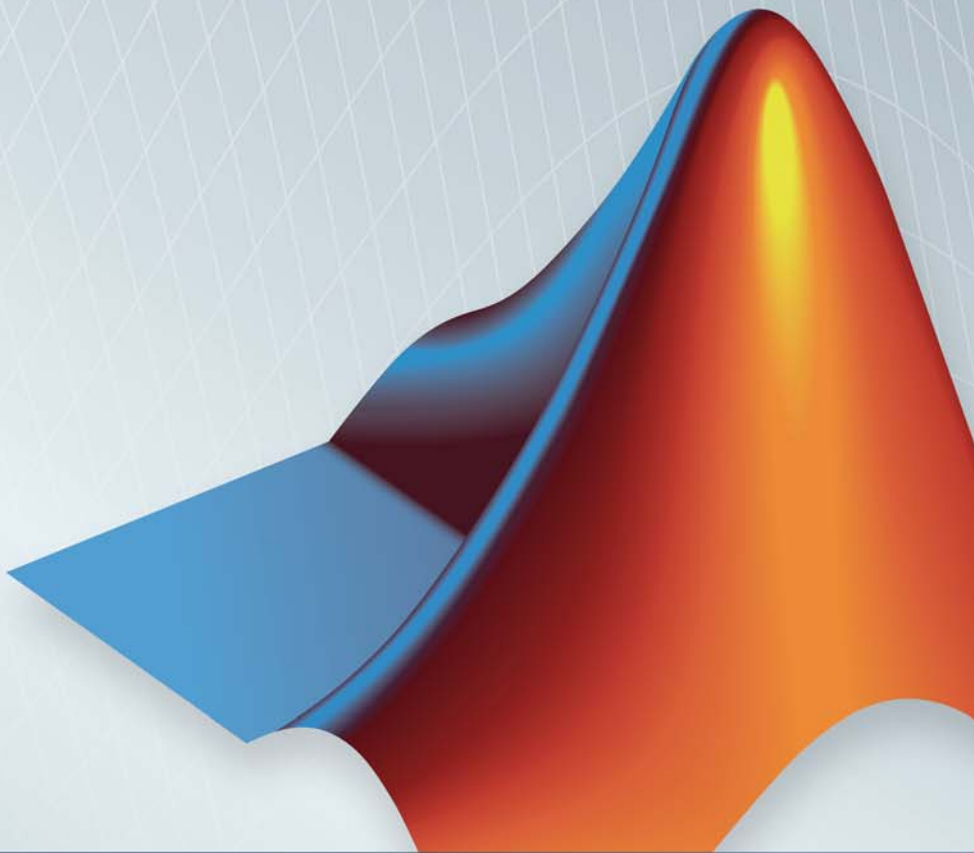


**SimBiology<sup>®</sup>**

Reference

**R2014a**



**MATLAB<sup>®</sup>**



## How to Contact MathWorks



[www.mathworks.com](http://www.mathworks.com) Web  
[comp.soft-sys.matlab](mailto:comp.soft-sys.matlab) Newsgroup  
[www.mathworks.com/contact\\_TS.html](http://www.mathworks.com/contact_TS.html) Technical Support



[suggest@mathworks.com](mailto:suggest@mathworks.com) Product enhancement suggestions  
[bugs@mathworks.com](mailto:bugs@mathworks.com) Bug reports  
[doc@mathworks.com](mailto:doc@mathworks.com) Documentation error reports  
[service@mathworks.com](mailto:service@mathworks.com) Order status, license renewals, passcodes  
[info@mathworks.com](mailto:info@mathworks.com) Sales, pricing, and general information



508-647-7000 (Phone)



508-647-7001 (Fax)



The MathWorks, Inc.  
3 Apple Hill Drive  
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

*SimBiology*<sup>®</sup> Reference

© COPYRIGHT 2005–2014 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

### Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

### Patents

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

## Revision History

|                |             |  |
|----------------|-------------|--|
| September 2005 | Online only | New for Version 1.0 (Release 14SP3+)         |
| March 2006     | Online only | Updated for Version 1.0.1 (Release 2006a)    |
| May 2006       | Online only | Updated for Version 2.0 (Release 2006a+)     |
| September 2006 | Online only | Updated for Version 2.0.1 (Release 2006b)    |
| March 2007     | Online only | Rereleased for Version 2.1.1 (Release 2007a) |
| September 2007 | Online only | Rereleased for Version 2.1.2 (Release 2007b) |
| October 2007   | Online only | Updated for Version 2.2 (Release 2007b+)     |
| March 2008     | Online only | Updated for Version 2.3 (Release 2008a)      |
| October 2008   | Online only | Updated for Version 2.4 (Release 2008b)      |
| March 2009     | Online only | Updated for Version 3.0 (Release 2009a)      |
| September 2009 | Online only | Updated for Version 3.1 (Release 2009b)      |
| March 2010     | Online only | Updated for Version 3.2 (Release 2010a)      |
| September 2010 | Online only | Updated for Version 3.3 (Release 2010b)      |
| April 2011     | Online only | Updated for Version 3.4 (Release 2011a)      |
| September 2011 | Online only | Updated for Version 4.0 (Release 2011b)      |
| March 2012     | Online only | Updated for Version 4.1 (Release 2012a)      |
| September 2012 | Online only | Updated for Version 4.2 (Release 2012b)      |
| March 2013     | Online only | Updated for Version 4.3 (Release 2013a)      |
| September 2013 | Online only | Updated for Version 4.3.1 (Release 2013b)    |
| March 2014     | Online only | Updated for Version 5.0 (Release 2014a)      |



## Functions — Alphabetical List

**1**

## Methods — Alphabetical List

**2**

## Properties — Alphabetical List

**3**

## Index



# Functions — Alphabetical List

---

# groupedData

---

|                        |   |
|------------------------|---|
| <b>Purpose</b>         | Create groupedData object   |
| <b>Syntax</b>          | <pre>grpData = groupedData grpData = groupedData(tbl) grpData = groupedData(tbl,groupVarName) grpData = groupedData(tbl,groupVarName,independentVarName)</pre>  |
| <b>Description</b>     | <p>grpData = groupedData creates an empty groupedData object.</p> <p>grpData = groupedData(tbl) creates a groupedData object by copying a table or dataset object tbl. The GroupVariableName and IndependentVariableName properties of the grpData object are implicitly determined by looking for the first case-insensitive match to a list of variable names of tbl (tbl.Properties.VariableNames). For the grouping variable, the list of names is ID, Group, I, and Run. For the independent variable, the list of names is Time, T, and IDV. If there are no match, GroupVariableName and IndependentVariableName are set to empty strings.</p> <p>grpData = groupedData(tbl,groupVarName) sets the GroupVariableName property of the grpData object to groupVarName. The IndependentVariableName property is implicitly set as in the previous syntax.</p> <p>grpData = groupedData(tbl,groupVarName,independentVarName) additionally sets the IndependentVariableName property of the grpData object to independentVarName.</p> |
| <b>Input Arguments</b> | <p><b>tbl - Data table</b><br/>table   dataset<br/>Data table, specified as a table or dataset.</p> <p><b>groupVarName - Grouping variable name</b><br/>string</p>  |



Grouping variable name, specified as a string. An empty string '' indicates there is no group variable.

### **independentVarName - Independent variable name**

string

Independent variable name, specified as a string. An empty string '' indicates there is no independent variable.

## **Output Arguments**

### **grpData - Grouped data**

groupedData object

Grouped data, returned as a groupedData object.

## **Examples**

### **Create a GroupedData Object from Dataset**

This example uses data collected on 59 preterm infants given phenobarbital during the first 16 days after birth. Each infant received an initial dose followed by one or more sustaining doses by intravenous bolus administration. A total of between 1 and 6 concentration measurements were obtained from each infant at times other than dose times, for a total of 155 measurements. Infant weights and APGAR scores (a measure of newborn health) were also recorded. Data is described in [1], a study funded by the NIH/NIBIB grant P41-EB01975.

Load the sample data set.

```
load pheno.mat ds
```

Create a groupedData object from the data set ds.

```
grpData = groupedData(ds);
```

Display the properties.

```
grpData.Properties
```

```
ans =
```

```
Description: 'This dataset was generated using sbionm
```

# groupedData

---

```
FileNa...'  
  VariableDescriptions: {}  
    VariableUnits: {}  
    DimensionNames: {'Observations' 'Variables'}  
    UserData: []  
    RowNames: {}  
    VariableNames: {'ID' 'Time' 'Dose' 'Response'}  
    GroupVariableName: 'ID'  
    IndependentVariableName: 'Time'
```

GroupVariableName and IndependentVariableName have been automatically assigned to 'ID' and 'Time' respectively.

## References

[1] Grasela Jr, T.H., Donn, S.M. (1985) Neonatal population pharmacokinetics of phenobarbital derived from routine clinical data. *Dev Pharmacol Ther.* 8(6), 374–83.

## See Also

[table](#) | [groupedData](#) object | [sbiofit](#) | [sbiofitmixed](#)

## Purpose

Create kinetic law definition

## Syntax

```
abstkineticlawObj = sbioabstractkineticlaw('Name')  
abstkineticlawObj = sbioabstractkineticlaw('Name','Expression')  
abstkineticlawObj = sbioabstractkineticlaw(...'PropertyName',  
PropertyValue...)
```

## Arguments

*Name*

Enter a name for the kinetic law definition. Name must be unique in the user-defined kinetic law library. Name is referenced by *kineticlawObj*.

*Expression*

The mathematical expression that defines the kinetic law.

## Description

*abstkineticlawObj* = `sbioabstractkineticlaw('Name')` creates an abstract kinetic law object, with the name *Name* and returns it to *abstkineticlawObj*. Use the abstract kinetic law object to specify a *kinetic law definition*.

The *kinetic law definition* provides a mechanism for applying a specific rate law to multiple reactions. It acts as a mapping template for the reaction rate. The kinetic law definition defines a reaction rate expression, which is shown in the property `Expression`, and the species and parameter variables used in the expression. The species variables are defined in the `SpeciesVariables` property, and the parameter variables are defined in the `ParameterVariables` property of the abstract kinetic law object.

To use the kinetic law definition, you must add it to the user-defined library with the `sbioaddtolibrary` function. To retrieve the kinetic law definitions from the user-defined library, first create a root object using `sbioroot`, then use the command `get(rootObj.UserDefinedLibrary, 'KineticLaws')`.

*abstkineticlawObj* = `sbioabstractkineticlaw('Name','Expression')` constructs a SimBiology® abstract kinetic law object,

# sbioabstractkineticlaw

---

*abstkineticlawObj* with the name '*Name*' and with the expression '*Expression*' and returns it to *abstkineticlawObj*.

*abstkineticlawObj* = `sbioabstractkineticlaw(...'PropertyName', PropertyValue...)` defines optional properties. The property name/property value pairs can be in any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs).

Additional *abstkineticlawObj* properties can be viewed with the `get` command. *abstkineticlawObj* properties can be modified with the `set` command.

---

**Note** If you use the `sbioabstractkineticlaw` constructor function to create an object containing a reaction rate expression that is not continuous and differentiable, see “Using Events to Address Discontinuities in Rule and Reaction Rate Expressions” before simulating your model.

---

## Method Summary

|                                   |                                      |
|-----------------------------------|--------------------------------------|
| <code>delete</code> (any object)  | Delete SimBiology object             |
| <code>display</code> (any object) | Display summary of SimBiology object |
| <code>get</code> (any object)     | Get object properties                |
| <code>set</code> (any object)     | Set object properties                |

## Property Summary

|  |  |
|--|--|
| Expression<br>( <code>AbstractKineticLaw</code> ,<br><code>KineticLaw</code> ) | Expression to determine reaction rate equation |
| Name   | Specify name of object                         |

|                    |  |
|--------------------|--|
| Notes              | HTML text describing SimBiology object |
| ParameterVariables | Parameters in kinetic law definition   |
| Parent             | Indicate parent object                 |
| SpeciesVariables   | Species in abstract kinetic law        |
| Tag                | Specify label for SimBiology object    |
| Type               | Display SimBiology object type         |
| UserData           | Specify data to associate with object  |

## Examples

- 1 Create a kinetic law definition.

```
abstkineticlawObj = sbioabstractkineticlaw('ex_myLaw1', '(k1*s)/(k2+k1+s)');
```

- 2 Assign the parameter and species variables in the expression.

```
set (abstkineticlawObj, 'SpeciesVariables', {'s'});  
set (abstkineticlawObj, 'ParameterVariables', {'k1', 'k2'});
```

- 3 Add the new kinetic law definition to the user-defined library.

```
sbioaddtolibrary(abstkineticlawObj);
```

`sbioaddtolibrary` adds the kinetic law definition to the user-defined library. You can verify this using `sbiowhos`.

```
sbiowhos -kineticlaw -userdefined
```

SimBiology Abstract Kinetic Law Array

| Index: | Library:    | Name:     | Expression:           |
|--------|-------------|-----------|-----------------------|
| 1      | UserDefined | ex_myLaw1 | $(k_1*s)/(k_2+k_1+s)$ |

# sbioabstractkineticlaw

---

- 4** Use the new kinetic law definition when defining a reaction's kinetic law.

```
modelObj = sbiomodel('cell');  
reactionObj = addreaction(modelObj, 'A + B <-> B + C');  
kineticlawObj = addkineticlaw(reactionObj, 'ex_myLaw1');
```

---

**Note** Remember to specify the `SpeciesVariableNames` and the `ParameterVariableNames` in `kineticlawObj` to fully define the `ReactionRate` of the reaction.

---

## See Also

[addkineticlaw](#) | [addparameter](#) | [addreaction](#) | [sbiomodel](#)

**Purpose** Prepare model object for accelerated simulations

**Syntax**

```
sbioaccelerate(modelObj)
sbioaccelerate(modelObj,optionObj)

sbioaccelerate(modelObj,csObj,dvObj)

sbioaccelerate(modelObj,csObj,variantObj,doseObj)
```

**Description** `sbioaccelerate(modelObj)` prepares a model object for an accelerated simulation using its active configuration set (configset), and, if available, active variants and active doses. A SimBiology model can contain multiple configsets with only one being active at any given time. The active configset contains the settings to use in model preparation for acceleration.

For accelerated simulations, use `sbioaccelerate` before running `sbiosimulate`. You must use the same model and configset for both functions.

Rerun `sbioaccelerate`, before calling `sbiosimulate`, if you modify this model, other than:

- Changing the variants
- Changing values for the `InitialAmount` of species
- Changing the `Capacity` of compartments
- Changing the `Value` of parameters

`sbioaccelerate(modelObj,optionObj)` uses an option object specified by `optionObj` as one of the following:

- Configset object
- Variant object
- ScheduleDose object
- RepeatDose object

- array of doses or variants

Currently, a particular dose object can only be accelerated with a single model. You cannot use the same dose object for multiple models to be accelerated. You must create a new copy of the dose for each model.

`sbioaccelerate(modelObj, csObj, dvObj)` uses a configset object `csObj` and dose, variant, or an array of doses or variants specified by `dvObj`. If `csObj` is set to `[]`, then the function uses the active configset object.

`sbioaccelerate(modelObj, csObj, variantObj, doseObj)` uses a configset object `csObj`, variant object or variant array specified by `variantObj` and dose object or dose array specified by `doseObj`.

---

## Requirements:

- Microsoft® Visual Studio® 2010 run-time libraries must be available on any computer running accelerated models generated using Microsoft Windows® SDK.
  - If you plan to redistribute your accelerated models to other MATLAB® users, be sure they have the same run-time libraries.
- 

## Input Arguments

### **modelObj - SimBiology model**

SimBiology model object

SimBiology model, specified as a SimBiology model object. The model minimally needs one reaction or rate rule to be accelerated for simulations.

### **optionObj - Option object**

configset object | variant object or array of variant objects | dose object or array of dose objects



Option object, specified as one of the following: `configset` object, `variant` object, array of variant objects, `scheduleDose` object, `repeatDose` object, or array of dose objects.

- When you accelerate the model using an array of dose objects, you can simulate the model using any subset of the dose objects from the array.
- You can use any or no variant input arguments when running `sbioaccelerate`.

### **csObj - Configuration set object**

`configset` object | `[]`

Configuration set object, specified as a `configset` object that stores simulation-specific information. When you specify `csObj` as `[]`, `sbioaccelerate` uses the currently active `configset`.

### **dvObj - Dose or variant object**

`dose` object or array of dose objects | `variant` object or array of variant objects | `[]`

Dose or variant object, specified as one of the following: `scheduleDose` object, `repeatDose` object, array of dose objects, `variant` object, or array of variant objects.

- Use `[]` when you want to explicitly exclude any variant objects from the `sbioaccelerate` function.
- When `dvObj` is a dose object, `sbioaccelerate` uses the specified dose object as well as any active variant objects if available.
- When `dvObj` is a variant object, `sbioaccelerate` uses the specified variant object as well as any active dose objects if available.

### **variantObj - Variant object**

`variant` object or array of variant objects | `[]`

Variant object, specified as a `variant` object or array of variant objects. Use `[]` when you want to explicitly exclude any variant object from `sbioaccelerate`.

## **doseObj - Dose object**

dose object or array of dose objects | []

Dose object, specified as a `scheduleDose` object, `repeatDose` object, or array of dose objects. A dose object defines additions that are made to species amounts or parameter values. Use [] when you want to explicitly exclude any dose objects from `sbioaccelerate`.

## **Examples**

### **Prepare a Model for Accelerated Simulation**

Create a SimBiology model from an SML file.

```
m = sbmlimport('lotka.xml');
```

Prepare the model for accelerated simulation.

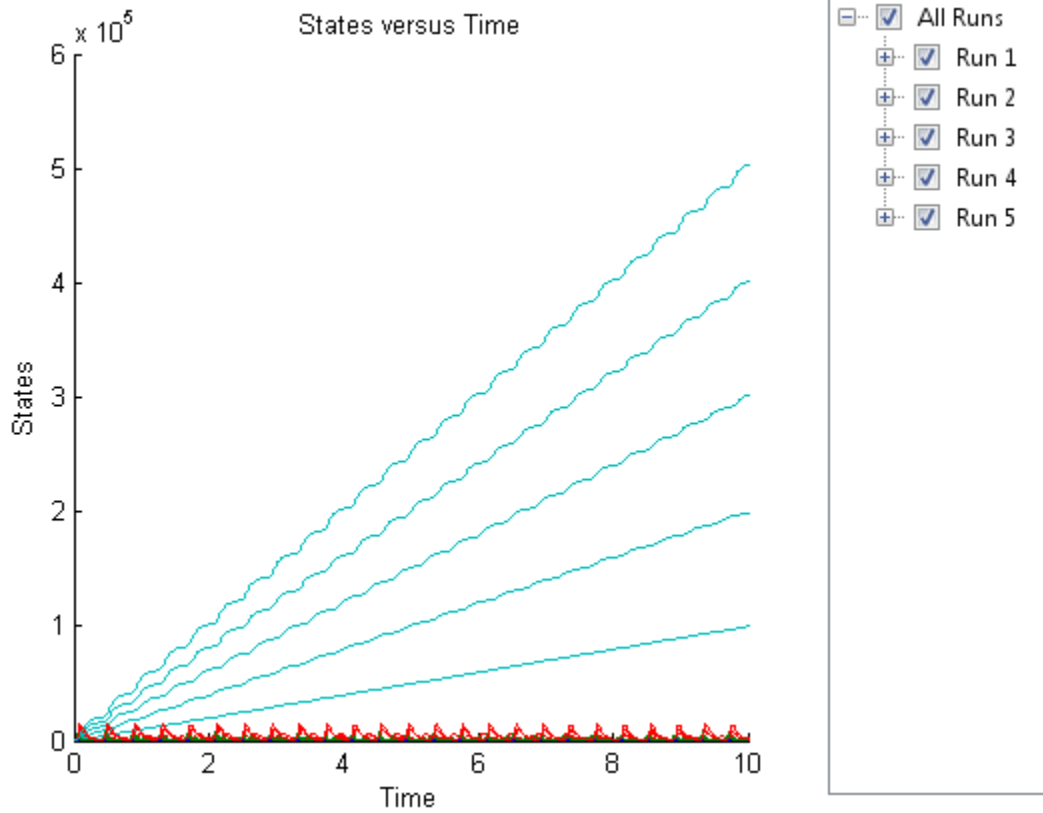
```
sbioaccelerate(m);
```

Simulate the model using different initial amounts of species `x`.

```
x = sbioselect(m,'type','species','name','x');  
for i=1:5  
    x.initialAmount = i;  
    sd(i) = sbiosimulate(m);  
end
```

Plot the results.

```
sbioplot(sd)
```



### Accelerate Simulation Using a User-Defined Configset Object

Load a sample SimBiology model.

```
sbioloadproject radiodecay.sbproj
```

Add a new configuration set using a different stop time of 15 seconds.

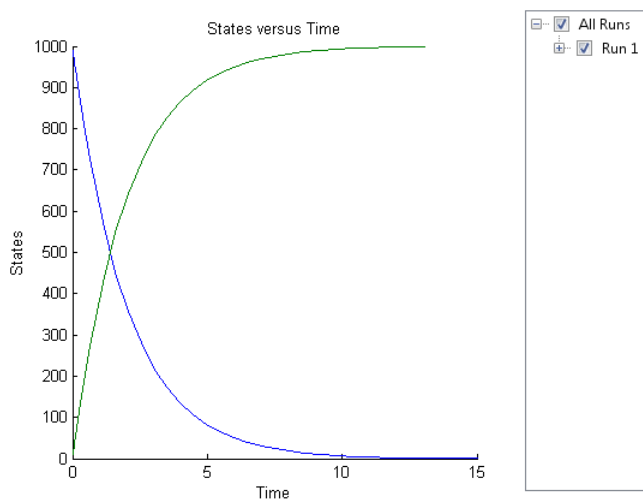
```
csObj = addconfigset(m1,'newStopTimeConfigSet');
csObj.StopTime = 15;
```

Prepare the model for accelerated simulation using the new configset object.

```
sbioaccelerate(m1,csObj);
```

Simulate the model using the same configset object.

```
sim = sbiosimulate(m1,csObj);  
sbioplot(sim)
```



## Accelerate Simulation Using an Array of Dose Objects

Load a sample SimBiology model.

```
sbioloadproject radiodecay.sbproj
```

Add two doses of 100 molecules each for species x, scheduled at 2 and 4 seconds respectively.

```
dObj1 = adddose(m1,'d1','schedule');  
dObj1.Amount = 100;  
dObj1.AmountUnits = 'molecule';
```

```
dObj1.TimeUnits = 'second';  
dObj1.Time = 2;  
dObj1.TargetName = 'unnamed.x';  
  
dObj2 = adddose(m1,'d2','schedule');  
dObj2.Amount = 100;  
dObj2.AmountUnits = 'molecule';  
dObj2.TimeUnits = 'second';  
dObj2.Time = 4;  
dObj2.TargetName = 'unnamed.x';
```

Prepare the model for accelerated simulation using the array of both doses.

```
sbioaccelerate(m1,[dObj1,dObj2]);
```

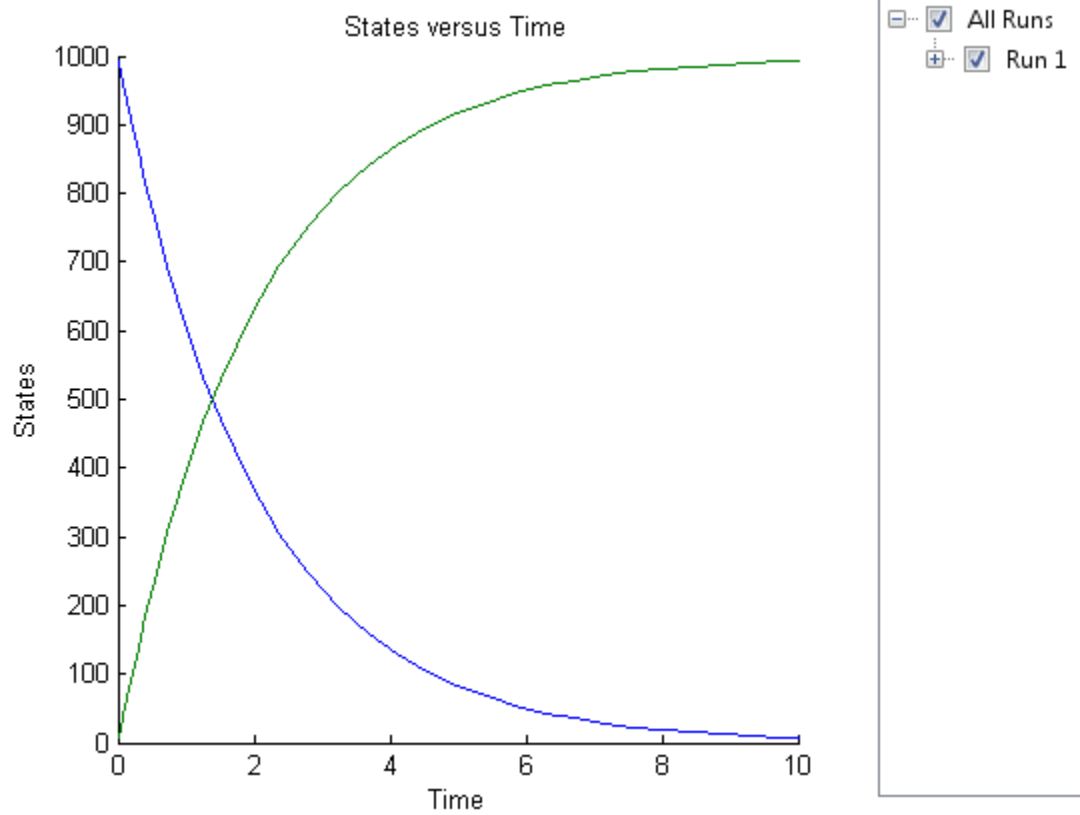
Simulate the model using no dose or any subset of the dose array.

```
sim1 = sbiosimulate(m1);  
sim2 = sbiosimulate(m1,dObj1);  
sim3 = sbiosimulate(m1,dObj2);  
sim4 = sbiosimulate(m1,[dObj1,dObj2]);
```

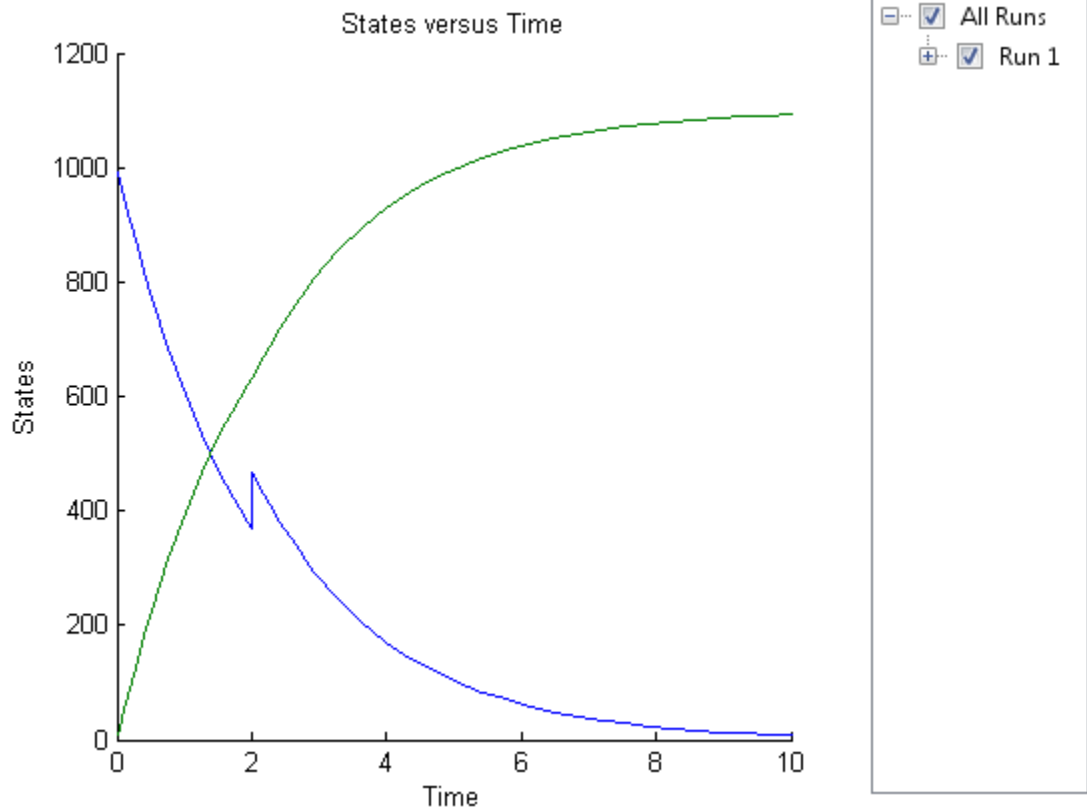
Plot the results.

```
sbioplot(sim1)
```

# sbiaccelerate

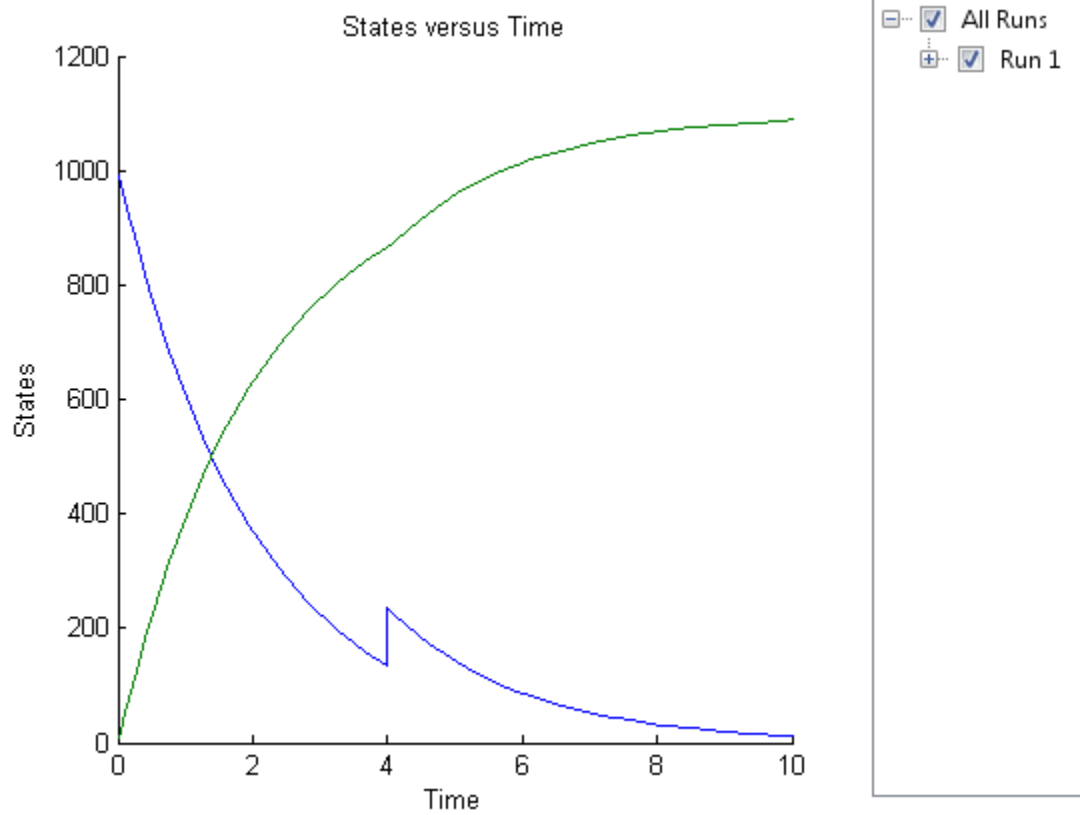


`sbioplot(sim2)`



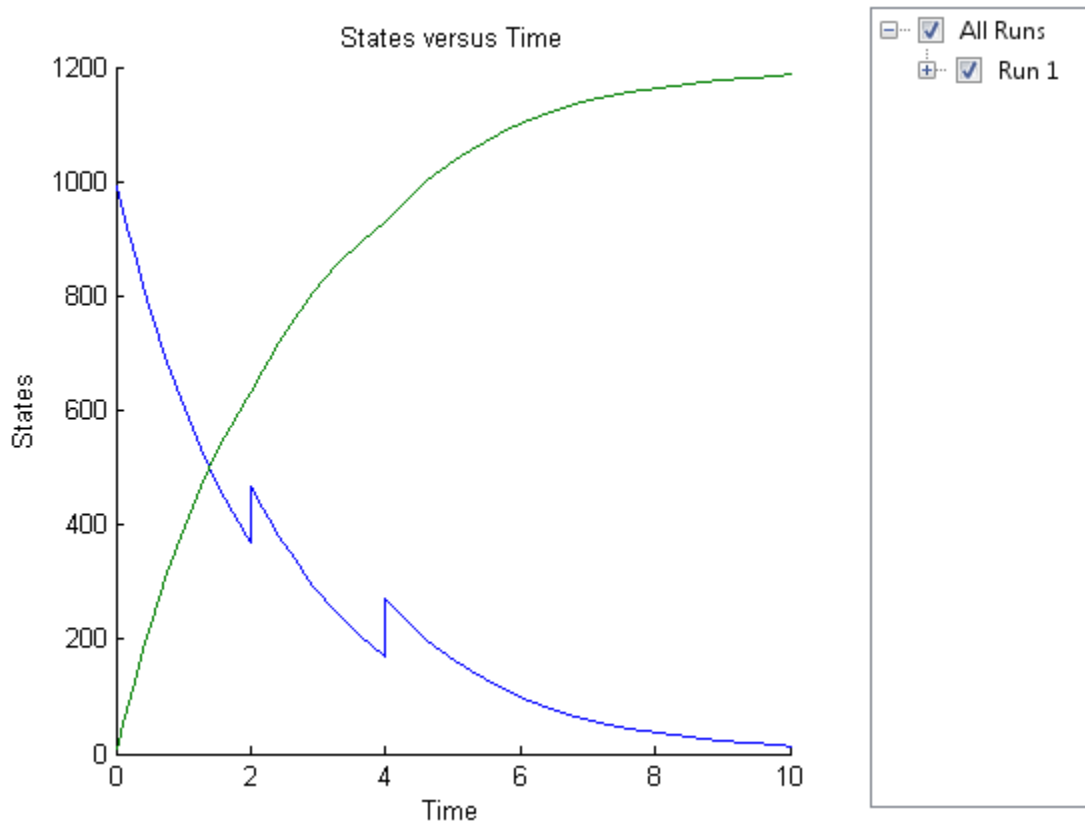
`sbioplot(sim3)`

# sbiaccelerate



`sbioplot(sim4)`





### **Accelerate Simulation Using Configset and Dose Objects**

Load a sample SimBiology model.

```
sbioloadproject radiodecay.sbproj
```

Get the default configuration set from the model.

```
defaultConfigSet = getconfigset(m1,'default');
```

Add a scheduled dose of 100 molecules at 2 seconds for species x.

# sbioaccelerate

---

```
dObj = adddose(m1,'d1','schedule');  
dObj.Amount = 100;  
dObj.AmountUnits = 'molecule';  
dObj.TimeUnits = 'second';  
dObj.Time = 2;  
dObj.TargetName = 'unnamed.x';
```

Prepare the model for accelerated simulation using the default configset object and added dose object.

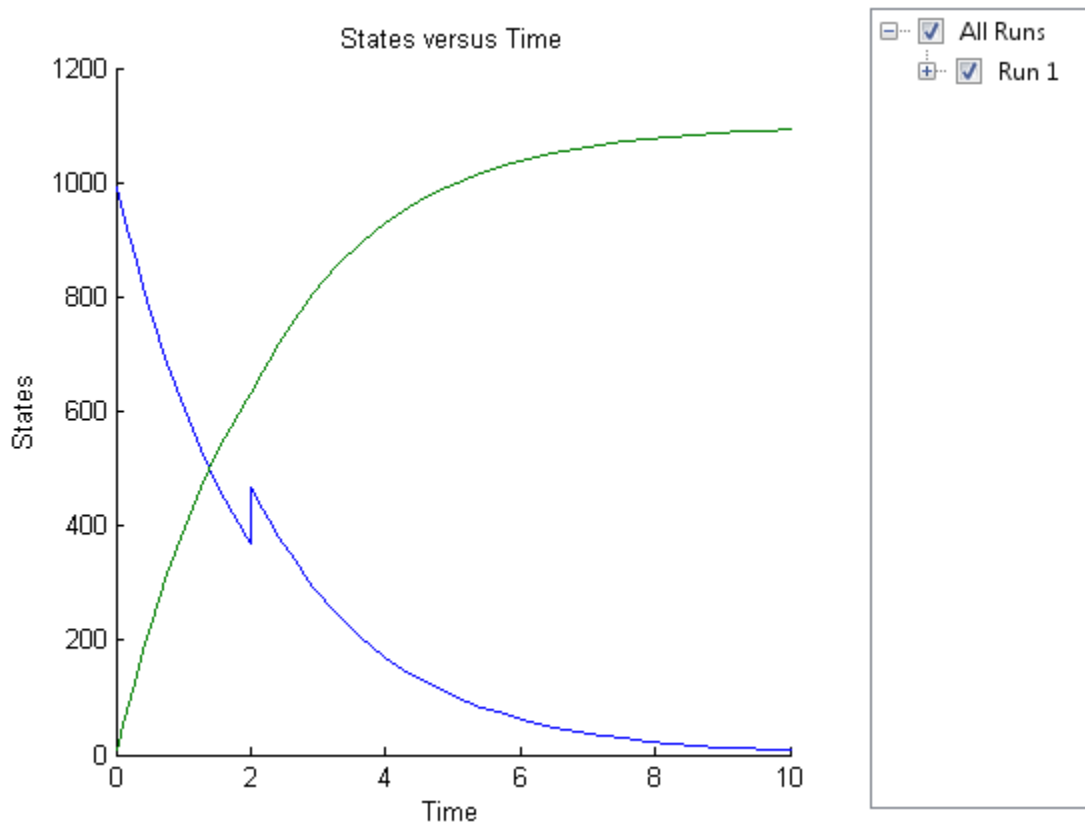
```
sbioaccelerate(m1,defaultConfigSet,dObj);
```

Simulate the model using the same configset and dose objects.

```
sim = sbiosimulate(m1,defaultConfigSet,dObj);
```

Plot the result.

```
sbioplot(sim);
```



### Accelerate Simulation Using Configset, Dose, and Variant Objects

Load a sample SimBiology model.

```
sbioloadproject radiodecay.sbproj
```

Add a new configuration set using a different stop time of 15 seconds.

```
csObj = m1.addconfigset('newStopTimeConfigSet');
csObj.StopTime = 15;
```

Add a scheduled dose of 100 molecules at 2 seconds for species x.

```
dObj = adddose(m1,'d1','schedule');  
dObj.Amount = 100;  
dObj.AmountUnits = 'molecule';  
dObj.TimeUnits = 'second';  
dObj.Time = 2;  
dObj.TargetName = 'unnamed.x';
```

Add a variant of species x using a different initial amount of 500 molecules.

```
vObj = addvariant(m1,'v1');  
addcontent(vObj,{'species','x','InitialAmount',500});
```

Prepare the model for accelerated simulation using the configset, dose, and variant objects. In this case, the third argument of `sbioaccelerate` must be the variant object.

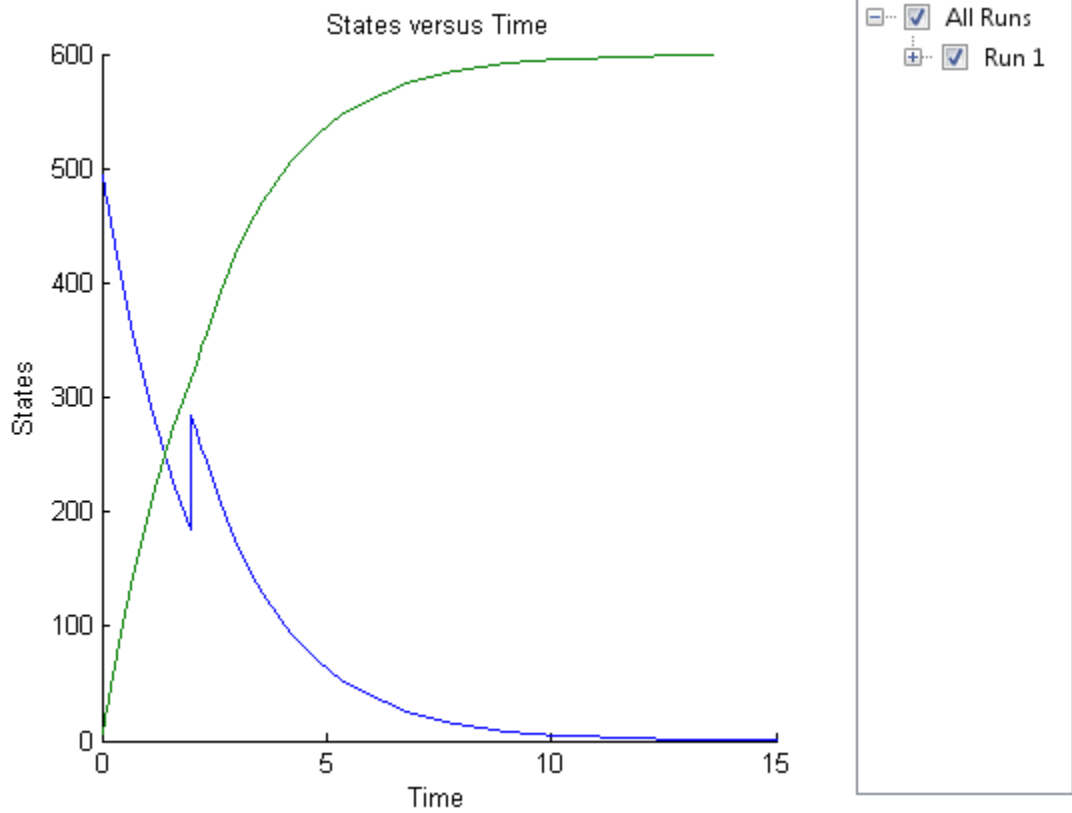
```
sbioaccelerate(m1,csObj,vObj,dObj);
```

Simulate the model using the same configset, variant, and dose objects.

```
sim = sbiosimulate(m1,csObj,vObj,dObj);
```

Plot the result.

```
sbioplot(sim);
```



**See Also**      sbiosimulate

# sbioaddtolibrary

---

**Purpose** Add to user-defined library

**Syntax**  
`sbioaddtolibrary (abstkineticlawObj)`  
`sbioaddtolibrary (unitObj)`  
`sbioaddtolibrary (unitprefixObj)`

## Arguments

|                          |  |
|--------------------------|--|
| <i>abstkineticlawObj</i> | Specify the abstract kinetic law object that holds the kinetic law definition. The Name of the kinetic law must be unique in the user-defined kinetic law library. Name is referenced by <i>kineticlawObj</i> . For more information about creating <i>kineticlawObj</i> , see <code>sbioabstractkineticlaw</code> . |
| <i>unitObj</i>           | Specify the user-defined unit to add to the library. For more information about creating <i>unitObj</i> , see <code>sbiounit</code> .  |
| <i>unitprefixObj</i>     | Specify the user-defined unit prefix to add to the library. For more information about creating <i>unitprefixObj</i> , see <code>sbiounitprefix</code> .   |

**Description** The function `sbioaddtolibrary` adds kinetic law definitions, units, and unit prefixes to the user-defined library.

`sbioaddtolibrary (abstkineticlawObj)` adds the abstract kinetic law object (*abstkineticlawObj*) to the user-defined library.

`sbioaddtolibrary (unitObj)` adds the user-defined unit (*unitObj*) to the user-defined library.

`sbioaddtolibrary (unitprefixObj)` adds the user-defined unit prefix (*unitprefixObj*) to the user-defined library.

The `sbioaddtolibrary` function adds any kinetic law definition, unit, or unit prefix to the root object's `UserDefinedLibrary` property. These

library components become available automatically in future MATLAB sessions.

Use the kinetic law definitions in the built-in and user-defined library to construct a kinetic law object with the method `addkineticlaw`.

To get a component of the built-in and user-defined libraries, use the commands `get(sbioroot, 'BuiltInLibrary')` and `(get(sbioroot, 'UserDefinedLibrary'))`.

To remove the library component from the user-defined library, use the function `sbioremovefromlibrary`. You cannot remove a kinetic law definition being used by a reaction.

## Examples

This example shows how to create a kinetic law definition and add it to the user-defined library.

- 1 Create a kinetic law definition.

```
abstkineticlawObj = sbioabstractkineticlaw('ex_myLaw1', '(k1*s)/(k2+k1+s)');
```

- 2 Assign the parameter and species variables in the expression.

```
set (abstkineticlawObj, 'SpeciesVariables', {'s'});
set (abstkineticlawObj, 'ParameterVariables', {'k1', 'k2'});
```

- 3 Add the new kinetic law definition to the user-defined library.

```
sbioaddtolibrary(abstkineticlawObj);
```

The function adds the kinetic law definition to the user-defined library. You can verify this using `sbiowhos`.

```
sbiowhos -kineticlaw -userdefined
```

```
SimBiology Abstract Kinetic Law Array
```

| Index: | Library:    | Name:  | Expression:        |
|--------|-------------|--------|--------------------|
| 1      | UserDefined | myLaw1 | $(k1*s)/(k2+k1+s)$ |

- 4 Use the new kinetic law definition when defining a reaction's kinetic law.

```
modelObj = sbiomodel('cell');  
reactionObj = addreaction(modelObj, 'A + B <-> B + C');  
kineticlawObj = addkineticlaw(reactionObj, 'ex_myLaw1');
```

---

**Note** Remember to specify the `SpeciesVariableNames` and the `ParameterVariableNames` in `kineticlawObj` to fully define the `ReactionRate` of the reaction.

---

## See Also

[addkineticlaw](#) | [sbioabstractkineticlaw](#) | [sbioremovefromlibrary](#)  
| [sbioroot](#) | [sbiounit](#) | [sbiounitprefix](#)



## Purpose

Find conserved moieties in SimBiology model

## Syntax

```
[G, Sp] = sbioconsmoiety(mode1Obj)
[G, Sp] = sbioconsmoiety(mode1Obj, alg)
H = sbioconsmoiety(mode1Obj, alg, 'p')
H = sbioconsmoiety(mode1Obj, alg, 'p', FormatArg)
[SI, SD, LO, NR, ND] = sbioconsmoiety(mode1Obj, 'link')
```

## Arguments

|                  |   |
|------------------|---|
| <i>G</i>         | An m-by-n matrix, where m is the number of conserved quantities found and n is the number of species in the model. Each row of <i>G</i> specifies a linear combination of species whose rate of change over time is zero.   |
| <i>Sp</i>        | Cell array of species names that labels the columns of <i>G</i> . If the species are in multiple compartments, species names are qualified with the compartment name in the form <code>compartmentName.speciesName</code> . For example, <code>nucleus.DNA</code> , <code>cytoplasm.mRNA</code> . |
| <i>mode1Obj</i>  | Model object to be evaluated for conserved moieties.  |
| <i>alg</i>       | Specify algorithm to use during evaluation of conserved moieties. Valid values are 'qr', 'rreduce', or 'semipos'.   |
| <i>H</i>         | Cell array of strings containing the conserved moieties.  |
| p                | Prints the output to a cell array of strings.   |
| <i>FormatArg</i> | Specifies formatting for the output <i>H</i> . <i>FormatArg</i> should either be a C-style format string, or a positive integer specifying the maximum number of digits of precision used.  |
| <i>SI</i>        | Cell array containing the names of independent species in the model.  |

|           |  |
|-----------|--|
| <i>SD</i> | Cell array containing the names of dependent species in the model.   |
| <i>LO</i> | Link matrix relating <i>SI</i> and <i>SD</i> . The link matrix <i>LO</i> satisfies $ND = LO * NR$ . For the 'link' functionality, species with their <code>BoundaryCondition</code> or <code>ConstantAmount</code> properties set to true are treated as having stoichiometry of zero in all reactions.<br><br><i>LO</i> is a sparse matrix. To convert it to a full matrix, use the <code>full</code> function. |
| <i>NR</i> | Reduced stoichiometry matrices containing one row for each independent species. The concatenated matrix [ <i>NR</i> ; <i>ND</i> ] is a row-permuted version of the full stoichiometry matrix of <i>modelObj</i> .<br><br><i>NR</i> is a sparse matrix. To convert it to a full matrix, use the <code>full</code> function.   |
| <i>ND</i> | Reduced stoichiometry matrices containing one row for each dependent species. The concatenated matrix [ <i>NR</i> ; <i>ND</i> ] is a row-permuted version of the full stoichiometry matrix of <i>modelObj</i> .<br><br><i>ND</i> is a sparse matrix. To convert it to a full matrix, use the <code>full</code> function.   |

## Description

`[G, Sp] = sbioconsmoiety(modelObj)` calculates a complete set of linear conservation relations for the species in the SimBiology model object *modelObj*.

`sbioconsmoiety` computes conservation relations by analyzing the structure of the model object's stoichiometry matrix. Thus, `sbioconsmoiety` does not include species that are governed by algebraic or rate rules.

`[G, Sp] = sbioconsmoiety(modelObj, alg)` provides an algorithm specification. For *alg*, specify 'qr', 'rreduce', or 'semipos'.

- When you specify 'qr', `sbioconsmoiety` uses an algorithm based on QR factorization. From a numerical standpoint, this is the most efficient and reliable approach.
- When you specify 'rreduce', `sbioconsmoiety` uses an algorithm based on row reduction, which yields better numbers for smaller models. This is the default.
- When you specify 'semipos', `sbioconsmoiety` returns conservation relations in which all the coefficients are greater than or equal to 0, permitting a more transparent interpretation in terms of physical quantities.

For larger models, the QR-based method is recommended. For smaller models, row reduction or the semipositive algorithm may be preferable. For row reduction and QR factorization, the number of conservation relations returned equals the row rank degeneracy of the model object's stoichiometry matrix. The semipositive algorithm may return a different number of relations. Mathematically speaking, this algorithm returns a generating set of vectors for the space of semipositive conservation relations.

`H = sbioconsmoiety(modeObj, alg, 'p')` returns a cell array of strings `H` containing the conserved quantities in `modeObj`.

`H = sbioconsmoiety(modeObj, alg, 'p', FormatArg)` specifies formatting for the output `H`. `FormatArg` should either be a C-style format string, or a positive integer specifying the maximum number of digits of precision used.

`[SI, SD, LO, NR, ND] = sbioconsmoiety(modeObj, 'link')` uses a QR-based algorithm to compute information relevant to the dimensional reduction, via conservation relations, of the reaction network in `modeObj`.

## Examples

### Example 1

This example shows conserved moieties in a cycle.

# sbioconsmoiety

---

- 1 Create a model with a cycle. For convenience use arbitrary reaction rates, as this will not affect the result.

```
modelObj = sbiomodel('cycle');
modelObj.addreaction('a -> b','ReactionRate','1');
modelObj.addreaction('b -> c','ReactionRate','b');
modelObj.addreaction('c -> a','ReactionRate','2*c');
```

- 2 Look for conserved moieties.

```
[g sp] = sbioconsmoiety(modelObj)

g =

     1     1     1

sp =

     'a'
     'b'
     'c'
```

## Example 2

Explore semipositive conservation relations in the oscillator model.

```
modelObj = sbmlimport('oscillator');
sbioconsmoiety(modelObj,'semipos','p')

ans =

     'p01 + p01_0pA + p01_0pB + p01_0pC'
     '0pB + p01_0pB + pA_0pB1 + pA_0pB_pA + pA_0pB2'
     '0pA + p01_0pA + pC_0pA1 + pC_0pA2 + pC_0pA_pC'
     '0pC + p01_0pC + pB_0pC1 + pB_0pC2 + pB_0pC_pB'
```

## See Also

`getstoichmatrix`

**How To**

- “Conserved Moiety Determination”

# sbioconvertunits

---

**Purpose** Convert unit and unit value to new unit

**Syntax** `sbioconvertunits(Obj, 'unit')`

**Description** `sbioconvertunits(Obj, 'unit')` converts the current `*Units` property on SimBiology object, `Obj` to the unit, `unit`. This function configures the `*Units` property to `unit` and updates the corresponding value property. For example, `sbioconvertunits` on a `speciesObj` updates the `InitialAmount` property value and the `InitialAmountUnits` property value.

`Obj` can be an array of SimBiology objects. `Obj` must be a SimBiology object that contains a unit property. The SimBiology objects that contain a unit property are compartment, parameter, and species objects. For example, if `Obj` is a species object with `InitialAmount` configured to 1 and `InitialAmountUnits` configured to mole, after the call to `sbioconvertunits` with `unit` specified as molecule, `speciesObj` `InitialAmount` is 6.0221e23 and `InitialAmountUnits` is molecule.

**Examples** Convert the units of the initial amount of glucose from molecule to mole.

- 1 Create the species 'glucose' and assign an initial amount of 23 molecule.

At the command prompt, type:

```
modelObj = sbiomodel('cell');
compObj = addcompartment(modelObj, 'C');
speciesObj = addspecies(compObj, 'glucose', 23, 'InitialAmountUnits', 'molecule')
```

SimBiology Species Array

| Index: | Compartment: | Name:   | InitialAmount: | InitialAmountUnits: |
|--------|--------------|---------|----------------|---------------------|
| 1      | C            | glucose | 23             | molecule            |

**2** Convert the InitialAmountUnits of glucose from molecule to mole.

```
sbioconvertunits (speciesObj, 'mole')
```

**3** Verify the conversion of units and InitialAmount value.

Units are converted from molecule to mole.

```
get (speciesObj, 'InitialAmountUnits')
```

```
ans =
```

```
mole
```

The InitialAmount value is changed.

```
get (speciesObj, 'InitialAmount')
```

```
ans =
```

```
3.8192e-023
```

## See Also

sbioshowunits

## How To

- sbioshowunits

# sbiocopylibrary

---

**Purpose** Copy library to disk

**Syntax**  
`sbiocopylibrary ('kineticlaw', 'LibraryFileName')`  
`sbiocopylibrary ('unit', 'LibraryFileName')`

**Description** `sbiocopylibrary ('kineticlaw', 'LibraryFileName')` copies all user-defined kinetic law definitions to the file `LibraryFileName.sbklib` and places the copied file in the current directory.

`sbiocopylibrary ('unit', 'LibraryFileName')` copies all user-defined units and unit prefixes to the file `LibraryFileName.sbulib`.

To get the kinetic law definitions that are in the built-in or user-defined libraries, first create a root object using `sbioroot`, then use the commands `get(rootObj.BuiltInLibrary, 'KineticLaws')` or `get(rootObj.UserDefinedLibrary, 'KineticLaws')`.

To add a kinetic law definition to the user-defined library, use `sbioaddtolibrary`.

To add a unit to the user-defined library, use `sbiounit` followed by `sbioaddtolibrary`. To add a unit prefix to the user-defined library, use `sbiounitprefix` followed by `sbioaddtolibrary`.

## Examples

Create a kinetic law definition, add it to the user-defined library, and then copy the user-defined kinetic law library to a `.sbklib` file.

- 1 Create a kinetic law definition.

```
abstkineticlawObj = sbioabstractkineticlaw('mylaw1', '(k1*s)/(k2+k1+s)');
```

- 2 Add the new a kinetic law definition to the user-defined library.

```
sbioaddtolibrary(abstkineticlawObj);
```

`sbioaddtolibrary` adds the kinetic law definition to the user-defined library. You can verify this using `sbiowhos`.



```
sbiowhos -kineticlaw -userdefined
```

```
SimBiology Abstract Kinetic Law Array
```

| Index: | Library:    | Name:  | Expression:        |
|--------|-------------|--------|--------------------|
| 1      | UserDefined | mylaw1 | $(k1*s)/(k2+k1+s)$ |

**3** Copy the user-defined kinetic law library.

```
sbiocopylibrary ('kineticlaw','myLibFile')
```

**4** Verify with sbiowhos.

```
sbiowhos -kineticlaw myLibFile
```

## See Also

```
sbioaddtolibrary | sbioabstractkineticlaw |  
sbioremovefromlibrary | sbiounit | sbiounitprefix
```

# sbiodesktop

---

**Purpose** Open SimBiology desktop for modeling and simulation

**Syntax**  
sbiodesktop  
sbiodesktop(*modelObj*)  
sbiodesktop(*File*)

**Input Arguments**

|                 |  |
|-----------------|--|
| <i>modelObj</i> | SimBiology model object or an array of model objects.  |
| <i>File</i>     | String specifying a file name or path and file name of an sbproj file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder. |

**Description** sbiodesktop opens the SimBiology desktop, which lets you:

- Build a SimBiology model by representing reaction pathways and entering kinetic data for the reactions.
- Import or export SimBiology models to and from the MATLAB workspace or from a Systems Biology Markup Language (SBML) file.
- Modify an existing SimBiology model.
- Simulate a SimBiology model through individual or ensemble runs.
- View results from the simulation.
- Perform analysis tasks such as sensitivity analysis, parameter and species scans, and calculation of conserved moieties.
- Create and/or modify user-defined units and unit prefixes.
- Create and/or modify user-defined kinetic laws.

sbiodesktop(*modelObj*) opens the SimBiology desktop with *modelObj*, a SimBiology model object. If there is a project open in the SimBiology desktop, this command adds *modelObj* to the project.

sbiodesktop(*File*) opens the project specified by *File* in the SimBiology desktop. *File* is a string specifying a file name or path and file name of

an sbproj file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder. If a project is open in the desktop, the software replaces it with the new project, after prompting you to save any changes.

The Parent property of a SimBiology model object is set to the SimBiology root object. The root object contains a list of model objects that are accessible from the MATLAB command line and from the SimBiology desktop. Because both the command line and the desktop point to the same model object in the `Root` object, changes you make to the model at the command line are reflected in the desktop, and vice versa.

---

**Note** The `sbioreset` command removes all models from the root object. Therefore, this command also removes all models from the SimBiology desktop.

---

## Examples

Create a SimBiology model in the MATLAB workspace, and then open the SimBiology desktop with the model:

```
modelObj = sbiomodel('cell');  
sbidesktop(modelObj)
```

## See Also

`sbioroot` | `sbiofittool` | `simbiology`

# sbiodose

---

**Purpose** Construct dose object

**Syntax**

```
doseObj = sbiodose('DoseName')  
doseObj = sbiodose('DoseName', 'DoseType')  
doseObj = sbiodose(...'PropertyName', PropertyValue...)
```

## Inputs

|                 |   |
|-----------------|---|
| <i>DoseName</i> | Name of the dose object.  |
| <i>DoseType</i> | Selects which type of dose object to construct. Enter either 'schedule' or 'repeat' |

- 'schedule' creates a ScheduleDose object and defines the dose with a time array, amount array, and rate array.
- 'repeat' creates a RepeatDose object and defines the dose with a dose amount, number of dose repetitions, and the time between doses.

## Output

**Arguments** *doseObj* ScheduleDose or RepeatDose object.

## Description

*doseObj* = sbiodose('DoseName') constructs a SimBiology RepeatDose object (*doseObj*), assigns *DoseName* to the property Name, and assigns [] to the property Parent.

*doseObj* = sbiodose('DoseName', 'DoseType') constructs either a SimBiology ScheduleDose object or RepeatDose object (*doseObj*).

*doseObj* = sbiodose(...'PropertyName', PropertyValue...) defines dose object properties. You can enter the property name/property value pairs in any format supported by the function set (for example, name-value string pairs, structures, and name-value cell array pairs).

You can view additional *doseObj* properties with the `get` command and modify *doseObj* properties with the `set` command.

Before you use a dose object in a simulation, you must add the object to a SimBiology model with the method `adddose`.

## Examples

Construct a repeat dose object:

- 1 In the MATLAB Command Window, enter:

```
doseObj1 = sbiodose('dose1', 'repeat');
```

- 2 Define a repeating dose:

```
doseObj1.Amount      = 5;  
doseObj1.Repeat      = 6;  
doseObj1.Interval    = 24;  
doseObj1.TimeUnits   = 'hour';  
doseObj1.TargetName  = 'Central.x';
```

---

Construct a schedule dose object:

- 1 In the MATLAB Command Window, enter:

```
doseObj2 = sbiodose('dose2', 'schedule');
```

- 2 Define a dose schedule:

```
doseObj2.Time        = [0 24 48];  
doseObj2.Amount      = [10 5 5];  
doseObj2.TargetName  = 'Central.Drug';
```

## See Also

`adddose` | `getdose` | `removedose` | `copyobj` | `get` | `set`

## How To

- Model object
- ScheduleDose object
- RepeatDose object

# sbioensembleplot

---

**Purpose** Show results of ensemble run using 2-D or 3-D plots

**Syntax**

```
sbioensembleplot(simdataObj)
sbioensembleplot(simdataObj, Names)
sbioensembleplot(simdataObj, Names, Time)
FH = sbioensembleplot(simdataObj, Names)
FH = sbioensembleplot(simdataObj, Names, Time)
```

## Arguments

|                   |  |
|-------------------|--|
| <i>simdataObj</i> | An object that contains simulation data. You can generate a <i>simdataObj</i> object using the function <code>sbioenssemblerun</code> . All elements of <i>simdataObj</i> must contain data for the same states in the same model.   |
| <i>Names</i>      | Either a string or a cell array of strings. <i>Names</i> may include qualified names such as ' <i>CompartmentName.SpeciesName</i> ' or ' <i>ReactionName.ParameterName</i> ' to resolve ambiguities. Specifying {} for <i>Names</i> plots data for all states contained in <i>simdataObj</i> . |
| <i>Time</i>       | A numeric scalar value. If the specified <i>Time</i> is not an element of the time vectors in <i>simdataObj</i> , then the function resamples <i>simdataObj</i> as necessary using linear interpolation.   |
| <i>FH</i>         | Array of handles to figure windows.  |

## Description

`sbioensembleplot(simdataObj)` shows a 3-D shaded plot of time-varying distribution of all logged states in the SimData array *simdataObj*. The `sbioenssemblerun` function plots an approximate distribution created by fitting a normal distribution to the data at every time step.

`sbioensembleplot(simdataObj, Names)` plots the distribution for the data specified by *Names*.

`sbioensembleplot(simdataObj, Names, Time)` plots a 2-D histogram of the actual data of the ensemble distribution of the states specified by *Names* at the particular time point *Time*.

`FH = sbioensembleplot(simdataObj, Names)` returns an array of handles *FH*, to the figure window for the 3-D distribution plot.

`FH = sbioensembleplot(simdataObj, Names, Time)` returns an array of handles *FH*, to the figure window for the 2-D histograms.

## Examples

This example shows how to plot data from an ensemble run without interpolation.

- 1 The project file, `radiodecay.sbproj`, contains a model stored in a variable called `m1`. Load `m1` into the MATLAB workspace.

```
sbioloadproject('radiodecay.sbproj', 'm1');
```

- 2 Change the solver of the active configuration set to be `ssa`. Also, adjust the `LogDecimation` property on the `SolverOptions` property of the configuration set to reduce the size of the data generated.

```
cs = getconfigset(m1, 'active');
set(cs, 'SolverType', 'ssa');
so = get(cs, 'SolverOptions');
set(so, 'LogDecimation', 10);
```

- 3 Perform an ensemble of 20 runs with no interpolation.

```
simdataObj = sbioensemblerrun(m1, 20);
```

- 4 Create a 2-D distribution plot of the species 'z' at time = 1.0.

```
FH1 = sbioensembleplot(simdataObj, 'z', 1.0);
```

- 5 Create a 3-D shaded plot of both species.

```
FH2 = sbioensembleplot(simdataObj, {'x', 'z'});
```

## See Also

[sbioensemblerrun](#) | [sbioensemblestats](#) | [sbiomodel](#)

# sbioensemblerun

---

## Purpose

Multiple stochastic ensemble runs of SimBiology model

## Syntax

```
simdataObj = sbioensemblerun(modelObj, Numruns)  
simdataObj = sbioensemblerun(modelObj, Numruns, Interpolation)  
simdataObj = sbioensemblerun(modelObj, Numruns, configsetObj)  
simdataObj = sbioensemblerun(modelObj, Numruns, configsetObj,  
    Interpolation)  
simdataObj = sbioensemblerun(modelObj, Numruns, variantObj)  
simdataObj = sbioensemblerun(modelObj, Numruns, variantObj,  
    Interpolation)  
simdataObj = sbioensemblerun(modelObj, Numruns, configsetObj,  
    variantObj)  
simdataObj = sbioensemblerun(modelObj, Numruns, configsetObj,  
    variantObj, Interpolation)
```

## Arguments

|                      |  |
|----------------------|--|
| <i>simdataObj</i>    | An array of SimData objects containing simulation data generated by sbioensemblerun. All elements of <i>simdataObj</i> contain data for the same states in the same model.   |
| <i>modelObj</i>      | Model object to be simulated.  |
| <i>Numruns</i>       | Integer scalar representing the number of stochastic runs to make.   |
| <i>Interpolation</i> | String variable denoting the interpolation scheme to be used if data should be interpolated to get a consistent time vector. Valid values are 'linear' (linear interpolation), 'zoh' (zero-order hold), or 'off' (no interpolation). Default is 'off'. If interpolation is on, the data is interpolated to match the time vector with the smallest simulation stop time. |



- configsetObj* Specify the configuration set object to use in the ensemble simulation. For more information about configuration sets, see `Configset` object.
- variantObj* Specify the variant object to apply to the model during the ensemble simulation. For more information about variant objects, see `Variant` object.

## Description

`simdataObj = sbioensemblerun(modelObj, Numruns)` performs a stochastic ensemble run of the SimBiology model object (`modelObj`), and returns the results in `simdataObj`, an array of `SimData` objects. The active `configset` and the active variants are used during simulation and are saved in the output, `SimData` object (`simdataObj`).

`sbioensemblerun` uses the settings in the active `configset` on the model object (`modelObj`) to perform the repeated simulation runs. The `SolverType` property of the active `configset` must be set to one of the stochastic solvers: 'ssa', 'expltau', or 'impltau'. `sbioensemblerun` generates an error if the `SolverType` property is set to any of the deterministic (ODE) solvers.

`simdataObj = sbioensemblerun(modelObj, Numruns, Interpolation)` performs a stochastic ensemble run of a model object (`modelObj`), and interpolates the results of the ensemble run onto a common time vector using the interpolation scheme (`Interpolation`).

`simdataObj = sbioensemblerun(modelObj, Numruns, configsetObj)` performs an ensemble run of a model object (`modelObj`), using the specified configuration set (`configsetObj`).

`simdataObj = sbioensemblerun(modelObj, Numruns, configsetObj, Interpolation)` performs an ensemble run of a model object (`modelObj`), using the specified configuration set (`configsetObj`), and interpolates the results of the ensemble run onto a common time vector using the interpolation scheme (`Interpolation`).

`simdataObj = sbioensemblerun(modelObj, Numruns, variantObj)` performs an ensemble run of a model object (*modelObj*), using the variant object or array of variant objects (*variantObj*).

`simdataObj = sbioensemblerun(modelObj, Numruns, variantObj, Interpolation)` performs an ensemble run of a model object (*modelObj*), using the variant object or array of variant objects (*variantObj*), and interpolates the results of the ensemble run onto a common time vector using the interpolation scheme (*Interpolation*).

`simdataObj = sbioensemblerun(modelObj, Numruns, configsetObj, variantObj)` performs an ensemble run of a model object (*modelObj*), using the configuration set (*configsetObj*), and the variant object or array of variant objects (*variantObj*). If the configuration set object (*configsetObj*) is empty, the active configset on the model is used for simulation. If the variant object (*variantObj*) is empty, then no variant (not even the active variants in the model) is used for the simulation.

`simdataObj = sbioensemblerun(modelObj, Numruns, configsetObj, variantObj, Interpolation)` performs an ensemble run of a model object (*modelObj*), using the configuration set (*configsetObj*), and the variant object or array of variant objects (*variantObj*), and interpolates the results of the ensemble run onto a common time vector using the interpolation scheme (*Interpolation*).

## Examples

This example shows how to perform an ensemble run and generate a 2-D distribution plot.

- 1 The project file, `radiodecay.sbproj`, contains a model stored in a variable called `m1`. Load `m1` into the MATLAB workspace.

```
sbioloadproject('radiodecay.sbproj','m1');
```

- 2 Change the solver of the active configset to be `ssa`. Also, adjust the `LogDecimation` property on the `SolverOptions` property of the configuration set.

```
cs = getconfigset(m1, 'active');  
set(cs, 'SolverType', 'ssa');
```

```
so = get(cs, 'SolverOptions');  
set(so, 'LogDecimation', 10);
```

---

**Tip** The `LogDecimation` property lets you define how often the simulation data is recorded as output. If your model has high concentrations or amounts of species, or a long simulation time (for example, 600s), you can record simulation data less often to manage the amount of data generated. Be aware that by doing so you might miss some transitions if your model is very dynamic. Try setting `LogDecimation` to 10 or more.

---

- 3 Perform an ensemble of 20 runs with linear interpolation to get a consistent time vector.

```
simdata = sbioensemblerun(m1, 20, 'linear');
```

- 4 Create a 2-D distribution plot of the species 'z' at a time = 1.0.

```
FH = sbioensembleplot(simdata, 'z', 1.0);
```

## See Also

[addconfigset](#) | [getconfigset](#) | [sbioensemblestats](#) | [sbioensembleplot](#) | [setactiveconfigset](#) | [SimData](#) object

# sbioensemblestats

---

**Purpose** Get statistics from ensemble run data

**Syntax**  
`[t,m] = sbioensemblestats(simDataObj)`  
`[t,m,v] = sbioensemblestats(simDataObj)`  
`[t,m,v,n] = sbioensemblestats(simDataObj)`

## Arguments

|                   |  |
|-------------------|--|
| <i>t</i>          | Vector of doubles that holds the common time vector after interpolation.   |
| <i>m</i>          | Matrix of mean values from the ensemble data. The number of rows in <i>m</i> is the length of the common time vector <i>t</i> after interpolation and the number of columns is equal to the number of species. The species order corresponding to the columns of <i>m</i> can be obtained from any of the SimData objects in <i>simDataObj</i> using <code>selectbyname</code> . |
| <i>simDataObj</i> | A cell array of SimData objects, where each SimData object holds data for a separate simulation run. All elements of <i>simDataObj</i> must contain data for the same states in the same model. When the time vectors of the elements of <i>simDataObj</i> are not identical, <i>simDataObj</i> is first resampled onto a common time vector (see <i>interpolation</i> below).   |
| <i>v</i>          | Matrix of variance obtained from the ensemble data. <i>v</i> has the same dimensions as <i>m</i> .   |
| <i>n</i>          | Cell array of strings that holds names whose mean and variance are returned in <i>m</i> and <i>v</i> , respectively. The number of elements in <i>n</i> is the same as the number of columns of <i>m</i> and <i>v</i> . The order of names in <i>n</i> corresponds to the order of columns of <i>m</i> and <i>v</i> .  |

*names* Either a string or a cell array of strings. *names* may include qualified names such as '*CompartmentName.SpeciesName*' or '*ReactionName.ParameterName*' to resolve ambiguities. If you specify empty {} for *names*, *sbioensemblestats* returns statistics on all time courses contained in *simDataObj*.

*interpolation* String variable denoting the interpolation method to be used if data is to be interpolated to get a consistent time vector. See *resample* for a list of interpolation methods. Default is 'off'. If interpolation is on, the data is interpolated to match the time vector with the smallest simulation stop time.

## Description

`[t,m] = sbioensemblestats(simDataObj)` computes the time-dependent ensemble mean *m* of the ensemble data *simDataObj* obtained by running *sbioenssemblerun*.

`[t,m,v] = sbioensemblestats(simDataObj)` computes the time-dependent ensemble mean *m* and variance *v* for the ensemble run data *simDataObj*.

`[t,m,v,n] = sbioensemblestats(simDataObj)` computes the time-dependent ensemble mean *m* and variance *v* for the ensemble run data *simDataObj*. Each column of *m* or *v* describes the ensemble mean or variance of some state as a function of time.

## Examples

The project file, *radiodecay.sbproj*, contains a model stored in a variable called *m1*. Load *m1* into the MATLAB workspace.

- 1 Load a SimBiology model *m1* from a SimBiology project file.

```
sbioloadproject('radiodecay.sbproj','m1');
```

- 2 Change the solver of the active configuration set to be `ssa`. Also, adjust the `LogDecimation` property on the `SolverOptions` property of the configuration set.

```
cs = getconfigset(m1, 'active');  
set(cs, 'SolverType', 'ssa');  
so = get(cs, 'SolverOptions');  
set(so, 'LogDecimation', 10);
```

- 3 Perform an ensemble of 20 runs with no interpolation.

```
simDataObj = sbioensemblerrun(m1, 20);
```

- 4 Get ensemble statistics for all species using the default interpolation method.

```
[T,M,V] = sbioensemblestats(simDataObj);
```

- 5 Get ensemble statistics for a specific species using the default interpolation scheme.

```
[T2,M2,V2] = sbioensemblestats(simDataObj, {'z'});
```

## See Also

`sbioensemblerrun` | `sbioensembleplot` | `sbioemodel`

## Purpose

Perform nonlinear least-squares regression

## Syntax

```
fitResults = sbiofit(sm,grpData,responseMap,estiminfo)
fitResults =
sbiofit(sm,grpData,responseMap,estiminfo,dosing)
fitResults =
sbiofit(sm,grpData,responseMap,estiminfo,dosing,
        functionName)
fitResults =
sbiofit(sm,grpData,responseMap,estiminfo,dosing,
        functionName,options)
fitResults =
sbiofit(sm,grpData,responseMap,estiminfo,dosing,
        functionName,options,variants)
fitResults = sbiofit(_,Name,Value)

[fitResults,simdata] = sbiofit(_)
```

## Description

`fitResults = sbiofit(sm,grpData,responseMap,estiminfo)` estimates parameters of a SimBiology model `sm` using nonlinear least-squares regression.

`grpData` is a `groupedData` object specifying the data to fit. `responseMap` is a string or cell array of strings that maps model components to response data in `grpData`. `estimatedInfo` is an `estimatedInfo` object that defines the estimated parameters in the model `sm`.

If Statistics Toolbox™ is available, the function uses `nlinfit` for estimation as the default algorithm. If not, but Optimization Toolbox™ is available, `fminunc` is used. If both are unavailable, `fminsearch` is used.

If the model contains active doses and variants, they are applied during the simulation.

`fitResults = sbiofit(sm,grpData,responseMap,estiminfo,dosing)` uses the

dosing information specified by a matrix of SimBiology dose objects `dosing` instead of using the active doses of the model `sm` if there is any.

```
fitResults =  
sbiofit(sm,grpData,responseMap,estiminfo,dosing,  
functionName)
```

uses the estimation function specified by `functionName`. If the specified function is unavailable, a warning is issued and the first available default function is used.

```
fitResults =  
sbiofit(sm,grpData,responseMap,estiminfo,dosing,  
functionName,options)
```

uses the additional options specified by `options` for the function `functionName`.

```
fitResults =  
sbiofit(sm,grpData,responseMap,estiminfo,dosing,  
functionName,options,variants)
```

applies variant objects specified as `variants` instead of using any active variants of the model.

```
fitResults = sbiofit(_,Name,Value)
```

uses additional options specified by one or more `Name,Value` pair arguments.

```
[fitResults,simdata] = sbiofit(_)
```

returns a vector of result objects `fitResults` and a vector of SimData objects `simdata` using any of the input arguments in the previous syntaxes.

---

**Note** `sbiofit` unifies `sbionlinfit` and `sbioparamestim` estimation functions. Use `sbiofit` to perform nonlinear least-squares regression.

---



## Input Arguments

### **sm - SimBiology model**

SimBiology model object

SimBiology model, specified as a `SimBiology model` object. The active `configset` object of the model contains solver settings for simulation. Any active doses and variants are applied to the model during simulation unless specified otherwise using the `dosing` and `variants` input arguments, respectively.

### **grpData - Data to fit**

groupedData object

Data to fit, specified as a `groupedData` object.

The name of the time variable must be defined in the `IndependentVariableName` property of `grpData`. For instance, if the time variable name is 'TIME', then specify it as follows.

```
grpData.Properties.IndependentVariableName = 'TIME';
```

If the data contains more than one group of measurements, the grouping variable name must be defined in the `GroupVariableName` property of `grpData`. For example, if the grouping variable name is 'GROUP', then specify it as follows.

```
grpData.Properties.GroupVariableName = 'GROUP';
```

A group usually refers to a set of measurements that represent a single time course, often corresponding to a particular individual or experimental condition.

### **responseMap - Mapping information of model components to grpData**

string | cell array of strings

Mapping information of model components to `grpData`, specified as a string or cell array of strings.

Each string is an equation-like expression, similar to assignment rules in SimBiology. It contains the name (or qualified name) of a quantity

(species, compartment, or parameter) in the model `sm`, followed by the character '=' and the name of a variable in `grpData`. For clarity, white spaces are allowed between names and '='.

For example, if you have the concentration data 'CONC' in `grpData` for a species 'Drug\_Central', you can specify it as follows.

```
responseMap = 'Drug_Central = CONC';
```

To unambiguously name a species, use the qualified name, which includes the name of the compartment. To name a reaction-scoped parameter, use the reaction name to qualify the parameter. If the name is not a valid MATLAB variable name, surround it by square brackets such as `[reaction 1].[parameter 1]`.

An error is issued if any (qualified) name matches two components of the same type. However, you can use a (qualified) name that matches two components of different types, and the function first finds the species with the given name, followed by compartments and then parameters.

### **estiminfo - Estimated parameters**

`estimatedInfo` object | vector of `estimatedInfo` objects

Estimated parameters, specified as an `estimatedInfo` object or vector of `estimatedInfo` objects that defines the estimated parameters in the model `sm` and their initial estimates (optional). You can also specify parameter transformations if necessary. Supported transforms are `log`, `logit`, and `probit`. For details, see `estimatedInfo` object.

### **dosing - Dosing information**

`[]` | 2-D matrix of dose objects

Dosing information, specified as an empty array or 2-D matrix of dose objects (`ScheduleDose` object or `RepeatDose` object). If empty, no doses are applied during simulation, even if the model has active doses. If not empty, the matrix must have a single row or one row per group in the data `grpData`. If it has a single row, same doses are applied to all groups during simulation. If it has multiple rows, each row is applied to a separate group, in the same order as the groups appear in `grpData`.

Multiple columns are allowed so that you can apply multiple dose objects to each group. Each column of doses must reference the same components in the model `sm`. Specifically, all doses (except for empty doses) in a column must have the same values for `TargetName`, `DurationParameterName`, and `LagParameterName`. If some groups require more doses than others, then fill in the matrix with dummy doses that are either default doses or empty doses.

A default dose has default values for all properties, except for the `Name` property. An empty dose has a dose amount of 0, thus having no effect on the model. Create a default dose as follows.

```
d1 = sbiodose('d1');
```

In addition to manually constructing dose objects, if `grpData` has dosing information, you can use the `createDoses` method to construct doses.

### **functionName - Estimation function name**

string

Estimation function name, specified as a string. Choices are as follows.

- 'nlinfit' (Statistics Toolbox is required.)
- 'fminunc' (Optimization Toolbox is required.)
- 'fminsearch'
- 'lsqcurvefit' (Optimization Toolbox is required.)
- 'lsqnonlin' (Optimization Toolbox is required.)
- 'patternsearch' (Global Optimization Toolbox is required.)
- 'ga' (Global Optimization Toolbox is required.)
- 'pso' (Global Optimization Toolbox is required.)

If Statistics Toolbox is available, the function uses the `nlinfit` algorithm for estimation as the default. If not, but Optimization Toolbox is available, `fminunc` is used. If both are unavailable, `fminsearch` is used as the default.

## **options - Options specific to estimation function**

struct | optimoptions object

Options specific to the estimation function, specified as a struct or optimoptions object.

- optimset struct for fminsearch
- optimoptions object for lsqcurvefit, lsqnonlin, fminunc, and pso
- gaoptimset struct for ga
- psoptimset struct for patternsearch
- statset struct for nlinfit

See “Default Options for Estimation Algorithms” on page 1-61 for more details and default options associated with each estimation function.

## **variants - Variants**

[] | vector of variant objects

Variants, specified as an empty array or vector of variant objects. If empty, no variants are used even if the model has active variants.

## **Name-Value Pair Arguments**

Specify optional comma-separated pairs of **Name**, **Value** arguments. **Name** is the argument name and **Value** is the corresponding value. **Name** must appear inside single quotes ( ' '). You can specify several name and value pair arguments in any order as **Name1**, **Value1**, ..., **NameN**, **ValueN**.

**Example:** 'ErrorModel', 'constant', 'UseParallel', true specifies a constant error model and to run simulations in parallel during parameter estimation.

### **'ErrorModel' - Error model**

'constant' (default) | 'proportional' | 'combined' | 'exponential'

Error model used for estimation, specified as a string. There are four built-in error models. Each model defines the error using a standard

mean-zero and unit-variance (Gaussian) variable  $e$ , simulation results  $f$ , and one or two parameters  $a$  and  $b$ .

- 'constant':  $y = f + ae$
- 'proportional':  $y = f + b|f|e$
- 'combined':  $y = f + (a+b|f|)e$
- 'exponential':  $y = f * \exp(ae)$

---

**Note** If you specify an error model, you cannot specify weights except for the constant error model.

---

### **'Weights' - Weights**

matrix | function handle

Weights used for estimation, specified as a matrix of real positive weights, where the number of columns corresponds to the number of responses, or a function handle that accepts a vector of predicted response values and returns a vector of real positive weights.

If you specify an error model, you cannot use weights except for the constant error model. If neither the 'ErrorModel' or 'Weights' is specified, by default the function uses the constant error model with equal weights.

### **'Pooled' - Fit option flag**

false (default) | true

Fit option flag, specifying whether to fit each individual (false) or pool all individual data (true).

When true, the function performs fitting for all individuals or groups simultaneously using the same parameter estimates, and `fitResults` is a scalar results object.

When `false`, the function fits each group or individual separately using group- or individual-specific parameters, and `fitResults` is a vector of results objects with one result for each group.

### **'UseParallel' - Flag for parallel simulations**

`false` (default) | `true`

Flag for parallel simulations during fitting, specified as `true` or `false`. If `true` and Parallel Computing Toolbox™ is available, the function runs simulations in parallel.

### **Output Arguments**

#### **fitResults - Estimation results**

`OptimResults` object | `NLINResults` object | Vector of results objects

Estimation results, returned as a `OptimResults` object or `NLINResults` object or a vector of these objects. Both results objects are subclasses of the `LeastSquaresResults` object.

If the function uses `nlinfit`, then `fitResults` is a `NLINResults` object. Otherwise, `fitResults` is an `OptimResults` object.

When 'Pooled' is set to `false`, the function fits each group separately using group-specific parameters, and `fitResults` is a vector of results objects with one results object for each group.

#### **simdata - Simulation results**

Vector of `SimData` objects

Simulation results, returned as a vector of `SimData` objects representing simulation results for each group or individual.

If the 'Pooled' option is `false`, then each simulation uses group-specific parameter values. If `true`, then all simulations use the same (population-wide) parameter values.

The states reported in `simdata` are the states that were included in the `responseMap` input argument as well as any other states listed in

the `StatesToLog` property of the runtime options (`RuntimeOptions`) of the SimBiology model `sm`.

## Examples

### Estimate a Parameter from the Yeast G protein Model

This example uses the yeast heterotrimeric G protein model and experimental data reported by [1]. For details about the model, see the **Background** section in “Parameter Scanning, Parameter Estimation, and Sensitivity Analysis in the Yeast Heterotrimeric G Protein Cycle”.

Load the G protein model.

```
sbioloadproject gprotein
```

Store the experimental data containing the time course for the fraction of active G protein as reported in the reference paper [1].

```
time = [0 10 30 60 110 210 300 450 600]';  
GaFracExpt = [0 0.35 0.4 0.36 0.39 0.33 0.24 0.17 0.2]';
```

Create a `groupedData` object based on the experimental data.

```
tbl = table(time,GaFracExpt);  
grpData = groupedData(tbl);
```

Map the appropriate model component to the experimental data. In other words, indicate which species in the model corresponds to which response variable in the data. In this example, map the model parameter `GaFrac` to the experimental data variable `GaFracExpt` from `grpData`.

```
responseMap = 'GaFrac = GaFracExpt';
```

Use an `estimatedInfo` object to define the model parameter `kGd` as a parameter to be estimated.

```
estimatedParam = estimatedInfo('kGd');
```

Perform the parameter estimation.

```
fitResult = sbiofit(m1,grpData,responseMap,estimatedParam);
```

View the estimated parameter value of kGd.

```
fitResult.ParameterEstimates
```

```
ans =
```

| Name  | Estimate | StandardError |
|-------|----------|---------------|
| 'kGd' | 0.11     | 0.00037969    |

## Definitions

### Objective Functions

The function `sbiofit` uses maximum likelihood estimation (MLE) for fitting. Mathematically, the problem is typically formulated as the minimization of the negative of the logarithm of the likelihood function. For the constant error model, this is equivalent to the method of least-squares, which minimizes the sum of squares of the residuals. For other error models, the objective function has additional terms. When the error models are normally distributed, the negative log-likelihood can be written as follows.

$$-\ln L = \sum_i \frac{(y_i - f(x_i; p))^2}{2\sigma_i^2} + \sum_i \ln \sqrt{2\pi\sigma_i^2}$$

- $L$             Log likelihood
- $N$             Number of experimental observations
- $y_i$           The  $i$ th experimental observation



$f(x_i; p)$  Predicted value of the  $i$ th observation, which is a function of independent variables  $x_i$  and estimated parameters  $p$

$\sigma_i$  Standard deviation of the  $i$ th observation. For instance, for the combined error model  $y = f + (a + b|f|)e$ , the standard deviation is  $\sigma = a + b|f|$ .

When you use numeric weights or the weight function, the weights are assumed to be proportional to the variance of the error, i.e.,  $\sigma_i^2 = a^2 \cdot w_i$  where  $a$  is the constant error parameter. If you use weights, you cannot specify an error model except the constant error model.

Various parameter estimation methods have different requirements on the objective function. For some methods, the estimation of model parameters is performed independently of the estimation of the error model parameters. The following table summarizes the objective function  $F$  and any separate formulas used for the estimation of error model parameters.  $r_i$  is the  $i$ th residual,  $a$  and  $b$  are error model parameters, and  $f_i$  is the predicted value of the  $i$ th observation.

| Method    | Supported Error Model     | Objective Function                 | Error Parameter Estimation Function        |
|-----------|---------------------------|------------------------------------|--|
| lsqnonlin | constant                  | $\bar{F} = r_i$                    | $a^2 = \frac{1}{N} \sum \frac{r_i^2}{w_i}$ |
| lsqnonlin | numeric weights ( $w_i$ ) | $\bar{F} = \frac{r_i}{\sqrt{w_i}}$ | $a^2 = \frac{1}{N} \sum \frac{r_i^2}{w_i}$ |
| lsqnonlin | exponential               | $\bar{F} = \ln r_i$                | $a^2 = \frac{1}{N} \sum \frac{r_i^2}{w_i}$ |

(Continued)

| Method     | Supported Error Model | Objective Function  | Error Parameter Estimation Function  |
|------------|-----------------------|---|--|
| nlinfit    | All                   | $\bar{F} = f_i$   | nlinfit also estimates error parameters.   |
| All others | 'constant'            | $F = -\ln L$  | $a^2 = \frac{1}{N} \sum \frac{r_i^2}{w_i}$   |
| All others | 'proportional'        | $F = -\ln L$  | $b^2 = \frac{1}{N} \sum \left( r_i / f_i \right)^2$  |
| All others | 'combined'            | $F = -\ln L$  | Error parameters are included in minimization.   |
| All others | 'exponential'         | $F = \sum_i \frac{(\ln y_i - \ln f(x_i; p))^2}{2a^2}$ ,<br>where $a$ is the standard deviation of the log-transformed $i$ th observation. | $\sum_i \ln \sqrt{2\pi a^2} = \frac{N}{2} \ln \frac{2\pi a^2}{N} + \frac{1}{2} \sum_i \frac{r_i^2}{w_i}$ |
| All others | Weights               | $F = -\ln L$  | $a^2 = \frac{1}{N} \sum \frac{r_i^2}{w_i}$   |

### Default Options for Estimation Algorithms

| Function    | Default Options  |
|-------------|--|
| nlinfit     | <p>sbiofit uses the default options structure associated with nlinfit, except for:</p> <p>FunValCheck = 'off'<br/>           DerivStep = max(eps^(1/3), min(1e-4, solverOptions.RelativeTolerance)), where solverOptions property corresponds to the model's active configset object.</p>  |
| fminsearch  | <p>sbiofit uses the default options structure associated with fminsearch, except for:</p> <p>Display = 'off'<br/>           TolFun = 1e-6 * f0, where f0 is the initial value of objective function.</p>   |
| lsqcurvefit | <p>Requires Optimization Toolbox.</p> <p>sbiofit uses the default options structure associated with lsqcurvefit, except for:</p> <p>Display = 'off'<br/>           TolFun = 1e-6 * norm(f0), where f0 is the initial value of objective function.<br/>           FinDiffRelStep = max(eps^(1/3), min(1e-4, solverOptions.RelativeTolerance)), where solverOptions property corresponds to the model's active configset object.<br/>           TypicalX = 1e-6 * x0, where x0 is an array of transformed initial estimates.</p> |

(Continued)

| Function      | Default Options   |
|---------------|---|
| lsqnonlin     | <p>Requires Optimization Toolbox.</p> <p>sbiofit uses the default options structure associated with lsqnonlin, except for:</p> <p>Display = 'off'</p> <p>TolFun = 1e-6 * norm(f0), where f0 is the initial value of objective function.</p> <p>FinDiffRelStep = max(eps^(1/3), min(1e-4, solverOptions.RelativeTolerance)) , where solverOptions property corresponds to the model's active configset object.</p> <p>TypicalX = 1e-6 * x0, where x0 is an array of transformed initial estimates.</p> |
| patternsearch | <p>Requires Global Optimization Toolbox.</p> <p>sbioparamestim uses the default options structure associated with patternsearch, except for:</p> <p>Display = 'off'</p> <p>TolFun = 1e-6 * f0, where f0 is the initial value of the objective function.</p> <p>TolMesh = 1.0e-3</p> <p>Cache = 'on'</p> <p>MeshAccel = 'on'</p>   |

**(Continued)**

| <b>Function</b> | <b>Default Options</b>   |
|-----------------|--|
| ga              | <p>Requires Global Optimization Toolbox.</p> <p>sbioparamestim uses the default options structure associated with ga, except for:</p> <p>Display = 'off'</p> <p>TolFun = 1e-6 * f0, where f0 is the initial value of objective function.</p> <p>MutationFcn = @mutationadaptfeasible</p>   |
| pso             | <p>Requires Global Optimization Toolbox.</p> <p>sbioparamestim uses the following default options for the pso (particle swarm optimization) algorithm, except for:</p> <p>Display = 'off'</p> <p>TolFun = 1e-6 * f0, where f0 is the initial value of objective function.</p> <p>See the Function Descriptions table from the sbioparamestim reference page for more details about this algorithm.</p> |

## References

[1] Yi, T-M., Kitano, H., and Simon, M. (2003). A quantitative characterization of the yeast heterotrimeric G protein cycle. PNAS. 100, 10764–10769.

## See Also

estimatedInfo object | groupedData object |  
 LeastSquaresResults object | NLINResults object |

OptimResults object | sbiofitmixed | nlinfit | fminunc |  
fminsearch | lsqcurvefit | lsqnonlin | patternsearch | ga | pso

## **Related Examples**

- “Parameter Scanning, Parameter Estimation, and Sensitivity Analysis in the Yeast Heterotrimeric G Protein Cycle”

**Purpose** Fit nonlinear mixed-effects model

**Syntax**

```

fitResults =
sbiofitmixed(sm,grpData,responseMap,covEstiminfo)
fitResults =
sbiofitmixed(sm,grpData,responseMap,covEstiminfo,dosing)
fitResults =
sbiofitmixed(sm,grpData,responseMap,covEstiminfo,dosing,
    functionName)
fitResults =
sbiofitmixed(sm,grpData,responseMap,covEstiminfo,dosing,
    functionName,opt)
fitResults =
sbiofitmixed(sm,grpData,responseMap,covEstiminfo,dosing,
    functionName,opt,variants)
fitResults = sbiofitmixed(_, 'UseParallel', tf)

[fitResults,simDataI,simDataP] = sbiofitmixed(_)
    
```

## Description

---

**Note** This function requires `nlmefit` and `nlmefitsa` in Statistics Toolbox.

---

`fitResults` = `sbiofitmixed(sm,grpData,responseMap,covEstiminfo)` performs nonlinear mixed-effects estimation using the SimBiology model `sm` and returns a `NLMEResults` object `fitResults`.

`grpData` is a `groupedData` object specifying the data to fit. `responseMap` is a string or cell array of strings that maps model components to response data in `grpData`. `covEstiminfo` is a `CovariateModel` object or an array of `estimatedInfo` objects that defines the parameters to be estimated.

If the model contains active doses and variants, they are applied during the simulation.

# sbiofitmixed

---

`fitResults = sbiofitmixed(sm,grpData,responseMap,covEstiminfo,dosing)` uses the dosing information specified by a matrix of SimBiology dose objects `dosing` instead of using the active doses of the model `sm` if there are any.

`fitResults = sbiofitmixed(sm,grpData,responseMap,covEstiminfo,dosing,functionName)` uses the estimation function specified by `functionName` that must be either `'nlmefit'` or `'nlmefitsa'`.

`fitResults = sbiofitmixed(sm,grpData,responseMap,covEstiminfo,dosing,functionName,opt)` uses the additional options specified by `opt` for the estimation function `functionName`.

`fitResults = sbiofitmixed(sm,grpData,responseMap,covEstiminfo,dosing,functionName,opt,variants)` applies variant objects specified as `variants` instead of using any active variants of the model.

`fitResults = sbiofitmixed(_, 'UseParallel', tf)` provides an option to estimate parameters in parallel if Parallel Computing Toolbox is available.

`[fitResults,simDataI,simDataP] = sbiofitmixed(_)` returns a vector of results objects `fitResults`, vector of simulation results `simDataI` using individual-specific parameter estimates, and vector of simulation results `simDataP` using population parameter estimates.



---

**Note** `sbiofitmixed` unifies `sbionlmeffit` and `sbionlmefitsa` estimation functions. Use `sbiofitmixed` to perform nonlinear mixed-effects modeling and estimation.

---

## Input Arguments

### **sm - SimBiology model**

SimBiology model object

SimBiology model, specified as a `SimBiology model` object. The active `configset` object of the model contains solver settings for simulation. Any active doses and variants are applied to the model during simulation unless specified otherwise using the `dosing` and `variants` input arguments, respectively.

### **grpData - Data to fit**

groupedData object

Data to fit, specified as a `groupedData` object.

The name of the time variable must be defined in the `IndependentVariableName` property of `grpData`. For instance, if the time variable name is 'TIME', then specify it as follows.

```
grpData.Properties.IndependentVariableName = 'TIME';
```

`grpData` must at least two groups, and the name of grouping variable name must be defined in the `GroupVariableName` property of `grpData`. For example, if the grouping variable name is 'GROUP', then specify it as follows.

```
grpData.Properties.GroupVariableName = 'GROUP';
```

A group usually refers to a set of measurements that represent a single time course, often corresponding to a particular individual or experimental condition.

### **responseMap - Mapping information of model components to response data**

string | cell array of strings

Mapping information of model components to response data in `grpData`, specified as a string or cell array of strings.

Each string is an equation-like expression, similar to assignment rules in SimBiology. It contains the name (or qualified name) of a quantity (species, compartment, or parameter) in the model `sm`, followed by the character '=' and the name of a variable in `grpData`. For clarity, white spaces are allowed between names and '='.

For example, if you have the concentration data 'CONC' in `grpData` that corresponds to a model species 'Drug\_Central', you can specify the mapping information as follows.

```
responseMap = 'Drug_Central = CONC';
```

To unambiguously name a species, use the qualified name, which includes the name of the compartment. To name a reaction-scoped parameter, use the reaction name to qualify the parameter. If the name is not a valid MATLAB variable name, surround it by square brackets such as `[reaction 1].[parameter 1]`.

An error is issued if any (qualified) name matches two components of the same type. However, you can use a (qualified) name that matches two components of different types, and the function first finds the species with the given name, followed by compartments and then parameters.

### **covEstiminfo - Estimated parameters**

vector of `estimatedInfo` objects | `CovariateModel` object

Estimated parameters, specified as a vector of `estimatedInfo` objects or a `CovariateModel` object that defines the estimated parameters in the model `sm`, their initial estimates (optional), and their relationship to group-specific covariates included in `grpData` (optional). If this is a vector of `estimatedInfo` objects, then no covariates are used, and all parameters are estimated with group-specific random effects.

You can also specify parameter transformations if necessary. Supported transforms are `log`, `logit`, and `probit`. For details, see `estimatedInfo` object and `CovariateModel` object.

**dosing - Dosing information**

`[]` | 2-D matrix of dose objects

Dosing information, specified as an empty array or 2-D matrix of dose objects (`ScheduleDose` object or `RepeatDose` object). If empty, no doses are applied during simulation, even if the model has active doses. If not empty, the matrix must have a single row or one row per group in the data `grpData`. If it has a single row, the same doses are applied to all groups during simulation. If it has multiple rows, each row is applied to a separate group, in the same order as the groups appear in `grpData`.

Multiple columns are allowed so that you can apply multiple dose objects to each group. Each column of doses must reference the same components in the model `sm`. Specifically, all doses (except for empty doses) in a column must have the same values for `TargetName`, `DurationParameterName`, and `LagParameterName`. If some groups require more doses than others, then fill in the matrix with dummy doses that are either default doses or empty doses.

A default dose has default values for all properties, except for the `Name` property. An empty dose has a dose amount of 0, thus having no effect on the model. Create a default dose as follows.

```
d1 = sbiodose('d1');
```

In addition to manually constructing dose objects, if `grpData` has dosing information, you can use the `createDoses` method of `groupedData` object `grpData` to construct doses.

**functionName - Estimation function name**

string

Estimation function name, specified as a string. Choices are `'nlmefit'` or `'nlmefitsa'`.

## **opt - Options specific to estimation function**

struct

Options specific to the estimation function, specified as a structure. The structure contains fields and default values that are the name-value pair arguments accepted by `nlmefit` and `nlmefitsa`, except the following that are not supported.

- 'FEConstDesign'
- 'FEGroupDesign'
- 'FEObsDesign'
- 'FEParamsSelect'
- 'ParamTransform'
- 'REConstDesign'
- 'REGroupDesign'
- 'REObsDesign'
- 'Vectorization'

'REParamsSelect' is only supported when `covEstiminfo` is a vector of `estimatedInfo` objects.

Use the `statset` function only to set the 'Options' field of the `opt` structure as follows.

```
opt.Options = statset('Display','iter','TolX',1e-3,'TolFun',1e-3);
```

For other supported name-value pair arguments (see `nlmefit` and `nlmefitsa`), set them as follows.

```
opt.ErrorModel = 'proportional';  
opt.ApproximationType = 'LME';
```

## **variants - Variants**

[] | vector of variant objects

Variants, specified as an empty array or vector of variant objects. If empty, no variants are used even if the model has active variants.

**tf - 'UseParallel' option**

true | false

'UseParallel' option, specified as true or false. If true, and Parallel Computing Toolbox is available, the function performs parameter estimation in parallel.

**Output Arguments**

**fitResults - Estimation results**

NLMEResults object

Estimation results, returned as an NLMEResults object.

**simDataI - Simulation results**

vector of SimData objects

Simulation results, returned as a vector of SimData objects representing simulation results for each group (or individual) using fixed-effect and random-effect estimates (individual-specific parameter estimates).

The states reported in `simDataI` are the states that were included in the `responseMap` input argument as well as any other states listed in the `StatesToLog` property of the runtime options (`RuntimeOptions`) of the SimBiology model `sm`.

**simDataP - Simulation results**

vector of SimData objects

Simulation results, returned as a vector of SimData objects representing simulation results for each group (or individual) using only fixed-effect estimates (population parameter estimates).

The states reported in `simDataP` are the states that were included in the `responseMap` input argument as well as any other states listed in the `StatesToLog` property of the runtime options (`RuntimeOptions`) of the SimBiology model `sm`.

## Examples

### Fit a One-Compartment PK Model to the Phenobarbital Data

This example uses data collected on 59 preterm infants given phenobarbital during the first 16 days after birth. Each infant received an initial dose followed by one or more sustaining doses by intravenous bolus administration. A total of between 1 and 6 concentration measurements were obtained from each infant at times other than dose times, for a total of 155 measurements. Infant weights and APGAR scores (a measure of newborn health) were also recorded. Data is described in [1], a study funded by the NIH/NIBIB grant P41-EB01975.

Load the data.

```
load pheno.mat ds
```

Convert the dataset to a `groupedData` object, a container for holding tabular data that is divided into groups. It can automatically identify commonly used variable names as the grouping variable or independent (time) variable. Display the properties of the data and confirm that `GroupVariableName` and `IndependentVariableName` are correctly identified as 'ID' and 'TIME', respectively.

```
data = groupedData(ds);  
data.Properties
```

```
ans =
```

```
          Description: ''  
VariableDescriptions: {}  
   VariableUnits: {}  
DimensionNames: {'Observations' 'Variables'}  
      UserData: []  
      RowNames: {}  
VariableNames: {'ID' 'TIME' 'DOSE' 'WEIGHT' 'APGAR' 'C'}  
GroupVariableName: 'ID'  
IndependentVariableName: 'TIME'
```

Create a simple one-compartment PK model with bolus dosing and linear clearance to fit such data. Use the `PKModelDesign` object to

construct the model. Each compartment is defined by a name, dosing type, a clearance type, and whether or not the dosing requires a lag parameter. After constructing the model, you can also get a `PKModelMap` object map that lists the names of species and parameters in the model that are most relevant for fitting.

```
pkmd = PKModelDesign;
pkmd.addCompartment('Central', 'DosingType', 'Bolus', 'EliminationType',
[onecomp, map] = pkmd.construct;
```

Describe the experimentally measured response by mapping the appropriate model component to the response variable. In other words, indicate which species in the model corresponds to which response variable in the data. The `PKModelMap` property `Observed` indicates that the relevant species in the model is `Drug_Central`, which represents the drug concentration in the system. The relevant data variable is `CONC`, which you visualized previously.

```
map.Observed
responseMap = 'Drug_Central = CONC';
```

```
ans =

    'Drug_Central'
```

The parameters to estimate in this model are the volume of the central compartment `Central` and the clearance rate `Cl_Central`. The `PKModelMap` property `Estimated` lists these relevant parameters. The underlying algorithm of `sbiofit` assumes parameters are normally distributed, but this assumption may not be true for biological parameters that are constrained to be positive, such as volume and clearance. Specify a log transform for the estimated parameters so that the transformed parameters follow a normal distribution. Use an `estimatedInfo` object to define such transforms and initial values (optional).

```
map.Estimated
estimatedParams = estimatedInfo({'log(Central)', 'log(Cl_Central)'}, 'In
```

```
ans =  
  
  'Central'  
  'Cl_Central'
```

Each infant received a different schedule of dosing. The amount of drug is listed in the data variable DOSE. To specify these dosing during fitting, create dose objects from the data. These objects use the property TargetName to specify which species in the model receives the dose. In this example, the target species is Drug\_Central, as listed by the PKModelMap property Dosed.

```
map.Dosed
```

```
ans =  
  
  'Drug_Central'
```

Create a sample dose with this target name and then use the createDoses method of groupedData object data to generate doses for each infant based on the dosing data DOSE.

```
sampleDose = sbiodose('sample','TargetName','Drug_Central');  
doses = createDoses(data,'DOSE','',sampleDose);
```

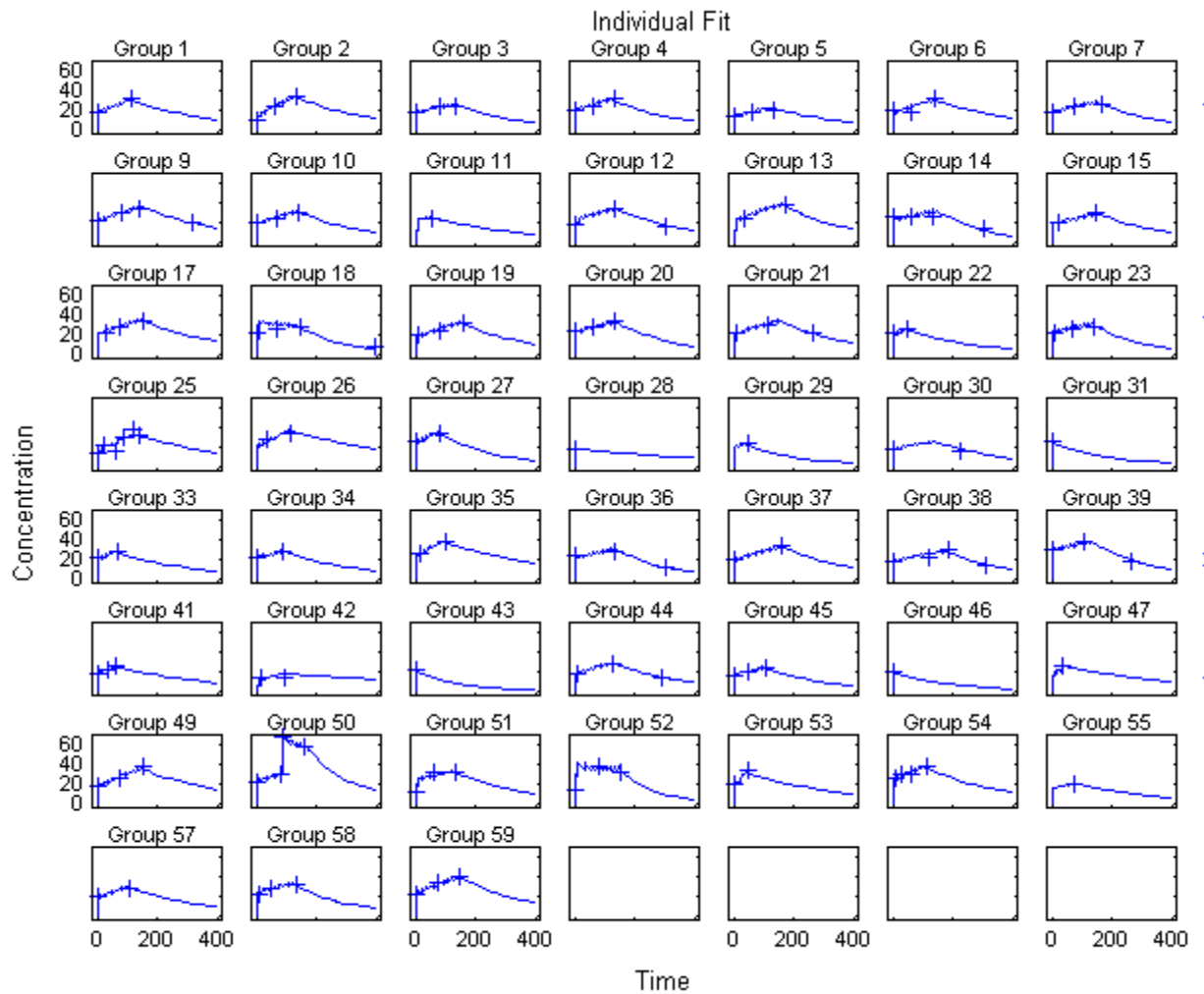
Fit the model.

```
[nlmeResults,simI,simP] = sbiofitmixed(onecomp,data,responseMap,estimated)
```

Visualize the fitted results using individual-specific parameter estimates.

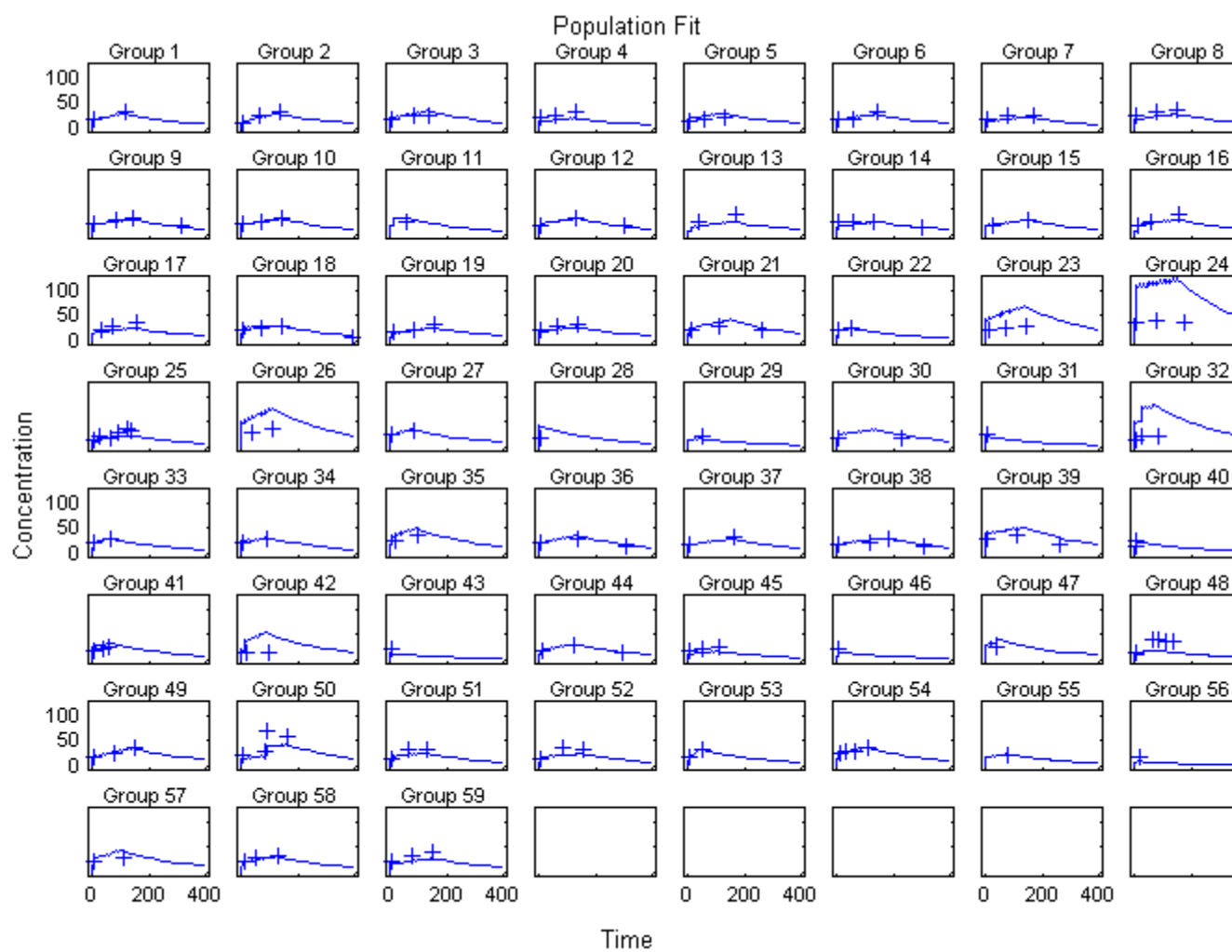
```
plot(nlmeResults,'ParameterType','individual');
```





Visualize the fitted results using population parameter estimates.

```
plot(nlmeResults, 'ParameterType', 'population');
```



## References

[1] Grasela Jr, T.H., Donn, S.M. (1985) Neonatal population pharmacokinetics of phenobarbital derived from routine clinical data. *Dev Pharmacol Ther.* 8(6), 374–83.

## See Also

`sbiofit` | `nlmefit` | `nlmefitsa` | `groupedData` | `estimatedInfo` object | `NLMEResults` object | `sbiofitstatusplot` | `CovariateModel` object

## Related Examples

- “Modeling the Population Pharmacokinetics of Phenobarbital in Neonates”

# sbiofitstatusplot

---

**Purpose** Plot status of sbionlmeffit or sbionlmeffitsa

**Syntax** stop = sbiofitstatusplot(beta, status, state)

**Description** stop = sbiofitstatusplot(beta, status, state) initializes or updates a plot with the fixed effects, beta, the log likelihood status.fval, and the variance of the random effects, diag(status.Psi).

The function returns an output (stop) to satisfy requirements for the 'OutputFcn' option of nlmefit or nlmefitsa. For sbiofitstatusplot, the value of stop is always false.

Use sbiofitstatusplot to obtain status information about NLME fitting when using the sbionlmeffit or sbionlmeffitsa function . Specify sbiofitstatusplot as a function handle by using the optionStruct (options structure) input argument to sbionlmeffit or sbionlmeffitsa. Use sbiofitstatusplot or customize your own function to use in the options structure.

## Input Arguments

**beta**  
The current fixed effects.

**status**  
Structure containing several fields.

| Field | Value  |
|-------|--|
| inner | Structure describing the current status of the inner iterations within the ALT and LAP procedures, with the fields: <ul style="list-style-type: none"><li>• procedure<ul style="list-style-type: none"><li>▪ 'PNLS', 'LME', or 'none' when the procedure is 'ALT'</li><li>▪ 'PNLS', 'PLM', or 'none' when the procedure is 'LAP'</li></ul></li></ul> |

| Field     | Value  |
|-----------|--|
|           | <ul style="list-style-type: none"> <li>• state — 'init', 'iter', 'done', or 'none'</li> <li>• iteration — Integer starting from 0, or NaN</li> </ul> |
| procedure | 'ALT' or 'LAP'   |
| iteration | Integer starting from 0  |
| fval      | Current log-likelihood   |
| Psi       | Current random-effects covariance matrix   |
| theta     | Current parameterization of Psi  |
| mse       | Current error variance   |

**state**

Either 'init', 'iter', or 'done'.

**Definitions**

**Alt**

Alternating algorithm for the optimization of the LME or RELME approximations

**FO**

First-order estimate

**FOCE**

First-order conditional estimate

**LAP**

Optimization of the Laplacian approximation for FO or FOCE

**LME**

Linear mixed-effects estimation

**NLME**

Nonlinear mixed effects

# sbiofitstatusplot

---

## **PLM**

Profiled likelihood maximization

## **PNLS**

Penalized nonlinear least squares

## **RELME**

Restricted likelihood for the linear mixed-effects model

## **Examples**

Obtain status information for NLME fitting:

```
% Create options structure with 'OutputFcn'.  
optionStruct.Options.OutputFcn = @sbiofitstatusplot;  
% Pass options structure with OutputFcn to sbionlmeft function.  
results = sbionlmeft(..., optionStruct);
```

## **See Also**

[nlmeft](#) | [sbionlinfit](#) | [sbionlmeft](#) | [sbionlmeftsa](#)

## **How To**

- “Obtaining the Status of Fitting”

|                    |  |
|--------------------|--|
| <b>Purpose</b>     | Open SimBiology desktop for population fitting   |
| <b>Syntax</b>      | <code>sbiofittool</code>   |
| <b>Description</b> | <p><code>sbiofittool</code> opens the SimBiology desktop in a state designed for:</p> <ul style="list-style-type: none"><li>• Importing and plotting data for fitting</li><li>• Selecting from a library of pharmacokinetic models</li><li>• Performing population fit tasks using <code>sbionlmeffit</code> or <code>sbionlmefitsa</code></li><li>• Performing individual fit tasks using <code>sbionlinfit</code></li></ul> <p><code>sbiofittool</code> opens a simplified configuration of the SimBiology desktop. However, all desktop functionality is available.</p> <p>If you opened the SimBiology desktop using the <code>simbiology</code> function, then <code>sbiofittool</code> changes the desktop layout to optimize it for population fitting.</p> |
| <b>See Also</b>    | <code>simbiology</code>  |

# sbiogetmodel

---

**Purpose** Get model object that generated simulation data

**Syntax** `modelObj = sbiogetmodel(simDataObj)`

## Arguments

|                         |   |
|-------------------------|---|
| <code>simDataObj</code> | SimData object returned by the function <code>sbiosimulate</code> or by <code>sbioensemblrun</code> . |
| <code>modelObj</code>   | Model object associated with the SimData object.  |

## Description

`modelObj = sbiogetmodel(simDataObj)` returns the SimBiology model (`modelObj`) associated with the results from a simulation run (`simDataObj`). You can use this function to find the model object associated with the specified SimData object when you load a project with several model objects and SimData objects.

If the SimBiology model used to generate the SimData object (`simDataObj`) is not currently loaded, `modelObj` is empty.

## Examples

Retrieve the model object that generated the SimData object.

- 1 Create a model object, simulate, and then return the results as a SimData object.

```
modelObj = sbmlimport('oscillator');  
simDataObj = sbiosimulate(modelObj);
```

- 2 Get the model that generated the simulation results.

```
modelObj2 = sbiogetmodel(simDataObj)  
SimBiology Model - Oscillator
```

Model Components:

|             |    |
|-------------|----|
| Models:     | 0  |
| Parameters: | 0  |
| Reactions:  | 42 |



```
Rules:          0
Species:       23
```

**3** Check that the two models are the same.

```
modelObj == modelObj2
ans =
     1
```

## See Also

`sbiosimulate`

# sbiolasterror

---

**Purpose** SimBiology last error message

**Syntax**  
sbiolasterror  
*diagstruct* = sbiolasterror  
sbiolasterror([])  
sbiolasterror(*diagstruct*)

**Arguments**

|                   |  |
|-------------------|--|
| <i>diagstruct</i> | The diagnostic structure holding Type, Message ID, and Message for the errors. |
|-------------------|--|

**Description** sbiolasterror or *diagstruct* = sbiolasterror return a SimBiology diagnostic structure array containing the last error(s) generated by the software. The fields of the diagnostic structure are:

|                  |   |
|------------------|---|
| <b>Type</b>      | 'error'   |
| <b>MessageID</b> | The message ID for the error (for example, 'SimBiology:ConfigSetNameClash') |
| <b>Message</b>   | The error message   |

sbiolasterror([]) resets the SimBiology last error so that it will return an empty array until the next SimBiology error is encountered.

sbiolasterror(*diagstruct*) will set the SimBiology last error(s) to those specified in the diagnostic structure (*diagstruct*).

**Examples** This example shows how to use verify and sbiolasterror.

```
1 Import a model.  
  
a = sbmlimport('radiodecay.xml')  
  
SimBiology Model - RadioactiveDecay  
  
Model Components:  
Models: 0
```

```

Parameters:      1
Reactions:      1
Rules:          0
Species:        2

```

**2** Change the ReactionRate of a reaction to make the model invalid.

```
a.reactions(1).reactionrate = 'x*y'
```

```
SimBiology Model - RadioactiveDecay
```

```

Model Components:
Models:          0
Parameters:     1
Reactions:      1
Rules:          0
Species:        2

```

**3** Use the function verify to validate the model.

```
a.verify
```

```

??? Error using==>simbio\private\odebuilder>buildPatternSubStrings
The object y does not resolve on reaction with expression 'x*y'.

```

```

Error in ==> sbiogate at 22
feval(varargin{:});

```

```

??? --> Error reported from Expression Validation :
The object 'y' in reaction 'Reaction1' does not resolve
to any in-scope species or parameters.
--> Error reported from Dimensional Analysis :
Could not resolve species, parameter or model object 'y'
during dimensional analysis.
--> Error reported from ODE Compilation:
Error using==>simbio\private\odebuilder>buildPatternSubStrings
The object y does not resolve on reaction with expression 'x*y'.

```

**4** Retrieve the error diagnostic struct.

```
p = sbiolasterror
```

```
p =
```

```
1x3 struct array with fields:
```

```
    Type
```

```
    MessageID
```

```
    Message
```

**5** Display the first error ID and Message.

```
p(1)
```

```
ans =
```

```
    Type: 'Error'
```

```
MessageID: 'SimBiology:ReactionObjectDoesNotResolve'
```

```
Message: 'The object 'y' in reaction 'Reaction1'  
does not resolve to any in-scope  
species or parameters.'
```

**6** Reset the sbiolasterror.

```
sbiolasterror([])
```

```
ans =
```

```
    []
```

**7** Set sbiolasterror to the diagnostic struct.

```
sbiolasterror(p)
```

```
ans =
```

1x3 struct array with fields:  
Type  
MessageID  
Message

## See Also

[sbiolastwarning](#) | [verify](#)

## How To

- [sbioroot](#)

# sbiolastwarning

---

**Purpose** SimBiology last warning message

**Syntax**  
`sbiolastwarning`  
`diagstruct = sbiolastwarning`  
`sbiolastwarning([])`  
`sbiolastwarning(diagstruct)`

**Arguments**

|                   |  |
|-------------------|--|
| <i>diagstruct</i> | The diagnostic structure holding Type, Message ID, and Message for the warnings. |
|-------------------|--|

**Description** `sbiolastwarning` or `diagstruct = sbiolastwarning` return a SimBiology diagnostic structure array containing the last warnings generated by the software. The fields of the diagnostic structure are:

|                  |   |
|------------------|---|
| <b>Type</b>      | 'warning'   |
| <b>MessageID</b> | The message ID for the warning (for example, 'SimBiology:DANotPerformedReactionRate') |
| <b>Message</b>   | The warning message   |

`sbiolastwarning([])` resets the SimBiology last warning so that it will return an empty array until the next SimBiology warning is encountered.

`sbiolastwarning(diagstruct)` will set the SimBiology last warnings to those specified in the diagnostic structure (*diagstruct*).

**See Also** `sbiolasterror` | `verify`

**How To**

- `sbiroot`

## Purpose

Load project from file

## Syntax

```
sbioloadproject('projFilename')  
sbioloadproject ('projFilename','variableName')  
sbioloadproject projFilename variableName1 variableName2...  
s = sbioloadproject (...)
```

## Description

`sbioloadproject('projFilename')` loads a SimBiology project from a project file (*projFilename*). If no extension is specified, `sbioloadproject` assumes a default extension of `.sbproj`. Alternatively, the command syntax is `sbioloadproject projFilename`.

`sbioloadproject ('projFilename','variableName')` loads only the variable *variableName* from the project file.

`sbioloadproject projFilename variableName1 variableName2...` loads the specified variables from the project.

`s = sbioloadproject (...)` returns the contents of *projFilename* in a variable `s`. `s` is a struct containing fields matching the variables retrieved from the SimBiology project.

You can display the contents of the project file using the `sbiowhos` command.

## See Also

`sbioaddtolibrary` | `sbioremovefromlibrary` | `sbiosaveproject` | `sbiowhos`

## How To

- `sbiosaveproject`
- `sbiowhos`
- `sbioaddtolibrary`
- `sbioremovefromlibrary`

**Purpose** Construct model object

**Syntax**  
`modelObj = sbiomodel('NameValue')`  
`modelObj = sbiomodel(...'PropertyName', PropertyValue...)`

## Arguments

|                      |  |
|----------------------|--|
| <i>NameValue</i>     | Required property to specify a unique name for a model object. Enter a character string. |
| <i>PropertyName</i>  | Property name for a Model object from “Property Summary” on page 1-92.                   |
| <i>PropertyValue</i> | Property value. Valid value for the specified property.                                  |

## Description

`modelObj = sbiomodel('NameValue')` creates a model object and returns the model object (*modelObj*). In the model object, this method assigns a value (*NameValue*) to the property Name.

`modelObj = sbiomodel(...'PropertyName', PropertyValue...)` defines optional properties. The property name/property value pairs can be in any format supported by the function set (for example, name-value string pairs, structures, and name-value cell array pairs).

Simulate *modelObj* with the function `sbiosimulate`.

Add objects to a model object using the methods `addkineticlaw`, `addparameter`, `addreaction`, `addrule`, and `addspecies`.

All SimBiology model objects can be retrieved from the SimBiology root object. A SimBiology model object has its Parent property set to the SimBiology root object.

## Method Summary

|  |   |
|--|---|
| <code>addcompartment (model, compartment)</code> | Create compartment object                               |
| <code>addconfigset (model)</code>                | Create configuration set object and add to model object |



|                                  |   |
|----------------------------------|---|
| adddose (model)                  | Add dose object to model  |
| addevent (model)                 | Add event object to model object  |
| addparameter (model, kineticlaw) | Create parameter object and add to model or kinetic law object          |
| addreaction (model)              | Create reaction object and add to model object                          |
| addrule (model)                  | Create rule object and add to model object                              |
| addspecies (model, compartment)  | Create species object and add to compartment object within model object |
| addvariant (model)               | Add variant to model  |
| copyobj (any object)             | Copy SimBiology object and its children                                 |
| createSimFunction (model)        | Create SimFunction object   |
| delete (any object)              | Delete SimBiology object  |
| display (any object)             | Display summary of SimBiology object                                    |
| export (model)                   | Export SimBiology models for deployment and standalone applications     |
| get (any object)                 | Get object properties   |
| getadjacencymatrix (model)       | Get adjacency matrix from model object                                  |
| getconfigset (model)             | Get configuration set object from model object                          |
| getdose (model)                  | Return SimBiology dose object   |
| getequations                     | Return system equations for model object                                |

|   |   |
|---|---|
| getstoichmatrix (model)                   | Get stoichiometry matrix from model object    |
| getvariant (model)                        | Get variant from model                        |
| removeconfigset (model)                   | Remove configuration set from model           |
| removedose (model)                        | Add dose object to model                      |
| removevariant (model)                     | Remove variant from model                     |
| reorder (model, compartment, kinetic law) | Reorder component lists                       |
| set (any object)                          | Set object properties                         |
| setactiveconfigset (model)                | Set active configuration set for model object |
| verify (model, variant)                   | Validate and verify SimBiology model          |

## Property Summary

|              |   |
|--------------|---|
| Compartments | Array of compartments in model or compartment |
| Events       | Contain all event objects                     |
| Name         | Specify name of object                        |
| Notes        | HTML text describing SimBiology object        |
| Parameters   | Array of parameter objects                    |
| Parent       | Indicate parent object                        |
| Reactions    | Array of reaction objects                     |
| Rules        | Array of rules in model object                |
| Tag          | Specify label for SimBiology object           |

|          |                                       |
|----------|---------------------------------------|
| Type     | Display SimBiology object type        |
| UserData | Specify data to associate with object |

## Examples

- 1 Create a SimBiology model object.

```
modelObj = sbiomodel('cell', 'Tag', 'mymodel');
```

- 2 List all modelObj properties and the current values.

```
get(modelObj)
```

MATLAB returns:

```

Annotation: ''
  Models: [0x1 double]
  Name: 'cell'
  Notes: ''
Parameters: [0x1 double]
  Parent: [1x1 SimBiology.Root]
Species: [0x1 double]
Reactions: [0x1 double]
  Rules: [0x1 double]
  Tag: 'mymodel'
  Type: 'sbiomodel'
UserData: []

```

- 3 Display a summary of modelObj contents.

```
modelObj
```

```
SimBiology Model - cell
```

```

Model Components:
  Models:          0
Parameters:       0
Reactions:       0

```

# sbimodel

---

Rules: 0  
Species: 0

## See Also

`addcompartment` | `addconfigset` | `addevent` | `addkineticlaw` |  
`addparameter` | `addreaction` | `addrule` | `addspecies` | `copyobj` |  
`get` | `sbioroot` | `sbiosimulate` | `set`

|                      |  |
|----------------------|--|
| <b>Purpose</b>       | Perform nonlinear least-squares regression using SimBiology models   |
| <b>Compatibility</b> | sbionlinfit will be removed in a future release. Use sbiofit instead.  |
| <b>Syntax</b>        | <pre> results = sbionlinfit(modelObj, pkModelMapObject,     pkDataObj,         InitEstimates) results = sbionlinfit(modelObj, pkModelMapObject,     pkDataObj,         InitEstimates, Name, Value) results = sbionlinfit(modelObj, pkModelMapObject,     pkDataObj,         InitEstimates, optionStruct) [results, SimDataI] = sbionlinfit(...) </pre> |

## Description

---

**Note** This function requires nlinfit in Statistics Toolbox (Version 7.0 or later).

---

`results = sbionlinfit(modelObj, pkModelMapObject, pkDataObj, InitEstimates)` performs least-squares regression using the SimBiology model, `modelObj`, and returns estimated results in the `results` structure.

`results = sbionlinfit(modelObj, pkModelMapObject, pkDataObj, InitEstimates, Name, Value)` performs least-squares regression, with additional options specified by one or more `Name, Value` pair arguments.

Following is an alternative to the previous syntax:

`results = sbionlinfit(modelObj, pkModelMapObject, pkDataObj, InitEstimates, optionStruct)` specifies `optionStruct`, a structure containing fields and values used by the options input structure to the `nlinfit` function.

`[results, SimDataI] = sbionlinfit(...)` returns simulations of the SimBiology model, `modelObj`, using the estimated values of the parameters.

## Input Arguments

### **modelObj**

SimBiology model object used to fit observed data.

---

**Note** If using a model object containing active doses (that is, containing dose objects created using the `adddose` method, and specified as active using the `Active` property of the dose object), be aware that these active doses are ignored by the `sbionlinfit` function.

---

### **pkModelMapObject**

PKModelMap object that defines the roles of the model components in the estimation. For details, see `PKModelMap` object.

---

**Note** If using a PKModelMap object that specifies multiple doses, ensure each element in the `Dosed` property is unique.

---

### **pkDataObj**

PKData object that defines the data to use in fitting, and the roles of the data columns used for estimation. For details, see `PKData` object.

---

**Note** For each subset of data belonging to a single group (as defined in the data column specified by the `GroupLabel` property), the software allows multiple observations made at the same time. If this is true for your data, be aware that:

- These data points are not averaged, but fitted individually.
  - Different numbers of observations at different times cause some time points to be weighted more.
- 

### **InitEstimates**

Vector of initial parameter estimates for each parameter estimated in `pkModelMapObject.Estimated`. The length of `InitEstimates` must equal at least the length of `pkmodelMapObject.Estimated`. The elements of `InitEstimates` are transformed as specified by the `ParamTransform` name-value pair argument.

For details on specifying initial estimates, see “Set Initial Estimates”.

### **optionStruct**

Structure containing fields and values used by the options input structure to the `nlinfit` function. The structure can also use the name-value pairs listed below as fields and values. Defaults for `optionStruct` are the same as for the options input structure to `nlinfit`, except for:

- `DerivStep` — Default is the lesser of  $1e-4$ , or the value of the `SolverOptions.RelativeTolerance` property of the configuration set associated with `modelObj`, with a minimum of  $\text{eps}^{(1/3)}$ .
- `FunValCheck` — Default is `off`.

If you have Parallel Computing Toolbox, you can enable parallel computing for faster data fitting by setting the name-value pair argument 'UseParallel' to `true` in the `statset` options structure as follows:

```
parpool; % Open a parpool for parallel computing
opt = statset(...,'UseParallel',true); % Enable parallel computing
results = sbionlinfit(...,opt); % Perform data fitting
```

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

The `Name`, `Value` arguments are the same as the fields and values in the `options` structure accepted by `nlinfit`. For a complete list, see the `options` input argument in the `nlinfit` reference page in the Statistics Toolbox documentation. The defaults for `Name`, `Value` arguments are the same as for the `options` structure accepted by `nlinfit`, except for:

- `DerivStep` — Default is the lesser of  $1e-4$ , or the value of the `SolverOptions.RelativeTolerance` property of the configuration set associated with `modelObj`, with a minimum of  $\text{eps}^{(1/3)}$ .
- `FunValCheck` — Default is `off`.

Following are additional `Name`, `Value` arguments that you can use with `sbionlinfit`.

### 'ParamTransform'

Vector of integers specifying a transformation function for each estimated parameter. The transformation function, `f`, takes `estimate` as an input and returns `beta`:

```
beta = f(estimate)
```

Each element in the vector must be one of these integers specifying the transformation for the corresponding value of `estimate`:

- 0 – `beta` = `estimate`
- 1 – `beta` = `log(estimate)` (default)



- 2 – beta = probit(estimate)
- 3 – beta = logit(estimate)

For details, see “Specify Parameter Transformations”.

### ‘ErrorModel’

String specifying the form of the error term. Default is 'constant'. Each model defines the error using a standard normal (Gaussian) variable  $e$ , the function value  $f$ , and one or two parameters  $a$  and  $b$ . Choices are:

- 'constant':  $y = f + a*e$
- 'proportional':  $y = f + b*abs(f)*e$
- 'combined':  $y = f + (a+b*abs(f))*e$
- 'exponential':  $y = f*exp(a*e)$ , or equivalently  $\log(y) = \log(f) + a*e$

If you specify an error model, the results output argument includes an errorparam property, which has the value:

- $a$  for 'constant' and 'exponential'
- $b$  for 'proportional'
- [ $a$   $b$ ] for 'combined'

---

**Note** If you specify an error model, you cannot specify weights.

---

### ‘Weights’

Either of the following:

- A matrix of real positive weights, where the number of columns corresponds to the number of responses. That is, the number of columns must equal the number of entries in the DependentVarLabel property of *pkDataObj*. The number of rows in the matrix must equal the number of rows in the data set.

- A function handle that accepts a vector of predicted response values and returns a vector of real positive weights.

---

**Note** If using a function handle, the weights must be a function of the response (dependent variable).

---

Default is no weights. If you specify weights, you cannot specify an error model.

### 'Pooled'

Logical specifying whether `sbionlinfit` does fitting for each individual (`false`) or if it pools all individual data and does one fit (`true`). If set to `true`, `sbionlinfit` uses the same model parameters for each dose level.

**Default:** `false`

## Output Arguments

### results

1-by- $N$  array of objects, where  $N$  is the number of groups in `pkDataObj`. There is one object per group, and each object contains these properties:

- `ParameterEstimates` — A `dataset` array containing fitted coefficients and their standard errors.
- `CovarianceMatrix` — Estimated covariance matrix for the fitted coefficients.
- `beta` — Vector of scalars specifying the fitted coefficients in transformed space.
- `R` — Vector of scalars specifying the residual values, where  $R(i,j)$  is the residual for the  $i$ th time point and the  $j$ th response in the group of data. If your model includes:
  - A single response, then `R` is a column vector of residual values associated with time points in the group of data.

- Multiple responses, then R is a matrix of residual values associated with time points in the group of data, for each response.
- J — Matrix specifying the Jacobian of the model, with respect to an estimated parameter, that is

$$J(i, j, k) = \left. \frac{\partial y_k}{\partial \beta_j} \right|_{t_i}$$

where  $t_i$  is the  $i$ th time point,  $\beta_j$  is the  $j$ th estimated parameter in the transformed space, and  $y_k$  is the  $k$ th response in the group of data.

If your model includes:

- A single response, then J is a matrix of Jacobian values associated with time points in the group of data.
- Multiple responses, then J is a 3-D array of Jacobian values associated with time points in the group of data, for each response.
- COVB — Estimated covariance matrix for the transformed coefficients.
- mse — Scalar specifying the estimate of the error of the variance term.
- errorparam — Estimated parameters of the error model. This property is a scalar if you specify 'constant', 'exponential', or 'proportional' for the error model. This property is a two-element vector if you specify 'combined' for the error model. This property is an empty array if you specify weights using the 'Weights' name-value pair argument.

### SimData

SimData object containing data from simulating the model using estimated parameter values for individuals. This object includes observed states and logged states.

### See Also

PKData object | PKModelDesign object | PKModelDesign object | PKModelMap object | Model object | sbionlmeftit | nlinfit | sbionlmeftitsa

## How To

- “Perform Data Fitting with PKPD Models”

|                      |   |
|----------------------|---|
| <b>Purpose</b>       | Estimate nonlinear mixed effects using SimBiology models  |
| <b>Compatibility</b> | sbionlmeft will be removed in a future release. Use sbiofitmixed instead.   |
| <b>Syntax</b>        | <pre> results = sbionlmeft(modelObj, pkModelMapObject,                     pkDataObject,                     InitEstimates) results = sbionlmeft(modelObj, pkModelMapObject,                     pkDataObject,                     CovModelObj) results = sbionlmeft(..., Name, Value) results = sbionlmeft(..., optionStruct) [results, SimDataI, SimDataP] = sbionlmeft(...) </pre> |

## Description

---

**Note** This function requires nlmefit in Statistics Toolbox (Version 7.0 or later).

---

`results = sbionlmeft(modelObj, pkModelMapObject, pkDataObject, InitEstimates)` performs nonlinear mixed-effects estimation using the SimBiology model, `modelObj`, and returns estimated results in the `results` structure.

`results = sbionlmeft(modelObj, pkModelMapObject, pkDataObject, CovModelObj)` specifies the relationship between parameters and covariates using `CovModelObj`, a `CovariateModel` object. The `CovariateModel` object also provides the parameter transformation.

`results = sbionlmeft(..., Name, Value)` performs nonlinear mixed-effects estimation, with additional options specified by one or more `Name, Value` pair arguments.

Following is an alternative to the previous syntax:

`results = sbionlmeft(..., optionStruct)` specifies `optionStruct`, a structure containing fields and values, that are the name-value pair arguments accepted by `nlmeft`. The defaults for `optionStruct` are the same as the defaults for the arguments used by `nlmeft`, with the exceptions explained in “Input Arguments” on page 1-104.

`[results, SimDataI, SimDataP] = sbionlmeft(...)` returns simulation data of the SimBiology model, `modelObj`, using the estimated values of the parameters.

## Input Arguments

### **modelObject**

SimBiology model object used to fit observed data.

---

**Note** If using a model object containing active doses (that is, containing dose objects created using the `adddose` method, and specified as active using the `Active` property of the dose object), be aware that these active doses are ignored by the `sbionlmeft` function.

---

### **pkModelMapObject**

PKModelMap object that defines the roles of the model components used for estimation. For details, see PKModelMap object.

---

**Note** If using a PKModelMap object that specifies multiple doses, ensure each element in the `Dosed` property is unique.

---

### **pkDataObject**

PKData object that defines the data to use in fitting, and the roles of the columns used for estimation. `pkDataObject` must define target data for at least two groups. For details, see PKData object.

---

**Note** For each subset of data belonging to a single group (as defined in the data column specified by the `GroupLabel` property), the software allows multiple observations made at the same time. If this is true for your data, be aware that:

- These data points are not averaged, but fitted individually.
  - Different numbers of observations at different times cause some time points to be weighted more.
- 

### **InitEstimates**

Vector of initial estimates for the fixed effects. The first  $P$  elements of *InitEstimates* correspond to the fixed effects for each  $P$  element of *pkModelMapObject.Estimated*. Additional elements correspond to the fixed effects for covariate factors. The first  $P$  elements of *InitEstimates* are transformed as specified by the `ParamTransform` name-value pairs (log transformed by default). For details on specifying initial estimates, see “Set Initial Estimates”.

### **CovModelObj**

`CovariateModel` object that defines the relationship between parameters and covariates. For details, see `CovariateModel` object.

---

**Tip** To simultaneously fit data from multiple dose levels, omit the random effect (`eta`) from the expressions in the `CovariateModel` object.

---

### **optionStruct**

Structure containing fields and values that are the name-value pairs accepted by the `nlmefit` function. The defaults for *optionStruct* are the same as the defaults for the arguments used by `nlmefit`, with the exceptions noted in “Name-Value Pair Arguments” on page 1-106.

If you have Parallel Computing Toolbox, you can enable parallel computing for faster data fitting by setting the name-value pair argument 'UseParallel' to true in the statset options structure as follows:

```
parpool; % Open a parpool for parallel computing
opt = statset(...,'UseParallel',true); % Enable parallel computing
results = sbionlmeft(...,'Options',opt); % Perform data fitting
```

---

**Tip** SimBiology software includes the `sbiofitstatusplot` function, which you can specify in the `OutputFcn` field of the `Options` field. This function lets you monitor the status of fitting.

---

---

**Tip** To simultaneously fit data from multiple dose levels, use the `InitEstimates` input argument and set the value of the `REParamsSelect` field to a 1-by- $n$  logical vector, with all entries set to `false`, where  $n$  equals the number of fixed effects.

---

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name, Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

The `sbionlmeft` function uses the name-value pair arguments supported by the `nlmeft` function.

These `nlmeft` name-value pairs are hard-coded in `sbionlmeft`, and therefore, you cannot set them:

- `FEParamsSelect`
- `FEConstDesign`
- `FEGroupDesign`



- FEObsDesign
- REConstDesign
- REGroupDesign
- REObsDesign
- Vectorization

If you provide a `CovariateModel` object as input to `sbionlmeft`, then these `nlmeft` name-value pairs are computed from the covariate model, and therefore, you cannot set them:

- FEGroupDesign
- ParamTransform
- REParamsSelect

You can set all other `nlmeft` name-value pairs. For details, see the `nlmeft` reference page.

Be aware that the defaults for these `nlmeft` name-value pairs differ when used by `sbionlmeft`:

### **‘FEGroupDesign’**

Numeric array specifying the design matrix for each group. For details, see “Specify a Nonlinear, Mixed-Effects Model”.

**Default:** `repmat(eye(P), [1 1 nGroups])`, where  $P$  = the number of estimated parameters, and `nGroups` = the number of groups in the observed data.

### **‘ParamTransform’**

Vector of integers specifying how the parameters are distributed. For details, see “Specify Parameter Transformations”.

---

**Note** Do not use the `ParamTransform` option to specify parameter transformations when providing a `CovariateModel` object to a fitting function. The `CovariateModel` object provides the parameter transformation.

---

**Default:** Vector of ones, which specifies all parameters are log transformed.

### 'OptimFun'

String specifying the optimization function used in maximizing the likelihood.

**Default:** `fminunc`, if you have Optimization Toolbox installed. Otherwise, the default is `fminsearch`.

### 'Options'

Structure containing multiple fields, including `DerivStep`, a scalar or vector specifying the relative difference used in the finite difference gradient calculation, and `FunValCheck`, a logical specifying whether to check for invalid values, such as NaN or Inf, from `modelfun`.

**Default:** The default for `DerivStep` is the lesser of  $1e-4$ , or the value of the `SolverOptions.RelativeTolerance` property of the configuration set associated with `modelObj`, with a minimum of  $\text{eps}^{(1/3)}$ . The default for `FunValCheck` is `off`.

---

**Tip** SimBiology software includes the `sbiofitstatusplot` function, which you can specify in the `OutputFcn` field of the `Options` name-value pair input argument. This function lets you monitor the status of fitting.

---

---

**Tip** To simultaneously fit data from multiple dose levels, use the `InitEstimates` input argument and set the `REParamsSelect` name-value pair input argument to a 1-by- $n$  logical vector, with all entries set to `false`, where  $n$  equals the number of fixed effects.

---

## Output Arguments

### results

Structure containing these fields:

- `FixedEffects` — A dataset array containing estimated fixed effects, including standard errors.
- `RandomEffects` — A dataset array containing sampled random effects for each group in the observed data in *pkDataObject*.
- `IndividualParameterEstimates` — A dataset array containing estimated parameter values for individuals, including random effects.
- `PopulationParameterEstimates` — A dataset array containing estimated parameter values for the population, without random effects.
- `RandomEffectCovarianceMatrix` — A dataset array containing the estimated covariance matrix of the random effects.
- `EstimatedParameterNames` — Cell array of strings specifying names of the estimated parameters.
- `CovariateNames` — Cell array of strings specifying names of the covariates in *CovModelObj*.
- `FixedEffectsStruct` — Structure containing the values of the estimated fixed effects.
- `stats` — Structure containing information such as AIC, BIC, and weighted residuals. For details on the fields in this structure, see the `stats` structure in `nlmefit` in the Statistics Toolbox documentation. However, the fields in the `stats` structure returned by `sbionlmeffit` vary slightly from those returned by `nlmefit`, namely:

- `ires`, `pres`, `iwres`, `pwres`, and `cwres` each contain a matrix of raw or weighted residuals, with the number of columns equal to the number of responses in the model.
- The `stats` structure returned by `sbionlmeft` includes an additional field, `Observed`. This field contains a string or cell array of strings specifying the measured responses that correspond to the columns in the matrices of the `ires`, `pres`, `iwres`, `pwres`, and `cwres` fields. The `Observed` field is the same as the `Observed` property of the `PKModelMap` input argument.

## **SimDataI**

`SimDataI` object containing data from simulating the model using the estimated parameter values for individuals. This object includes observed states and logged states.

## **SimDataP**

`SimDataP` object containing data from simulating the model using the estimated parameter values for the population. This object includes observed states and logged states.

## **See Also**

`Model` object | `nlmeft` | `PKData` object | `SimData` object | `PKModelDesign` object | `PKModelMap` object | `sbiofitstatusplot` | `sbionlinfit` | `sbionlmeftsa`

## **How To**

- “Perform Data Fitting with PKPD Models”
- “Specify a Nonlinear, Mixed-Effects Model”
- “Specify Parameter Transformations”

|                      |  |
|----------------------|--|
| <b>Purpose</b>       | Estimate nonlinear mixed effects with stochastic EM algorithm  |
| <b>Compatibility</b> | sbionlmefitsa will be removed in a future release. Use sbiofitmixed instead.   |
| <b>Syntax</b>        | <pre> results = sbionlmefitsa(modelObj, pkModelMapObject,     pkDataObject,         InitEstimates) results = sbionlmefitsa(modelObj, pkModelMapObject,     pkDataObject,         CovModelObj) results = sbionlmefitsa(..., Name, Value) results = sbionlmefitsa(..., optionStruct) [results, SimDataI, SimDataP] = sbionlmefitsa(...) </pre> |

## Description

---

**Note** This function requires nlmefitsa in Statistics Toolbox (Version 7.0 or later).

---

`results = sbionlmefitsa(modelObj, pkModelMapObject, pkDataObject, InitEstimates)` performs estimations using the Stochastic Approximation Expectation-Maximization (SAEM) algorithm for fitting population data with the SimBiology model, `modelObj`, and returns the estimated results in the `results` structure.

`results = sbionlmefitsa(modelObj, pkModelMapObject, pkDataObject, CovModelObj)` specifies the relationship between parameters and covariates using `CovModelObj`, a `CovariateModel` object. The `CovariateModel` object also provides the parameter transformation.

`results = sbionlmefitsa(..., Name, Value)` performs estimations using the SAEM algorithm, with additional options specified by one or more `Name, Value` pair arguments.

Following is an alternative to the previous syntax:

`results = sbionlmefitsa(..., optionStruct)` specifies `optionStruct`, a structure containing fields and values, that are the name-value pair arguments accepted by `nlmefitsa`. The defaults for `optionStruct` are the same as the defaults for the name-value pair arguments used by `nlmefitsa`, with the exceptions explained in “Input Arguments” on page 1-112.

`[results, SimDataI, SimDataP] = sbionlmefitsa(...)` returns simulation data of the SimBiology model, `modelObj`, using the estimated values of the parameters.

## Input Arguments

### **modelObject**

SimBiology model object used to fit observed data.

---

**Note** If using a model object containing active doses (that is, containing dose objects created using the `adddose` method, and specified as active using the `Active` property of the dose object), be aware that these active doses are ignored by the `sbionlmefitsa` function.

---

### **pkModelMapObject**

PKModelMap object that defines the roles of the model components used for estimation. For details, see `PKModelMap` object.

---

**Note** If using a `PKModelMap` object that specifies multiple doses, ensure each element in the `Dosed` property is unique.

---

### **pkDataObject**

PKData object that defines the data to use in fitting and the roles of the columns used for estimation. `pkDataObject` must define target data for at least two groups. For details, see `PKData` object.

---

**Note** For each subset of data belonging to a single group (as defined in the data column specified by the `GroupLabel` property), the software allows multiple observations made at the same time. If this is true for your data, be aware that:

- These data points are not averaged, but fitted individually.
  - Different numbers of observations at different times cause some time points to be weighted more.
- 

### **InitEstimates**

Vector of initial estimates for the fixed effects. The first  $P$  elements of *InitEstimates* correspond to the fixed effects for each  $P$  element of *pkModelMapObject.Estimated*. Additional elements correspond to the fixed effects for covariate factors. The first  $P$  elements of *InitEstimates* are transformed as specified by the `ParamTransform` name-value pair argument (log transformed by default). For details on specifying initial estimates, see “Set Initial Estimates”.

### **CovModelObj**

`CovariateModel` object that defines the relationship between parameters and covariates. For details, see `CovariateModel` object.

### **optionStruct**

Structure containing fields and values that are name-value pair arguments accepted by the `nlmefitsa` function. The defaults for *optionStruct* are the same as the defaults for the arguments used by `nlmefitsa`, with the exceptions noted in “Name-Value Pair Arguments” on page 1-114.

If you have Parallel Computing Toolbox, you can enable parallel computing for faster data fitting by setting the name-value pair argument 'UseParallel' to true in the `statset` options structure as follows:

```
parpool; % Open a parpool for parallel computing
opt = statset(...,'UseParallel',true); % Enable parallel computing
results = sbionlmefitsa(...,'Options',opt); % Perform data fitting
```

---

**Tip** SimBiology software includes the `sbiofitstatusplot` function, which you can specify in the `OutputFcn` field of the `Options` field. This function lets you monitor the status of fitting.

---

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes ( `' '`). You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

The `sbionlmefitsa` function uses the name-value pair arguments supported by the `nlmefitsa` function.

These `nlmefitsa` name-value pair arguments are hard-coded in `sbionlmefitsa`, and therefore, you cannot set them:

- `FEPARAMSSELECT`
- `FECONSTDESIGN`
- `FEGROUPDESIGN`
- `FEOBSDSIGN`
- `RECONSTDESIGN`
- `REGROUPDESIGN`
- `REOBSDSIGN`
- `VECTORIZATION`

If you provide a `CovariateModel` object as input to `sbionlmefitsa`, then these `nlmefitsa` name-value pairs are computed from the covariate model, and therefore, you cannot set them:



- FEGroupDesign
- ParamTransform
- REParamsSelect

You can set all other `nlmefitsa` name-value pair arguments. For details on these arguments, see the `nlmefitsa` reference page.

Be aware that the defaults for these `nlmefitsa` name-value pair arguments differ when used by `sbionlmefitsa`:

#### **‘FEGroupDesign’**

Numeric array specifying the design matrix for each group. For details, see “Specify a Nonlinear, Mixed-Effects Model”.

**Default:** `repmat(eye(P), [1 1 nGroups])`, where `P` = the number of estimated parameters, and `nGroups` = the number of groups in the observed data.

#### **‘ParamTransform’**

Vector of integers specifying how the parameters are distributed. For details, see “Specify Parameter Transformations”.

---

**Note** Do not use the `ParamTransform` option to specify parameter transformations when providing a `CovariateModel` object to a fitting function. The `CovariateModel` object provides the parameter transformation.

---

**Default:** Vector of ones, which specifies all parameters are log transformed.

#### **‘OptimFun’**

String specifying the optimization function used in maximizing the likelihood.

**Default:** `fminunc`, if you have Optimization Toolbox installed. Otherwise, the default is `fminsearch`.

## 'Options'

Structure containing multiple fields, including `DerivStep`, a scalar or vector specifying the relative difference used in the finite difference gradient calculation, and `FunValCheck`, a logical specifying whether to check for invalid values, such as NaN or Inf, from `modelfun`.

**Default:** The default for `DerivStep` is the lesser of  $1e-4$ , or the value of the `SolverOptions.RelativeTolerance` property of the configuration set associated with `modelObj`, with a minimum of  $\text{eps}^{(1/3)}$ . The default for `FunValCheck` is `off`.

---

**Tip** SimBiology software includes the `sbiofitstatusplot` function, which you can specify in the `OutputFcn` field of the `Options` name-value pair input argument. This function lets you monitor the status of fitting.

---

## Output Arguments

### results

Structure containing these fields:

- `FixedEffects` — A dataset array containing estimated fixed effects, including standard errors.
- `RandomEffects` — A dataset array containing sampled random effects for each group in the observed data in `pkDataObject`.
- `IndividualParameterEstimates` — A dataset array containing estimated parameter values for individuals, including random effects.
- `PopulationParameterEstimates` — A dataset array containing estimated parameter values for the population, without random effects.
- `RandomEffectCovarianceMatrix` — A dataset array containing the estimated covariance matrix of the random effects.

- `EstimatedParameterNames` — Cell array of strings specifying names of the estimated parameters.
- `CovariateNames` — Cell array of strings specifying names of the covariates in *CovModelObj*.
- `FixedEffectsStruct` — Structure containing the values of the estimated fixed effects.
- `stats` — Structure containing information such as AIC, BIC, and weighted residuals. For details on the fields in this structure, see the `stats` structure in `nlmefitsa` in the Statistics Toolbox documentation. However, the fields in the `stats` structure returned by `sbionlmefitsa` vary slightly from those returned by `nlmefitsa`, namely:
  - `ires`, `pres`, `iwres`, `pwres`, and `cwres` each contain a matrix of raw or weighted residuals, with the number of columns equal to the number of responses in the model.
  - The `stats` structure returned by `sbionlmefit` includes an additional field, `Observed`. This field contains a string or cell array of strings specifying the measured responses that correspond to the columns in the matrices of the `ires`, `pres`, `iwres`, `pwres`, and `cwres` fields. The `Observed` field is the same as the `Observed` property of the `PKModelMap` input argument.

### **SimDataI**

`SimData` object containing data from simulating the model using the estimated parameter values for individuals. This object includes observed states and logged states.

### **SimDataP**

`SimData` object containing data from simulating the model using the estimated parameter values for the population. This object includes observed states and logged states.

## See Also

Model object | nlmefitsa | PKData object | SimData object |  
PKModelDesign object | PKModelMap object | sbiofitstatusplot |  
sbionlinfit | sbionlmefit

## How To

- “Perform Data Fitting with PKPD Models”
- “Specify an Error Model”
- “Specify a Nonlinear, Mixed-Effects Model”
- “Specify Parameter Transformations”

## Purpose

NONMEM file definition object for sbionmimport

## Syntax

```
nndefObj = sbionmfiledef
nndefObj = sbionmfiledef('Property Name', Property Value)
```

## Description

*nndefObj* = sbionmfiledef creates a NONMEM® file definition object. The NONMEM file definition object contains properties for specifying the NONMEM data items such as group, time, and dependent variable. The NONMEM file definition object lets you configure the properties to the column heading or the index of the column. Use the NONMEM file definition object in conjunction with the sbionmimport function to import NONMEM formatted files for use in fitting.

*nndefObj* = sbionmfiledef('Property Name', Property Value) accepts one or more comma-separated property name/value pairs. Specify *Property Name* inside single quotes. To see the default interpretations for NONMEM formatted files see “Support for Importing NONMEM Formatted Files” in the SimBiology documentation.

## Tips

- Use sbionmfiledef with sbionmimport if you want to apply NONMEM interpretation of headers, and the data file has column header labels different from the table shown in “Support for Importing NONMEM Formatted Files”
- Use sbionmimport if the data file has column header labels identical to the table shown in “Support for Importing NONMEM Formatted Files”.

## Input Arguments

### Filename

If *Filename* extension is .xls or .xlsx it is assumed to be an Excel® file, otherwise it is assumed to be a text file. sbionmfiledef file reads the file using the dataset constructor.

### Property Name/Value Pairs

'CompartmentLabel'

Identifies the column in the NONMEM formatted file that contains the compartment. Specify the header name as a `char` string or specify the index number of the header. During import the `sbionmimport` function uses the information in the column to interpret which compartment receives a dose or measured an observation. The `EventIDLabel` property specifies whether the value is a dose or an observation.

**Default:** ''

### **'ContinuousCovariateLabels'**

Identifies the column in the NONMEM formatted file that contains continuous covariates. Specify the header name as a `char` string or specify the index number of the header.

**Default:** {}

### **'DateLabel'**

Identifies the column in the NONMEM formatted file that contains the date. Specify the header name as a `char` string or specify the index number of the header. During import the `sbionmimport` function uses the information in the column to interpret time information for each dose, response and covariate measurement.

**Default:** ''

### **'DependentVariableLabel'**

Identifies the column in the NONMEM formatted file that contains observations. Specify the header name as a `char` string or specify the index number of the header.

**Default:** ''

### **'DoseLabel'**

Identifies the column in the NONMEM formatted file that contains the dosing information. Specify the header name as a `char` string or specify the index number of the header.

**Default:** ''

**'DoseIntervalLabel'**

Identifies the column in the NONMEM formatted file that contains the time between doses. Specify the header name as a `char` string or specify the index number of the header.

**Default:** ''

**'DoseRepeatLabel'**

Identifies the column in the NONMEM formatted file that contains the number of times (excluding the initial dose) that the dose is repeated. Specify the header name as a `char` string or specify the index number of the header.

**Default:** ''

**'EventIDLabel'**

Identifies the column in the NONMEM formatted file that contains the event identification specifying whether the value is a dose, observation, or covariate. Specify the header name as a `char` string or specify the index number of the header.

**Default:** ''

**'GroupLabel'**

Identifies the column in the NONMEM formatted file that contains the Group ID. Specify the header name as a `char` string or specify the index number of the header.

**Default:** ''

# sbionmfiledef

---

## **'MissingDependentVariableLabel'**

Identifies the column in the NONMEM formatted file that contains information about whether a row contains an observation event (0), or not (1). Specify the header name as a `char` string or specify the index number of the header.

**Default:** ''

## **'RateLabel'**

Identifies the column in the NONMEM formatted file that contains the rate of infusion. Specify the header name as a `char` string or specify the index number of the header.

**Default:** ''

## **'TimeLabel'**

Identifies the column in the NONMEM formatted file that contains the time or date of observation. During `import` the `sbionmimport` function uses this information to interpret when a dose was given, an observation or covariate measurement recorded. Specify the header name as a `char` string or specify the index number of the header.

**Default:** ''

## **'Type'**

Identifies the object as 'NMFileDef', (Read-only).

## **Output Arguments**

### **nmdefObj**

Defines the meanings of the file column headings. It contains properties for specifying data items such as group, time and date. `TimeLabel` and `DependentVariableLabel` must be specified.

## **Examples**

Configure a NONMEM file definition object and import data from a NONMEM formatted file.



```
% Configure a NMFileDef object.
def = sbionmfiledef;
def.CompartmentLabel      = 'CPT';
def.DoseLabel             = 'AMT';
def.DoseIntervalLabel    = 'II';
def.DoseRepeatLabel      = 'ADDL';
def.GroupLabel            = 'ID';
def.TimeLabel             = 'TIME';
def.DependentVariableLabel = 'DV';
def.EventIDLabel         = 'EVID';

filename = 'C:\work\datafiles\dose.xls';
ds = sbionmimport(filename, def);
```

**See Also**      sbionmimport

**How To**      • “Importing Data”

# sbionmimport

---

**Purpose** Import NONMEM-formatted data

**Syntax**

```
data = sbionmimport('Filename')
data = sbionmimport(nmds)
data = sbionmimport('Filename', nmdefObj)
data = sbionmimport(_, 'OutputFormat', dataFormat)
data = sbionmimport(_, 'ParameterName', ParameterValue)
data = sbionmimport(nmds, nmdefObj)
data =
sbionmimport(nmds, nmdefObj, 'OutputFormat', dataFormat)
[data, PKDataObj] = sbionmimport(_)
```

**Description** *data* = sbionmimport('Filename') or *data* = sbionmimport(*nmds*) converts a NONMEM formatted file, and assumes that the file is configured to use the following default values for column headers: ADDL, AMT, CPT, DATE, DAT1, DAT2, or DAT3, DV, EVID, ID, II, MDV, RATE, TIME. See “Support for Importing NONMEM Formatted Files” in the SimBiology documentation for more information on each of the headers.

*data* = sbionmimport('Filename', *nmdefObj*) imports a NONMEM formatted file named *Filename*, into a SimBiology formatted dataset *data* using the meanings of the file column headings defined in the NONMEM file definition object *nmdefObj*.

*data* = sbionmimport(\_, 'OutputFormat', *dataFormat*) returns a SimBiology formatted dataset *data* in the specified format *dataFormat* which must be either 'dataset' or 'groupedData'. If it is 'groupedData', the function uses the readtable method to read the data file and returns a groupedData object. If it is 'dataset', the dataset function is used and returns a dataset.

*data* = sbionmimport(\_, 'ParameterName', *ParameterValue*) accepts one or more comma-separated name-value pairs that are accepted by the dataset function or readtable method depending on what you have specified has the 'OutputFormat'. If additional information is required to read the file such as the delimiter, specify required name-value pairs. See dataset or readtable for a list of supported name-value pairs.

`data = sbionmimport(nmfs, nmdefObj)` reads a NONMEM formatted dataset `nmfs` and returns a SimBiology formatted dataset `data`.

`data = sbionmimport(nmfs, nmdefObj, 'OutputFormat', dataFormat)` reads a NONMEM formatted dataset `nmfs` and returns a dataset or `groupedData` object as specified by `dataFormat`.

`[data, PKDataObj] = sbionmimport(_)` returns a `PKData` object, `PKDataObj` containing the dataset `data`. The `PKDataObj` properties show the labels specified in `data`.

## Input Arguments

### Filename

If extension of `Filename` is `.xls` or `.xlsx`, `sbionmimport` assumes it to be an Excel file. Otherwise `sbionmimport` assumes `Filename` is a text file. `sbionmimport` reads the file using `dataset` or `readtable`.

### nmfs

NONMEM-formatted data, specified as a `dataset`, `table`, or `groupedData` object.

### dataFormat

Data format, specified as a string which must be one of the following: `'groupedData'` or `'dataset'` (default). If `'groupedData'`, `data` is a `groupedData` object. If `'dataset'`, it is a `dataset`.

### nmdefObj

`nmdefObj` defines the meanings of the file column headings. `nmdefObj` is a NONMEM file definition object created using the `sbionmfiledef` function. It contains properties for specifying data items such as `group`, `time`, and `date`. You must specify the `TimeLabel` and the `DependentVariableLabel` properties.

When this argument is omitted or empty `[]`, the default NONMEM interpretation is used.

## Output Arguments

### **data**

dataset or groupedData object. It contains a separate column for each dose and observation. The Description property of *data* contains a list of warnings, if any, that occurred while constructing *data*. To view the warnings, enter the following in the command line.

```
data.Properties.Description
```

### **PkDataObj**

The PKData object defines the data to use in fitting and the roles of the columns used for estimation. For more information, see PKData object.

## Examples

### **Import a Dataset**

Load a sample dataset.

```
load pheno ds;
```

The dataset contains 6 variables (columns). Display the names of these variables.

```
ds.Properties.VarNames
```

```
ans =
```

```
      'ID'      'TIME'      'DOSE'      'WEIGHT'      'APGAR'      'CONC'
```

Define what these variables mean according to the NONMEM definition.

```
def = sbionmfiledef;  
def.GroupLabel = 'ID';  
def.TimeLabel = 'TIME';  
def.DependentVariableLabel = 'CONC';  
def.DoseLabel = 'DOSE';  
def.ContinuousCovariateLabels = {'WEIGHT', 'APGAR'};
```

Import the dataset.

```
data = sbionmimport(ds,def);
```

## Import a Dataset to a GroupedData object

Load a sample dataset.

```
load pheno ds;
```

The dataset contains 6 variables (columns). Display the names of these variables.

```
ds.Properties.VarNames
```

```
ans =
```

```
      'ID'      'TIME'      'DOSE'      'WEIGHT'      'APGAR'      'CONC'
```

Define what these variables mean according to the NONMEM definition.

```
def = sbionmfiledef;
def.GroupLabel = 'ID';
def.TimeLabel = 'TIME';
def.DependentVariableLabel = 'CONC';
def.DoseLabel = 'DOSE';
def.ContinuousCovariateLabels = {'WEIGHT', 'APGAR'};
```

Import the dataset to a groupedData object.

```
grpData = sbionmimport(ds,def,'OutputFormat','groupedData');
```

## Import Data from a GroupedData object

Load a sample dataset.

```
load pheno ds
```

Create a groupedData object.

```
grpData = groupedData(ds);
```

# sbionmimport

---

Use the `groupedData` object variable names and define what column headings or variables mean according to the NONMEM definition.

```
def = sbionmfiledef;  
def.GroupLabel = grpData.Properties.GroupVariableName;  
def.TimeLabel = grpData.Properties.IndependentVariableName;  
def.DependentVariableLabel = 'CONC';  
def.DoseLabel = 'DOSE';  
def.ContinuousCovariateLabels = {'WEIGHT', 'APGAR'};
```

Import the dataset.

```
data = sbionmimport(grpData,def);
```

## See Also

`sbionmfiledef`

## How To

- “Importing Data”

**Purpose** Perform parameter estimation

**Compatibility** sbioparamestim will be removed in a future release. Use sbiofit instead.

**Syntax**

```
[k, result]= sbioparamestim(modelObj, tspan,  
xtarget, observed_array,  
estimated_array)  
[ __ ]= sbioparamestim( __ , observed_array,  
estimated_array, k0)  
[ __ ]= sbioparamestim( __ , observed_array,  
estimated_array, k0, method)
```

**Arguments**

|          |  |
|----------|--|
| k        | Vector of estimated parameter values. For all optimization methods except 'fminsearch', the parameters are constrained to be greater than or equal to 0.   |
| result   | Structure with fields that provide information about the progress of optimization.   |
| modelObj | SimBiology model object.   |
| tspan    | <i>n</i> -by-1 vector representing the time span of the target data xtarget.   |
| xtarget  | <i>n</i> -by- <i>m</i> matrix, where <i>n</i> is the number of time samples and <i>m</i> is the number of states to match during the simulation. The number of rows in xtarget must equal the number of rows in tspan. |

`observed_array` Either of the following:

- Array of objects (species, compartment, or nonconstant parameter) in `modelObj`, whose values should be matched during the estimation process
- Cell array of object names (species, compartment, or nonconstant parameter) in `modelObj`, whose values should be matched during the estimation process

---

**Note** If duplicate names exist for any species or parameters, ensure there are no ambiguities by specifying either an array of objects or a cell array of qualified names, such as *compartmentName.speciesName* or *reactionName.parameterName*. For example, for a species named `sp1` that is in a compartment named `comp2`, the qualified name is `comp2.sp1`.

---

The length of `observed_array` must equal the number of columns in `xtarget`. `sbioparamestim` assumes that the order of elements in `observed_array` is the same as the order of columns in `xtarget`.



`estimated_array` Either of the following:

- Array of objects (compartment, species, or parameter) in `modelObj` whose initial values should be estimated
- Cell array of names of objects (compartment, species, or parameter) in `modelObj` whose initial values should be estimated

---

**Note** If duplicate names exist for any compartments, species, or parameters, ensure there are no ambiguities by specifying either an array of objects or a cell array of qualified names, such as *compartmentName.speciesName* or *reactionName.parameterName*. For example, for a parameter named `param1` scoped to a reaction named `reaction1`, the qualified name is `reaction1.param1`.

---

`k0` Numeric vector containing the initial values of compartments, species, or parameters to be estimated. The length of `k0` must equal that of `estimated_array`. If you do not specify `k0`, or specify an empty vector for `k0`, then `sbioparamestim` takes initial values for compartments, species, or parameters from `modelObj`, or, if there are active variants, `sbioparamestim` uses any initial values specified in the active variants. For details about variants, see `Variant` object.

`method` Optimization algorithm to use during the estimation process, specified by either of the following:

- String specifying one of the following functions:
  - 'fminsearch'
  - 'lsqcurvefit'
  - 'lsqnonlin'
  - 'fmincon'
  - 'patternsearch'
  - 'patternsearch\_hybrid'
  - 'ga'
  - 'ga\_hybrid'
  - 'pso'
  - 'pso\_hybrid'

For descriptions of how `sbioparamestim` uses the previous functions, see the Function Descriptions on page 1-133 table.

- Two-element cell array, with the first element being one of the previous functions, and the second element being an options structure or object. Use an appropriate options structure or object for each method listed next.

| Method                                 | Options Structure or Object |
|--|-----------------------------|
| 'fminsearch'                           | optimset                    |
| 'fmincon'                              | optimoptions                |
| 'lsqcurvefit' 'lsqnonlin'              | lsqoptions                  |
| 'pso' 'pso_hybrid'                     | psooptions                  |
| 'patternsearch' 'patternsearch_hybrid' | patternsearchoptions        |
| 'ga' 'ga_hybrid'                       | gaoptimset                  |

---

**Tip** Use a two-element cell array to provide your own options structure for the optimization algorithm.

---

## Function Descriptions

| Function    | Description  |
|-------------|--|
| fminsearch  | <p>sbioparamestim uses the default options structure associated with fminsearch, except for:</p> <p>Display = 'off'<br/> TolFun = 1e-6* (Initial value of objective function)</p> <hr/> <p><b>Note</b> 'fminsearch' is an unconstrained optimization method, which can result in negative values for parameters.</p> <hr/>   |
| lsqcurvefit | <p>Requires Optimization Toolbox.</p> <p>sbioparamestim uses the default options structure associated with lsqcurvefit, except for:</p> <p>Display = 'off'<br/> FinDiffRelStep = value of the SolverOptions.RelativeTolerance property of the configuration set associated with modelObj, with a minimum of <math>\text{eps}^{(1/3)}</math><br/> TolFun = 1e-6* (Initial value of objective function)<br/> TypicalX = 1e-6* (Initial values of components to be estimated)</p> |

## Function Descriptions (Continued)

| Function  | Description  |
|-----------|--|
| lsqnonlin | Requires Optimization Toolbox.<br><br>sbioparamestim uses the default options structure associated with lsqnonlin, except for:<br><br>Display = 'off'<br>FinDiffRelStep = value of the SolverOptions.RelativeTolerance property of the configuration set associated with <i>mode1Obj</i> , with a minimum of $\text{eps}^{(1/3)}$<br>TolFun = $1\text{e-}6*$ (Initial value of objective function)<br>TypicalX = $1\text{e-}6*$ (Initial values of components to be estimated)                               |
| fmincon   | Requires Optimization Toolbox.<br><br>sbioparamestim uses the default options structure associated with fmincon, except for:<br><br>Algorithm = 'interior-point'<br>Display = 'off'<br>FinDiffRelStep = value of the SolverOptions.RelativeTolerance property of the configuration set associated with <i>mode1Obj</i> , with a minimum of $\text{eps}^{(1/3)}$<br>TolFun = $1\text{e-}6*$ (Initial value of objective function)<br>TypicalX = $1\text{e-}6*$ (Initial values of components to be estimated) |

## Function Descriptions (Continued)

| Function            | Description  |
|---------------------|--|
| patternsearch       | <p>Requires Global Optimization Toolbox.</p> <p>sbioparamestim uses the default options structure associated with patternsearch, except for:</p> <pre> Display = 'off' TolFun = 1e-6* (Initial value of objective function) TolMesh = 1.0e-3 Cache = 'on' MeshAccel = 'on'                     </pre>  |
| patternsearchHybrid | <p>Requires Global Optimization Toolbox.</p> <p>sbioparamestim calls the patternsearch function with the additional option SearchMethod = {@searchlhs,10,15}. This option adds an additional search step that uses Latin hypercube sampling.</p> <p>The sbioparamestim function uses the default options structure associated with patternsearch, except for:</p> <pre> Display = 'off' TolFun = 1e-6* (Initial value of objective function) TolMesh = 1.0e-3 Cache = 'on' MeshAccel = 'on' SearchMethod = {@searchlhs,10,15}                     </pre> |

## Function Descriptions (Continued)

| Function  | Description   |
|-----------|---|
| ga        | <p>Requires Global Optimization Toolbox.</p> <p>sbioparamestim uses the default options structure associated with ga, except for:</p> <p>Display = 'off'<br/>TolFun = 1e-6* (Initial value of objective function)<br/>PopulationSize = 10<br/>Generations = 30<br/>MutationFcn = @mutationadaptfeasible</p>   |
| ga_hybrid | <p>Requires Global Optimization Toolbox.</p> <p>sbioparamestim calls the ga function with the additional option HybridFcn = {@fmincon, fminopt}, where fminopt is the same set of default options sbioparamestim uses for fmincon. This option causes an additional gradient-based minimization after the genetic algorithm step ends.</p> <p>The sbioparamestim function uses the default options structure associated with ga, except for:</p> <p>Display = 'off'<br/>TolFun = 1e-6* (Initial value of objective function)<br/>PopulationSize = 10<br/>Generations = 30<br/>MutationFcn = @mutationadaptfeasible<br/>HybridFcn = {@fmincon, <i>structure of name/value pairs for fmincon</i>}</p> |

## Function Descriptions (Continued)

| Function    | Description  |                                 |             |        |             |                        |                                 |
|-------------|--|---------------------------------|-------------|--------|-------------|------------------------|---------------------------------|
| ps0         | <p>Requires Global Optimization Toolbox.<br/> sbioparamestim uses the following default options for the PSO (particle swarm optimization) algorithm [2][3]:</p> <pre style="margin-left: 40px;"> CreationFcn: @psocreationuniform Display: 'off' DisplayInterval: 1 FunValCheck: 'off' HybridFcn: [] InertiaRange: [0.1000 1.1000] InitialPositions: [] MaxFunEvals: 3000*numberofvariable MaxIter: 30 MinNeighborhoodFraction: 0.2500 NumParticles: 10 ObjectiveLimit: -Inf OutputFcns: [] PositionInitSpan: 2000 SelfRecognitionCoefficient: 1.4900 SocialRecognitionCoefficient: 1.4900 StallIterLimit: 20 StallTimeLimit: Inf TimeLimit: Inf TolFun: 1e-6*(Initial objective UseParallel: 0 Vectorized: 'off'                     </pre> <p>Description of options</p> <table border="1" style="width: 100%; border-collapse: collapse; margin-top: 10px;"> <thead> <tr style="background-color: #cccccc;"> <th style="text-align: left; padding: 5px;">Option</th> <th style="text-align: left; padding: 5px;">Description</th> <th style="text-align: left; padding: 5px;">Values</th> </tr> </thead> <tbody> <tr> <td style="padding: 5px;">CreationFcn</td> <td style="padding: 5px;">Handle to the function</td> <td style="padding: 5px;">@psocreationuniformThe function</td> </tr> </tbody> </table> | Option                          | Description | Values | CreationFcn | Handle to the function | @psocreationuniformThe function |
| Option      | Description  | Values                          |             |        |             |                        |                                 |
| CreationFcn | Handle to the function   | @psocreationuniformThe function |             |        |             |                        |                                 |

## Function Descriptions (Continued)

| Function | Description  |
|----------|--|
|          | <p>that creates the initial population</p> <p>creates initial positions of particles for the pso function and has the following format:</p> <p>positions=psocreationuniform<br/>where:</p> <ul style="list-style-type: none"><li>• pos is the numParticles*nvars matrix of particle positions.</li><li>• nvars is the number of variables.</li><li>• problemStruct is the problem struct with the following fields:<br/>solver for the solver name, 'pso',<br/>objective for the objective</li></ul> |



## Function Descriptions (Continued)

| Function | Description     |                                 |   |
|----------|-----------------|---------------------------------|---|
|          |                 |                                 | <p>function, nvars for the number of design variables, lb for the vector of lower bounds, and ub for the vector of upper bounds.</p> <ul style="list-style-type: none"> <li>options is an options object containing options created by calling <code>optimoptions('pso',...</code></li> </ul> |
|          | Display         | Level of display                | 'off'   'none'   'final'   'it  |
|          | DisplayInterval | Interval for iterative display  | Positive integer  |
|          | FunValCheck     | Indicator specifying whether to | 'off'   'on'  |

## Function Descriptions (Continued)

| Function         | Description   |  |  |
|------------------|---|--|--|
|                  |   | check objective function values are valid    |  |
| HybridFcn        | Function that is automatically run at the end of iterations of the solver | @fminsearch @patternsearch @fminunc @fmincon | 1-by-2 cell array of the form {@solver, hybridoptions}, where solver is: fminsearch, patternsearch, fminunc, or fmincon and hybridoptions is a structure of options for these functions and their values |
| InertiaRange     | Lower and upper bound of the adaptive inertia                             |  | Two-element vector of same-signed values in increasing order   |
| InitialPositions | Initial particle positions used to initiate the pso function. If any      |  | Matrix   |

**Function Descriptions (Continued)**

| Function                | Description |   |   |
|-------------------------|-------------|---|---|
|                         |             | initial particle positions are outside of the bounds, they are truncated to the bounds. |   |
| MaxFunEvals             |             | Maximum number of objective function evaluations allowed                                | Positive integer                                    |
| MaxIter                 |             | Maximum number of iterations allowed  | Positive integer                                    |
| MinNeighborhoodFraction |             | Minimum adaptive neighborhood size  | Scalar value from 0 and 1                           |
| NumParticles            |             | Number of particles   | Positive integer > = 2                              |
| ObjectiveLimit          |             | Minimum objective function value wanted   | Scalar  |
| OutputFcns              |             | Functions that the pso function handles   | Function handle   cell array of function handlesThe |

## Function Descriptions (Continued)

| Function | Description  |
|----------|--|
|          | <p>calls at each iteration</p> <p>function requires three input arguments and has the following form:</p> <pre>[stopflag,options,optchanged] = myFcn(optimValues,options,stopflag)</pre> <p>where:</p> <ul style="list-style-type: none"><li>• <b>optimValues</b> is a structure containing the following fields:<ul style="list-style-type: none"><li>▪ <b>Iteration</b> (iteration number)</li><li>▪ <b>StartTime</b> (identifier returned by <code>tic</code> when the algorithm starts)</li><li>▪ <b>LastImprovement</b> (iteration of last improvement)</li></ul></li></ul> |

## Function Descriptions (Continued)

| Function | Description   |
|----------|---|
|          | <ul style="list-style-type: none"> <li data-bbox="1159 427 1475 678">▪ LastImprovementTime (time of last improvement, as returned by <code>toc(StartTime)</code>)</li> <li data-bbox="1159 699 1343 887">▪ FunEval (total number of objective function evaluations)</li> <li data-bbox="1159 907 1322 1064">▪ Positions (matrix of current particle positions)</li> <li data-bbox="1159 1085 1343 1241">▪ Velocities (matrix of current particle velocities)</li> <li data-bbox="1159 1262 1307 1487">▪ Fvals (vector of current objective function values seen by</li> </ul> |

## Function Descriptions (Continued)

| Function | Description  |
|----------|--|
|          | <p>each particle)</p> <ul style="list-style-type: none"><li>▪ <code>IndividualBestFvals</code> (vector of the best objective function values seen by each particle)</li><li>▪ <code>IndividualBestPositions</code> (matrix of the best positions seen by each particle).</li><li>• <code>options</code> is an options object.</li><li>• <code>state</code> is the state of the algorithm with possible values of <code>'init'</code> (initialization state), <code>'iter'</code></li></ul> |

## Function Descriptions (Continued)

| Function | Description     |  |   |
|----------|-----------------|--|---|
|          |                 |  | <p>(iteration state), and 'done' (finished state).</p> <ul style="list-style-type: none"> <li>• stopflag is a logical indicating whether the optimization should continue or end at the current iteration.</li> <li>• optchanged is a logical indicating whether the options argument was changed.</li> </ul> |
|          | PositionInitSpa | Range of initial particle positions for variables that do not have finite bounds | Positive scalar   vector with n elements, where n is the number of design   |

## Function Descriptions (Continued)

| Function                     | Description   |                        |
|------------------------------|---|------------------------|
|                              |   | variables in the model |
| SelfRecognitionCoefficient   | Coefficient that controls the weighting of each particle's best position when updating the velocity                     | Finite scalar          |
| SocialRecognitionCoefficient | Coefficient that controls the weighting of the neighborhood's best position when updating the velocity                  | Finite scalar          |
| StallIterLimit               | Number of iterations over which average change in objective function value at current point is less than options.TolFun | Positive integer       |
| StallTimeLimit               | Maximum number of seconds, as measured by tic/toc,  | Positive scalar        |



**Function Descriptions (Continued)**

| Function    | Description |   |                 |
|-------------|-------------|---|-----------------|
|             |             | permitted without an improvement in the best known objective function value                 |                 |
| TimeLimit   |             | Maximum number of seconds, as measured by tic/toc, allowed for the optimization             | Positive scalar |
| TolFun      |             | Termination tolerance on function value   | Positive scalar |
| UseParallel |             | Indicator specifying whether to compute objective functions of a particle swarm in parallel | true   false    |
| Vectorized  |             | String specifying whether the computation of the objective function is vectorized.          | 'off'           |

## Function Descriptions (Continued)

| Function   | Description  |
|------------|--|
|            | Currently, the algorithm does not support any vectorization.   |
| pso_hybrid | Requires Global Optimization Toolbox.<br><br>sbioparamestim calls the pso function with the additional option HybridFcn = {@objFcn, options}. The objective function, objFcn, is one of these supported functions: patternsearch, fminsearch, fminunc, or fmincon. options is a structure of options for these functions and their values. |

---

**Note** sbioparamestim does not support setting the Vectorized option to 'on' in algorithms that support this option.

---

## Description

[k, result]= sbioparamestim(modelObj, tspan, xtarget, observed\_array, estimated\_array) estimates the initial values of compartments, species, and parameters of modelObj, a SimBiology model object, specified in estimated\_array, so as to match the values of species and nonconstant parameters given by observed\_array with the target state, xtarget, whose time variation is given by the time span tspan. If you have Optimization Toolbox installed, sbioparamestim uses the lsqnonlin function as the default method for the parameter estimation. If you do not have Optimization Toolbox

installed, `sbioparamestim` uses the MATLAB function `fminsearch` as the default method for the parameter estimation.

`[ ___ ]= sbioparamestim( ___, observed_array, estimated_array, k0)` specifies the initial values of compartments, species, and parameters listed in `estimated_array`.

`[ ___ ]= sbioparamestim( ___, observed_array, estimated_array, k0, method)` specifies the optimization method to use.

## Algorithms

`sbioparamestim` estimates parameters by attempting to minimize the discrepancy between simulation results and the data to fit. The minimization uses one of these optimization algorithms: `fminsearch` (from MATLAB); `lsqcurvefit`, `lsqnonlinfit`, or `fmincon` (from Optimization Toolbox); or `patternsearch` or `ga` (from Global Optimization Toolbox). All optimization methods require an objective function as an input. This objective function takes as input a vector of parameter values and returns an estimate of the discrepancy between simulation and data. When using `lsqcurvefit` or `lsqnonlinfit` as the optimization method, this objective function returns a vector of the residuals. For other optimization methods, the objective function returns the 2-norm of the residuals.

## Examples

Given a model and some target data, estimate all of its parameters without explicitly specifying any initial values:

- 1 Load a model from the project, `gprotein_norules.sbproj`. The project contains two models, one for the wild-type strain (stored in variable `m1`), and one for the mutant strain (stored in variable `m2`). Load the G protein model for the wild-type strain.

```
sbioloadproject gprotein_norules m1;
```

- 2 Store the target data in a variable:

```
Gt = 10000;
tspan = [0 10 30 60 110 210 300 450 600]';
Ga_frac = [0 0.35 0.4 0.36 0.39 0.33 0.24 0.17 0.2]';
```

# sbioparamestim

---

```
xtarget = Ga_frac * Gt;
```

**3** Store all model parameters in an array:

```
p_array = sbioselect(m1, 'Type', 'parameter');
```

**4** Store the species that should match target:

```
Ga = sbioselect(m1, 'Type', 'species', 'Name', 'Ga');  
% In this example only one species is selected.  
% To match more than one targeted species data  
% replace with selected species array.
```

**5** Estimate the parameters:

```
[k, result] = sbioparamestim(m1, tspan, xtarget, Ga, p_array)
```

```
k =
```

```
0.0100  
0.0000  
0.0004  
4.0000  
0.0040  
1.0000  
0.0000  
0.1100
```

```
result =
```

```
    fval: 1.4193e+06  
  residual: [9x1 double]  
  exitflag: 2  
 iterations: 2  
  funccount: 27  
 algorithm: 'trust-region-reflective'  
  message: [1x413 char]
```

---

Estimate parameters specified in `p_array` for species Ga using different algorithms. This example uses data from the first example.

```
[k1,r1] = sbioparamestim(m1,tspan,xtarget,Ga,p_array, ...
    {}, 'fmincon');
[k2,r2] = sbioparamestim(m1,tspan,xtarget,Ga,p_array, ...
    {}, 'patternsearch');
[k3,r3] = sbioparamestim(m1,tspan,xtarget,Ga,p_array, ...
    {}, 'ga');
[k4,r4] = sbioparamestim(m1,tspan,xtarget,Ga,p_array, ...
    {}, 'pso');
```

---

Estimate parameters specified in `p_array` for species Ga, and change default optimization options to use user-specified options. This example uses data from the first example.

```
myopt1 = optimset('Display','iter');
[k1,r1] = sbioparamestim(m1,tspan,xtarget, ...
    Ga,p_array,{},{'fmincon',myopt1});

myopt2 = psoptimset('TolMesh',1.0e-4);
[k2,r2] = sbioparamestim(m1,tspan,xtarget, ...
    Ga,p_array,{},{'patternsearch',myopt2});

myopt3 = gaoptimset('PopulationSize',25, 'Generations', 10);
[k3,r3] = sbioparamestim(m1,tspan,xtarget, ...
    Ga,p_array,{},{'ga',myopt3});

myopt4 = optimoptions('pso','Display','iter');
[k4,r4] = sbioparamestim(m1,tspan,xtarget,Ga,p_array,{},{'pso',myopt4});
```

## References

- [1] Yi, T-M., Kitano, H., and Simon, M.I. (2003) A quantitative characterization of the yeast heterotrimeric G protein cycle. *PNAS* *100*, 10764–10769.
- [2] Iadevaia, S., Lu, Y., Morales, F.C., Mills, G.B., and Ram, P.T. (2010) Identification of Optimal Drug Combinations Targeting Cellular Networks: Integrating Phospho-Proteomics and Computational Network Analysis. *Cancer Research* *70*, 6704–6714.
- [3] Abraham, A., Guo, H., and Liu, H. (2006) *Swarm Intelligence: Foundations, Perspectives and Applications*. *Studies in Computational Intelligence*, 3–25.

## See Also

`sbiomodel` | `optimset` | `gaoptimset` | `psoptimset`

**Purpose**

Plot simulation results in one figure

**Syntax**

```
sbioplot(simDataObj)
sbioplot(simDataObj, fcnHandleValue, xArgsValue, yArgsValue)
```

**Arguments**

|                       |  |
|-----------------------|--|
| <i>simDataObj</i>     | A SimData object or an array of SimData objects, containing data from simulation of a model. |
| <i>fcnHandleValue</i> | Function handle.   |
| <i>xArgsValue</i>     | Cell array with the names of the states.   |
| <i>yArgsValue</i>     | Cell array with the names of the states.   |

**Description**

`sbioplot(simDataObj)` plots each simulation run for *simDataObj*, a SimBiology data object or array of data objects, in the same figure. The plot is a time plot of each state in *simDataObj*. The figure also shows a hierarchical display of all the runs in a tree, with the ability to choose which trajectories to display.

`sbioplot(simDataObj, fcnHandleValue, xArgsValue, yArgsValue)` plots each simulation run for the SimBiology data object, *simDataObj*, in the same figure. The plot is created by calling the function handle, *fcnHandleValue*, with input arguments *simDataObj*, *xArgsValue*, and *yArgsValue*.

*xArgsValue* and *yArgsValue* should be cell arrays with the names of the states. The function represented by the function handle should return an array of handles and names. The signature of the function is shown below.

```
function [handles, names] = functionName(simDataObj, xArgsValue, YArgsValue)
```

The output argument `handles` is a two-dimensional array of handles to the lines plotted by the function. Each column corresponds to a run and each row corresponds to the lines being plotted for a state. `names` is a one-dimensional cell array that contains the names to be displayed on

the nodes which are children of a Run Node. The length of names should be equal to the number of rows in the `handles` array returned.

## Examples

### Plot Data from an Ensemble Run

This example shows how to plot data from an ensemble run.

Load the radiodecay model.

```
sbioloadproject('radiodecay.sbproj','m1');
```

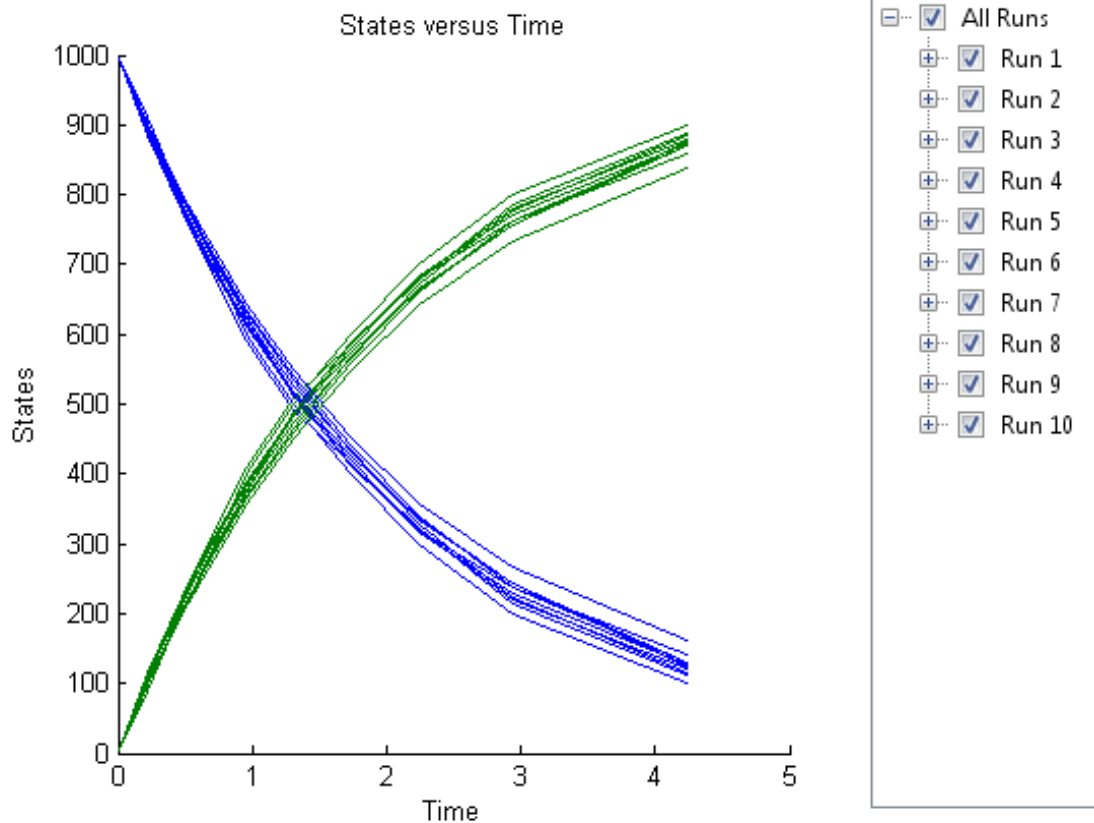
Configure the model to run with the stochastic solver. By default, there will be no interpolation.

```
cs = getconfigset(m1, 'active');  
set(cs, 'SolverType', 'ssa');  
set(cs.SolverOptions, 'LogDecimation', 100);
```

Run an ensemble simulation and view the results.

```
simDataObj = sbioensamplerun(m1, 10, 'linear');  
sbioplot(simDataObj);
```





**See Also** `sbiosubplot` | SimData object

# sbioremovefromlibrary

---

**Purpose** Remove kinetic law, unit, or unit prefix from library

**Syntax**  
`sbioremovefromlibrary (Obj)`  
`sbioremovefromlibrary ('Type', 'Name')`

**Description** `sbioremovefromlibrary (Obj)` removes the kinetic law definition, unit, or unit prefix object (Obj) from the user-defined library. The removed component will no longer be available automatically in future MATLAB sessions.

`sbioremovefromlibrary` does not remove a kinetic law definition that is being used in a model.

You can use a built-in or user-defined kinetic law definition when you construct a kinetic law object with the method `addkineticlaw`.

`sbioremovefromlibrary ('Type', 'Name')` removes the object of type 'Type' with name 'Name' from the corresponding user-defined library. Type can be 'kineticlaw', 'unit' or 'unitprefix'.

To get a component of the built-in and user-defined libraries, use the commands `get(sbioroot, 'BuiltInLibrary')` and `get(sbioroot, 'UserDefinedLibrary')`.

To create a kinetic law definition, unit, or unit prefix, use `sbioabstractkineticlaw`, `sbiounit`, or `sbiounitprefix` respectively.

To add a kinetic law definition, unit, or unit prefix to the user-defined library, use the function `sbioaddtolibrary`.

## Examples

This example shows how to remove a kinetic law definition from the user-defined library.

- 1 Create a kinetic law definition.

```
abstkineticlawObj = sbioabstractkineticlaw('mylaw1', '(k1*s)/(k2+k1+s)');
```

- 2 Add the new kinetic law definition to the user-defined library.

```
sbioaddtolibrary(abstkineticlawObj);
```

`sbioaddtolibrary` adds the kinetic law definition to the user-defined library. You can verify this using `sbiowhos`.

```
sbiowhos -kineticlaw -userdefined
```

SimBiology Abstract Kinetic Law Array

| Index: | Library:    | Name:  | Expression:        |
|--------|-------------|--------|--------------------|
| 1      | UserDefined | mylaw1 | $(k1*s)/(k2+k1+s)$ |

**3** Remove the kinetic law definition.

```
sbioremovefromlibrary('kineticlaw', 'mylaw1');
```

## See Also

`sbioaddtolibrary` | `sbioabstractkineticlaw` | `sbiounit` | `sbiounitprefix`

# sbioreset

---

**Purpose** Delete all model objects

**Syntax** sbioreset

**Description** sbioreset deletes all SimBiology model objects at the root level. You cannot use a SimBiology model object after it is deleted.

---

**Tip** To remove a SimBiology model object from the MATLAB workspace, without deleting it from the root level, use the `clear` function.

---

---

**Note** If the SimBiology desktop is open, calling `sbioreset` at the command line deletes all model objects that are open in the desktop.

---

The SimBiology root object contains a list of SimBiology model objects, available units, unit prefixes, and kinetic law objects. A SimBiology model object has its `Parent` property set to the SimBiology root object.

To add a kinetic law definition to the SimBiology root user-defined library, use the `sbioaddtolibrary` function. To add a unit to the SimBiology user-defined library on the root, use `sbiounit` followed by `sbioaddtolibrary`. To add a unit prefix to the SimBiology user-defined library on the root, use `sbiounitprefix` followed by `sbioaddtolibrary`.

## Examples

This example shows the difference between `sbioreset` and `clear all`.

**1** Import a model into the workspace.

```
modelObj = sbmlimport('oscillator');
```

Note that the workspace contains `modelObj` and if you query the SimBiology root, there is one model on the root object.

```
rootObj = sbioroot
```

SimBiology Root Contains:

|                                |    |
|--------------------------------|----|
| Models:                        | 1  |
| Builtin Abstract Kinetic Laws: | 3  |
| User Abstract Kinetic Laws:    | 0  |
| Builtin Units:                 | 54 |
| User Units:                    | 0  |
| Builtin Unit Prefixes:         | 13 |
| User Unit Prefixes:            | 0  |

- 2** Use `clear all` to clear the workspace. The `modelObj` still exists on the `rootObj`.

```
clear all
```

```
rootObj
```

SimBiology Root Contains:

|                                |    |
|--------------------------------|----|
| Models:                        | 1  |
| Builtin Abstract Kinetic Laws: | 3  |
| User Abstract Kinetic Laws:    | 0  |
| Builtin Units:                 | 54 |
| User Units:                    | 0  |
| Builtin Unit Prefixes:         | 13 |
| User Unit Prefixes:            | 0  |

- 3** Use `sbioreset` to delete the `modelObj` from the root.

```
sbioreset
```

```
rootObj
```

SimBiology Root Contains:

|         |   |
|---------|---|
| Models: | 0 |
|---------|---|

# sbioreset

---

|                                |    |
|--------------------------------|----|
| Builtin Abstract Kinetic Laws: | 3  |
| User Abstract Kinetic Laws:    | 0  |
| Builtin Units:                 | 54 |
| User Units:                    | 0  |
| Builtin Unit Prefixes:         | 13 |
| User Unit Prefixes:            | 0  |

## See Also

[sbioaddtolibrary](#) | [sbioroot](#) | [sbiounit](#) | [sbiounitprefix](#)

## How To

- [sbioroot](#)

**Purpose** Return SimBiology root object

**Syntax** `rootObj = sbioroot`

**Arguments** `rootObj` Return sbioroot to this object.

**Description** `rootObj = sbioroot` returns the SimBiology root object to root. The SimBiology root object contains a list of the SimBiology model objects, available units, unit prefixes, and available kinetic laws.

The units define the set of built-in units and user-defined units. See `Unit` object for more information.

The unit prefixes define the set of built-in prefixes and user-defined prefixes. See `Unit Prefix` object for more information.

The kinetic laws define the built-in kinetic laws and user-defined kinetic laws. See `AbstractKineticLaw` object for more information.

To add a unit, prefix or kinetic law to the root (in the user-defined library), use the `sbioaddtolibrary` function. To remove, use `sbiorremovefromlibrary`.

The models opened in the SimBiology desktop are stored in the root object.

**Method Summary**

- `copyobj` (any object) Copy SimBiology object and its children
- `get` (any object) Get object properties
- `reset` (root) Delete all model objects from root object
- `set` (any object) Set object properties

## Property Summary

|                    |                                    |
|--------------------|------------------------------------|
| BuiltInLibrary     | Library of built-in components     |
| Models             | Contain all model objects          |
| Type               | Display SimBiology object type     |
| UserDefinedLibrary | Library of user-defined components |

## See Also

`addkineticlaw` | `sbiomodel` | `sbioreset` | `Unit object` | `UnitPrefix object`

## How To

- `sbiomodel`
- `addkineticlaw`
- `sbioreset`



## Purpose

Generate parameters by sampling covariate model

## Syntax

```
phi = sbiosampleparameters(covexpr, fe, omega, ds)
phi = sbiosampleparameters(covexpr, fe, omega, n)
[phi, covmodel] = sbiosampleparameters(_)
```

## Description

`phi = sbiosampleparameters(covexpr, fe, omega, ds)` generates a matrix `phi` containing sampled parameter values using the covariate model specified by the covariate expression `covexpr`, fixed effects `fe`, covariance matrix `omega`, and covariate data `ds`.

`phi = sbiosampleparameters(covexpr, fe, omega, n)` uses a scalar `n` that specifies the number of rows in `phi` when the parameters are not dependent on any covariate.

`[phi, covmodel] = sbiosampleparameters(_)` returns a matrix `phi` and a covariate model object `covmodel` using any of the input arguments from previous syntaxes.

## Input Arguments

### **covexpr - Covariate expressions**

cell array of strings

Covariate expressions, specified as a cell array of strings that defines the parameter-covariate relationships.

---

**Note** See `CovariateModel` object to learn more about covariate expressions.

---

### **fe - Fixed effects**

vector | dataset | table

Fixed effects, specified as a vector, dataset, or table containing values for fixed effect parameters defined in the covariate expressions `covexpr`. Fixed effect parameter names must start with 'theta'.

# sbiosampleparameters

---

When `fe` is specified as a vector, it must be in the increasing order of the suffixes in `'theta'` if they are numeric such as `'theta1'`, `'theta2'`, etc. If the suffixes are nonnumeric or mixed, `fe` must be specified in ascending ASCII dictionary order.

---

**Tip** Use the `sort` function to sort a cell array of strings to see the ASCII dictionary order.

```
sort({'thetaone', 'theta2', 'thetax', 'theta4'})
```

```
ans =
```

```
    'theta2'    'theta4'    'thetaone'    'thetax'
```

Then specify parameter values in the same order.

```
thetas = [0.1 1.3 0.3 4.1];
```

where `'theta2'` has the value of 0.1, `'theta4'`, 1.3, etc.

---

## **omega - Covariance matrix of random effects**

matrix | dataset | table

Covariance matrix of random effects, specified as a matrix or dataset or table. Random effect parameter names must start with `'eta'`.

When `omega` is specified as a matrix, it must be in the increasing order of suffixes in `'eta'` such as `'eta1'`, `'eta2'`, etc. If the suffixes are nonnumeric or mixed, `omega` must be specified in ascending ASCII dictionary order. An example of a diagonal covariance matrix with three random effect parameters (`eta1`, `eta2`, and `eta3`) is

$$\begin{bmatrix} \text{Cov}(\eta_1, \eta_1) & \text{Cov}(\eta_1, \eta_2) & \text{Cov}(\eta_1, \eta_3) \\ \text{Cov}(\eta_2, \eta_1) & \text{Cov}(\eta_2, \eta_2) & \text{Cov}(\eta_2, \eta_3) \\ \text{Cov}(\eta_3, \eta_1) & \text{Cov}(\eta_3, \eta_2) & \text{Cov}(\eta_3, \eta_3) \end{bmatrix} = \begin{bmatrix} \text{eta1} & 0 & 0 \\ 0 & \text{eta2} & 0 \\ 0 & 0 & \text{eta3} \end{bmatrix}$$

If `omega` is a dataset, `omega.Properties.VarNames` must match the names of the random effects.

If `omega` is a table, `omega.Properties.VariableNames` must match the names of the random effects.

## **ds - Covariate data**

dataset | table

Covariate data, specified as a dataset or table containing the covariate data for all groups.

`ds` must have a column named 'Group' or 'GROUP' specifying the group labels as well as a column each for all covariates used in the covariate model. The column names must match the names of the corresponding covariates used in the covariate expressions.

## **n - Number of rows in phi**

scalar

Number of rows in `phi`, specified as a scalar.

## **Output Arguments**

### **phi - Sampled parameter values**

matrix

Sampled parameter values, returned as a matrix of size S-by-P, where S is the number of groups specified in `ds` or specified by `n` and P is the number of parameters which is equal to the number of elements in `covexpr`.

### **covmodel - Covariate model**

CovariateModel object

Covariate model, returned as a `CovariateModel` object which represents the model defined by `covexpr`.

## Examples

### Sample Parameter Values from a Covariate Model

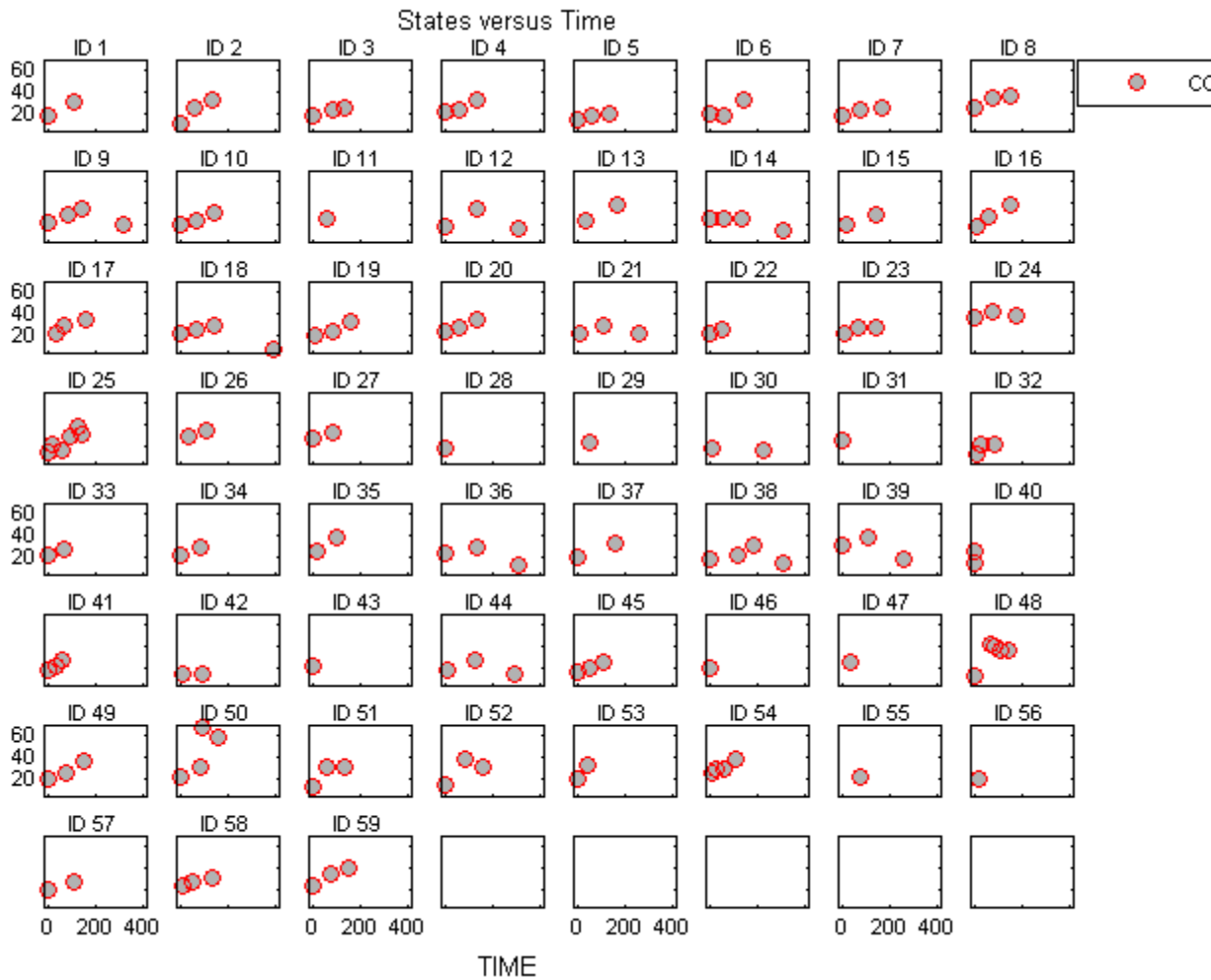
This example uses data collected on 59 preterm infants given phenobarbital during the first 16 days after birth. Each infant received an initial dose followed by one or more sustaining doses by intravenous bolus administration. A total of between 1 and 6 concentration measurements were obtained from each infant at times other than dose times, for a total of 155 measurements. Infant weights and APGAR scores (a measure of newborn health) were also recorded. Data was described in [1], a study funded by the NIH/NIBIB grant P41-EB01975.

Load the data.

```
load pheno.mat ds
```

Visualize the data.

```
t = sbiotrellis(ds, 'ID', 'TIME', 'CONC', 'marker', 'o', 'markerfacecolor', [.7  
t.plottitle = 'States versus Time';  
t.updateFigureForPrinting();
```



Create a one-compartment PK model with bolus dosing and linear clearance to model such data.

```
pkmd = PKModelDesign;
```

## sbiosampleparameters

---

```
pkmd.addCompartment('Central','DosingType','Bolus','EliminationType','lin  
onecomp = pkmd.construct;
```

Suppose there is a correlation between the volume of the central compartment (Central) and the weight of infants. You can define this parameter-covariate relationship using a covariate model that can be described as

$$\log(V_i) = \theta_V + \theta_V / \text{WEIGHT} * \text{WEIGHT}_i + \eta_{V,i},$$

where, for each  $i$ th infant,  $V$  is the volume,  $\theta$ s (thetas) are fixed effects,  $\eta$  (eta) represents random effects, and  $\text{WEIGHT}$  is the covariate.

```
covM = CovariateModel;  
covM.Expression = {'Central = exp(theta1+theta2*WEIGHT+eta1)'};
```

Define the fixed and random effects.

```
thetas = [1.4 0.05];  
eta1 = [0.2];
```

Change the group label of  $ds$  to 'GROUP' as required by the `sbiosampleparameters` function.

```
ds.Properties.VarNames{1} = 'GROUP';
```

Generate parameter values for the volumes of central compartments Central based on the covariate model for all infants in the data set.

```
phi = sbiosampleparameters(covM.Expression,thetas,eta1,ds);
```

You can then simulate the model using the sampled parameter values. For convenience, use the function-like interface provided by a `SimFunction` object.

First, construct a `SimFunction` object using the `createSimFunction` method, specifying the volume (Central) as the parameter, and the drug concentration in the compartment (Drug\_Central) as the output of the `SimFunction` object, and the dosed species.

```
f = createSimFunction(onecomp,covM.ParameterNames,'Drug_Central','Drug')
```

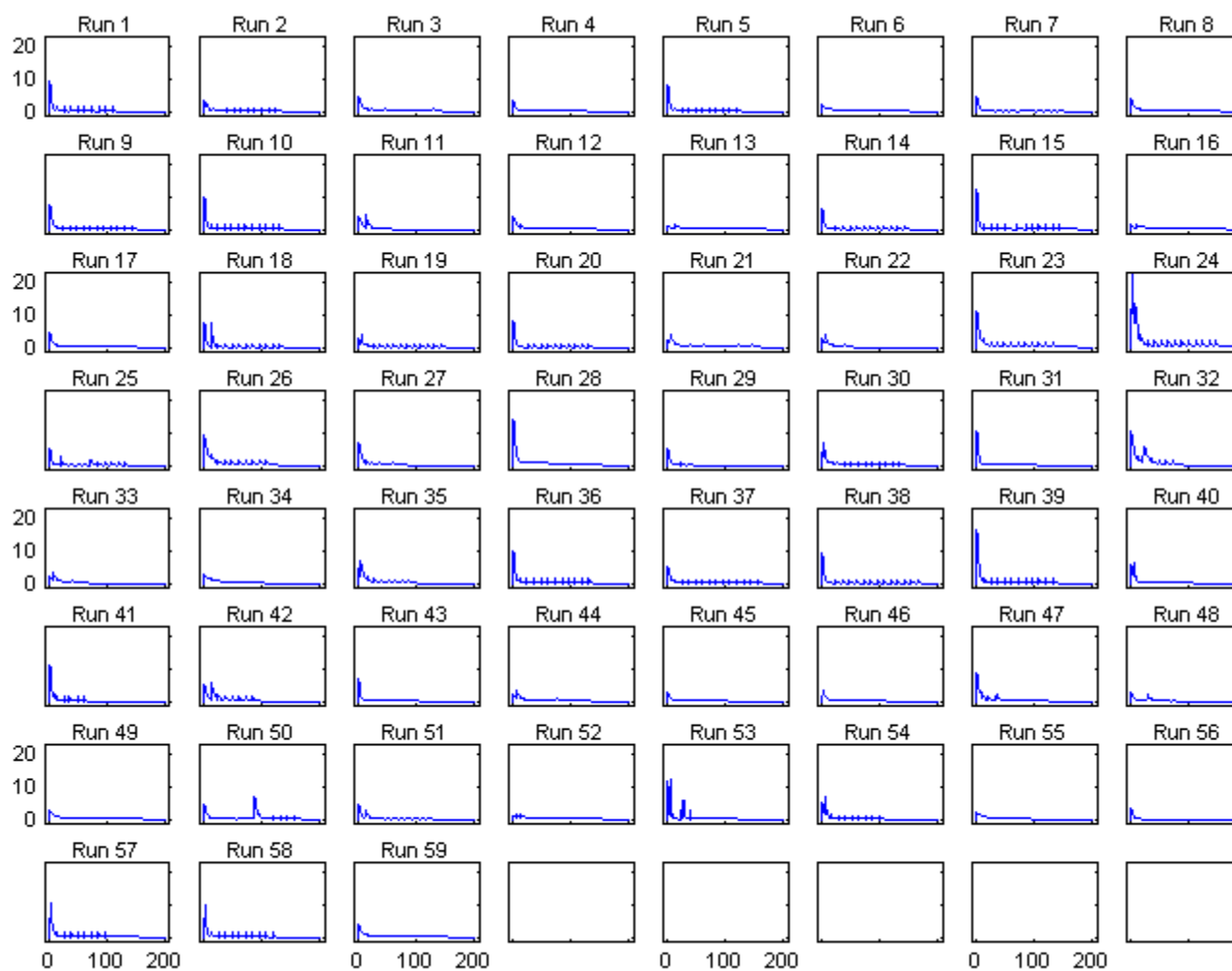
The data set `ds` contains dosing information for each infant, and the `groupedData` object provides a convenient way to extract such dosing information. Convert `ds` to a `groupedData` object and extract dosing information.

```
grpData = groupedData(ds);  
doses = createDoses(grpData,'DOSE');
```

Simulate the model using the sampled parameter values from `phi` and the extracted dosing information of each infant, and plot the results. The `ith` run uses the `ith` parameter value in `phi` and dosing information of the `ith` infant.

```
sbiotrellis(f(phi,200, doses.getTable), [], 'TIME', 'Drug_Central');
```

# sbiosampleparameters





## References

[1] Grasela Jr, T.H., Donn, S.M. (1985) Neonatal population pharmacokinetics of phenobarbital derived from routine clinical data. *Dev Pharmacol Ther.* 8(6), 374–83.

## See Also

`sbiosampleerror` | `createSimFunction` | `SimFunction` object | `CovariateModel` object

# sbiosampleerror

---

**Purpose** Sample error based on error model and add noise to simulation data

**Syntax** `sdN = sbiosampleerror(sd,errormodel,errorparam)`

**Description** `sdN = sbiosampleerror(sd,errormodel,errorparam)` adds noise to the simulation data `sd` using one or more error models `errormodel` and error parameters `errorparam`.

## Input Arguments

### **sd - Simulation results**

`SimData` object | matrix

Simulation results, specified as a `SimData` object or matrix.

### **errormodel - Error model**

string | function handle | cell array

Error model(s), specified as a string, function handle, or cell array containing strings, function handles, or a mixture of strings and function handles.

If it is a cell array, its length must match the number of columns (responses) in `sd`, and each error model is applied to the corresponding column in `sd`. If it is a single string or function handle, the same error model is applied to all columns in `sd`.

The first argument of a function handle must be a matrix of simulation results. The subsequent arguments are the parameters of the error model supplied in the `errorparam` input argument. The output of the function handle must be a matrix of the same size as the first input argument (simulation results).

For example, suppose you have a custom error model with a standard mean-zero and unit-variance (Gaussian) normal variable  $e$ , simulation results  $f$ , and two parameters  $p1$  and  $p2$ :  $y = f + p1 + p2 * e$ . You can define the corresponding function handle as follows.

```
em = @(y,p1,p2) y+p1+p2*randn(size(y));
```

where  $y$  is the matrix of simulation results and  $p1$  and  $p2$  are the error parameters. The output of the function handle must be the same size as  $y$ , which is the same as the simulation results specified in the `sd` input argument. The parameters  $p1$  and  $p2$  are specified in the `errorparam` argument.

There are four built-in error models. Each model defines the error using a standard mean-zero and unit-variance (Gaussian) variable  $e$ , simulation results  $f$ , and one or two parameters  $a$  and  $b$ . The models are:

- 'constant':  $y = f + ae$
- 'proportional':  $y = f + b|f|e$
- 'combined':  $y = f + (a+b|f|)e$
- 'exponential':  $y = f * \exp(ae)$

### **errorparam - Error model parameter**

scalar | vector | cell array

Error model parameter(s), specified as a scalar, vector, or cell array. If `errormodel` is 'constant', 'proportional', or 'exponential', then `errorparam` is specified as a numeric scalar. If it is 'combined', then `errorparam` is specified as a row vector with two elements [a b].

If `errormodel` is a cell array, then `errorparam` must be a cell array of the same length. In other words, `errorparam` must contain  $N$  elements, where  $N$  is the number of error models in `errormodel`. Each element must have the correct number of parameters for the corresponding error model.

For example, suppose you have three columns in `sd`, and you are applying a different error model (constant, proportional, and exponential error models with error parameters 0.1, 2, and 0.5 respectively) to each column, then `errormodel` and `errorparam` must be cell arrays with three elements as follows.

```
errormodel = {'constant','proportional','exponential'};
```

```
00errorparam = {0.1,2,0.5};
```

## Output Arguments

### **sdN - Simulation results with added noise**

vector of SimData objects | matrix

Simulation results with added noise, returned as a vector of SimData objects or numeric matrix. If `sd` is a vector of SimData objects, `sdN` is also a vector of SimData objects, and the error is added to each column in the `sd.Data` property. If `sd` is specified as a matrix, `sdN` is a matrix, and the error is added to each column in the matrix.

## Examples

### **Add Noise to Simulation Data**

This example adds noise/error to the simulation data from a radioactive

decay model with the first-order reaction  $\frac{dz}{dt} = c \cdot x$ , where  $x$  and  $z$  are species and  $c$  is the forward rate constant.

Load the sample project containing the radiodecay model `m1`.

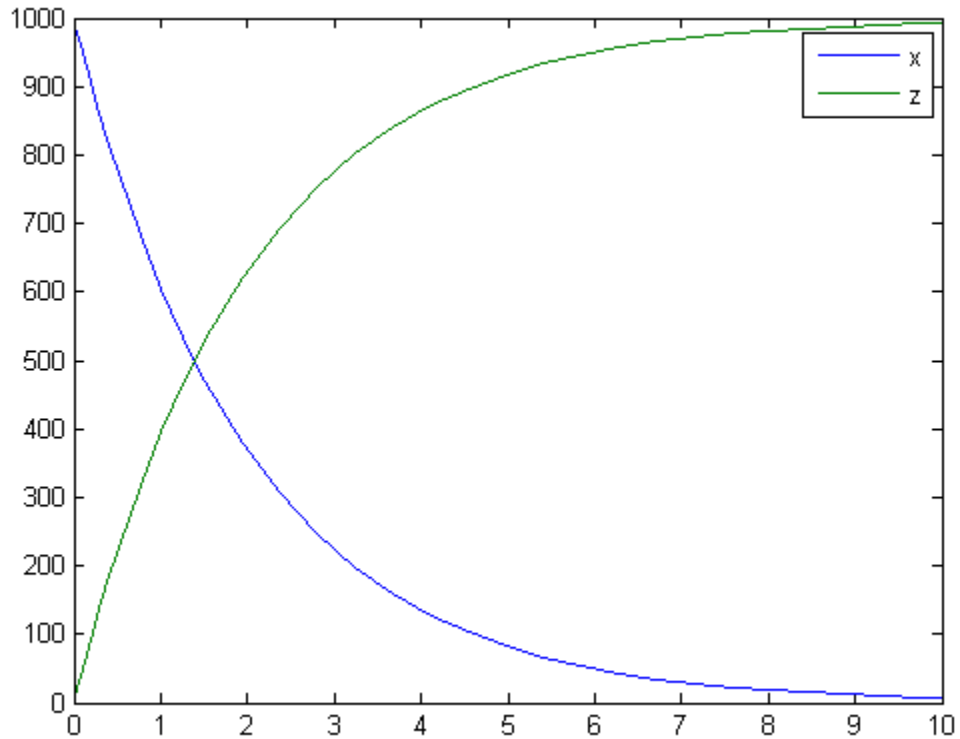
```
sbioloadproject radiodecay;
```

Simulate the model.

```
[t,sd,names] = sbiosimulate(m1);
```

Plot the simulation results.

```
plot(t,sd);  
legend(names);  
hold on
```

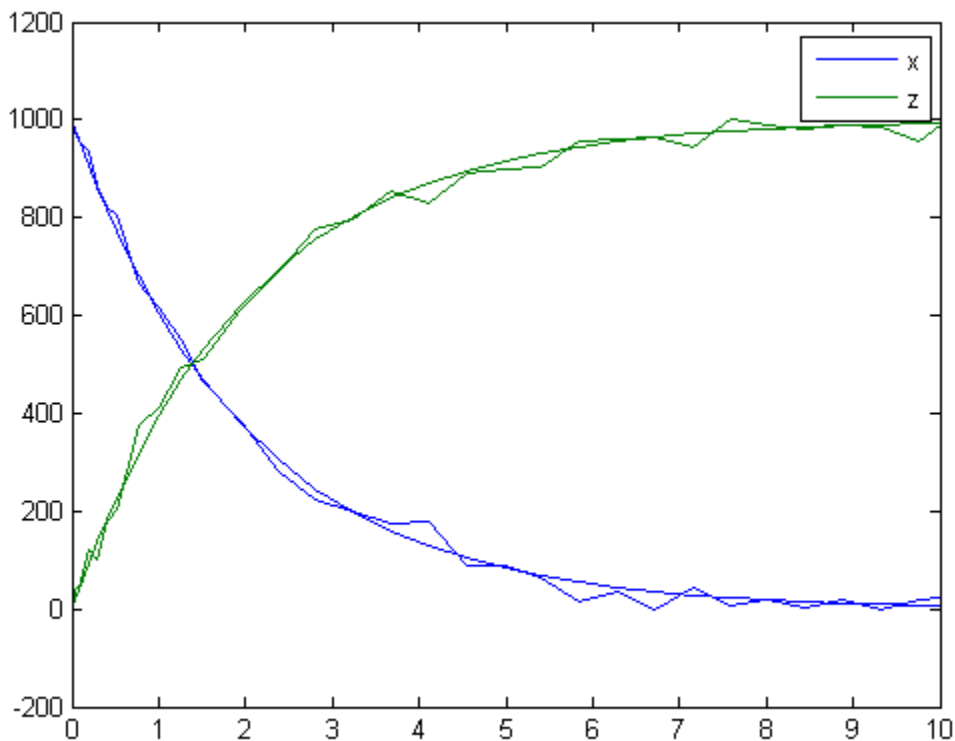


Add noise to the simulation results using the constant error model with the error parameter set to 20.

```
sdNoisy = sbiosampleerror(sd,'constant',20);
```

Plot the noisy simulation data.

```
plot(t,sdNoisy);
```



## Define a Custom Error Model Using a Function Handle

This example defines a custom error model using a function handle and adds noise to simulation data of a radioactive decay model with

the first-order reaction  $\frac{dz}{dt} = c \cdot x$ , where  $x$  and  $z$  are species, and  $c$  is the forward rate constant.

Load the sample project containing the radiodecay model `m1`.

```
sbioloadproject radiodecay;
```

Suppose you have a simple custom error model with a standard mean-zero and unit-variance (Gaussian) normal variable  $e$ , simulation results  $f$ , and two parameters  $p1$  and  $p2$ :  $y = f + p1 + p2 * e$ .

Define a function handle that represents the error model.

```
em = @(y,p1,p2) y+p1+p2*randn(size(y));
```

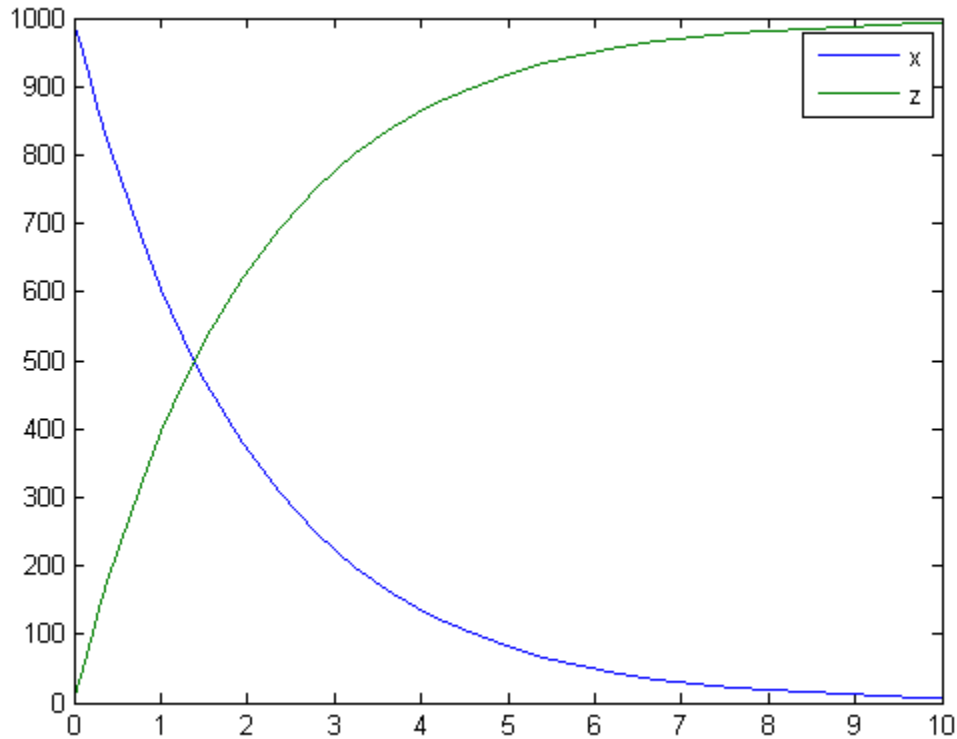
Simulate the model.

```
[t,sd,names] = sbiosimulate(m1);
```

Plot the simulation results and hold the plot.

```
plot(t,sd);  
legend(names);  
hold on
```

# sbiosampleerror



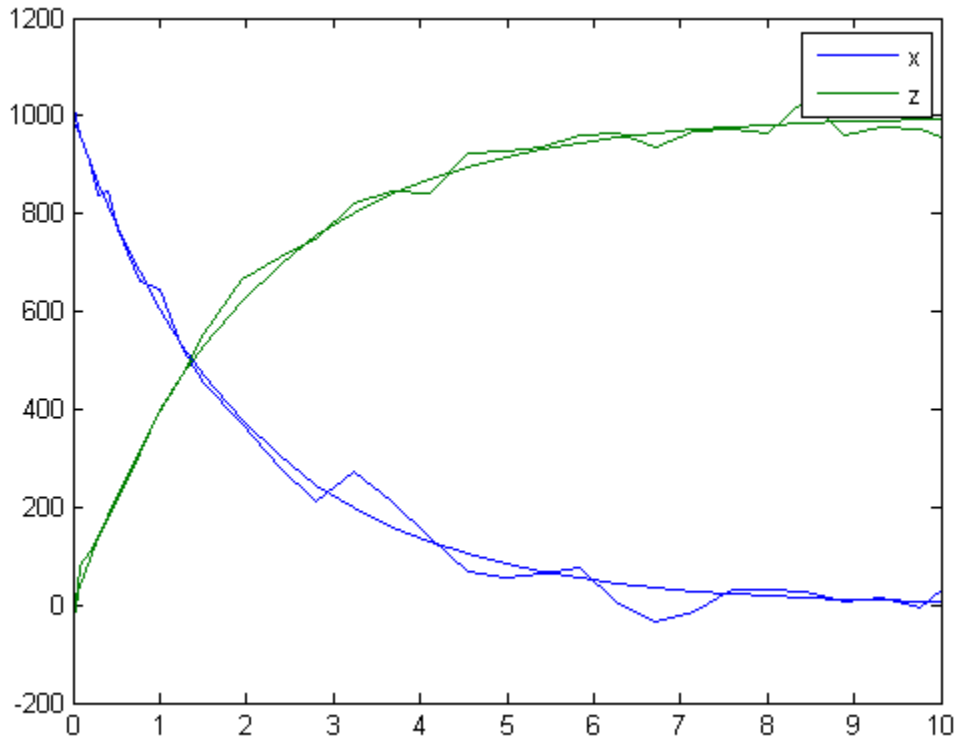
Sample the error using the previously defined custom function with two parameters set to 0.5 and 30, respectively.

```
sdNoisy = sbiosampleerror(sd,em,{0.5,30});
```

Plot the noisy simulation data.

```
plot(t,sdNoisy);
```



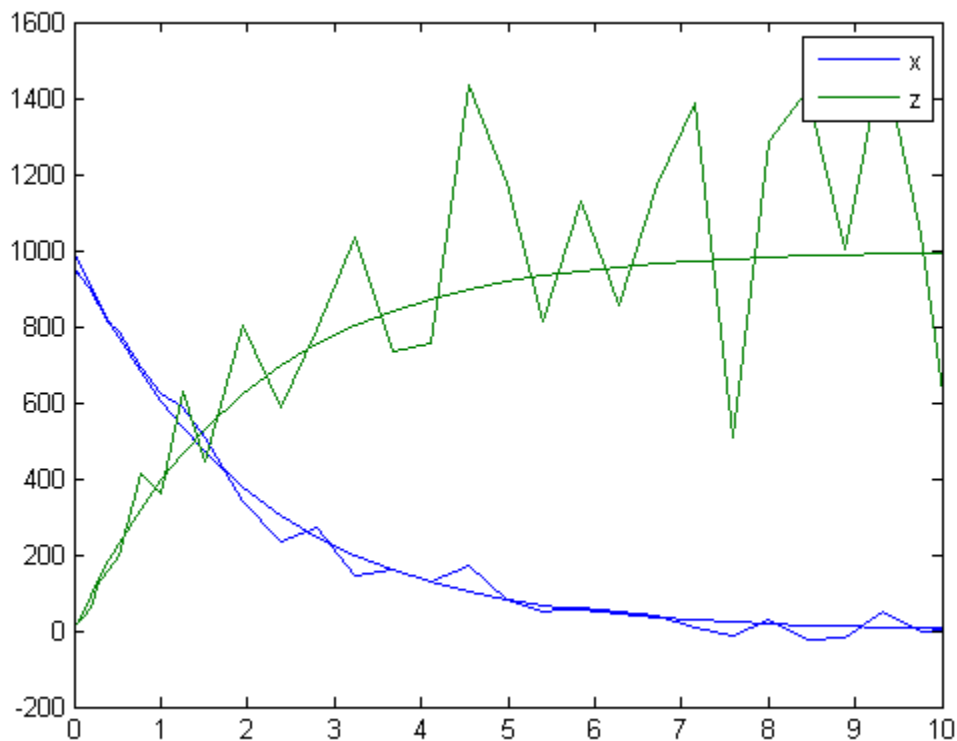


You can also apply a different error model to each state, which is a column in `sd`. Suppose you want to apply the custom error model (`em`) to the first column (species `x` data) and the proportional error model to the second column (species `z` data).

```
hold off
sdNoisy = sbiosampleerror(sd,{em,'proportional'},{{0.5,30},0.3});
plot(t,sd);
legend(names);
hold on
```

# sbiosampleerror

```
plot(t,sdNoisy);
```



## See Also

[sbiosampleparameters](#) | [createSimFunction](#) | [SimFunction](#) object

**Purpose** Save all models in root object

**Syntax**

```
sbiosaveproject projFilename
sbiosaveproject projFilename variableName
sbiosaveproject projFilename variableName1 variableName2 ...
```

**Description**

`sbiosaveproject projFilename` saves all models in the SimBiology root object to the binary SimBiology project file named `projFilename.sbproj`. The project can be loaded with `sbioloadproject`. `sbiosaveproject` returns an error if `projFilename.sbproj` is not writable.

`sbiosaveproject` creates the binary SimBiology project file named `simbiology.sbproj`. `sbiosaveproject` returns an error if this is not writable.

`sbiosaveproject projFilename variableName` saves only `variableName`. `variableName` can be a SimBiology model or any MATLAB variable.

`sbiosaveproject projFilename variableName1 variableName2 ...` saves the specified variables in the project.

Use the functional form of `sbiosaveproject` when the file name or variable names are stored in a string. For example, if the file name is stored in the variable `fileName` and you want to store MATLAB variables `variableName1` and `variableName2`, type `sbiosaveproject(projFileName, 'variableName1', 'variableName2')` at the command line.

**Examples**

- 1 Import an SBML file and simulate (default configset object is used).

```
modelObj = sbmlimport('oscillator.xml');
timeseriesObj = sbiosimulate(modelObj);
```

- 2 Save the model and the simulation results to a project.

```
sbiosaveproject myprojectfile modelObj timeseriesObj
```

# sbiosaveproject

---

## See Also

[sbioaddtolibrary](#) | [sbioloadproject](#) | [sbioremovefromlibrary](#)  
| [sbiowhos](#)

## How To

- [sbioloadproject](#)
- [sbiowhos](#)
- [sbioaddtolibrary](#)
- [sbioremovefromlibrary](#)

**Purpose**

Search for objects with specified constraints

**Syntax**

```

Out = sbioselect('PropertyName', PropertyValue)
Out = sbioselect('Where', 'PropertyName', 'Condition',
PropertyValue)
Out = sbioselect(Obj, 'PropertyName', PropertyValue)
Out = sbioselect(Obj, 'Type', 'TypeValue', 'PropertyName',
PropertyValue)
Out = sbioselect(Obj, 'Where', 'PropertyName', 'Condition',
PropertyValue)
Out = sbioselect(Obj, 'Where', 'PropertyNameCondition',
'PropertyNamePattern', 'Condition', PropertyValue)
Out = sbioselect(Obj, 'Where', 'PropertyName1', 'Condition1',
PropertyValue1, 'Where', 'PropertyName2', 'Condition2',
PropertyValue2,...)
Out = sbioselect(Obj, 'Where', 'PropertyName1', 'Condition1',
PropertyValue1,Bool_Operator, 'Where', 'PropertyName2',
'Condition2', PropertyValue2,...)
Out = sbioselect(Obj, 'Depth', DepthValue,...)

```

**Arguments**

- Out* Object or array of objects returned by the sbioselect function. *Out* might contain a mixture of object types (for example, species and parameters), depending on the selection you specify. If *PropertyValue* is a cell array, then the function returns all objects with the property '*PropertyName*' that matches any element of *PropertyValue*.
- Obj* SimBiology object or array of objects to search. If an object is not specified, sbioselect searches the root.
- PropertyName* Any property of the object being searched.
- PropertyValue* Specify *PropertyValue* to include in the selection criteria.
- TypeValue* Type of object to include in the selection, for example, sbiomodel, species, reaction, or kineticlaw.

|                              |  |
|------------------------------|--|
| <i>Condition</i>             | The search condition. See the table under “Description” on page 1-184 for a list of conditions.  |
| <i>PropertyNameCondition</i> | Search condition that applies only to property names (which are strings). See the table listing “Conditions for Properties Names or String Values” below.  |
| <i>PropertyNamePattern</i>   | String used to select the property name according to the condition imposed by <i>PropertyNameCondition</i> .   |
| <i>DepthValue</i>            | Specify the depth number to search. Valid numbers are positive integer values and <i>inf</i> . If <i>DepthValue</i> is <i>inf</i> , <i>sbioselect</i> searches <i>Obj</i> and all of its children. If <i>DepthValue</i> is 1, <i>sbioselect</i> only searches <i>Obj</i> and not its children. By default, <i>DepthValue</i> is <i>inf</i> . |

## Description

*sbioselect* searches for objects with specified constraints.

*Out* = *sbioselect*('PropertyName', *PropertyValue*) searches the root object (including all model objects contained by the root object) and returns the objects with the property name (*PropertyName*) and property value (*PropertyValue*) contained by the root object.

*Out* = *sbioselect*('Where', 'PropertyName', 'Condition', *PropertyValue*) searches the root object and finds objects that have a property name (*PropertyName*) and value (*PropertyValue*) that matches the condition (*Condition*).

*Out* = *sbioselect*(*Obj*, 'PropertyName', *PropertyValue*) returns the objects with the property name (*PropertyName*) and property value (*PropertyValue*) found in any object (*Obj*). If the property name in a property-value pair contains either a '?' or '\*', then the name is automatically interpreted as a wildcard expression, equivalent to the where clause ('Where', 'wildcard', 'PropertyName', '==', *PropertyValue*).

*Out* = *sbioselect*(*Obj*, 'Type', 'TypeValue', 'PropertyName', *PropertyValue*) finds the objects of type (*TypeValue*), with the property name (*PropertyName*) and property value (*PropertyValue*) found in any

object (*Obj*). *TypeValue* is the type of SimBiology object to be included in the selection, for example, species, reaction, or kineticlaw.

`Out = sbioselect(Obj, 'Where', 'PropertyName', 'Condition', PropertyValue)` finds objects that have a property name (*PropertyName*) and value (*PropertyValue*) that match the condition (*Condition*).

If you search for a string property value without specifying a condition, you must use the same format as `get` returns. For example, if `get` returns the Name as 'MyObject', `sbioselect` will not find an object with a Name property value of 'myobject'. Therefore, for this example, you must specify:

```
modelObj = sbioselect ('Name', 'MyObject')
```

Instead, if you use a condition, you can specify:

```
modelObj = sbioselect ('Where', 'Name', '==i', 'myobject')
```

Thus, conditions let you control the specificity of your selection.

`sbioselect` searches for model objects on the root in both cases.

`Out = sbioselect(Obj, 'Where', 'PropertyNameCondition', 'PropertyNamePattern', 'Condition', PropertyValue)` finds objects with a property name that matches the pattern in (*PropertyNamePattern*) with the condition (*PropertyNameCondition*) and matches the value (*PropertyValue*) with the condition (*Condition*). Use this syntax when you want search conditions on both property names and property values.

# sbioselect

The conditions, with examples of property names and corresponding examples of property values that you can use, are listed in the following tables. This table shows you conditions for numeric properties.

| Conditions for Numeric Properties | Example Syntax   |
|-----------------------------------|--|
| ==                                | <p>Search in the model object (<code>modelObj</code>), and return parameter objects that have <code>Value</code> equal to 0.5. <code>sbioselect</code> returns parameter objects because only parameter objects have a property called <code>Value</code>.</p> <pre>parameterObj = sbioselect (modelObj,...<br/>    'Where', 'Value', '==', 0.5)</pre> <p>In the case of <code>==</code>, this is equivalent to omitting the condition as shown:</p> <pre>parameterObj = sbioselect (modelObj,...<br/>    'Value', 0.5)</pre> <p>Search in the model object (<code>modelObj</code>), and return parameter objects that have <code>ConstantValue</code> false (nonconstant parameters).</p> <pre>parameterObj = sbioselect (modelObj,...<br/>    'Where', 'ConstantValue', '==', false)</pre> |
| ~=                                | <p>Search in the model object (<code>modelObj</code>), and return parameter objects that do not have <code>Value</code> equal to 0.5.</p> <pre>parameterObj = sbioselect (modelObj,...<br/>    'Where', 'Value', '~=', 0.5)</pre>  |



| <b>Conditions for Numeric Properties</b> | <b>Example Syntax</b>  |
|--|--|
| <p>&gt;,&lt;,&gt;=,&lt;=</p>             | <p>Search in the model object (modelObj), and return species objects that have an initial amount (InitialAmount) greater than 50.</p> <pre>speciesObj = sbioselect (modelObj, ...     'Where', 'InitialAmount', '&gt;', 50)</pre> <p>Search in the model object (modelObj), and return species objects that have an initial amount (InitialAmount) less than or equal to 50.</p> <pre>speciesObj = sbioselect (modelObj,...     'Where', 'InitialAmount', '&lt;=', 50)</pre> |
| <p>between</p>                           | <p>Search in the model object (modelObj), and return species objects that have an initial amount (InitialAmount) between 200 and 300.</p> <pre>speciesObj = sbioselect (modelObj,...     'Where', 'InitialAmount',...     'between', [200 300])</pre>  |
| <p>-between</p>                          | <p>Search in the model object (modelObj), and return species objects that have an initial amount (InitialAmount) that is not between 200 and 300.</p> <pre>speciesObj = sbioselect (modelObj,...     'Where', 'InitialAmount',...     '~between', [200 300])</pre>   |

| <b>Conditions for Numeric Properties</b> | <b>Example Syntax</b>   |
|--|---|
| equal_and_same_type                      | <p>Similar to ==, but in addition requires the property value to be of the same type. Search in the model object (modelObj), and return all objects containing a property of type double and a value equal to 0. (Using '==' would also select objects containing a property with a value of false.)</p> <pre>zeroObj = sbioselect(modelObj, ...<br/>'Where', '*', 'equal_and_same_type', 0);</pre> |
| unequal_and_same_type                    | <p>Similar to ~=, but in addition requires the property value to be of the same type. Select all objects containing a property of type double and value not equal to 0. (Using '~=' would also select objects containing a property with a value of true.)</p> <pre>nonzeroObj = sbioselect(modelObj, ...<br/>'Where', '*', 'unequal_and_same_type', 0);</pre>                                      |

The following table shows you conditions for properties names or for properties whose values are strings.

| <b>Conditions for Properties Names or String Values</b> | <b>Example Syntax</b>  |
|---|--|
| ==  | <p>Search in the model object (<code>modelObj</code>), and return species objects named 'Glucose'.</p> <pre>speciesObj = sbioselect (modelObj,...     'Type', 'species', 'Where',...     'Name', '==', 'Glucose')</pre>                              |
| ~=  | <p>Search in the model object (<code>modelObj</code>), and return species objects that are not named 'Glucose'.</p> <pre>speciesObj = sbioselect (modelObj,...     'Type', 'species', 'Where',...     'Name', '~=', 'Glucose')</pre>                 |
| ==i   | <p>Same as ==; in addition, this is case insensitive.</p>  |
| ~=i   | <p>Search in the model object (<code>modelObj</code>), and return species objects that are not named 'Glucose', ignoring case.</p> <pre>speciesObj = sbioselect (modelObj,...     'Type', 'species', 'Where',...     'Name', '~=i', 'glucose')</pre> |

# sbioselect

| Conditions for Properties Names or String Values                            | Example Syntax  |
|---|---|
| regexp. Supports expressions supported by the functions regexp and regexpi. | <p>Search in the model object (modelObj), and return objects that have 'ese' or 'ase' anywhere within the name.</p> <pre>Obj = sbioselect (modelObj, 'Where',...<br/>  'Name', 'regexp', '[ea]se')</pre> <p>Search in the root, and return objects that have kinase anywhere within the name.</p> <pre>Obj = sbioselect ('Where',...<br/>  'Name', 'regexp', 'kinase')</pre> <p>Note that this query could result in a mixture of object types (for example, species and parameters).</p> |
| regexpi   | Same as regexp; in addition, this is case insensitive.  |
| ~regexp   | <p>Search in the model object (modelObj), and return objects that do not have kinase anywhere within the name.</p> <pre>Obj = sbioselect (modelObj, 'Where',...<br/>  'Name', '~regexp', 'kinase')</pre>  |
| ~regexpi  | Same as ~regexp; in addition, this is case insensitive.   |
| wildcard  | Supports DOS-style wildcards ('?' matches any single character, '*' matches any number of characters, and the pattern must match the entire string). See <a href="#">regexptranslate</a> for more information.  |
| wildcardi   | Same as wildcard; in addition, this is case insensitive.  |

| <b>Conditions for Properties Names or String Values</b> | <b>Example Syntax</b>  |
|---|--|
| -wildcard   | <p>Search in the model object (modelObj), and return objects that have names that do not begin with kin*.</p> <pre>Obj = sbioselect (modelObj, 'Where', ...     'Name', '~wildcard', 'kin*')</pre> |
| -wildcardi  | <p>Same as -wildcard; in addition, this is case insensitive.</p>   |

# sbioselect

Use the condition type function for any property. The specified value should be a function handle that, when applied to a property value, returns a boolean indicating whether there is a match. The following table shows an example of using function.

| Condition  | Example Syntax  |
|------------|---|
| 'function' | <p>Search in the model object and return reaction objects whose Stoichiometry property contains the specified stoichiometry.</p> <pre>Out = sbioselect(modelObj, 'Where', ...<br/>  'Stoichiometry', 'function', ...<br/>  @(x)any(x&gt;2))</pre> <p>Select all objects with a numeric value that is even.</p> <pre>iseven = @(x) isnumeric(x)...<br/>  &amp;&amp; isvector(x) &amp;&amp; mod(x, 2) == 0;<br/>evenValuedObj = sbioselect(modelObj, ...<br/>  'where', 'Value', 'function', iseven);</pre> |

The condition 'contains' can be used only for those properties whose values are an array of SimBiology objects. The following table shows an example of using contains.

| Condition  | Example Syntax   |
|------------|--|
| 'contains' | <p>Search in the model object and return reaction objects whose Reactant property contains the specified species.</p> <pre>Out = sbioselect(modelObj, 'Where', ...<br/>  'Reactants', 'contains', ...<br/>  modelObj.Species(1))</pre> |

```
Out = sbioselect(Obj, 'Where', 'PropertyName1', 'Condition1',  
  PropertyValue1, 'Where', 'PropertyName2', 'Condition2',
```

*PropertyValue2*,...) finds objects contained by *Obj* that matches all the conditions specified.

You can combine any number of property name/property value pairs and conditions in the `sbioselect` command.

`Out = sbioselect(Obj, 'Where', 'PropertyName1', 'Condition1', PropertyValue1, Bool_Operator, 'Where', 'PropertyName2', 'Condition2', PropertyValue2, ...)` finds objects contained by *Obj* that matches all the conditions specified. Supported strings for *Bool\_Operator* are as follows.

'and' True if ( 'Where',  
                   'PropertyName1','Condition1',PropertyValue1) and (  
                   'Where', 'PropertyName2','Condition2',PropertyValue2)  
 are both true.

'or' True if either ( 'Where',  
                   'PropertyName1','Condition1',PropertyValue1) or (  
                   'Where', 'PropertyName2','Condition2',PropertyValue2) is

'xor' True if exactly one of ( 'Where', 'PropertyName1','Condition1',PropertyValue1) or (  
                   'Where', 'PropertyName2','Condition2',PropertyValue2)  
 is true.

'not' True if ( 'Where',  
                   'PropertyName1','Condition1',PropertyValue1) is true and  
                   ( 'Where', 'PropertyName2','Condition2',PropertyValue2)  
 is not true.

Compound expressions with multiple boolean operators are supported. Precedence of the operators follows the order of operations for boolean algebra not → and → xor → or.

`Out = sbioselect(Obj, 'Depth', DepthValue, ...)` finds objects using a model search depth of *DepthValue*.

---

**Note** The order of results from `sbioselect` is not guaranteed when returning results of multiple types. Hence it is not recommended to depend on the order of results.

---

## Examples

### Find Species from a SimBiology Model

Import a model.

```
modelObj = sbmlimport('oscillator');
```

Find and return an object named `pA`.

```
Obj = sbioselect(modelObj, 'Name', 'pA');
```

Find and return species objects whose `Name` starts with `p` and have `A` or `B` as the next letter in the name.

```
speciesObj = sbioselect(modelObj, 'Type', 'species', 'Where', ...  
    'Name', 'regexp', '^p[AB]');
```

Find a cell array. Note how cell array values must be specified inside another cell array.

```
modelObj.Species(2).UserData = {'a' 'b'};  
Obj = sbioselect(modelObj, 'UserData', {'a' 'b'})
```

SimBiology Species Array

| Index: | Compartment: | Name: | InitialAmount: | InitialAmountUnits: |
|--------|--------------|-------|----------------|---------------------|
| 1      | unnamed      | pB    | 0              |                     |

Find and return objects that do not have their units set.

```
unitlessObj = sbioselect(modelObj, 'Where', 'wildcard', '*Units', '==', '');
```

Alternatively, you can do the following.



```
unitlessObj = sbioselect(modelObj, '*Units', '');
```

## See Also

[regexp](#)

# sbioshowunitprefixes

---

**Purpose** Show unit prefixes in library

**Syntax**

```
UnitPrefixObjs = sbioshowunitprefixes  
[Name, Multiplier] = sbioshowunitprefixes  
[Name, Multiplier, Builtin] = sbioshowunitprefixes  
[Name, Multiplier, Builtin] = sbioshowunitprefixes('Name')
```

## Arguments

|                       |  |
|-----------------------|--|
| <i>unitPrefixObjs</i> | Vector of unit prefix objects from the BuiltInLibrary and UserDefinedLibrary properties of the Root object.  |
| <i>Name</i>           | Name of the built-in or user-defined unit prefix. Built-in prefixes are defined based on the International System of Units (SI).   |
| <i>Multiplier</i>     | Shows the value of $10^{\text{Exponent}}$ that defines the relationship of the unit prefix <i>Name</i> to the base unit. For example, the multiplier in picomole is $10e-12$ .       |
| <i>Builtin</i>        | An array of logical values. If <i>Builtin</i> is true for a unit prefix, the unit prefix is built in. If <i>Builtin</i> is false for a unit prefix, the unit prefix is user defined. |

**Description** sbioshowunitprefixes returns information about unit prefixes in the SimBiology library.

*UnitPrefixObjs* = sbioshowunitprefixes returns the unit prefixes in the library as a vector of unit prefix objects in *UnitPrefixObjs*.

[*Name*, *Multiplier*] = sbioshowunitprefixes returns the multiplier for each prefix in *Name* to *Multiplier* as a cell array of strings.

[*Name*, *Multiplier*, *Builtin*] = sbioshowunitprefixes returns whether the unit prefix is built in or user defined for each unit prefix in *Name* to *Builtin*.

`[Name, Multiplier, Builtin] = sbioshowunitprefixes('Name')`  
returns the name, multiplier, and built-in status for the unit prefix with name *Name*. *Name* can be a cell array of strings.

## Examples

```
[name, multiplier] = sbioshowunitprefixes;  
[name, multiplier] = sbioshowunitprefixes('nano');
```

## See Also

[sbioconvertunits](#) | [sbioshowunits](#) | [sbiounitprefix](#)

# sbioshowunits

---

**Purpose** Show units in library

**Syntax**

```
unitObjs = sbioshowunits  
[Name, Composition] = sbioshowunits  
[Name, Composition, Multiplier] = sbioshowunits  
[Name, Composition, Multiplier, Offset] = sbioshowunits  
[Name, Composition, Multiplier, Offset, Builtin] = sbioshowunits  
[Name, Composition, Multiplier, Offset,  
  Builtin] = sbioshowunits('Name')
```

## Arguments

|                    |   |
|--------------------|---|
| <i>unitObjs</i>    | Vector of unit objects from the <code>BuiltInLibrary</code> and <code>UserDefinedLibrary</code> properties of the <code>Root</code> object.   |
| <i>Name</i>        | Name of the built-in or user-defined unit.  |
| <i>Composition</i> | Shows the combination of base and derived units that defines the unit <i>Name</i> . For example, molarity is mole/liter.  |
| <i>Multiplier</i>  | The numerical value that defines the relationship between the unit <i>Name</i> and the base or derived unit as a product of the <i>Multiplier</i> and the base unit or derived unit. For example, 1 mole is $6.0221e23$ *molecule. The <i>Multiplier</i> is $6.0221e23$ . |
| <i>Offset</i>      | Numerical value by which the unit composition is modified from the base unit. For example, <code>Celsius = (5/9)*( Fahrenheit-32)</code> ; <i>Multiplier</i> is 5/9 and <i>Offset</i> is 32.  |
| <i>Builtin</i>     | An array of logical values. If <i>Builtin</i> is true for a unit, the unit is built in. If <i>Builtin</i> is false for a unit, the unit is user defined.  |

## Description

`unitObjs = sbioshowunits` returns the units in the library to `unitObjs` as a vector of unit objects.

`[Name, Composition] = sbioshowunits` returns the composition for each unit in `Name` to `Composition` as a cell array of strings.

`[Name, Composition, Multiplier] = sbioshowunits` returns the multiplier for the unit with name `Name` to `Multiplier`.

`[Name, Composition, Multiplier, Offset] = sbioshowunits` returns the offset for the unit with name `Name` to `Offset`. The unit is defined as  $Multiplier * Composition + Offset$ .

`[Name, Composition, Multiplier, Offset, Builtin] = sbioshowunits` returns whether the unit is built in or user defined for each unit in `Name` to `Builtin`.

`[Name, Composition, Multiplier, Offset, Builtin] = sbioshowunits('Name')` returns the name, composition, multiplier, offset and built-in status for the unit with name `Name`. `Name` can be a cell array of strings.

## Examples

```
[name, composition] = sbioshowunits;  
[name, composition] = sbioshowunits('molecule');
```

## See Also

`sbioconvertunits` | `sbioshowunitprefixes` | `sbiounit`

# sbiosimulate

---

**Purpose** Simulate SimBiology model

**Syntax**

```
[time,x, names] = sbiosimulate(modelObj)
[time,x, names] = sbiosimulate(modelObj,optionObj)
[time,x, names] = sbiosimulate(modelObj,csObj,dvObj)
[time,x, names] =
sbiosimulate(modelObj,csObj,variantObj,doseObj)

simDataObj = sbiosimulate( __ )
```

**Description** [time,x, names] = sbiosimulate(modelObj) returns simulation results in three outputs, **time**, vector of time samples, **x**, simulation data, and **names**, column labels of simulation data **x**. This function simulates the SimBiology model **modelObj** while using the model's active configuration set along with its active doses and variants, if available.

[time,x, names] = sbiosimulate(modelObj,optionObj) returns simulation results using an option object specified by **optionObj** that can be one of the following:

- Configset object
- Variant object
- ScheduleDose object
- RepeatDose object
- Array of doses or variants

[time,x, names] = sbiosimulate(modelObj,csObj,dvObj) returns simulation results using a configset object **csObj** and dose, variant, or an array of doses or variants specified by **dvObj**. If you set **csObj** to [], then the function uses the model's currently active configset object.

[time,x, names] = sbiosimulate(modelObj,csObj,variantObj,doseObj) returns

simulation results using a configset object `csObj`, variant object or variant array specified by `variantObj`, and dose object or dose array specified by `doseObj`.

`simDataObj = sbiosimulate( ___ )` returns simulation results in a `SimData` object `simDataObj` using any of the input arguments in the preceding syntaxes.

## Input Arguments

### **modelObj - SimBiology model**

SimBiology model object

SimBiology model, specified as a SimBiology model object. The model minimally needs one reaction or rate rule for simulations.

### **optionObj - Option object**

configset object | variant object or array of variant objects | dose object or array of dose objects

Option object, specified as a `configset` object, `variant` object, an array of variant objects, `ScheduleDose` object, `RepeatDose` object, or an array of dose objects.

### **csObj - Configuration set object**

configset object | []

Configuration set object, specified as a `configset` object that stores simulation-specific information. When you specify `csObj` as [], `sbiosimulate` uses the currently active configset object.

If your model contains events, the `csObj` object cannot specify `'expltau'` or `'impltau'` for the `SolverType` property.

If your model contains doses, the `csObj` object cannot specify `'ssa'`, `'expltau'`, or `'impltau'` for the `SolverType` property.

### **dvObj - Dose or variant object**

dose object or array of dose objects | variant object or array of variant objects | []

# sbiosimulate

---

Dose or variant object, specified as a `ScheduleDose` object , `RepeatDose` object , an array of dose objects, `variant` object , or an array of variant objects.

- Use `[]` when you want to explicitly exclude any variant objects from the `sbiosimulate` function.
- When `dvObj` is a dose object, `sbiosimulate` uses the specified dose object as well as any active variant objects if available.
- When `dvObj` is a variant object, `sbiosimulate` uses the specified variant object as well as any active dose objects if available.

## **variantObj - Variant object**

variant object or array of variant objects | `[]`

Variant object, specified as a `variant` object or an array of variant objects. Use `[]` when you want to explicitly exclude any variant objects from `sbiosimulate`.

## **doseObj - Dose object**

dose object or array of dose objects | `[]`

Dose object, specified as a `ScheduleDose` object , `RepeatDose` object , or an array of dose objects. A dose object defines additions that are made to species amounts or parameter values. Use `[]` when you want to explicitly exclude any dose objects from `sbiosimulate`.

## **Output Arguments**

### **time - Vector of time samples**

vector

Vector of time samples, returned as an  $n$ -by-1 vector containing the simulation time steps.  $n$  is the number of time samples.

### **x - Simulation data**

array

Simulation data, returned as an  $n$ -by- $m$  data array, where  $n$  is the number of time samples and  $m$  is the number of states logged in the



simulation. Each column of `x` describes the variation in the quantity of a species, compartment, or parameter over time.

### **names - Names of species, compartments, or parameters**

cell array of strings

Names of species, compartments, or parameters, returned as an `m-by-1` cell array of strings. In other words, `names` contains the column labels of the simulation data, `x`. If the species are in multiple compartments, species names are qualified with the compartment name in the form `compartmentName.speciesName`.

### **simDataObj - Simulation data**

`SimData` object

Simulation data, returned as a `SimData` object that holds time and state data as well as metadata, such as the types and names for the logged states or the configuration set used during simulation. You can access time, data, and names stored in a `SimData` object by using its properties.

## **Examples**

### **Simulate a SimBiology Model**

Load a sample `SimBiology` model.

```
sbioloadproject radiodecay.sbproj
```

Change the simulation stop time to 15 seconds.

```
csObj = getconfigset(m1,'active');  
set(csObj,'Stoptime',15);
```

Simulate the model and return outputs in an array.

```
[t,x,n] = sbiosimulate(m1);
```

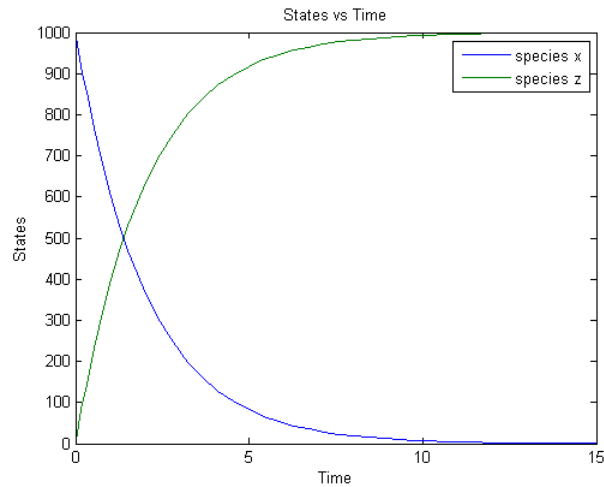
Plot the simulated results for species `x` and `z`.

```
figure;
```

# sbiosimulate

---

```
plot(t,x)
xlabel('Time')
ylabel('States')
title('States vs Time')
legend('species x','species z')
```

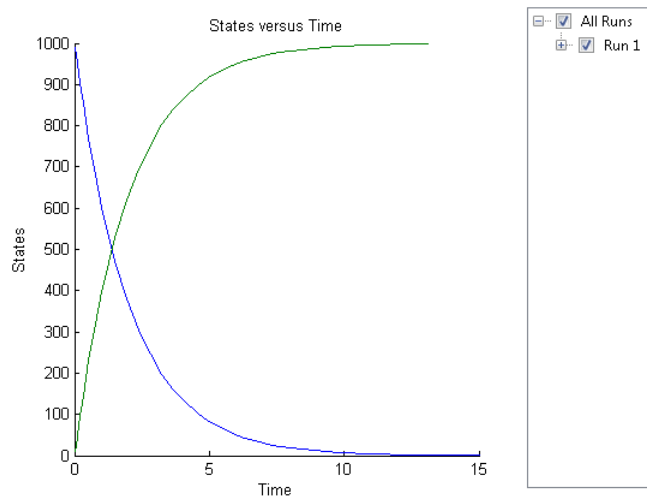


You can also return the results to a `SimData` object .

```
simData = sbiosimulate(m1);
```

Plot the simulated results.

```
sbioplot(simData);
```



## Simulate a SimBiology Model Using an Array of Dose Objects

Load a sample SimBiology model.

```
sbioloadproject radiodecay.sbproj
```

Add two doses of 100 molecules each for species x, scheduled at 2 and 4 seconds respectively.

```
dObj1 = adddose(m1,'d1','schedule');
dObj1.Amount = 100;
dObj1.AmountUnits = 'molecule';
dObj1.TimeUnits = 'second';
dObj1.Time = 2;
dObj1.TargetName = 'unnamed.x';
```

```
dObj2 = adddose(m1,'d2','schedule');
dObj2.Amount = 100;
dObj2.AmountUnits = 'molecule';
dObj2.TimeUnits = 'second';
dObj2.Time = 4;
```

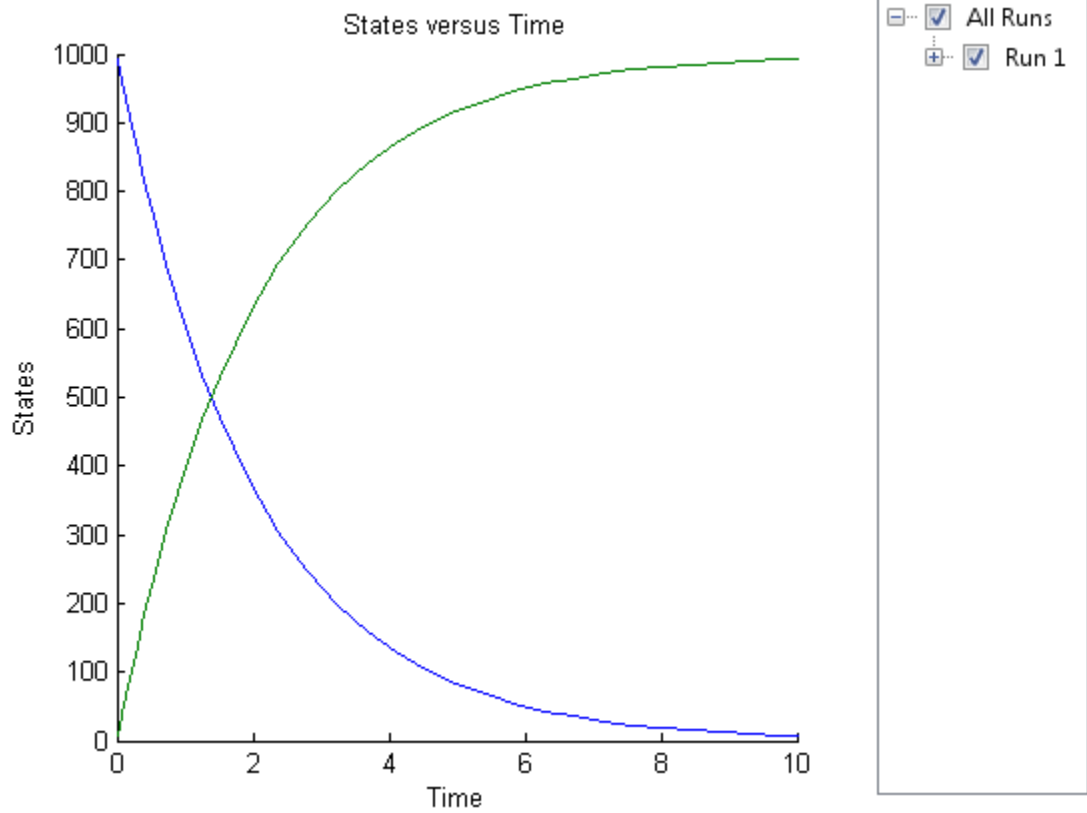
```
dObj2.TargetName = 'unnamed.x';
```

Simulate the model using no dose or any subset of the dose array.

```
sim1 = sbiosimulate(m1);  
sim2 = sbiosimulate(m1,dObj1);  
sim3 = sbiosimulate(m1,dObj2);  
sim4 = sbiosimulate(m1,[dObj1,dObj2]);
```

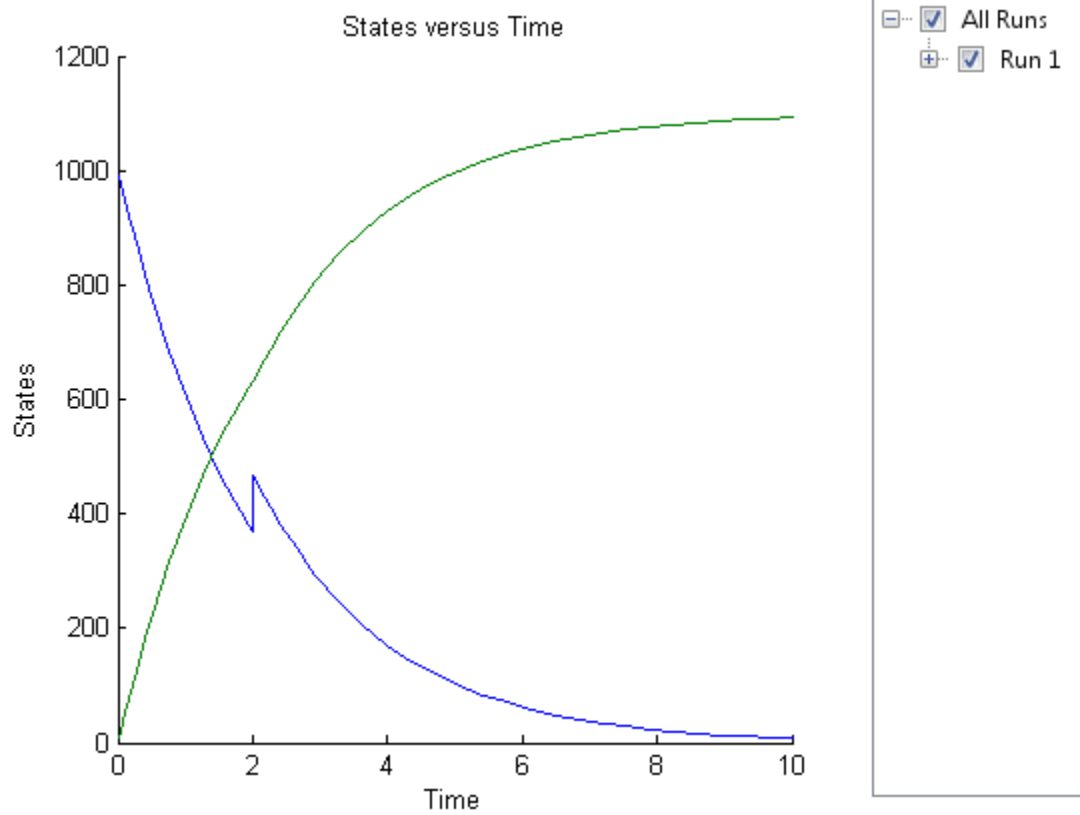
Plot the results.

```
sbioplot(sim1)
```

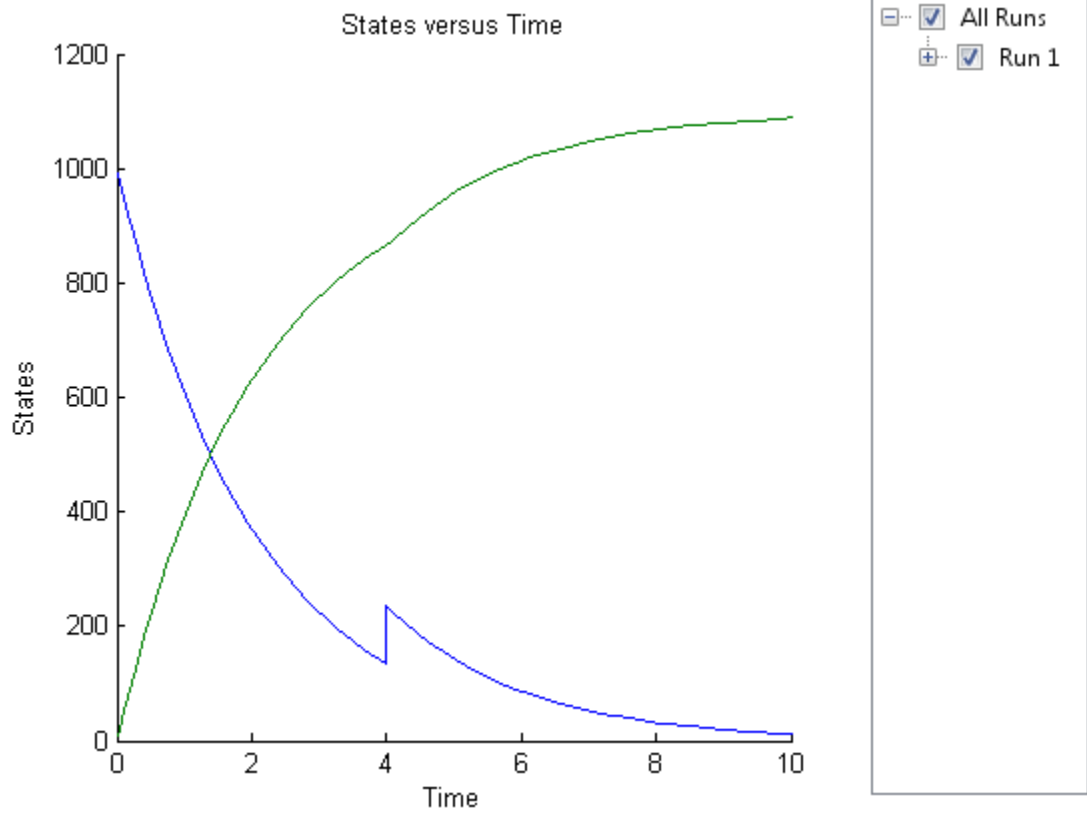


```
sbioplot(sim2)
```

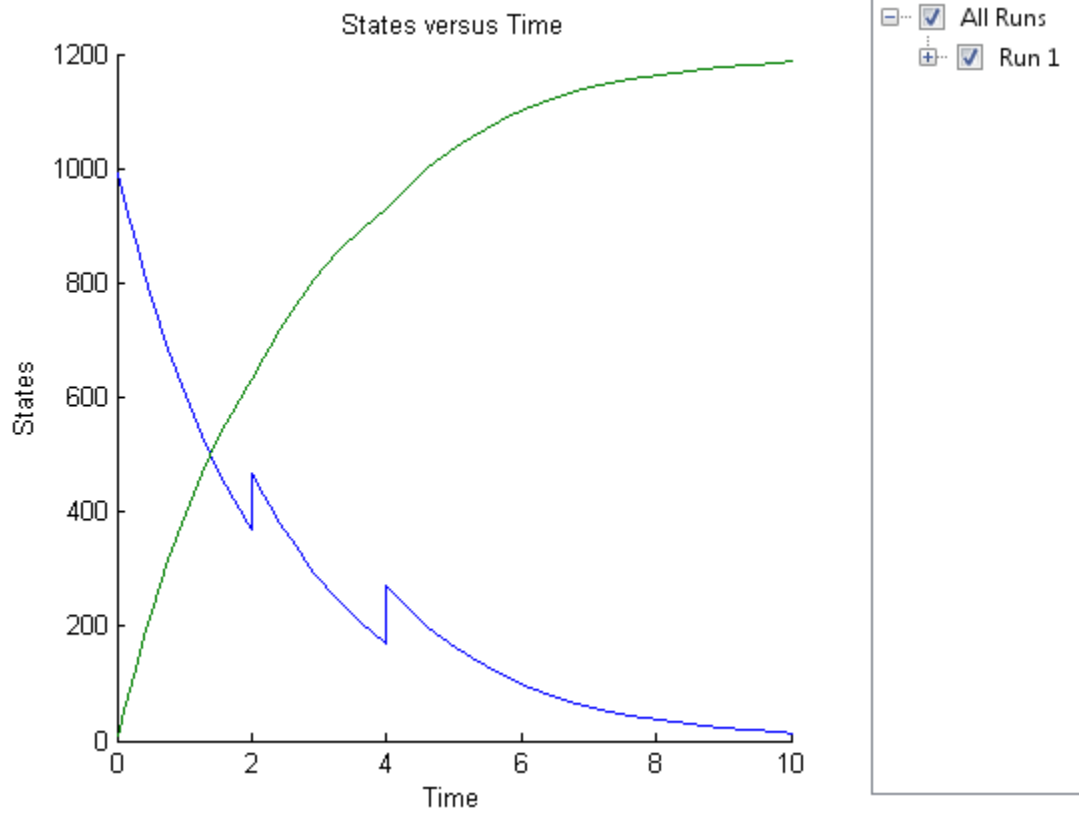
# sbiosimulate



```
sbioplot(sim3)
```



```
sbioplot(sim4)
```



## Simulate a SimBiology Model Using Configset and Dose Objects

Load a sample SimBiology model.

```
sbioimportproject radiodecay.sbproj
```

Get the default configuration set from the model.

```
defaultConfigSet = getconfigset(m1,'default');
```



Add a scheduled dose of 100 molecules at 2 seconds for species x.

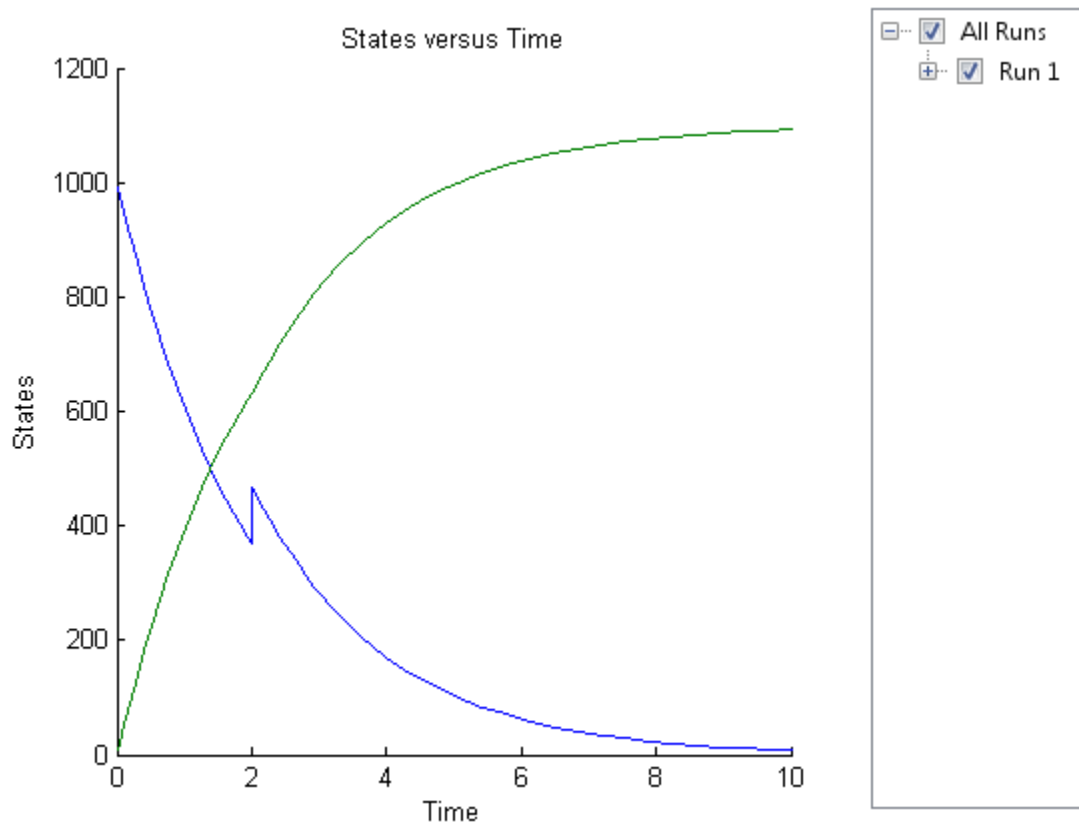
```
dObj = adddose(m1,'d1','schedule');  
dObj.Amount = 100;  
dObj.AmountUnits = 'molecule';  
dObj.TimeUnits = 'second';  
dObj.Time = 2;  
dObj.TargetName = 'unnamed.x';
```

Simulate the model using configset and dose objects.

```
sim = sbiosimulate(m1,defaultConfigSet,dObj);
```

Plot the result.

```
sbioplot(sim);
```



## Simulate a SimBiology Model Using Configset, Dose, and Variant Objects

Load a sample SimBiology model.

```
sbioloadproject radiodecay.sbproj
```

Add a new configuration set using a stop time of 15 seconds.

```
csObj = m1.addconfigset('newStopTimeConfigSet');  
csObj.StopTime = 15;
```

Add a scheduled dose of 100 molecules at 2 seconds for species x.

```
dObj = adddose(m1,'d1','schedule');  
dObj.Amount = 100;  
dObj.AmountUnits = 'molecule';  
dObj.TimeUnits = 'second';  
dObj.Time = 2;  
dObj.TargetName = 'unnamed.x';
```

Add a variant of species x using a different initial amount of 500 molecules.

```
vObj = addvariant(m1,'v1');  
addcontent(vObj,{'species','x','InitialAmount',500});
```

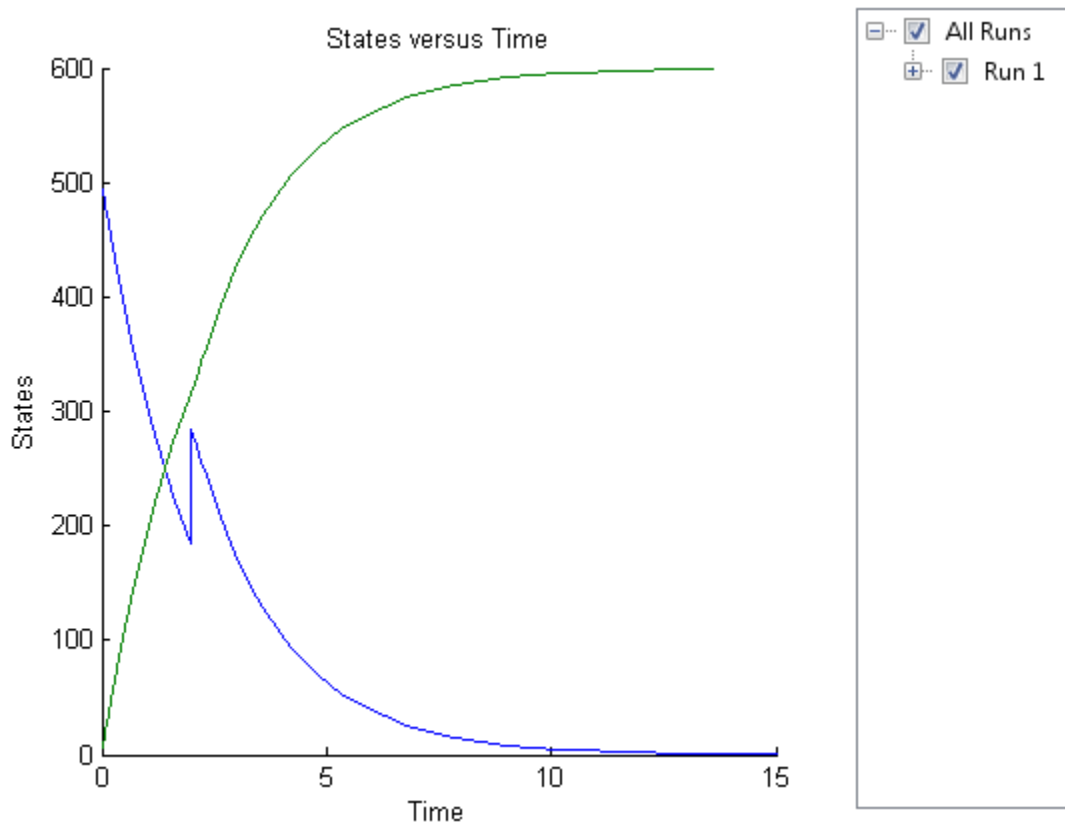
Simulate the model using the same configset, variant, and dose objects. Use the same order of input arguments as shown next.

```
sim = sbiosimulate(m1,csObj,vObj,dObj);
```

Plot the result.

```
sbioplot(sim);
```

# sbiosimulate



## See Also

[addconfigset](#) | [sbioaccelerate](#) | [sbiomodel](#) | [SimData](#) object | [Configset](#) object | [getconfigset](#) | [setactiveconfigset](#) | [Variant](#) object | [ScheduleDose](#) object | [RepeatDose](#) object | [Model](#) object

**Purpose**

Plot simulation results in subplots

**Syntax**

```
sbiosubplot(simDataObj)  
sbiosubplot(simDataObj, fcnHandleValue, xArgsValue, yArgsValue)  
sbiosubplot(simDataObj, fcnHandleValue, xArgsValue, yArgsValue,  
            showLegendValue)
```

**Arguments**

|                        |  |
|------------------------|--|
| <i>simDataObj</i>      | SimBiology data object.                  |
| <i>fcnHandleValue</i>  | Function handle.                         |
| <i>xArgsValue</i>      | Cell array with the names of the states. |
| <i>yArgsValue</i>      | Cell array with the names of the states. |
| <i>showLegendValue</i> | Boolean (default is false).              |

**Description**

`sbiosubplot(simDataObj)` plots each simulation run for SimBiology data object, *simDataObj* into its own subplot. The subplot is a time plot of each state in *simDataObj*. A legend is included.

`sbiosubplot(simDataObj, fcnHandleValue, xArgsValue, yArgsValue)` plots each simulation run for the SimBiology data object, *simDataObj*, into its own subplot. The subplot is plotted by calling the function handle, *fcnHandleValue*, with input arguments *simDataObj*, *xArgsValue*, and *yArgsValue*.

`sbiosubplot(simDataObj, fcnHandleValue, xArgsValue, yArgsValue, showLegendValue)` plots each simulation run for the SimBiology data object, *simDataObj*, into its own subplot. The subplot is plotted by calling the function handle, *fcnHandleValue*, with input arguments *simDataObj*, *xArgsValue*, and *yArgsValue*. *showLegendValue* indicates if a legend is shown in the plot. *showLegendValue* can be either true or false. By default, *showLegendValue* is false.

**Examples**

This example shows how to plot data from an ensemble run without interpolation.

# sbiosubplot

---

```
% Load the radiodecay model.
    sbioloadproject('radiodecay.sbproj','m1');

% Configure the model to run with the stochastic solver.
cs = getconfigset(m1, 'active');
set(cs, 'SolverType', 'ssa');
set(cs.SolverOptions, 'LogDecimation', 100);

% Run an ensemble simulation and view the results.
simDataObj = sbioensamplerun(m1, 10, 'linear');
sbiosubplot(simDataObj);
```

**See Also**      sbiplot

**Purpose**

Plot data or simulation results in trellis plot

**Syntax**

```
trellisplot = sbiotrellis(data,groupCol,xCol,yCol)
trellisplot =
sbiotrellis(data,groupCol,xCol,yCol,Name,Value)
trellisplot =
sbiotrellis(data,fcnHandle,groupCol,xCol,yCol)
trellisplot = sbiotrellis(simData,fcnHandle,xCol,yCol)
```

**Arguments**

|                          |  |
|--------------------------|--|
| <code>trellisplot</code> | Object returned by <code>sbiotrellis</code> . Use <code>trellisplot</code> together with the <code>plot</code> method to overlay trellis plots. See “Description” on page 1-218 for information about the <code>plot</code> method.  |
| <code>data</code>        | A dataset containing grouped data or a <code>groupedData</code> object.  |
| <code>groupCol</code>    | String specifying a column in <code>data</code> that contains grouping information or an empty string <code>''</code> which implies there is only one group in <code>data</code> .   |
| <code>xCol</code>        | String specifying a column in <code>data</code> to plot on the <i>x</i> -axis.<br><br>If <code>data</code> is <code>groupedData</code> , then <code>xCol</code> can also be an empty string <code>''</code> , and the <i>x</i> -coordinates of the data are determined by the variable specified in <code>DATA.Properties.IndependentVariableName</code> .<br><br>If <code>data</code> is <code>dataset</code> , then <code>xCol</code> cannot be empty. |
| <code>yCol</code>        | String or cell array of strings specifying column(s) in <code>data</code> to plot on the <i>y</i> -axis.   |

|                        |                  |
|------------------------|------------------|
| <code>fcnHandle</code> | Function handle. |
| <code>simData</code>   | SimData object.  |

## Description

`trellisplot = sbiotrellis(data,groupCol,xCol,yCol)` plots each group in `data` as defined by the group column variable `groupCol` into its own subplot. The data defined by column `xCol` is plotted against the data defined by column(s) `yCol`.

`trellisplot = sbiotrellis(data,groupCol,xCol,yCol,Name,Value)` uses additional options specified by one or more `Name,Value` pair arguments that are supported by the `plot` command.

`trellisplot = sbiotrellis(data,fcnHandle,groupCol,xCol,yCol)` plots each group in `data` as defined by the group column variable `groupCol` into its own subplot. `sbiotrellis` creates the subplot by calling the function handle, `fcnHandle`, with input arguments defined by the `data` columns `xCol` and `yCol`. The `fcnHandle` cannot be empty and must be specified.

`trellisplot = sbiotrellis(simData,fcnHandle,xCol,yCol)` plots each group in `simData` into its own subplot. `sbiotrellis` creates the subplot by calling the function handle, `fcnHandle` with input arguments defined by the columns `xCol` and `yCol`. The `fcnHandle` can be empty ( ' ' or [ ] ). If empty, the default time plot is created using the handle `@plot`.



---

**Tip** Use the `plot` method to overlay a `SimData` object or a dataset on an existing `sbiotrellis` plot. For example, `plot(trellisplot, ...)` adds a plot to the `sbiotrellis` object `trellisplot`. The `SimData` or dataset that is being plotted must have the same number of elements/groups as the `trellisplot` object. The `plot` method has the same input arguments as `sbiotrellis`. For an example, see “Perform Population Fitting”.

---

## Examples

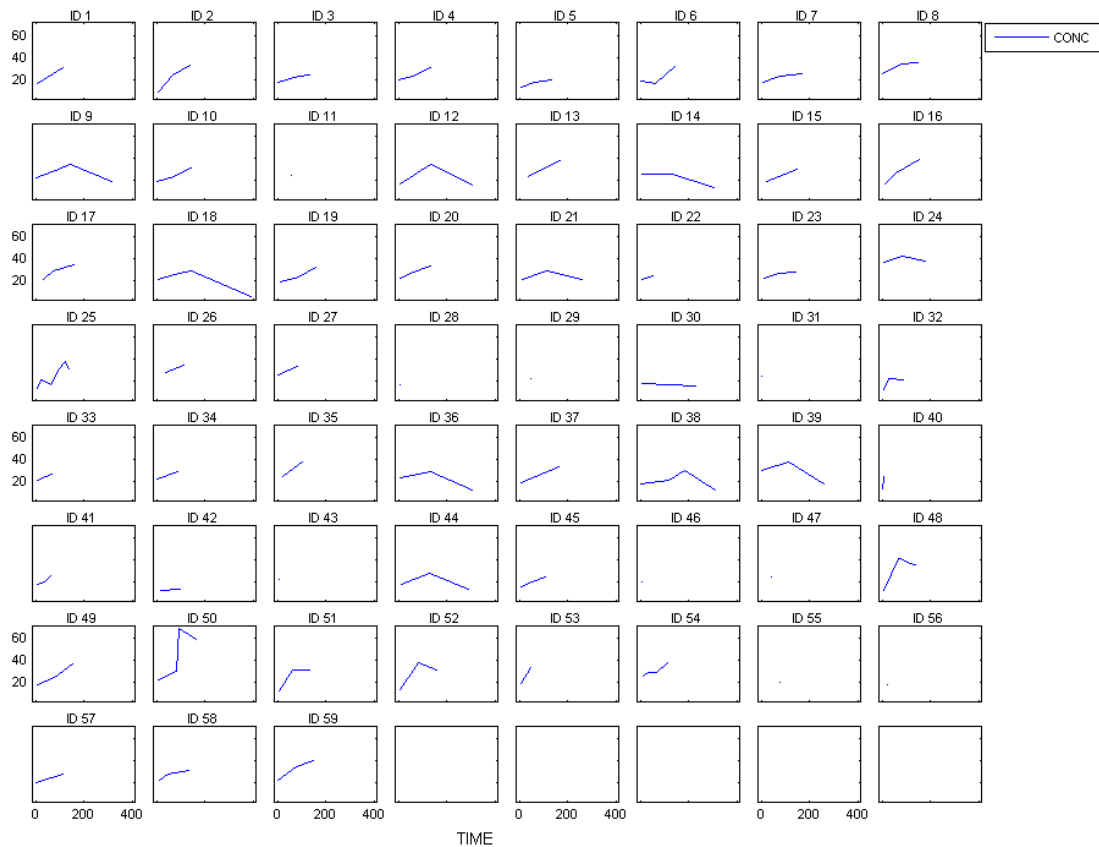
### Create a Trellis Plot for Dataset Containing Grouped Data

Load a sample dataset.

```
load pheno.mat ds
```

Plot a trellisplot using the dataset.

```
sbiotrellis(ds, 'ID', 'TIME', 'CONC');
```



## Plot Using a GroupedData Object

Load a sample dataset.

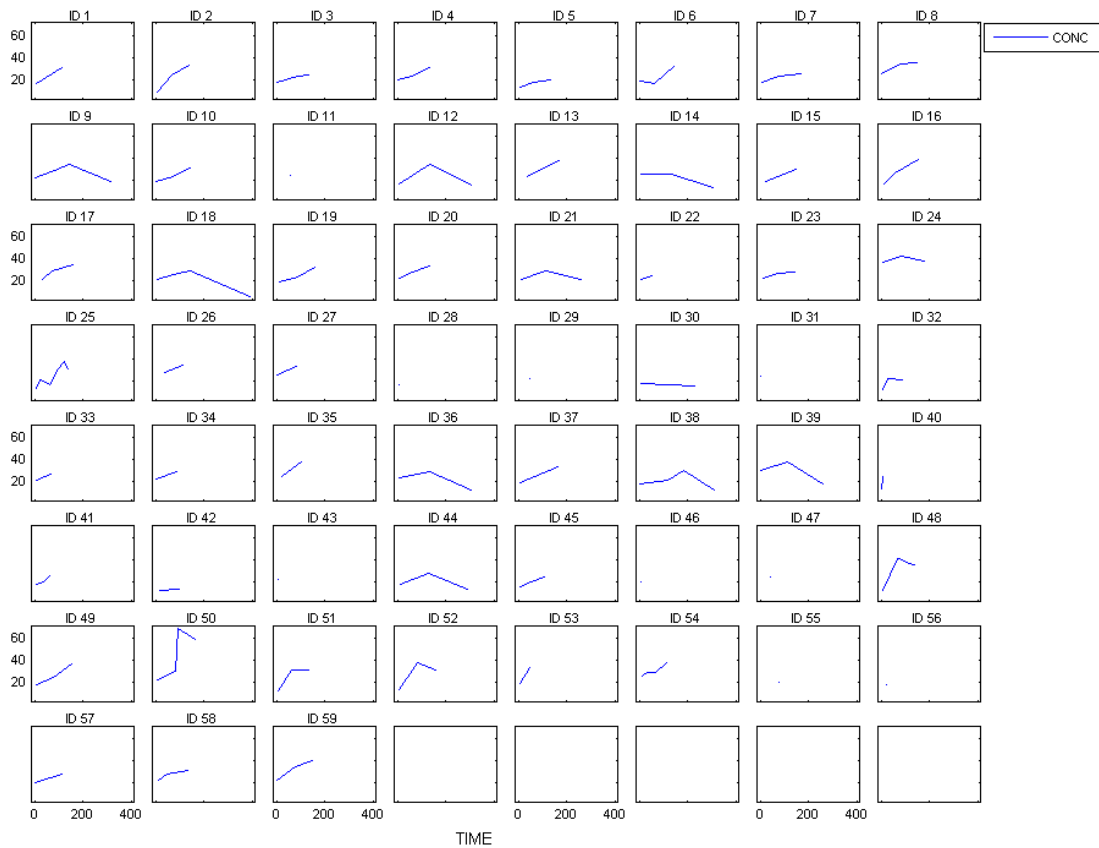
```
load pheno.mat ds
```

Convert the dataset to a groupedData object.

```
gData = groupedData(ds);
```

Plot a trellisplot using the groupedData.

```
sbiotrellis(gData, 'ID', 'TIME', 'CONC');
```



### Plot Simulation Data

Load a sample project.

```
sbioloadproject('radiodecay');
```

Set the solver to be a stochastic solver.

```
m1.getConfigset('active').SolverType = 'ssa';
```

Set the initial amount of species x to 100.

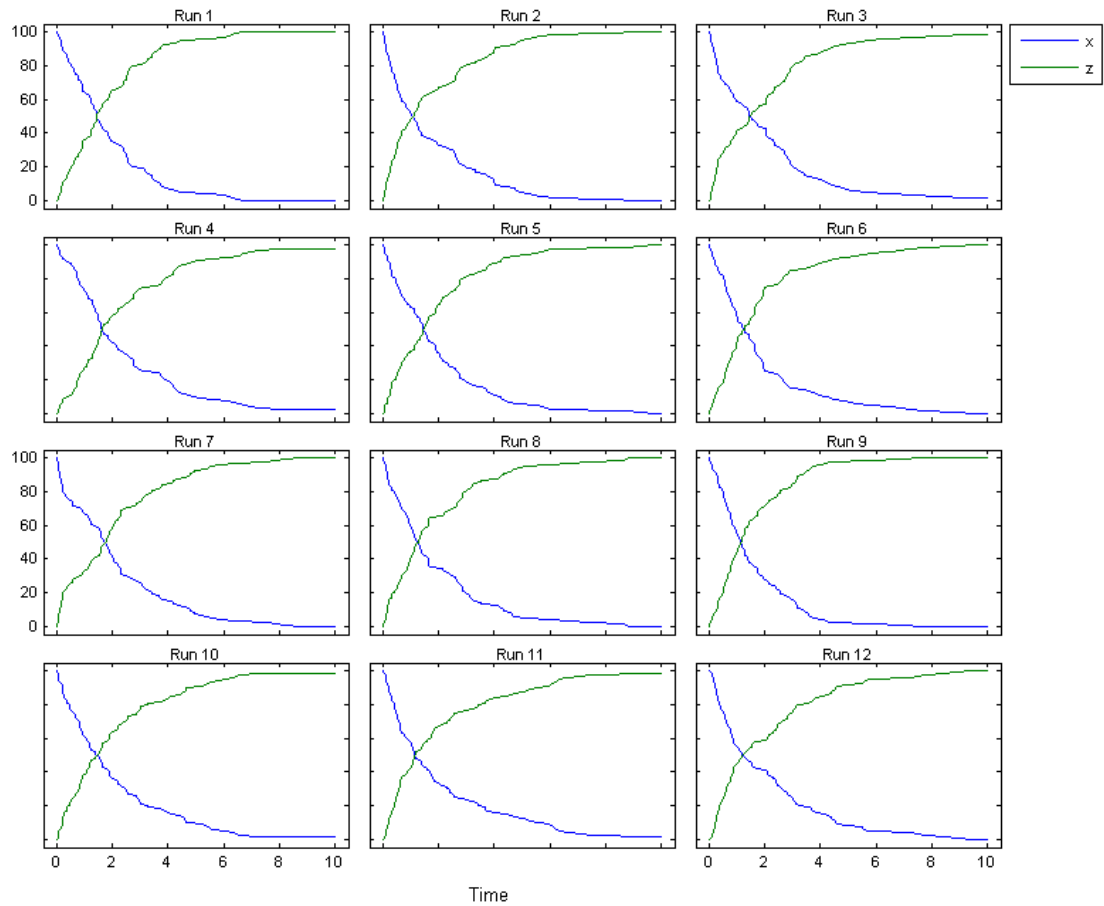
```
m1.Compartments(1).Species(1).InitialAmount = 100;
```

Run an ensemble simulation.

```
sd = sbioenssemblerun(m1, 12);
```

Plot the simulation data.

```
sbiotrellis(sd, '', 'Time', {'x', 'z'});
```



**See Also** [sbioplot](#) | [sbiosubplot](#)

**Purpose** Create user-defined unit

**Syntax**

```
unitObject = sbiunit('NameValue')
unitObject = sbiunit('NameValue', 'CompositionValue')
unitObject =
sbiunit('NameValue','CompositionValue',MultiplierValue)
unitObject =
sbiunit('NameValue','CompositionValue',MultiplierValue,
OffsetValue)
unitObject = sbiunit('NameValue','CompositionValue',
...'PropertyName', PropertyValue...)
```

## Arguments

|                         |  |
|-------------------------|--|
| <i>NameValue</i>        | Name of the user-defined unit. <i>NameValue</i> must begin with characters and can contain characters, underscores, or numbers. <i>NameValue</i> can be any valid MATLAB variable name.  |
| <i>CompositionValue</i> | Shows the combination of base and derived units that defines the unit <i>NameValue</i> . For example molarity is mole/liter. Base units are the set of units used to define all unit quantity equations. Derived units are defined using base units or mixtures of base and derived units. |
| <i>MultiplierValue</i>  | Numerical value that defines the relationship between the user-defined unit <i>NameValue</i> and the base unit as a product of the <i>MultiplierValue</i> and the base unit. For example, 1 mole is 6.0221e23*molecule. The <i>MultiplierValue</i> is 6.0221e23.                           |

|                      |   |
|----------------------|---|
| <i>OffsetValue</i>   | Numerical value by which the unit composition is modified. For example, <code>Celsius = (5/9)*( Fahrenheit-32)</code> ; <i>Fahrenheit</i> is <i>Composition</i> ; <i>MultiplierValue</i> is 5/9 and <i>OffsetValue</i> is 32. |
| <i>PropertyName</i>  | Name of the unit object property, for example, 'Notes'.   |
| <i>PropertyValue</i> | Value of the unit object property, for example, 'New unit for GPCR model'.  |

## Description

*unitObject* = `sbiounit('NameValue')` constructs a SimBiology unit object named *NameValue*. Valid names must begin with a letter, and be followed by letters, underscores, or numbers.

*unitObject* = `sbiounit('NameValue', 'CompositionValue')` allows you to specify the name and the composition of the unit.

*unitObject* = `sbiounit('NameValue', 'CompositionValue', 'MultiplierValue')` creates a unit with the name *NameValue* where the unit is defined as  $\text{MultiplierValue} \times \text{CompositionValue}$ .

*unitObject* = `sbiounit('NameValue', 'CompositionValue', 'MultiplierValue', 'OffsetValue')` creates a unit with the specified offset.

*unitObject* = `sbiounit('NameValue', 'CompositionValue', ... 'PropertyName', PropertyValue...)` defines optional properties. The property name/property value pairs can be in any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs).

In order to use *unitObject*, you must add it to the user-defined library with the `sbioaddtolibrary` function. To get the unit object into the user-defined library, use the following command:

```
sbioaddtolibrary(unitObject);
```

You can view additional *unitObject* properties with the `get` command. You can modify additional properties with the `set` command. For more information about unit object properties and methods, see Unit object.

Use the `sbiowhos` function to list the units available in the user-defined library.

## Examples

This example shows you how to create a user-defined unit, add it to the user-defined library, and query the library.

- 1 Create units for the rate constants of a first-order and a second-order reaction.

```
unitObj1 = sbiounit('firstconstant', '1/second', 1);  
unitObj2 = sbiounit('secondconstant', '1/molarity*second', 1);
```

- 2 Add the unit to the user-defined library.

```
sbioaddtolibrary(unitObj1);  
sbioaddtolibrary(unitObj2);
```

- 3 Query the user-defined library in the root object.

```
rootObj = sbioroot;
```

```
rootObj.UserDefinedLibrary.Units
```

```
SimBiology UserDefined Units
```

| Index: | Name:          | Composition:      | Multiplier: | Offset:  |
|--------|----------------|-------------------|-------------|----------|
| 1      | firstconstant  | 1/second          | 1.000000    | 0.000000 |
| 2      | secondconstant | 1/molarity*second | 1.000000    | 0.000000 |

Alternatively, use the `sbiowhos` command.

```
sbiowhos -userdefined -unit
```



SimBiology UserDefined Units

| Index: | Name:          | Composition:      | Multiplier: | Offset:  |
|--------|----------------|-------------------|-------------|----------|
| 1      | firstconstant  | 1/second          | 1.000000    | 0.000000 |
| 2      | secondconstant | 1/molarity*second | 1.000000    | 0.000000 |

**See Also**

[sbioaddtolibrary](#) | [sbiowhos](#) | [sbiunitprefix](#) | [sbiunit](#) | [sbiunit](#)

# sbiunitcalculator

---

|                    |  |
|--------------------|--|
| <b>Purpose</b>     | Convert value between units  |
| <b>Syntax</b>      | <code>result = sbiunitcalculator('fromUnits', 'toUnits', Value)</code>   |
| <b>Description</b> | <code>result = sbiunitcalculator('fromUnits', 'toUnits', Value)</code><br>converts the value, <i>Value</i> , which is defined in the units, <i>fromUnits</i> , to the value, <i>result</i> , which is defined in the units, <i>toUnits</i> . |
| <b>Examples</b>    | <code>result = sbiunitcalculator('mile/hour', 'meter/second', 1)</code>  |
| <b>See Also</b>    | <code>sbioshowunits</code>   |

**Purpose** Create user-defined unit prefix

**Syntax**

```
unitprefixObject = sbiunitprefix('NameValue')
unitprefixObject = sbiunitprefix('NameValue', 'ExponentValue')
unitprefixObject = sbiunitprefix('NameValue', ...'PropertyName',
    PropertyValue ...)
```

## Arguments

|                      |  |
|----------------------|--|
| <i>NameValue</i>     | Name of the user-defined unit prefix. <i>NameValue</i> must begin with characters and can contain characters, underscores, or numbers. <i>NameValue</i> can be any valid MATLAB variable name. |
| <i>ExponentValue</i> | Shows the value of $10^{\text{Exponent}}$ that defines the relationship of the unit <i>Name</i> to the base unit. For example, for the unit picomole, Exponent is 12.                          |
| <i>PropertyName</i>  | Name of the unit prefix object property. For example, 'Notes'.   |
| <i>PropertyValue</i> | Value of the unit prefix object property. For example, 'New unitprefix for GPCR model'.  |

## Description

*unitprefixObject* = `sbiunitprefix('NameValue')` constructs a SimBiology unit prefix object with the name *NameValue*. Valid names must begin with a letter, and be followed by letters, underscores, or numbers.

*unitprefixObject* = `sbiunitprefix('NameValue', 'ExponentValue')` creates a unit-prefix object with a multiplicative factor of  $10^{\text{ExponentValue}}$ .

*unitprefixObject* = `sbiunitprefix('NameValue', ...'PropertyName', PropertyValue ...)` defines optional properties. The property name/property value pairs can be in any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs).

# sbiounitprefix

---

In order to use *unitprefixObject*, you must add it to the user-defined library with the `sbioaddtolibrary` function. To get the unit-prefix object into the user-defined library, use the following command:

```
sbioaddtolibrary(unitprefixObject);
```

You can view additional *unitprefixObject* properties with the `get` command. You can modify additional properties with the `set` command.

Use the `sbiowhosunitprefixes` function to list the units available in the user-defined library.

## Examples

This example shows how to create a user-defined unit prefix, add it to the user-defined library, and query the library.

- 1 Create a unit prefix.

```
unitprefixObj1 = sbiounitprefix('peta', 15);
```

- 2 Add the unit prefix to the user-defined library.

```
sbioaddtolibrary(unitprefixObj1);
```

- 3 Query the user-defined library in the root object.

```
rootObj = sbioroot;
```

```
rootObj.UserDefinedLibrary.UnitPrefixes
```

```
Unit Prefix Array
```

| Index: | Library:    | Name: | Exponent: |
|--------|-------------|-------|-----------|
| 1      | UserDefined | peta  | 15        |

Alternatively, use the `sbiowhos` command.

```
sbiowhos -userdefined -unitprefix
```

## SimBiology UserDefined Unit Prefixes

| Index: | Name: | Multiplier:   |
|--------|-------|---------------|
| 1      | peta  | 1.000000e+015 |

### See Also

[sbioaddtolibrary](#) | [sbioshowunits](#) | [sbiounit](#) | [sbiowhos](#)

### How To

- [sbioshowunits](#)

**Purpose** Construct variant object

**Syntax**

```
variantObj = sbiovariant('NameValue')
variantObj = sbiovariant('NameValue', 'ContentValue')
variantObj = sbiovariant(...'PropertyName', PropertyValue...)
```

## Arguments

|                   |  |
|-------------------|--|
| <i>modelObj</i>   | Specify the model object to which you want add a variant.  |
| <i>variantObj</i> | Variant object to create and add to the model object.  |
| <i>NameValue</i>  | Name of the variant object. <i>NameValue</i> is assigned to the Name property of the variant object. |

## Description

*variantObj* = sbiovariant('NameValue') creates a SimBiology variant object (*variantObj*) with the name *NameValue*. The variant object Parent property is assigned [ ] (empty).

*variantObj* = sbiovariant('NameValue', 'ContentValue') creates a SimBiology variant object (*variantObj*) with the Content property set to *ContentValue*.

To add a variant to a model use the copyobj method. A SimBiology variant object stores alternate values for properties on a SimBiology model. For more information on variants, see Variant object.

*variantObj* = sbiovariant(...'PropertyName', PropertyValue...) defines optional properties. The property name/property value pairs can be in any format supported by the function set (for example, name-value string pairs, structures, and name-value cell array pairs).

View properties for a variant object with the get command, and modify properties for a variant object with the set command.

---

**Note** Remember to use the `addcontent` method instead of using the `set` method on the `Content` property because the `set` method replaces the data in the `Content` property, whereas `addcontent` appends the data.

---

**Method  
Summary**

|                                      |   |
|--------------------------------------|---|
| <code>addcontent</code> (variant)    | Append content to variant object        |
| <code>commit</code> (variant)        | Commit variant contents to model        |
| <code>copyobj</code> (any object)    | Copy SimBiology object and its children |
| <code>delete</code> (any object)     | Delete SimBiology object                |
| <code>display</code> (any object)    | Display summary of SimBiology object    |
| <code>get</code> (any object)        | Get object properties                   |
| <code>rmcontent</code> (variant)     | Remove contents from variant object     |
| <code>set</code> (any object)        | Set object properties                   |
| <code>verify</code> (model, variant) | Validate and verify SimBiology model    |

**Property  
Summary**

|                      |  |
|----------------------|--|
| <code>Active</code>  | Indicate object in use during simulation |
| <code>Content</code> | Contents of variant object               |
| <code>Name</code>    | Specify name of object                   |
| <code>Notes</code>   | HTML text describing SimBiology object   |
| <code>Parent</code>  | Indicate parent object                   |

|          |                                       |
|----------|---------------------------------------|
| Tag      | Specify label for SimBiology object   |
| Type     | Display SimBiology object type        |
| UserData | Specify data to associate with object |

## Examples

- 1 Create a variant object.

```
variantObj = sbiovariant('p1');
```

- 2 Add content to the variant object that varies the InitialAmount property of a species named A.

```
addcontent(variantObj, {'species', 'A', 'InitialAmount', 5});
```

## See Also

[addvariant](#) | [copyobj](#) | [getvariant](#)



**Purpose** Show contents of project file, library file, or SimBiology root object

**Syntax**

```
sbiowhos flag
sbiowhos ('flag')
sbiowhos flag1 flag2...
sbiowhos FileName
```

**Description** sbiowhos shows contents of the SimBiology root object. This includes the built-in and user-defined kinetic laws, units, and unit prefixes.

sbiowhos flag shows specific information about the SimBiology root object as defined by flag. Valid flags are described in this table.

| Flag         | Description   |
|--------------|---|
| -builtin     | Built-in kinetic laws, units, and unit prefixes     |
| -data        | Data saved in file                                  |
| -kineticlaw  | Built-in and user-defined kinetic laws              |
| -unit        | Built-in and user-defined units                     |
| -unitprefix  | Built-in and user-defined unit prefixes             |
| -userdefined | User-defined kinetic laws, units, and unit prefixes |

You can also specify the functional form `sbiowhos ('flag')`.

`sbiowhos flag1 flag2...` shows information about the SimBiology root object as defined by flag1, flag2, ... .

`sbiowhos FileName` shows the contents of the SimBiology project or library defined by Name.

**Examples**

```
% Show contents of the SimBiology root object
sbiowhos
```

# sbiowhos

---

```
% Show kinetic laws on the SimBiology root object
sbiowhos -kineticlaw

% Show the builtin units of the SimBiology root object.
sbiowhos -builtin -unit

% Show all contents of project file.
sbiowhos myprojectfile

% Show kinetic laws from a library file.
sbiowhos -kineticlaw mylibraryfile

% Show all contents of multiple files.
sbiowhos myfile1 myfile2
```

## See Also

whos

**Purpose** Export SimBiology model to SBML file

**Syntax**  
`sbmlexport(modelObj)`  
`sbmlexport(modelObj, 'FileName')`

## Arguments

*modelObj* Model object. Enter a variable name for a model object.

*FileName* XML file with a Systems Biology Markup Language (SBML) format. Enter either a file name or a path and file name supported by your operating system. If the file name does not have the extension `.xml`, then `.xml` is appended to end of the file name.

## Description

`sbmlexport(modelObj)` exports a SimBiology model object (`modelObj`) to a file with a Systems Biology Markup Language (SBML) Level 2 Version 4 format. The default file extension is `.xml` and the file name matches the model name.

`sbmlexport(modelObj, 'FileName')` exports a SimBiology model object (`modelObj`) to an SBML file named *FileName*. The default file extension is `.xml`.

A SimBiology model can also be written to a SimBiology project with the `sbiosaveproject` function to save features not supported by SBML.

For more information about features that are supported by SimBiology but not by SBML or vice visa, see “SBML Support”.

## Examples

Export a model (`modelObj`) to a file (`gene_regulation.xml`) in the current working directory.

```
sbmlexport(modelObj, 'gene_regulation.xml');
```

## References

Finney, A., Hucka, M., (2003), *Systems Biology Markup Language (SBML) Level 2: Structures and facilities for model definitions*. Accessed from SBML.org

# sbmlexport

---

## See Also

[sbiomodel](#) | [sbiosaveproject](#) | [sbmlimport](#)

- [“SBML Support”](#)
- [sbiomodel](#)
- [sbiosaveproject](#)

|                        |  |
|------------------------|--|
| <b>Purpose</b>         | Import SBML-formatted file   |
| <b>Syntax</b>          | <code>modelObj = sbmlimport(<i>File</i>)</code>  |
| <b>Description</b>     | <p><code>modelObj = sbmlimport(<i>File</i>)</code> imports <i>File</i>, a Systems Biology Markup Language (SBML)-formatted file, into MATLAB and creates a model object <code>modelObj</code>.</p> <p><i>File</i> is a string specifying a file name or a path and file name supported by your operating system. <i>File</i> extensions are <code>.sbml</code> or <code>.xml</code>. <i>File</i> can also be a URL, if you have the Java® programming language.</p> <p><code>sbmlimport</code> supports SBML Level 2 Version 4 and earlier.</p> <p>For functional characteristics and limitations, see “SBML Support”.</p> |
| <b>Input Arguments</b> | <p><b>File</b></p> <p>String specifying either of the following:</p> <ul style="list-style-type: none"><li>• File name or path and file name supported by your operating system</li><li>• URL (if you have Java programming language)</li></ul>  |
| <b>Examples</b>        | <p>Import SBML model:</p> <pre>sbmlObj = sbmlimport('oscillator.xml');</pre>   |
| <b>References</b>      | <p>Finney, A., Hucka, M., (2003). <i>Systems Biology Markup Language (SBML) Level 2: Structures and facilities for model definitions</i>. SBML.org.</p>  |
| <b>Alternatives</b>    | <p>Use the SimBiology desktop to import an SBML-formatted file. For more information, see “Importing and Exporting Models, Tasks, and Data from the Desktop” and “Importing from SBML Files”.</p>  |
| <b>See Also</b>        | <code>get</code>   <code>sbiosimulate</code>   <code>sbmlexport</code>   <code>set</code>  |
| <b>How To</b>          | <ul style="list-style-type: none"><li>• “SBML Support”</li></ul>   |

# sbmlimport

---

- “Importing and Exporting Models, Tasks, and Data from the Desktop”
- “Importing from SBML Files”

**Purpose** Open SimBiology desktop for modeling and simulation

**Syntax**

```
simbiology
simbiology(modelObj)
simbiology(File)
```

**Input Arguments**

|                 |  |
|-----------------|--|
| <i>modelObj</i> | SimBiology model object or an array of model objects.  |
| <i>File</i>     | String specifying a file name or path and file name of an sbproj file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder. |

**Description**

`simbiology` opens the SimBiology desktop, which lets you:

- Build a SimBiology model by representing reaction pathways and entering kinetic data for the reactions.
- Import or export SimBiology models to and from the MATLAB workspace or from a Systems Biology Markup Language (SBML) file.
- Modify an existing SimBiology model.
- Simulate a SimBiology model using individual or ensemble runs.
- View results from the simulation.
- Perform analysis tasks such as sensitivity analysis, parameter and species scans, and calculation of conserved moieties.
- Import and plot data for analysis tasks.
- Create and/or modify user-defined units and unit prefixes.
- Create and/or modify user-defined kinetic laws.

`simbiology(modelObj)` opens the SimBiology desktop with *modelObj*, a SimBiology model object. If a project is open in the desktop, the `simbiology` function adds *modelObj* to the project.

`simbiology(File)` opens the project specified by *File* in the SimBiology desktop. *File* is a string specifying a file name or path and file name of an sbproj file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder. If a project is open in the desktop, the software replaces it with the new project, after prompting you to save any changes.

The `Parent` property of a SimBiology model object is set to the SimBiology root object. The root object contains a list of model objects that are accessible from the MATLAB command line and from the SimBiology desktop. Because both the command line and the desktop point to the same model object in the `Root` object, changes you make to the model at the command line are reflected in the desktop, and vice versa.

---

**Note** The `sbioreset` function removes all models from the root object. Therefore, the `sbioreset` function removes all models from the SimBiology desktop.

---

## Examples

Create a SimBiology model in the MATLAB workspace, and then open the SimBiology desktop with the model:

```
modelObj = sbiomodel('cell');  
simbiology(modelObj)
```

## See Also

`sbiroot` | `sbiofittool`



**Superclasses** matlab.mixin.Heterogeneous

**Purpose** Exported SimBiology model dose object

**Description** SimBiology.export.Dose is the superclass for modifiable export dose objects. An export dose is either of subclass SimBiology.export.RepeatDose or SimBiology.export.ScheduleDose.

**Construction** Export dose objects are created by the export method for SimBiology models. By default, all active doses are export doses, but you can specify which doses to export using the optional editdoses input argument to export.

|                |   |
|----------------|---|
| export (model) | Export SimBiology models for deployment and standalone applications |
|----------------|---|

## Properties

### Amount

Amount of dose, a nonnegative scalar value.

### AmountUnits

Dose amount units. This property is read only.

### DurationParameterName

Parameter specifying length of time to administer a dose. This property is read only.

### LagParameterName

Parameter specifying time lag for the dose. This property is read only.

### Name

Name of dose object. This property is read only.

### Notes

# SimBiology.export.Dose

---

Text describing dose object. This property is read only.

## **Parent**

Name of the parent export model. This property is read only.

## **Rate**

Rate of dose, a nonnegative scalar value.

## **RateUnits**

Units for dose rate. This property is read only.

## **TargetName**

Species receiving dose. This property is read only.

## **TimeUnits**

Time units for dosing. This property is read only.

## **Copy Semantics**

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB documentation.

## **Examples**

### **Exported SimBiology Model Dose Objects**

Open a sample SimBiology model project, and export the included model object.

```
sbioloadproject('AntibacterialPKPD')  
em = export(m1);
```

Get the editable doses from the exported model object.

```
doses = getdose(em)
```

```
doses =
```

```
1x4 RepeatDose array with properties:
```

```
Interval
```

```
RepeatCount
StartTime
TimeUnits
Amount
AmountUnits
DurationParameterName
LagParameterName
Name
Notes
Parent
Rate
RateUnits
TargetName
```

The exported model has 4 repeated dose objects.

Display the 3rd dose object from the exported model object.

```
doses(3)
```

```
ans =
```

```
RepeatDose with properties:
```

```
Interval: 12
RepeatCount: 27
StartTime: 0
TimeUnits: 'hour'
Amount: 500
AmountUnits: 'milligram'
DurationParameterName: 'TDose'
LagParameterName: ''
Name: '500 mg bid'
Notes: ''
Parent: 'Antibacterial'
Rate: 0
RateUnits: ''
TargetName: 'Central.Drug'
```

# SimBiology.export.Dose

---

Change the dosing amount for this dose object.

```
doses(3).Amount = 600;
```

```
doses(3)
```

```
ans =
```

```
RepeatDose with properties:
```

```
        Interval: 12
    RepeatCount: 27
      StartTime: 0
    TimeUnits: 'hour'
        Amount: 600
    AmountUnits: 'milligram'
DurationParameterName: 'TDose'
  LagParameterName: ''
          Name: '500 mg bid'
         Notes: ''
        Parent: 'Antibacterial'
          Rate: 0
    RateUnits: ''
    TargetName: 'Central.Drug'
```

## See Also

[SimBiology.export.RepeatDose](#) | [SimBiology.export.ScheduleDose](#)  
| [export](#)

## Related Examples

- “PK/PD Modeling and Simulation to Guide Dosing Strategy for Antibiotics”
- “Deploy a SimBiology Model”

# SimBiology.export.ExplicitTauSimulationOptions

---

|                     |   |                |   |
|---------------------|---|----------------|---|
| <b>Superclasses</b> | SimBiology.export.StochasticSimulationOptions   |                |   |
| <b>Purpose</b>      | Settings for explicit tau-leaping solver of exported SimBiology model   |                |   |
| <b>Description</b>  | SimBiology.export.ExplicitTauSimulationOptions is the class of simulation options associated with the explicit tau-leaping solver of an export model.   |                |   |
| <b>Construction</b> | <p>Explicit tau simulation options are created by the export method for SimBiology models with a stochastic SolverType set to 'expltau'.</p> <table><tr><td>export (model)</td><td>Export SimBiology models for deployment and standalone applications</td></tr></table>  | export (model) | Export SimBiology models for deployment and standalone applications |
| export (model)      | Export SimBiology models for deployment and standalone applications   |                |   |
| <b>Properties</b>   | <p><b>ErrorTolerance</b><br/>Error tolerance, a scalar value in the range (0,1).</p> <p><b>LogDecimation</b><br/>Frequency to log stochastic simulation output, a positive integer value.</p> <p><b>MaximumNumberofLogs</b><br/>Maximum number of logs criterion to stop simulation, a positive scalar value.</p> <p><b>MaximumWallClock</b><br/>Maximum elapsed wall clock time criterion to stop simulation, a positive scalar value.</p> <p><b>RandomState</b><br/>Random number generator, a positive integer value.</p> <p><b>SolverType</b></p> |                |   |

# SimBiology.export.ExplicitTauSimulationOptions

---

String indicating solver type to use for simulation, 'expltau'.  
This property is read only.

## **StopTime**

Simulation time criterion to stop simulation, a nonnegative scalar value.

## **TimeUnits**

Time units for simulation. This property is read only.

## **Copy Semantics**

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB documentation.

## **See Also**

`SimBiology.export.StochasticSimulationOptions` |  
`SimBiology.export.ImplicitTauSimulationOptions`  
| `SimBiology.export.SimulationOptions` |  
`SimBiology.export.ODESimulationOptions` | `export`

# SimBiology.export.ImplicitTauSimulationOptions

---

|                     |  |
|---------------------|--|
| <b>Superclasses</b> | SimBiology.export.StochasticSimulationOptions  |
| <b>Purpose</b>      | Settings for implicit tau-leaping stochastic simulation of exported SimBiology model   |
| <b>Description</b>  | SimBiology.export.ImplicitTauSimulationOptions is the class of simulation options associated with the implicit tau-leaping solver of an export model.  |
| <b>Construction</b> | Implicit tau-leaping simulation options are created by the export method for SimBiology models with a stochastic SolverType set to 'impltau'.<br><br>export (model)                      Export SimBiology models for deployment and standalone applications   |
| <b>Properties</b>   | <b>ErrorTolerance</b><br>Error tolerance, a scalar value in the range (0,1).<br><b>LogDecimation</b><br>Frequency to log stochastic simulation output, a positive integer value.<br><b>MaxIterations</b><br>Nonlinear solver maximum number of iterations, a positive integer value.<br><b>MaximumNumberofLogs</b><br>Maximum number of logs criterion to stop simulation, a positive scalar value.<br><b>MaximumWallClock</b><br>Maximum elapsed wall clock time criterion to stop simulation, a positive scalar value. |

# SimBiology.export.ImplicitTauSimulationOptions

---

## RandomState

Random number generator, a positive integer value.

## SolverType

String indicating solver type to use for simulation, 'impltau'.  
This property is read only.

## StopTime

Simulation time criterion to stop simulation, a nonnegative scalar value.

## TimeUnits

Time units for simulation. This property is read only.

## Copy Semantics

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB documentation.

## See Also

`SimBiology.export.StochasticSimulationOptions` |  
`SimBiology.export.ExplicitTauSimulationOptions`  
| `SimBiology.export.SimulationOptions` |  
`SimBiology.export.ODESimulationOptions` | `export`



**Purpose** Exported SimBiology model object

**Description** Exported SimBiology models are limited-access models that can be simulated and accelerated. You can speedup simulation of exported models using Parallel Computing Toolbox, and deploy exported models using MATLAB Compiler™.

By default, all active dose objects, species, parameters, and compartments export with a SimBiology model, and are editable in the exported model object. You can limit which doses, species, parameters, and compartments are editable using additional options during export. Reactions, rules, and events are never editable in an exported model.

**Construction** Use the export method to export a SimBiology model.

|                |   |
|----------------|---|
| export (model) | Export SimBiology models for deployment and standalone applications |
|----------------|---|

## Properties

### DependentFiles

Function files the model depends on. This property is read only.

### ExportNotes

Text with additional information associated with the exported model. This property is read only.

### ExportTime

Creation time of the exported model. This property is read only.

### InitialValues

Vector of initial values for modifiable species, compartments, and parameters.

### Name

Name of the exported model. This property is read only.

# SimBiology.export.Model

---

## Notes

HTML text describing the exported model object. This property is read only.

## SimulationOptions

SimBiology.export.SimulationOptions object specifying simulation options.

## ValueInfo

Array of SimBiology.export.ValueInfo objects of modifiable species, parameters, and compartments.

## Methods

|               |   |
|---------------|---|
| accelerate    | Prepare exported SimBiology model for acceleration            |
| getdose       | Return exported SimBiology model dose object                  |
| getIndex      | Get indices into ValueInfo and InitialValues properties       |
| isAccelerated | Determine whether an exported SimBiology model is accelerated |
| simulate      | Simulate exported SimBiology model                            |

## Copy Semantics

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB documentation.

## Examples

### Export SimBiology Model Object

Load a sample SimBiology model object, and export.

```
modelObj = sbmlimport('lotka');  
em = export(modelObj)
```

```
em =
```

```
Model with properties:
```

```
    Name: 'lotka'  
ExportTime: '12-Dec-2012 15:20:13'  
ExportNotes: ''
```

Display the editable values (compartments, species, and parameters) information.

```
em.ValueInfo
```

```
ans =
```

```
8x1 ValueInfo array with properties:
```

```
Constant  
InitialValue  
Name  
Parent  
QualifiedName  
Tag  
Type  
Units
```

There are 8 editable values. Display the names of the editable values.

```
{em.ValueInfo.Name}
```

```
ans =
```

```
'unnamed' 'x' 'y1' 'y2' 'z' 'c1' 'c2' 'c3'
```

Display the exported model simulation options.

```
em.SimulationOptions
```

# SimBiology.export.Model

---

```
ans =  
  
ODESimulationOptions with properties:  
  
    AbsoluteTolerance: 1.0000e-06  
    AbsoluteToleranceScaling: 1  
    AbsoluteToleranceStepSize: []  
        MaxStep: []  
        OutputTimes: []  
    RelativeTolerance: 1.0000e-03  
        SolverType: 'ode15s'  
    MaximumNumberOfLogs: Inf  
    MaximumWallClock: Inf  
        StopTime: 10  
        TimeUnits: 'second'
```

The exported model has a deterministic solver.

## See Also

[SimBiology.export.Dose](#) | [SimBiology.export.SimulationOptions](#)  
| [SimBiology.export.ValueInfo](#) | [export](#)

## Related Examples

- “PK/PD Modeling and Simulation to Guide Dosing Strategy for Antibiotics”
- “Deploy a SimBiology Model”

## Concepts

- Class Attributes
- Property Attributes

|                             |   |                             |   |
|-----------------------------|---|-----------------------------|---|
| <b>Superclasses</b>         | SimBiology.export.SimulationOptions   |                             |   |
| <b>Purpose</b>              | Settings for deterministic, ordinary differential equation simulation of exported SimBiology model  |                             |   |
| <b>Description</b>          | SimBiology.export.ODESimulationOptions is the class of simulation options associated with determinist, ordinary differential equation (ODE) solvers.  |                             |   |
| <b>Construction</b>         | Deterministic simulation options are created by the <code>export</code> method for SimBiology models with a deterministic SolverType (for example, <code>sundials</code> or <code>ode15s</code> ).<br><br><table><tr><td><code>export (model)</code></td><td>Export SimBiology models for deployment and standalone applications</td></tr></table>  | <code>export (model)</code> | Export SimBiology models for deployment and standalone applications |
| <code>export (model)</code> | Export SimBiology models for deployment and standalone applications   |                             |   |
| <b>Properties</b>           | <p><b>AbsoluteTolerance</b><br/>Absolute error tolerance applied to state value during simulation, a positive scalar value.</p> <p><b>AbsoluteToleranceScaling</b><br/>Control scaling of absolute error tolerance, a logical value.</p> <p><b>AbsoluteToleranceStepSize</b><br/>Initial guess for time step size for scaling of absolute error tolerance, [ ] or a scalar value.</p> <p><b>MaximumNumberofLogs</b><br/>Maximum number of logs criterion to stop simulation, a positive scalar value.</p> <p><b>MaximumWallClock</b><br/>Maximum elapsed wall clock time criterion to stop simulation, a positive scalar value.</p> |                             |   |

# SimBiology.export.ODESimulationOptions

---

## **MaxStep**

Upper bound on ODE solver step size, [] or a positive scalar value.

## **OutputTimes**

Times to log in simulation output, a vector of sorted nonnegative values.

## **RelativeTolerance**

Allowable error tolerance relative to state value during simulation, a scalar value in the range (0,1).

## **SolverType**

String indicating solver type to use for simulation. Possible deterministic solver types are:

- 'sundials'
- 'ode15s'
- 'ode23t'
- 'ode45'

## **StopTime**

Simulation time criterion to stop simulation, a nonnegative scalar value.

## **TimeUnits**

Time units for simulation. This property is read only.

## **Copy Semantics**

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB documentation.

## **See Also**

SimBiology.export.SimulationOptions |  
SimBiology.export.StochasticSimulationOptions | export

|                             |   |                             |   |
|-----------------------------|---|-----------------------------|---|
| <b>Superclasses</b>         | SimBiology.export.Dose  |                             |   |
| <b>Purpose</b>              | Repeated doses for exported SimBiology model  |                             |   |
| <b>Description</b>          | SimBiology.export.RepeatDose is the class for export repeat doses.  |                             |   |
| <b>Construction</b>         | <p>Export repeat dose objects are created by the <code>export</code> method for SimBiology models. By default, all active repeat doses are export repeat doses, but you can specify which repeat doses to export using the optional <code>editdoses</code> input argument to <code>export</code>.</p> <table><tr><td><code>export (model)</code></td><td>Export SimBiology models for deployment and standalone applications</td></tr></table>  | <code>export (model)</code> | Export SimBiology models for deployment and standalone applications |
| <code>export (model)</code> | Export SimBiology models for deployment and standalone applications   |                             |   |
| <b>Properties</b>           | <p><b>Amount</b><br/>Amount of dose, a nonnegative scalar value.</p> <p><b>AmountUnits</b><br/>Dose amount units. This property is read only.</p> <p><b>DurationParameterName</b><br/>Parameter specifying length of time to administer a dose. This property is read only.</p> <p><b>Interval</b><br/>Time between doses, a nonnegative scalar value.</p> <p><b>LagParameterName</b><br/>Parameter specifying time lag for the dose. This property is read only.</p> <p><b>Name</b><br/>Name of dose object. This property is read only.</p> <p><b>Notes</b></p> |                             |   |

# SimBiology.export.RepeatDose

---

Text describing dose object. This property is read only.

## **Parent**

Name of the parent export model. This property is read only.

## **Rate**

Rate of dose, a nonnegative scalar value.

## **RateUnits**

Units for dose rate. This property is read only.

## **RepeatCount**

Dose repetitions, a nonnegative integer value.

## **StartTime**

Start time for initial dose, a nonnegative scalar value.

## **TargetName**

Species receiving dose. This property is read only.

## **TimeUnits**

Time units for dosing. This property is read only.



---

**Note** You cannot change the `Rate` property of `RepeatDose` for exported SimBiology model if *all* of the following conditions are true:

- The `UnitConversion` property of the model is already set to `true`.
- The `Rate` property is empty or set to zero.
- The `RateUnits` is set to empty.

To change the `Rate`, do one of the following:

- Set the `UnitConversion` property of the original model to `false`. Then export the model again.
  - Set the `RateUnits` appropriately.
- 

## Copy Semantics

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB documentation.

## See Also

`SimBiology.export.Dose` | `SimBiology.export.ScheduleDose` | `export`

# SimBiology.export.ScheduleDose

---

**Superclasses** SimBiology.export.Dose

**Purpose** Schedule dose for exported SimBiology model

**Description** SimBiology.export.ScheduleDose is the class for export schedule doses.

**Construction** Export schedule dose objects are created by the `export` method for SimBiology models. By default, all active schedule doses are export schedule doses, but you can specify which schedule doses to export using the optional `editdoses` input argument to `export`.

|                             |   |
|-----------------------------|---|
| <code>export (model)</code> | Export SimBiology models for deployment and standalone applications |
|-----------------------------|---|

## Properties

### Amount

Amount of dose, a nonnegative scalar value.

### AmountUnits

Dose amount units. This property is read only.

### DurationParameterName

Parameter specifying length of time to administer a dose. This property is read only.

### LagParameterName

Parameter specifying time lag for the dose. This property is read only.

### Name

Name of dose object. This property is read only.

### Notes

Text describing dose object. This property is read only.

**Parent**

Name of the parent export model. This property is read only.

**Rate**

Rate of dose, a nonnegative scalar value.

**RateUnits**

Units for dose rate. This property is read only.

**TargetName**

Species receiving dose. This property is read only.

**Time**

Schedule dose times, a vector of nonnegative values.

**TimeUnits**

Time units for dosing. This property is read only.

---

**Note** You cannot change the `Rate` property of `ScheduleDose` for exported SimBiology model if *all* of the following conditions are true:

- The `UnitConversion` property of the model is already set to true.
- The `Rate` property is empty or set to zero.
- The `RateUnits` is set to empty.

To change the `Rate`, do one of the following:

- Set the `UnitConversion` property of the original model to false. Then export the model again.
  - Set the `RateUnits` appropriately.
-

# SimBiology.export.ScheduleDose

---

## **Copy Semantics**

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB documentation.

## **See Also**

`SimBiology.export.Dose` | `SimBiology.export.RepeatDose` | `export`

|                       |  |
|-----------------------|--|
| <b>Purpose</b>        | Simulation settings for exported SimBiology model  |
| <b>Description</b>    | <code>SimBiology.export.SimulationOptions</code> is the superclass of simulation options for exported models. Simulation options are either of subclass <code>SimBiology.export.ODESimulationOptions</code> for deterministic solvers, or <code>SimBiology.export.StochasticSimulationOptions</code> for stochastic solvers.   |
| <b>Construction</b>   | Simulation options are created by the <code>export</code> method for SimBiology models.<br><br><code>export (model)</code> Export SimBiology models for deployment and standalone applications   |
| <b>Properties</b>     | <b>MaximumNumberofLogs</b><br>Maximum number of logs criterion to stop simulation, a positive scalar value.<br><b>MaximumWallClock</b><br>Maximum elapsed wall clock time criterion to stop simulation, a positive scalar value.<br><b>StopTime</b><br>Simulation time criterion to stop simulation, a nonnegative scalar value.<br><b>TimeUnits</b><br>Time units for simulation. This property is read only. |
| <b>Copy Semantics</b> | <code>Handle</code> . To learn how handle classes affect copy operations, see <a href="#">Copying Objects in the MATLAB documentation</a> .  |
| <b>See Also</b>       | <code>SimBiology.export.ODESimulationOptions</code>   <code>SimBiology.export.StochasticSimulationOptions</code>   <code>export</code>   |

# SimBiology.export.StochasticSimulationOptions

---

|                     |  |                |   |
|---------------------|--|----------------|---|
| <b>Superclasses</b> | SimBiology.export.SimulationOptions  |                |   |
| <b>Purpose</b>      | Settings for stochastic simulation of exported SimBiology model  |                |   |
| <b>Description</b>  | SimBiology.export.StochasticSimulationOptions is the superclass of simulation options associated with stochastic solvers. The subclasses of SimBiology.export.StochasticSimulationOptions are SimBiology.export.ExplicitTauSimulationOptions and SimBiology.export.ImplicitTauSimulationOptions.   |                |   |
| <b>Construction</b> | <p>Stochastic simulation options are created by the export method for SimBiology models with a stochastic SolverType (ssa, expltau, or impltau).</p> <table><tr><td>export (model)</td><td>Export SimBiology models for deployment and standalone applications</td></tr></table>   | export (model) | Export SimBiology models for deployment and standalone applications |
| export (model)      | Export SimBiology models for deployment and standalone applications  |                |   |
| <b>Properties</b>   | <p><b>LogDecimation</b><br/>Frequency to log stochastic simulation output, a positive integer value.</p> <p><b>MaximumNumberofLogs</b><br/>Maximum number of logs criterion to stop simulation, a positive scalar value.</p> <p><b>MaximumWallClock</b><br/>Maximum elapsed wall clock time criterion to stop simulation, a positive scalar value.</p> <p><b>RandomState</b><br/>Random number generator, a positive integer value.</p> <p><b>SolverType</b></p> |                |   |

String indicating solver type to use for simulation. This property is read only. The stochastic solver type is one of:

- 'ssa'
- 'exptau'
- 'impltau'

## **StopTime**

Simulation time criterion to stop simulation, a nonnegative scalar value.

## **TimeUnits**

Time units for simulation. This property is read only.

## **Copy Semantics**

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB documentation.

## **See Also**

`SimBiology.export.SimulationOptions` |  
`SimBiology.export.ODESimulationOptions` |  
`SimBiology.export.ExplicitTauSimulationOptions` |  
`SimBiology.export.ImplicitTauSimulationOptions` | `export`

# SimBiology.export.ValueInfo

---

**Purpose** Modifiable species, compartments, or parameters in exported SimBiology model

**Description** SimBiology.export.ValueInfo is the class that describes the modifiable value components in a SimBiology.export.Model, including species, parameters, and compartments.

**Construction** ValueInfo objects are created by the export method for SimBiology models. By default, all model species, parameters, and compartments are ValueInfo objects, but you can specify which value components to export using the optional editvals input argument to export.

|                |   |
|----------------|---|
| export (model) | Export SimBiology models for deployment and standalone applications |
|----------------|---|

## Properties

### Constant

Display whether value is constant or time-varying. This property is read only.

### InitialValue

Initial value for the component, a scalar value.

### Name

Name of species, compartment, or parameter. This property is read only.

### Parent

Name of parent model, compartment, or reaction. This property is read only.

### QualifiedName

Qualified name of species, compartment, or parameter. This property is read only.



- For compartments and model-scoped parameters, the qualified name is the same as the name.
- For species, the qualified name is `CompartmentName.SpeciesName`.
- For reaction-scoped parameters, the qualified name is `ReactionName.ParameterName`.

**Tag**

Label for species, compartment, or parameter. This property is read only.

**Type**

Type of value (species, parameter, or compartment). This property is read only.

**Units**

Value units. This property is read only

**Copy Semantics**

Handle. To learn how handle classes affect copy operations, see Copying Objects in the MATLAB documentation.

**See Also**

`SimBiology.export.Model` | `export`



# Methods — Alphabetical List

---

The object that the methods apply to are listed in parenthesis after the method name.

# accelerate(SimFunction)

---

|                        |  |
|------------------------|--|
| <b>Purpose</b>         | Prepare SimFunction object for accelerated simulations                                 |
| <b>Syntax</b>          | accelerate(F)  |
| <b>Input Arguments</b> | F<br>SimFunction object created by the createSimFunction method of a SimBiology model. |
| <b>Description</b>     | accelerate(F) prepares SimFunction object F for accelerated simulations.               |

---

**Note** F is automatically accelerated at the first function execution. However, manually accelerate the object if you want it accelerated in your deployment applications.

---

## Examples

### Simulate SimFunction Object

This example uses the the Lotka-Volterra (predator-prey) model described by Gillespie [1].

Load the sample project containing the lotka model.

```
sbioloadproject lotka;
```

Create a SimFunction object f with c1 and c2 as input parameters to be scanned, and y1 and y2 as the output of the function with no dose.

```
f = createSimFunction(m1,{'Reaction1.c1', 'Reaction2.c2'},{'y1', 'y2'}, [
```

```
f =
```

```
SimFunction
```

```
Parameters:
```

| Name           | Value | Type        |
|----------------|-------|-------------|
| 'Reaction1.c1' | 10    | 'parameter' |
| 'Reaction2.c2' | 0.01  | 'parameter' |

Observables:

| Name | Type      |
|------|-----------|
| 'y1' | 'species' |
| 'y2' | 'species' |

Dosed: None

The `SimFunction` object `f` is not set for acceleration at the time of creation. But it will be automatically accelerated when executed.

```
f.isAccelerated
```

```
ans =
```

```
0
```

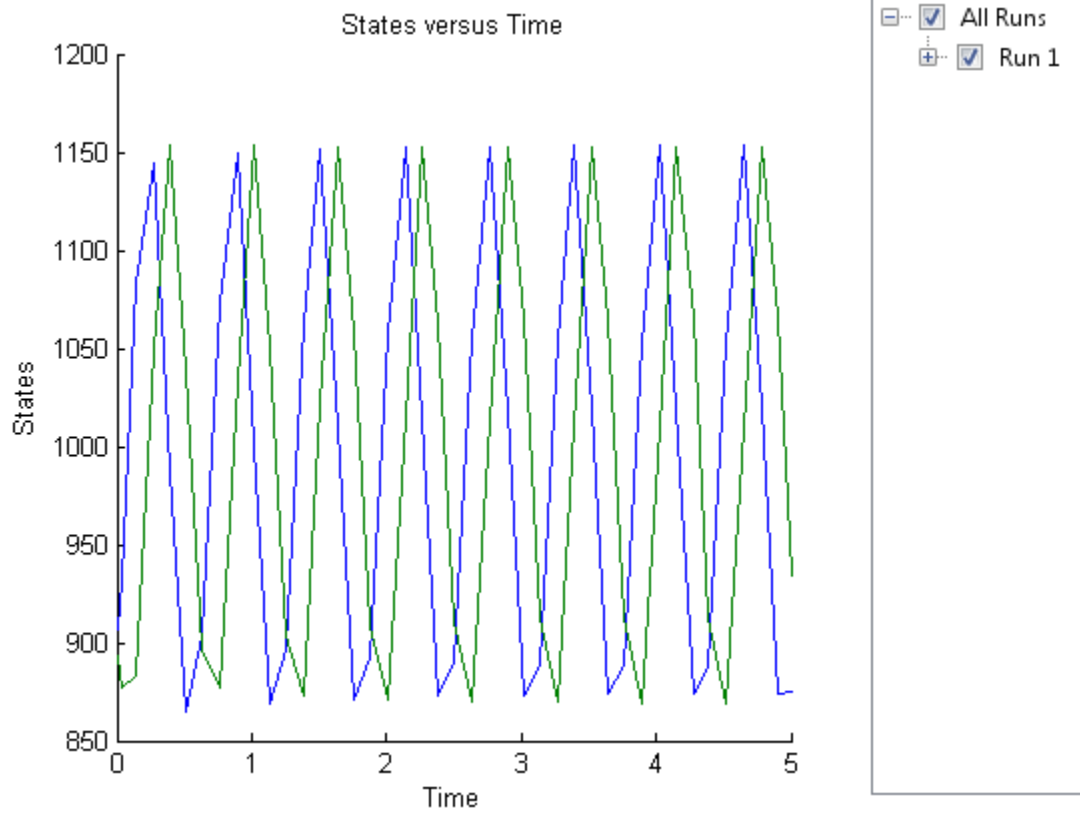
Define an input matrix that contains parameter values for `c1` and `c2`.

```
phi = [10 0.01];
```

Run simulations until the stop time is 5 and plot the simulation results.

```
sbioplot(f(phi,5))
```

# accelerate(SimFunction)



Confirm the SimFunction object `f` was accelerated during execution.

```
f.isAccelerated
```

```
ans =
```

```
1
```

## See Also

`createSimFunction`, `SimFunction` object

## References

- [1] Gillespie D.T. "Exact Stochastic Simulation of Coupled Chemical Reactions," (1977) The Journal of Physical Chemistry, 81(25), 2340-2361.

# SimBiology.export.Model.accelerate

---

**Purpose** Prepare exported SimBiology model for acceleration

**Syntax** `accelerate(model)`

**Description** `accelerate(model)` prepares the exported model for acceleration on the current type of computer.

---

**Note** Microsoft Visual Studio 2010 run-time libraries must be available on any computer running accelerated models generated using Microsoft Windows SDK. If you plan to redistribute your accelerated models to other MATLAB users, be sure they have the run-time libraries.

---

## Input Arguments

### **model**

SimBiology.export.Model object.

## Examples

### **Accelerate Exported SimBiology Model**

Load a sample SimBiology model object, and export.

```
modelObj = sbmlimport('lotka');  
em = export(modelObj)
```

```
em =
```

```
Model with properties:
```

```
    Name: 'lotka'  
ExportTime: '12-Dec-2012 15:20:13'  
ExportNotes: ''
```

Accelerate the exported model.

```
accelerate(em);  
em.isAccelerated
```



```
ans =
```

```
1
```

The logical value 1 indicates that the exported model is accelerated.

## See Also

[SimBiology.export.Model](#) |  
[SimBiology.export.Model.isAccelerated](#) | [export](#)

## Related Examples

- “PK/PD Modeling and Simulation to Guide Dosing Strategy for Antibiotics”
- “Deploy a SimBiology Model”

# AbstractKineticLaw object

---

**Purpose** Kinetic law information in library

**Description** The abstract kinetic law object represents a *kinetic law definition*, which provides a mechanism for applying a rate law to multiple reactions. The information in this object acts as a mapping template for the reaction rate. The kinetic law definition specifies a mathematical relationship that defines the rate at which reactant species are produced and product species are consumed in the reaction. The expression is shown in the `Expression` property. The species variables are defined in the `SpeciesVariables` property, and the parameter variables are defined in the `ParameterVariables` property of the abstract kinetic law object. For an explanation of how the kinetic law definition relates to the kinetic law object, see `KineticLaw` object.

Create your own kinetic law definition and add it to the kinetic law library with the `sbioaddtolibrary` function. You can then use the kinetic law to define a reaction rate. To retrieve a kinetic law definition from the user-defined library, first create a root object using `sbiroot`, then use the command `get(rootObj.UserDefinedLibrary, 'KineticLaws')`.

See “Property Summary” on page 2-9 for links to abstract kinetic law object property reference pages.

Properties define the characteristics of an object. For example, an abstract kinetic law object includes properties for the expression, the name of the law, parameter variables, and species variables. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the SimBiology desktop.

**Constructor Summary** `sbioabstractkineticlaw` Create kinetic law definition

## Method Summary

|                      |                                      |
|----------------------|--------------------------------------|
| delete (any object)  | Delete SimBiology object             |
| display (any object) | Display summary of SimBiology object |
| get (any object)     | Get object properties                |
| set (any object)     | Set object properties                |

## Property Summary

|   |  |
|---|--|
| Expression<br>(AbstractKineticLaw,<br>KineticLaw) | Expression to determine reaction rate equation |
| Name  | Specify name of object                         |
| Notes   | HTML text describing SimBiology object         |
| ParameterVariables                                | Parameters in kinetic law definition           |
| Parent  | Indicate parent object                         |
| SpeciesVariables                                  | Species in abstract kinetic law                |
| Tag   | Specify label for SimBiology object            |
| Type  | Display SimBiology object type                 |
| UserData  | Specify data to associate with object          |

## See Also

Configset object, KineticLaw object, Model object, Parameter object, Reaction object, Root object, Rule object, Species object

# addcompartment (model, compartment)

---

**Purpose** Create compartment object

**Syntax**

```
compartmentObj = addcompartment(modelObj, 'NameValue')
compartmentObj = addcompartment(owningCompObj, 'NameValue')
compartmentObj = addcompartment(modelObj, 'NameValue',
CapacityValue)
compartmentObj = addcompartment(...'PropertyName',
PropertyValue...)
```

## Arguments

|                      |   |
|----------------------|---|
| <i>modelObj</i>      | Model object.   |
| <i>owningCompObj</i> | Compartment object that contains the newly created compartment object.  |
| <i>NameValue</i>     | Name for a compartment object. Enter a character string unique to the model object. For information on naming compartments, see Name. |
| <i>CapacityValue</i> | Capacity value for the compartment object. Enter double. Positive real number, default = 1.   |
| <i>PropertyName</i>  | Enter the name of a valid property. Valid property names are listed in “Property Summary” on page 2-12.                               |
| <i>PropertyValue</i> | Enter the value for the property specified in <i>PropertyName</i> . Valid property values are listed on each property reference page. |

**Description** *compartmentObj* = addcompartment(*modelObj*, 'NameValue') creates a compartment object and returns the compartment object (*compartmentObj*). In the compartment object, this method assigns a value (*NameValue*) to the property Name, and assigns the model object (*modelObj*) to the property Parent. In the model object, this method assigns the compartment object to the property Compartments.

## addcompartment (model, compartment)

---

`compartmentObj = addcompartment(owningCompObj, 'NameValue')` in addition to the above, adds the newly created compartment within a compartment object (`owningCompObj`), and assigns this compartment object (`owningCompObj`) to the Owner property of the newly created compartment object (`compartmentObj`). The parent model is the model that contains the owning compartment (`owningCompObj`).

`compartmentObj = addcompartment(modelObj, 'NameValue', CapacityValue)`, in addition to the above, this method assigns capacity (`CapacityValue`) for the compartment.

If you define a reaction within a model object (`modelObj`) that does not contain any compartments, the process of adding a reaction generates a default compartment object and assigns the reaction species to the compartment. If there is more than one compartment, you must specify which compartment the species should be assigned to using the format `CompartmentName.SpeciesName`.

View properties for a compartment object with the `get` command, and modify properties for a compartment object with the `set` command. You can view a summary table of compartment objects in a model (`modelObj`) with `get(modelObj, 'Compartments')` or the properties of the first compartment with `get(modelObj.Compartments(1))`.

`compartmentObj = addcompartment(...'PropertyName', PropertyValue...)` defines optional properties. The property name/property value pairs can be in any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs). “Property Summary” on page 2-12 lists the properties. The Owner property is one exception; you cannot set the Owner property in the `addcompartment` syntax because, `addcompartment` requires the owning model or compartment to be specified as the first argument and uses this information to set the Owner property. After adding a compartment, you can change the owner using the function `set`.

# addcompartment (model, compartment)

---

## Method Summary

|  |   |
|--|---|
| Methods for compartment objects                    |   |
| addcompartment (model, compartment)                | Create compartment object   |
| addspecies (model, compartment)                    | Create species object and add to compartment object within model object |
| copyobj (any object)                               | Copy SimBiology object and its children                                 |
| delete (any object)                                | Delete SimBiology object  |
| display (any object)                               | Display summary of SimBiology object                                    |
| get (any object)                                   | Get object properties   |
| rename (compartment, parameter, species, reaction) | Rename object and update expressions                                    |
| reorder (model, compartment, kinetic law)          | Reorder component lists   |
| set (any object)                                   | Set object properties   |

## Property Summary

|                                    |   |
|------------------------------------|---|
| Properties for compartment objects |   |
| Capacity                           | Compartment capacity                              |
| CapacityUnits                      | Compartment capacity units                        |
| Compartments                       | Array of compartments in model or compartment     |
| ConstantCapacity                   | Specify variable or constant compartment capacity |
| Name                               | Specify name of object                            |
| Notes                              | HTML text describing SimBiology object            |

# addcompartment (model, compartment)

---

|          |  |
|----------|--|
| Owner    | Owning compartment                     |
| Parent   | Indicate parent object                 |
| Species  | Array of species in compartment object |
| Tag      | Specify label for SimBiology object    |
| Type     | Display SimBiology object type         |
| UserData | Specify data to associate with object  |

## Examples

- 1 Create a model object (modelObj).

```
modelObj = sbiomodel('cell');
```

- 2 Add two compartments to the model object.

```
compartmentObj1 = addcompartment(modelObj, 'nucleus');  
compartmentObj2 = addcompartment(modelObj, 'mitochondrion');
```

- 3 Add a compartment to one of the compartment objects.

```
compartmentObj3 = addcompartment(compartmentObj2, 'matrix');
```

- 4 Display the Compartments property in the model object.

```
get(modelObj, 'Compartments')
```

```
SimBiology Compartment Array
```

| Index: | Name:         | Capacity: | CapacityUnits: |
|--------|---------------|-----------|----------------|
| 1      | nucleus       | 1         |                |
| 2      | mitochondrion | 1         |                |
| 3      | matrix        | 1         |                |

- 5 Display the Compartments property in the compartment object.

## addcompartment (model, compartment)

---

```
get(compartmentObj2, 'Compartments')
```

```
SimBiology Compartment - matrix
```

```
Compartment Components:
```

```
Capacity:          1
```

```
CapacityUnits:
```

```
Compartments:     0
```

```
ConstantCapacity: true
```

```
Owner:            mitochondrion
```

```
Species:          0
```

### **See Also**

addproduct, addreactant, addreaction, addspecies, get, set



# addCompartment (PKModelDesign)

---

**Purpose** Add compartment to PKModelDesign object

**Syntax**

```
PKCompartmentObj = addCompartment(PKModelDesignObj,  
CompObjName)  
PKCompartmentObj = addCompartment(PKModelDesignObj,  
CompObjName,  
Name, Value)
```

**Description**

*PKCompartmentObj = addCompartment(PKModelDesignObj, CompObjName)* constructs a PK compartment with the specified name and adds it to *PKModelDesignObj*, a PKModelDesign object.

*PKCompartmentObj = addCompartment(PKModelDesignObj, CompObjName, Name, Value)* constructs a PK compartment with the specified name, and with additional options specified by one or more Name, Value pair arguments.

**Input Arguments**

|                         |   |
|-------------------------|---|
| <i>PKModelDesignObj</i> | PKModelDesign object to which you want to add a compartment |
| <i>CompObjName</i>      | Name of the PKCompartment object that is constructed        |

## Name-Value Pair Arguments

Optional comma-separated pairs of *Name*, *Value* arguments, where *Name* is the argument name and *Value* is the corresponding value. *Name* must appear inside single quotes (' '). You can specify several name-value pair arguments in any order as Name1,Value1, ,NameN,ValueN.

## addCompartment (PKModelDesign)

---

|                     |   |
|---------------------|---|
| DosingType          | <p>String specifying the mechanism for drug absorption. Choices are:</p> <ul style="list-style-type: none"><li>• 'Bolus'</li><li>• 'Infusion'</li><li>• 'ZeroOrder'</li><li>• 'FirstOrder'</li><li>• '' (default)</li></ul>   |
| EliminationType     | <p>For more information, see “Dosing Types”.</p> <p>String specifying the mechanism for drug elimination. Choices are:</p> <ul style="list-style-type: none"><li>• 'Linear'</li><li>• 'Linear-Clearance'</li><li>• 'Enzymatic'</li><li>• '' (default)</li></ul> <p>For more information, see “Elimination Types”.</p> |
| HasResponseVariable | <p>Logical indicating if the drug concentration in this compartment is reported. Multiple compartments in a model can have this property set to true. Default is false.</p>   |

# addCompartment (PKModelDesign)

---

---

**Note** If you perform a parameter fit on a model, at least one compartment in the model must have a `HasResponseVariable` property set to `true`.

---

`HasLag` Logical indicating if any dose targeting this compartment have a lag associated with them. Default is `false`.

These optional name-value pair arguments set the corresponding property of the `PKCompartment` object. You can also set these properties after creating the `PKCompartment` object by using the following syntax:

```
PKCompartmentObj.PropertyName = Value
```

For example:

```
PKCompartmentObj.DosingType = 'Bolus'
```

## Output Arguments

*PKCompartmentObj* PKCompartment object

## Method Summary

|                               |                       |
|-------------------------------|-----------------------|
| <code>get (any object)</code> | Get object properties |
| <code>set (any object)</code> | Set object properties |

## Property Summary

|                              |  |
|------------------------------|--|
| <code>DosingType</code>      | Drug dosing type in compartment                |
| <code>EliminationType</code> | Drug elimination type from compartment         |
| <code>HasLag</code>          | Lag associated with dose targeting compartment |

## addCompartment (PKModelDesign)

---

|                     |   |
|---------------------|---|
| HasResponseVariable | Compartment drug concentration reported |
| Name                | Specify name of object                  |

### See Also

“Create a Pharmacokinetic Model Using the Command Line”, HasLag, HasResponseVariable, PKCompartment object, PKModelDesign object

**Purpose** Create configuration set object and add to model object

**Syntax**

```
configsetObj = addconfigset(modelObj, 'NameValue')  
configsetObj = addconfigset(..., 'PropertyName',  
PropertyValue, ...)
```

## Arguments

*modelObj* Model object. Enter a variable name.

*NameValue* Descriptive name for a configuration set object. Reserved words 'active' and 'default' are not allowed.

*configsetObj* Configuration set object.

## Description

*configsetObj* = addconfigset(*modelObj*, 'NameValue') creates a configuration set object and returns to *configsetObj*.

In the configuration set object, this method assigns a value (*NameValue*) to the property Name.

*configsetObj* = addconfigset(..., 'PropertyName', *PropertyValue*, ...) constructs a configuration set object, *configsetObj*, and configures *configsetObj* with property value pairs. The property name/property value pairs can be in any format supported by the function set (for example, name-value string pairs, structures, and name-value cell array pairs). The *configsetObj* properties are listed in “Property Summary” on page 2-20.

A configuration set stores simulation specific information. A model object can contain multiple configuration sets, with one being active at any given time. The active configuration set contains the settings that are used during a simulation. *configsetObj* is not automatically set to active. Use the function setactiveconfigset to define the active configset for modelObj.

Use the method copyobj to copy a configset object and add it to the modelObj.

# addconfigset (model)

---

You can additionally view configuration set object properties with the command `get`. You can modify additional configuration set object properties with the command `set`.

## Method Summary

Methods for configuration set objects

|                                   |   |
|-----------------------------------|---|
| <code>copyobj</code> (any object) | Copy SimBiology object and its children |
| <code>delete</code> (any object)  | Delete SimBiology object                |
| <code>display</code> (any object) | Display summary of SimBiology object    |
| <code>set</code> (any object)     | Set object properties                   |

## Property Summary

Properties for configuration set objects

|   |  |
|---|--|
| <code>Active</code>                     | Indicate object in use during simulation           |
| <code>CompileOptions</code>             | Dimensional analysis and unit conversion options   |
| <code>MaximumNumberOfLogs</code>        | Maximum number of logs criteria to stop simulation |
| <code>MaximumWallClock</code>           | Maximum elapsed wall clock time to stop simulation |
| <code>Name</code>                       | Specify name of object                             |
| <code>Notes</code>                      | HTML text describing SimBiology object             |
| <code>RuntimeOptions</code>             | Options for logged species                         |
| <code>SensitivityAnalysisOptions</code> | Specify sensitivity analysis options               |
| <code>SolverOptions</code>              | Specify model solver options                       |

|            |   |
|------------|---|
| SolverType | Select solver type for simulation           |
| StartTime  | Start time for initial dose time            |
| StopTime   | Simulation time criteria to stop simulation |
| TimeUnits  | Show time units for dosing and simulation   |
| Type       | Display SimBiology object type              |

## Examples

- 1 Create a model object by importing the `oscillator.xml` file, and add a `Configset` object to the model.

```
modelObj = sbmlimport('oscillator');  
configsetObj = addconfigset(modelObj, 'myset');
```

- 2 Configure the simulation stop criteria by setting the `StopTime`, `MaximumNumberOfLogs`, and `MaximumWallClock` properties of the `Configset` object. Set the stop criteria to a simulation time of 3000 seconds, 50 logs, or a wall clock time of 10 seconds, whichever comes first.

```
set(configsetObj, 'StopTime', 3000, 'MaximumNumberOfLogs', 50,...  
    'MaximumWallClock', 10)  
get(configsetObj)
```

```
Active: 0  
CompileOptions: [1x1 SimBiology.CompileOptions]  
Name: 'myset'  
Notes: ''  
RuntimeOptions: [1x1 SimBiology.RuntimeOptions]  
SensitivityAnalysisOptions: [1x1 SimBiology.SensitivityAnalysisOptions]  
SolverOptions: [1x1 SimBiology.ODESolverOptions]  
SolverType: 'ode15s'  
StopTime: 3000  
MaximumNumberOfLogs: 50  
MaximumWallClock: 10
```

## addconfigset (model)

---

```
TimeUnits: 'second'  
Type: 'configset'
```

- 3 Set the new Configset object to be active, simulate the model using the new Configset object, and plot the result.

```
setactiveconfigset(modelObj, configsetObj);  
[t,x] = sbiosimulate(modelObj);  
plot (t,x)
```

### See Also

get, getconfigset, removeconfigset, set, setactiveconfigset



**Purpose** Append content to variant object

**Syntax**  
addcontent(*variantObj*, *contents*)  
addcontent(*variantObj1*, *variantObj2*)

**Arguments**

*variantObj* Specify the variant object to which you want to append data. The Content property is modified to add the new data.

*contents* Specify the data you want to add to a variant object. Contents can either be a cell array or an array of cell arrays. A valid cell array should have the form {'Type', 'Name', 'PropertyName', PropertyValue}, where PropertyValue is the new value to be applied for the PropertyName. Valid Type, Name, and PropertyName values are as follows.

| 'Type'        | 'Name'   | 'PropertyName'  |
|---------------|--|-----------------|
| 'species'     | Name of the species. If there are multiple species in the model with the same name, specify the species as [compartmentName.speciesName], where compartmentName is the name of the compartment containing the species. | 'InitialAmount' |
| 'parameter'   | If the parameter scope is a model, specify the parameter name. If the parameter scope is a kinetic law, specify [reactionName.parameterName].  | 'Value'         |
| 'compartment' | Name of the compartment.   | 'Capacity'      |

# addcontent (variant)

---

## Description

`addcontent(variantObj, contents)` adds the data stored in the variable `contents` to the variant object (`variantObj`).

`addcontent(variantObj1, variantObj2)` appends the data in the Content property of the variant object `variantObj2` to the Content property of variant object `variantObj1`.

---

**Note** Remember to use the `addcontent` method instead of using the `set` method on the Content property because the `set` method replaces the data in the Content property, whereas `addcontent` appends the data.

---

## Examples

- 1 Create a model containing one species.

```
modelObj = sbiomodel('mymodel');  
compObj = addcompartment(modelObj, 'comp1');  
speciesObj = addspecies(compObj, 'A');
```

- 2 Add a variant object that varies the `InitialAmount` property of a species named A.

```
variantObj = addvariant(modelObj, 'v1');  
addcontent(variantObj, {'species', 'A', 'InitialAmount', 5});
```

## See Also

`addvariant`, `rmcontent`, `sbiovariant`

**Purpose** Add dose object to model

**Syntax**

```
doseObj2 = adddose(mode1Obj, 'DoseName')  
doseObj2 = adddose(mode1Obj, 'DoseName', 'DoseType')  
doseObj2 = adddose(mode1Obj, doseObj)
```

## Arguments

|                 |  |
|-----------------|--|
| <i>mode1Obj</i> | Model object to which you add a dose object.   |
| <i>DoseName</i> | Name of a dose object to construct and add to a model object. <i>DoseName</i> is the value of the dose object property Name. |
| <i>DoseType</i> | Type of dose object to construct and add to a model object. Enter either 'schedule' or 'repeat'.                             |
| <i>doseObj</i>  | Dose object to add to a model object. Created with the constructor sbiodose.   |

## Outputs

|                 |  |
|-----------------|--|
| <i>doseObj2</i> | ScheduleDose or RepeatDose object. A RepeatDose or ScheduleDose object defines an increase (dose) to a species amount during a simulation. |
|-----------------|--|

## Description

Before using a dose object in a simulation, use the `adddose` method to add the dose object to a SimBiology model object. Then, set the `Active` dose object property to `true`.

`doseObj2 = adddose(mode1Obj, 'DoseName')` constructs a SimBiology RepeatDose object (*doseObj2*), assigns *DoseName* to the property Name, adds the dose object to a SimBiology model object (*mode1Obj*), and assigns *mode1Obj* to the property Parent.

# adddose (model)

---

`doseObj2 = adddose(modeObj, 'DoseName', 'DoseType')` constructs either a SimBiology ScheduleDose object or RepeatDose object (`doseObj`).

`doseObj2 = adddose(modeObj, doseObj)` adds a SimBiology dose object (`doseObj`) to a SimBiology model object (`modeObj`), copies the dose object to a second dose object (`doseObj2`), and assigns `modeObj` to the property Parent.

## Example

Add a dose object to a model object.

- 1 Create a model then add a dose to the model.

```
modelObj = sbiomodel('mymodel');  
doseObj = adddose(modelObj, 'dose1');
```

- 2 Define properties of the dose object.

```
doseObj.Amount = 5;  
doseObj.Repeat = 6;  
doseObj.Interval = 24;  
doseObj.TimeUnits = 'hour'
```

## See Also

Model object methods:

- `adddose` — add a dose object to a model object
- `getdose` — get dose information from a model object
- `removedose` — remove a dose object from a model object

Dose object constructor `sbiodose`.

ScheduleDose object and RepeatDose object methods:

- `copyobj` — copy a dose object from one model object to another model object
- `get` — view properties for a dose object
- `set` — define or modify properties for a dose object

**Purpose** Add event object to model object

**Syntax**

```
eventObj = addevent(modelObj, 'TriggerValue', 'EventFcnsValue')  
eventObj = addevent(...'PropertyName', PropertyValue...)
```

**Arguments**

|                       |   |
|-----------------------|---|
| <i>modelObj</i>       | Model object.   |
| <i>TriggerValue</i>   | Required property to specify a trigger condition. Must be a MATLAB expression that evaluates to a logical value. Use the keyword 'time' to specify that an event occurs at a specific time during the simulation. For more information, see <a href="#">Trigger</a> .                           |
| <i>EventFcnsValue</i> | A string or a cell array of strings, each of which specifies an assignment of the form ' <i>objectname</i> = <i>expression</i> ', where <i>objectname</i> is the name of a valid object. Defines what occurs when the event is triggered. For more information, see <a href="#">EventFcns</a> . |
| <i>PropertyName</i>   | Property name for an event object from “ <a href="#">Property Summary</a> ” on page 2-28.   |
| <i>PropertyValue</i>  | Property value. For more information on property values, see the property reference for each property listed in “ <a href="#">Property Summary</a> ” on page 2-28.  |

**Description**

`eventObj = addevent(modelObj, 'TriggerValue', 'EventFcnsValue')` creates an event object (*eventObj*) and adds the event to the model (*modelObj*). In the event object, this method assigns a value (*TriggerValue*) to the property `TriggerCondition`, assigns a value (*EventFcnsValue*) to the property `EventFcns`, and assigns the model object (*modelObj*) to the property `Parent`. In the model object, this method appends the event object to the property `Events`.

# addevent (model)

---

When the trigger expression in the property `Trigger` changes from false to true, the assignments in `EventFcns` are executed during simulation.

For details on how events are handled during a simulation, see “Event Object”.

`eventObj = addevent(...'PropertyName', PropertyValue...)` defines optional properties. The property name and property value pairs can be any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs).

You can view additional object properties with the `get` command. You can modify additional object properties with the `set` command. To view events of a model object (`modelObj`), use the command `get(modelObj, 'Events')`.

## Method Summary

|                                   |   |
|-----------------------------------|---|
| <code>copyobj</code> (any object) | Copy SimBiology object and its children |
| <code>delete</code> (any object)  | Delete SimBiology object                |
| <code>display</code> (any object) | Display summary of SimBiology object    |
| <code>get</code> (any object)     | Get object properties                   |
| <code>set</code> (any object)     | Set object properties                   |

## Property Summary

|                        |  |
|------------------------|--|
| <code>Active</code>    | Indicate object in use during simulation |
| <code>EventFcns</code> | Event expression                         |
| <code>Name</code>      | Specify name of object                   |
| <code>Notes</code>     | HTML text describing SimBiology object   |
| <code>Parent</code>    | Indicate parent object                   |

|          |                                       |
|----------|---------------------------------------|
| Tag      | Specify label for SimBiology object   |
| Trigger  | Event trigger                         |
| Type     | Display SimBiology object type        |
| UserData | Specify data to associate with object |

## Examples

- 1 Create a model object, and then add an event object.

```
modelObj = sbmlimport('oscillator')  
eventObj = addevent(modelObj, 'time>= 5', 'OpC = 200');
```

- 2 Get a list of properties for an event object.

```
get(modelObj.Events(1));
```

Or

```
get(eventObj)
```

MATLAB displays a list of event properties.

```
Active: 1  
Annotation: ''  
EventFcns: {'OpC = 200'}  
Name: ''  
Notes: ''  
Parent: [1x1 SimBiology.Model]  
Tag: ''  
Trigger: 'time >= 5'  
Type: 'event'  
UserData: []
```

## See Also

Event object

# addkineticlaw (reaction)

---

**Purpose** Create kinetic law object and add to reaction object

**Syntax**

```
kineticLawObj = addkineticlaw(reactionObj, 'KineticLawNameValue')  
kineticLawObj = addkineticlaw(..., 'PropertyName',  
PropertyValue, ...)
```

**Arguments**

|                            |  |
|----------------------------|--|
| <i>reactionObj</i>         | Reaction object. Enter a variable name for a reaction object.  |
| <i>KineticLawNameValue</i> | Property to select the type of kinetic law object to create. For built-in kinetic law, valid values are:<br><br>'Unknown', 'MassAction',<br>'Henri-Michaelis-Menten',<br>'Henri-Michaelis-Menten-Reversible',<br>'Hill-Kinetics', 'Iso-Uni-Uni',<br>'Ordered-Bi-Bi', 'Ping-Pong-Bi-Bi',<br>'Competitive-Inhibition',<br>'NonCompetitive-Inhibition', and<br>'UnCompetitive-Inhibition'.<br><br>Find valid <i>KineticLawNameValues</i> by using <code>sbioroot</code> to create a <code>SimBiology</code> root object, then query the object with the commands <code>get(rootObj.BuiltInLibrary, 'KineticLaws')</code> and <code>get(rootObj.UserDefinedLibrary, 'KineticLaws')</code> .<br><br><code>sbiowhos -kineticlaw</code> lists kinetic laws in the <code>SimBiology</code> root, which includes kinetic laws from both the <code>BuiltInLibrary</code> and the <code>UserDefinedLibrary</code> . |

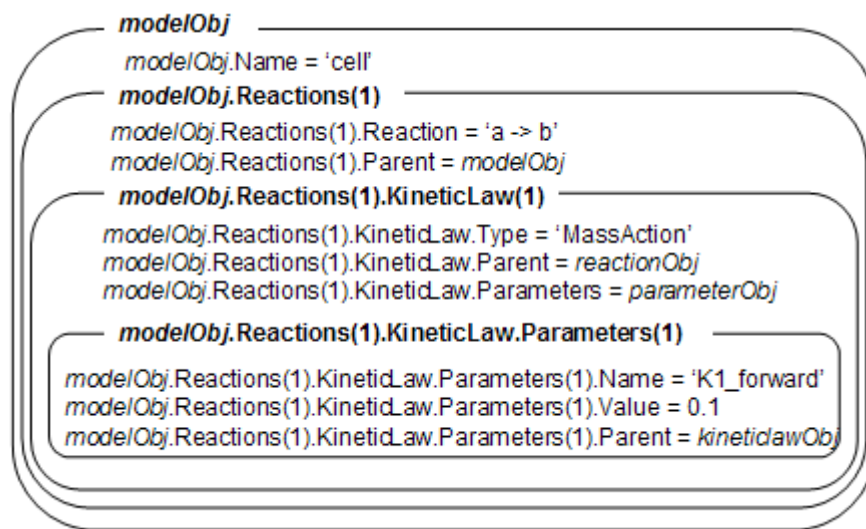


## Description

*kineticlawObj* = addkineticlaw(*reactionObj*, '*KineticLawNameValue*') creates a kinetic law object and returns the kinetic law object (*kineticlawObj*).

In the kinetic law object, this method assigns a name (*KineticLawNameValue*) to the property *KineticLawName* and assigns the reaction object to the property *Parent*. In the reaction object, this method assigns the kinetic law object to the property *KineticLaw*.

```
modelObj = sbiomodel('cell');  
reactionObj = addreaction(modelObj, 'a -> b');  
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');  
parameterObj = addparameter(kineticlawObj, 'K1_forward', 0.1);  
set(kineticlawObj, ParameterVariableName, 'K1_forward');
```



*KineticLawNameValue* is any valid kinetic law definition. See “Kinetic Law Definition” on page 3-71 for a definition of kinetic laws and more information about how they are used to get the reaction rate expression.

*kineticlawObj* = addkineticlaw(..., '*PropertyName*', *PropertyValue*, ...) constructs a kinetic law object, *kineticlawObj*, and configures

## addkineticlaw (reaction)

---

*kineticlawObj* with property value pairs. The property name/property value pairs can be in any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs). The *kineticlawObj* properties are listed in “Property Summary” on page 2-33.

You can view additional kinetic law object properties with the `get` command. You can modify additional kinetic law object properties with the `set` command. The kinetic law used to determine the `ReactionRate` of the `Reaction` can be viewed with `get(reactionObj, 'KineticLaw')`. Remove a SimBiology kinetic law object from a SimBiology reaction object with the `delete` command.

---

**Note** To define a Hill kinetic rate equation with a non-integer exponent that is compatible with `DimensionalAnalysis`, see “Define a Custom Hill Kinetic Law that Works with Dimensional Analysis” on page 3-149

---

### Method Summary

Methods for kinetic law objects

|   |  |
|---|--|
| <code>addparameter</code> (model, kineticlaw) | Create parameter object and add to model or kinetic law object |
| <code>copyobj</code> (any object)             | Copy SimBiology object and its children                        |
| <code>delete</code> (any object)              | Delete SimBiology object                                       |
| <code>display</code> (any object)             | Display summary of SimBiology object                           |
| <code>get</code> (any object)                 | Get object properties  |
| <code>getparameters</code> (kineticlaw)       | Get specific parameters in kinetic law object                  |
| <code>getspecies</code> (kineticlaw)          | Get specific species in kinetic law object                     |

|   |   |
|---|---|
| reorder (model, compartment, kinetic law) | Reorder component lists                           |
| set (any object)                          | Set object properties                             |
| setparameter (kineticlaw)                 | Specify specific parameters in kinetic law object |
| setspecies (kineticlaw)                   | Specify species in kinetic law object             |

### Property Summary

Properties for kinetic law objects

|   |   |
|---|---|
| Expression<br>(AbstractKineticLaw,<br>KineticLaw) | Expression to determine reaction rate equation  |
| KineticLawName                                    | Name of kinetic law applied to reaction         |
| Name  | Specify name of object                          |
| Notes   | HTML text describing SimBiology object          |
| Parameters  | Array of parameter objects                      |
| ParameterVariableNames                            | Cell array of reaction rate parameters          |
| ParameterVariables                                | Parameters in kinetic law definition            |
| Parent  | Indicate parent object                          |
| SpeciesVariableNames                              | Cell array of species in reaction rate equation |
| SpeciesVariables                                  | Species in abstract kinetic law                 |
| Tag   | Specify label for SimBiology object             |

# addkineticlaw (reaction)

---

|          |                                       |
|----------|---------------------------------------|
| Type     | Display SimBiology object type        |
| UserData | Specify data to associate with object |

## Examples

### Example 1

This example uses the built-in kinetic law Henri-Michaelis-Menten.

- 1 Create a model object, and add a reaction object to the model.

```
modelObj = sbiomodel ('Cell');  
reactionObj = addreaction (modelObj, 'Substrate -> Product');
```

- 2 Define a kinetic law for the reaction object and view the parameters to be set.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');  
get (kineticlawObj, 'Expression')
```

```
ans =  
    Vm*S/(Km + S)
```

The `addkineticlaw` method adds a kinetic law to the reaction object (*reactionObj*).

The Henri-Michaelis-Menten kinetic law has two parameters ( $V_m$  and  $K_m$ ) and one species ( $S$ ). You need to enter values for these parameters by first creating parameter objects, and then adding the parameter objects to the kinetic law object.

- 3 Add parameter objects to a kinetic law object. For example, create parameter objects named  $V_m_d$  and  $K_m_d$ , and add them to a kinetic law object (*kineticlawObj*).

```
parameterObj1 = addparameter(kineticlawObj, 'Vm_d', 'Value', 6.0);  
parameterObj2 = addparameter(kineticlawObj, 'Km_d', 'Value', 1.25);
```

The `addparameter` method creates two parameter objects with values that are associated with the kinetic law parameters.

- 4 Associate kinetic law parameters with the parameters in the kinetic law definition.

```
set(kineticlawObj, 'ParameterVariableNames', {'Vm_d' 'Km_d'});  
set(kineticlawObj, 'SpeciesVariableNames', {'Substrate'});
```

This method associates the parameters in the property `ParameterVariableNames` with the parameters in the property `ParameterVariables` using a one-to-one mapping in the order given.

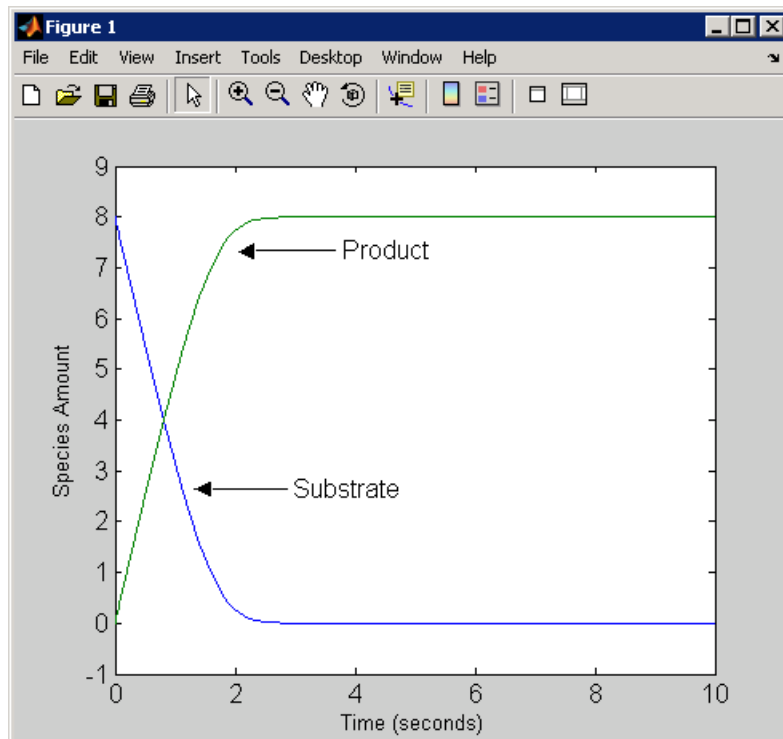
- 5 Verify that the reaction rate is expressed correctly in the reaction object `ReactionRate` property.

```
get (reactionObj, 'ReactionRate')  
  
ans =  
    Vm_d*Substrate/(Km_d+Substrate)
```

- 6 Enter an initial value for the substrate and simulate.

```
modelObj.Species(1).InitialAmount = 8;  
[T, X] = sbiosimulate(modelObj);  
plot(T,X)
```

# addkineticlaw (reaction)



## Example 2

This example uses the built-in kinetic law MassAction.

- 1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('Cell');  
reactionObj = addreaction (modelObj, 'a -> b');
```

- 2 Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');  
get(kineticlawObj, 'Expression')  
ans =
```

### MassAction

Notice, the property Expression for MassAction kinetic law does not show the parameters and species in the reaction rate.

- 3 Assign the rate constant for the reaction.

```
parameterObj = addparameter(kineticlawObj, 'k_forward');  
set (kineticlawObj, 'ParameterVariablenames', 'k_forward');  
get (reactionObj, 'ReactionRate')
```

```
ans =  
    k_forward*a
```

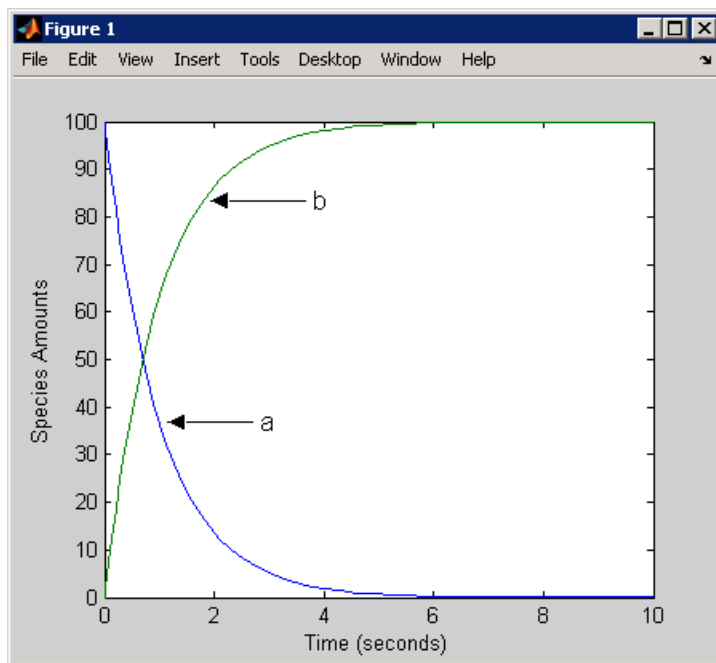
- 4 Enter an initial value for the substrate and simulate.

```
modelObj.Species(1).InitialAmount = 100;  
[T, X] = sbiosimulate(modelObj);plot(T,X)
```

The value used for k\_forward is the default value = 1.0.

# addkineticlaw (reaction)

---



**See Also** `addreaction`, `setparameter`



# addparameter (model, kineticlaw)

---

**Purpose** Create parameter object and add to model or kinetic law object

**Syntax**

```
parameterObj = addparameter(Obj, 'NameValue')  
parameterObj = addparameter(Obj, 'NameValue', ValueValue)  
parameterObj = addparameter(...'PropertyName', PropertyValue...)
```

## Arguments

|                   |  |
|-------------------|--|
| <i>Obj</i>        | Model or kinetic law object. Enter a variable name for the object.   |
| <i>NameValue</i>  | Property for a parameter object. Enter a unique character string. Since objects can use this property to reference a parameter, a parameter object must have a unique name at the level it is created. For example, a kinetic law object cannot contain two parameter objects named kappa. However, the model object that contains the kinetic law object can contain a parameter object named kappa along with the kinetic law object.<br><br>For information on naming parameters, see Name. |
| <i>ValueValue</i> | Property for a parameter object. Enter a number.   |

## Description

`parameterObj = addparameter(Obj, 'NameValue')` creates a parameter object and returns the object (*parameterObj*). In the parameter object, this method assigns a value (*NameValue*) to the property *Name*, assigns a value 1 to the property *Value*, and assigns the model or kinetic law object to the property *Parent*. In the model or kinetic law object, (*Obj*), this method assigns the parameter object to the property *Parameters*.

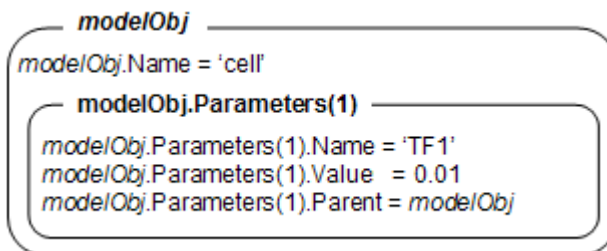
A parameter object defines an assignment that a model or a kinetic law can use. The scope of the parameter is defined by the parameter parent. If a parameter is defined with a kinetic law object, then only the kinetic law object and objects within the kinetic law object can use the

## addparameter (model, kineticlaw)

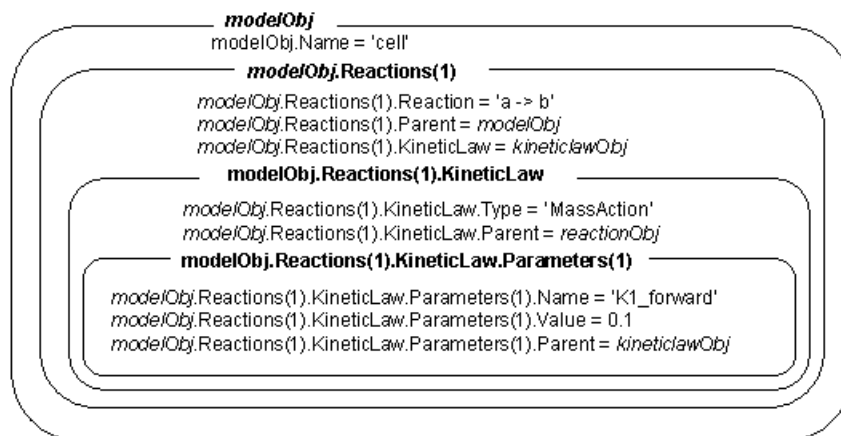
---

parameter. If a parameter object is defined with a model object as its parent, then all objects within the model (including all rules, events and kinetic laws) can use the parameter.

```
modelObj = sbiomodel('cell')
parameterObj = addparameter(modelObj, 'TF1', 0.01)
```



```
modelObj = sbiomodel('cell')
reactionObj = addreaction(modelObj, 'a -> b')
kineticlawObj = addkineticlaw (reactionObj, 'MassAction')
parameterObj = addparameter(kineticlawObj, 'K1_forward', 0.1)
```



## addparameter (model, kineticlaw)

---

`parameterObj = addparameter(Obj, 'NameValue', ValueValue)` creates a parameter object, assigns a value (*NameValue*) to the property *Name*, assigns the value (*ValueValue*) to the property *Value*, and assigns the model object or the kinetic law object to the property *Parent*. In the model or kinetic law object (*Obj*), this method assigns the parameter object to the property *Parameters*, and returns the parameter object to a variable (`parameterObj`).

`parameterObj = addparameter(...'PropertyName', PropertyValue...)` defines optional property values. The property name/property value pairs can be in any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs).

**Scope of a parameter** — A parameter can be *scoped* to either a model or a kinetic law.

- When a kinetic law searches for a parameter in its expression, it first looks in the parameter list of the kinetic law. If the parameter isn't found there, it moves to the model that the kinetic law object is in and looks in the model parameter list. If the parameter isn't found there, it moves to the model parent.
- When a rule searches for a parameter in its expression, it looks in the parameter list for the model. If the parameter isn't found there, it moves to the model parent. A rule cannot use a parameter that is scoped to a kinetic law. So for a parameter to be used in both a reaction rate equation and a rule, the parameter should be *scoped* to a model.

Additional parameter object properties can be viewed with the `get` command. Additional parameter object properties can be modified with the `set` command. The parameters of *Obj* can be viewed with `get(Obj, 'Parameters')`.

A SimBiology parameter object can be copied to a SimBiology model or kinetic law object with `copyobj`. A SimBiology parameter object can be removed from a SimBiology model or kinetic law object with `delete`.

# addparameter (model, kineticlaw)

---

## Method Summary

Methods for parameter objects

|  |   |
|--|---|
| copyobj (any object)                               | Copy SimBiology object and its children |
| delete (any object)                                | Delete SimBiology object                |
| display (any object)                               | Display summary of SimBiology object    |
| get (any object)                                   | Get object properties                   |
| rename (compartment, parameter, species, reaction) | Rename object and update expressions    |
| set (any object)                                   | Set object properties                   |

## Property Summary

Properties for parameter objects

|               |  |
|---------------|--|
| ConstantValue | Specify variable or constant parameter value |
| Name          | Specify name of object                       |
| Notes         | HTML text describing SimBiology object       |
| Parent        | Indicate parent object                       |
| Tag           | Specify label for SimBiology object          |
| Type          | Display SimBiology object type               |
| UserData      | Specify data to associate with object        |
| Value         | Assign value to parameter object             |
| ValueUnits    | Parameter value units                        |

## Example

- 1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');  
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

- 2 Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');
```

- 3 Add a parameter and assign it to the kinetic law object (kineticlawObj); add another parameter and assign to the model object (modelObj).

```
% Add parameter to kinetic law object  
parameterObj1 = addparameter (kineticlawObj, 'K1');
```

```
get (kineticlawObj, 'Parameters')
```

MATLAB returns:

SimBiology Parameter Array

| Index: | Name: | Value: | ValueUnits: |
|--------|-------|--------|-------------|
| 1      | K1    | 1      |             |

```
% Add parameter with value 0.9 to model object  
parameterObj1 = addparameter (modelObj, 'K2', 0.9);
```

```
get (modelObj, 'Parameters')
```

MATLAB returns:

SimBiology Parameter Array

| Index: | Name: | Value: | ValueUnits: |
|--------|-------|--------|-------------|
| 1      | K2    | 1      |             |

## See Also

[addreaction](#)

# addproduct (reaction)

---

**Purpose** Add product species object to reaction object

**Syntax**

```
speciesObj = addproduct(reactionObj, 'NameValue')
speciesObj = addproduct(reactionObj, speciesObj)
speciesObj = addproduct(reactionObj, 'NameValue',
Stoichcoefficient)
speciesObj = addproduct(reactionObj, speciesObj,
Stoichcoefficient)
```

## Arguments

|                          |  |
|--------------------------|--|
| <i>reactionObj</i>       | Reaction object. Enter a name for the reaction object.   |
| <i>NameValue</i>         | Property of a species object that names the object (not the reaction object). Enter a unique character string. For example, 'fructose 6-phosphate'. A species object can be referenced by other objects using this property. You can use the function <code>sbioselect</code> to find an object with a specific <i>NameValue</i> . |
| <i>speciesObj</i>        | Species object.  |
| <i>Stoichcoefficient</i> | Stoichiometric coefficients for products, length of array equal to length of <i>NameValue</i> , or length of <i>speciesObj</i> .   |

## Description

`speciesObj = addproduct(reactionObj, 'NameValue')` creates a species object and returns the species object (*speciesObj*). In the species object, this method assigns the value (*NameValue*) to the property *Name*. In the reaction object, this method assigns the species object to the property *Products*, modifies the reaction equation in the property *Reaction* to include the new species, and adds the stoichiometric coefficient 1 to the property *Stoichiometry*.

When you define a reaction with a new species:

- If no compartment objects exist in the model, the method creates a compartment object (called '*unnamed*') in the model and adds the newly created species to that compartment.
- If only one compartment object (*compObj*) exists in the model, the method creates a species object in that compartment.
- If there is more than one compartment object (*compObj*) in the model, you must qualify the species name with the compartment name.

For example, `cell.glucose` denotes that you want to put the species named `glucose` into a compartment named `cell`. Additionally, if the compartment named `cell` does not exist, the process of adding the reaction creates the compartment and names it `cell`.

Create and add a species object to a compartment object with the method `addspecies`.

`speciesObj = addproduct(reactionObj, speciesObj)`, in the species object (`speciesObj`), assigns the parent object of the `reactionObj` to the species property `Parent`. In the reaction object (`reactionObj`), it assigns the species object to the property `Products`, modifies the reaction equation in the property `Reaction` to include the new species, and adds the stoichiometric coefficient 1 to the property `Stoichiometry`.

`speciesObj = addproduct(reactionObj, 'NameValue', Stoichcoefficient)`, in addition to the description above, adds the stoichiometric coefficient (`Stoichcoefficient`) to the property `Stoichiometry`. If `NameValue` is a cell array of species names, then `Stoichcoefficient` must be a vector of doubles with the same length as `NameValue`.

`speciesObj = addproduct(reactionObj, speciesObj, Stoichcoefficient)`, in addition to the description above, adds the stoichiometric coefficient (`Stoichcoefficient`) to the property `Stoichiometry`.

Species names are referenced by reaction objects, kinetic law objects, and model objects. If you change the `Name` of a species the reaction also uses the new name. You must however configure all other applicable elements such as rules that use the species, and the kinetic law object.

## addproduct (reaction)

---

### Examples

- 1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');  
reactionObj = addreaction(modelObj, 'A + C -> U');
```

- 2 Modify the reaction of the reactionObj from  $A + C \rightarrow U$  to  $A + C \rightarrow U + 2 H$ .

```
speciesObj = addproduct(reactionObj, 'H', 2);
```

### See Also

addspecies



## Purpose

Add species object as reactant to reaction object

## Syntax

```
speciesObj = addreactant(reactionObj, 'NameValue')  
addreactant(reactionObj, speciesObj, StoichCoefficient)  
addreactant(reactionObj, 'NameValue', StoichCoefficient)
```

## Arguments

|                          |  |
|--------------------------|--|
| <i>reactionObj</i>       | Reaction object.   |
| <i>NameValue</i>         | Name property of a species object. Enter a unique character string, for example, 'glucose'. A species object can be referenced by other objects using this property. You can use the function <code>sbioselect</code> to find an object with a specific Name property value. |
| <i>speciesObj</i>        | Species object or cell array of species objects.   |
| <i>StoichCoefficient</i> | Stoichiometric coefficients for reactants, length of array equal to length of <i>NameValue</i> or length of <i>speciesObj</i> .  |

## Description

`speciesObj = addreactant(reactionObj, 'NameValue')` creates a species object and returns the species object (*speciesObj*). In the species object, this method assigns the value (*NameValue*) to the property `Name`. In the reaction object, this method assigns the species object to the property `Reactants`, modifies the reaction equation in the property `Reaction` to include the new species, and adds the stoichiometric coefficient -1 to the property `Stoichiometry`.

When you define a reaction with a new species:

- If no compartment objects exist in the model, the method creates a compartment object (called '*unnamed*') in the model and adds the newly created species to that compartment.
- If only one compartment object (`compObj`) exists in the model, the method creates a species object in that compartment.

## addreactant (reaction)

---

- If there is more than one compartment object (`compObj`) in the model, you must qualify the species name with the compartment name.

For example, `cell.glucose` denotes that you want to put the species named `glucose` into a compartment named `cell`. Additionally, if the compartment named `cell` does not exist, the process of adding the reaction creates the compartment and names it `cell`.

Create and add a species object to a compartment object with the method `addspecies`.

`addreactant(reactionObj, speciesObj, StoichCoefficient)`, in the species object (`speciesObj`), assigns the parent object to the `speciesObj` property `Parent`. In the reaction object (`reactionObj`), it assigns the species object to the property `Reactants`, modifies the reaction equation in the property `Reaction` to include the new species, and adds the stoichiometric coefficient `-1` to the property `Stoichiometry`. If `speciesObj` is a cell array of species objects, then `StoichCoefficient` must be a vector of doubles with the same length as `speciesObj`.

`addreactant(reactionObj, 'NameValue', StoichCoefficient)`, in addition to the description above, adds the stoichiometric coefficient (`StoichCoefficient`) to the property `Stoichiometry`. If `NameValue` is a cell array of species names, then `StoichCoefficient` must be a vector of doubles with the same length as `NameValue`.

Species names are referenced by reaction objects, kinetic law objects, and model objects. If you change the `Name` of a species the reaction also uses the new name. You must, however, configure all other applicable elements such as rules that use the species, and the kinetic law object.

See for more information on species names.

### Example

- 1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');  
reactionObj = addreaction(modelObj, 'A -> U');
```

**2** Modify the reaction of the `reactionObj` from  $A \rightarrow U$  to be  $A + 3C \rightarrow U$ .

```
speciesObj = addreactant(reactionObj, 'C', 3);
```

### See Also

`addspecies`

# addreaction (model)

---

**Purpose** Create reaction object and add to model object

**Syntax**

```
reactionObj = addreaction(modelObj, 'ReactionValue')
reactionObj = addreaction(modelObj, 'ReactantsValue',
    'ProductsValue')
reactionObj = addreaction(modelObj, 'ReactantsValue',
    RStoichCoefficients, 'ProductsValue', PStoichCoefficients)
reactionObj = addreaction(...'PropertyName', PropertyValue...)
```

## Arguments

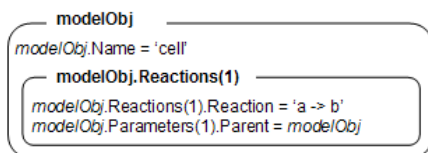
|                       |   |
|-----------------------|---|
| <i>modelObj</i>       | SimBiology model object.  |
| <i>ReactionValue</i>  | <p>Specify the reaction equation. Enter a character string. A hyphen preceded by a space and followed by a right angle bracket (-&gt;) indicates reactants going forward to products. A hyphen with left and right angle brackets (&lt;-&gt;) indicates a reversible reaction. Coefficients before reactant or product names must be followed by a space.</p> <p>Examples are 'A -&gt; B', 'A + B -&gt; C', '2 A + B -&gt; 2 C', and 'A &lt;-&gt; B'. Enter reactions with spaces between the species.</p> <p>If there are multiple compartments, or to specify the compartment name, use <i>compartmentName.speciesName</i>.</p> <p>Examples are 'cytoplasm.A -&gt; cytoplasm.B', 'cytoplasm.A -&gt; nucleus.A', and 'cytoplasm.A + cytoplasm.B -&gt; nucleus.AB'.</p> |
| <i>ReactantsValue</i> | A string defining the species name, a cell array of strings, a species object, or an array of species objects. If using name strings, qualify with compartment names if there are multiple compartments.  |

|                            |  |
|----------------------------|--|
| <i>ProductsValue</i>       | A string defining the species name, a cell array of strings, a species object, or an array of species objects. If using name strings, qualify with compartment names if there are multiple compartments. |
| <i>RStoichCoefficients</i> | Stoichiometric coefficients for reactants, length of array equal to length of <i>ReactantsValue</i> .  |
| <i>PStoichCoefficients</i> | Stoichiometric coefficients for products, length of array equal to length of <i>ProductsValue</i> .  |

## Description

`reactionObj = addreaction(modelObj, 'ReactionValue')` creates a reaction object, assigns a value (*ReactionValue*) to the property *Reaction*, assigns reactant species object(s) to the property *Reactants*, assigns the product species object(s) to the property *Products*, and assigns the model object to the property *Parent*. In the Model object (`modelObj`), this method assigns the reaction object to the property *Reactions*, and returns the reaction object (`reactionObj`).

```
reactionObj = addreaction(modelObj, 'a -> b')
```



When you define a reaction with a new species:

- If no compartment objects exist in the model, the method creates a compartment object (called '*unnamed*') in the model and adds the newly created species to that compartment.
- If only one compartment object (`compObj`) exists in the model, the method creates a species object in that compartment.

## addreaction (model)

---

- If there is more than one compartment object (`compObj`) in the model, you must qualify the species name with the compartment name.

For example, `cell.glucose` denotes that you want to put the species named `glucose` into a compartment named `cell`. Additionally, if the compartment named `cell` does not exist, the process of adding the reaction creates the compartment and names it `cell`.

You can manually add a species to a compartment object with the method `addspecies`.

You can add species to a reaction object using the methods `addreactant` or `addproduct`. You can remove species from a reaction object with the methods `rmreactant` or `rmproduct`. The property `Reaction` is modified by adding or removing species from the reaction equation.

You can copy a SimBiology reaction object to a model object with the function `copyobj`. You can remove the SimBiology reaction object from a SimBiology model object with the function `delete`.

You can view additional reaction object properties with the `get` command. For example, the reaction equation of `reactionObj` can be viewed with the command `get(reactionObj, 'Reaction')`. You can modify additional reaction object properties with the command `set`.

`reactionObj = addreaction(modelObj, 'ReactantsValue', 'ProductsValue')` creates a reaction object, assigns a value to the property `Reaction` using the reactant (`ReactantsValue`) and product (`ProductsValue`) names, assigns the species objects to the properties `Reactants` and `Products`, and assigns the model object to the property `Parent`. In the model object (`modelObj`), this method assigns the reaction object to the property `Reactions`, and returns the reaction object (`reactionObj`). The stoichiometric values are assumed to be 1.

`reactionObj = addreaction(modelObj, 'ReactantsValue', RStoichCoefficients, 'ProductsValue', PStoichCoefficients)` adds stoichiometric coefficients (`RStoichCoefficients`) for reactant species, and stoichiometric coefficients (`PStoichCoefficients`) for product species to the property `Stoichiometry`. The length of `Reactants`

and `RCoefficients` must be equal, and the length of `Products` and `PCoefficients` must be equal.

`reactionObj = addreaction(...'PropertyName', PropertyValue...)` defines optional properties. The property name/property value pairs can be in any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs).

---

**Note** If you use the `addreaction` method to create a reaction rate expression that is not continuous and differentiable, see “Using Events to Address Discontinuities in Rule and Reaction Rate Expressions” before simulating your model.

---

## Method Summary

### Methods for reaction objects

|   |  |
|---|--|
| <code>addkineticlaw (reaction)</code>                           | Create kinetic law object and add to reaction object |
| <code>addproduct (reaction)</code>                              | Add product species object to reaction object        |
| <code>addreactant (reaction)</code>                             | Add species object as reactant to reaction object    |
| <code>copyobj (any object)</code>                               | Copy SimBiology object and its children              |
| <code>delete (any object)</code>                                | Delete SimBiology object                             |
| <code>display (any object)</code>                               | Display summary of SimBiology object                 |
| <code>get (any object)</code>                                   | Get object properties                                |
| <code>rename (compartment, parameter, species, reaction)</code> | Rename object and update expressions                 |
| <code>rmproduct (reaction)</code>                               | Remove species object from reaction object products  |

# addreaction (model)

---

|                       |  |
|-----------------------|--|
| rmreactant (reaction) | Remove species object from reaction object reactants |
| set (any object)      | Set object properties                                |

## Property Summary

|                                 |  |
|---------------------------------|--|
| Properties for reaction objects |  |
| Active                          | Indicate object in use during simulation               |
| KineticLaw                      | Show kinetic law used for ReactionRate                 |
| Name                            | Specify name of object                                 |
| Notes                           | HTML text describing SimBiology object                 |
| Parent                          | Indicate parent object                                 |
| Products                        | Array of reaction products                             |
| Reactants                       | Array of reaction reactants                            |
| Reaction                        | Reaction object reaction                               |
| ReactionRate                    | Reaction rate equation in reaction object              |
| Reversible                      | Specify whether reaction is reversible or irreversible |
| Stoichiometry                   | Species coefficients in reaction                       |
| Tag                             | Specify label for SimBiology object                    |
| Type                            | Display SimBiology object type                         |
| UserData                        | Specify data to associate with object                  |



## Examples

Create a model, add a reaction object, and assign the expression for the reaction rate equation.

- 1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');  
reactionObj = addreaction(modelObj, 'a -> c + d');
```

- 2 Create a kinetic law object for the reaction object, of the type 'Henri-Michaelis-Menten'.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

reactionObj.KineticLaw property is configured to kineticlawObj.

- 3 The 'Henri-Michaelis-Menten' kinetic law has two parameter variables (Vm and Km) and one species variable (S) that should be set. To set these variables, first create the parameter variables as parameter objects (parameterObj1, parameterObj2) with names Vm\_d, and Km\_d, and assign the objects Parent property value to the kineticlawObj.

```
parameterObj1 = addparameter(kineticlawObj, 'Vm_d');  
parameterObj2 = addparameter(kineticlawObj, 'Km_d');
```

- 4 Set the variable names for the kinetic law object.

```
set(kineticlawObj, 'ParameterVariableNames', {'Vm_d' 'Km_d'});  
set(kineticlawObj, 'SpeciesVariableNames', {'a'});
```

- 5 Verify that the reaction rate is expressed correctly in the reaction object ReactionRate property.

```
get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =
```

## addreaction (model)

---

$$V_m \cdot d \cdot a / (K_m \cdot d + a)$$

### **See Also**

addkineticlaw, addproduct, addreactant, rmproduct, rmreactant

**Purpose** Create rule object and add to model object

**Syntax**

```
ruleObj = addrule(modelObj, Rule)
ruleObj = addrule(modelObj, Rule, RuleType)
ruleObj = addrule(..., 'PropertyName', PropertyValue,...)
```

## Arguments

|                 |  |
|-----------------|--|
| <i>modelObj</i> | Model object to which to add the rule.   |
| <i>Rule</i>     | String specifying the rule. For example, enter the algebraic rule 'Va*Ea + Vi*Ei - K2'.  |
| <i>RuleType</i> | String specifying the type of rule. Choices are: <ul style="list-style-type: none"><li>• 'algebraic'</li><li>• 'initialAssignment'</li><li>• 'repeatedAssignment'</li><li>• 'rate'</li></ul> |

For more information, see RuleType

## Description

A rule is a mathematical expression that changes the amount of a species or the value of a parameter. It also defines how species and parameters interact with one another.

*ruleObj* = *addrule(modelObj, Rule)* constructs and returns *ruleObj*, a rule object. In *ruleObj*, the rule object, this method assigns the *modelObj* input argument to the Parent property, assigns the *Rule* input argument to the Rule property, and assigns 'initialAssignment' or 'algebraic' to the RuleType property. (This method assigns 'initialAssignment' for all assignment rules and 'algebraic' for all other rules.) In *modelObj*, the model object, this method assigns *ruleObj*, the rule object, to the Rules property.

*ruleObj* = *addrule(modelObj, Rule, RuleType)* in addition to the assignments above, assigns the *RuleType* input argument to the

## addrule (model)

---

RuleType property. For more information on the types of rules, see RuleType.

ruleObj = addrule(..., 'PropertyName', PropertyValue,...) defines optional properties. The property name/property value pairs can be in any format supported by the function set (for example, name-value string pairs, structures, and name-value cell array pairs).

View additional rule properties with the function get, and modify rule properties with the function set. Copy a rule object to a model with the function copyobj, or delete a rule object from a model with the function delete.

---

**Note** If you use the addrule method to create an algebraic rule, rate rule, or repeated assignment rule, and the rule expression is not continuous and differentiable, see “Using Events to Address Discontinuities in Rule and Reaction Rate Expressions” before simulating your model.

---

### Method Summary

Methods for rule objects

|                      |   |
|----------------------|---|
| copyobj (any object) | Copy SimBiology object and its children |
| delete (any object)  | Delete SimBiology object                |
| display (any object) | Display summary of SimBiology object    |
| get (any object)     | Get object properties                   |
| set (any object)     | Set object properties                   |

## Property Summary

Properties for rule objects

|          |  |
|----------|--|
| Active   | Indicate object in use during simulation   |
| Name     | Specify name of object                     |
| Notes    | HTML text describing SimBiology object     |
| Parent   | Indicate parent object                     |
| Rule     | Specify species and parameter interactions |
| RuleType | Specify type of rule for rule object       |
| Tag      | Specify label for SimBiology object        |
| Type     | Display SimBiology object type             |
| UserData | Specify data to associate with object      |

## Examples

Add a rule with the default RuleType.

- 1 Create a model object, and then add a rule object.

```
modelObj = sbiomodel('cell');  
ruleObj = addrule(modelObj, '0.1*B-A')
```

- 2 Get a list of properties for a rule object.

```
get(modelObj.Rules(1)) or get(ruleObj)
```

MATLAB displays a list of rule properties.

```
Active: 1  
Annotation: ''  
Name: ''
```

## addrule (model)

---

```
Notes: ''
Parent: [1x1 SimBiology.Model]
Rule: '0.1*B-A'
RuleType: 'algebraic'
Tag: ''
Type: 'rule'
UserData: []
```

Add a rule with the RuleType property set to rate.

- 1 Create model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
reactionObj = addreaction (modelObj, 'a -> b');
```

- 2 Add a rule which defines that the quantity of a species c. In the rule expression, k is the rate constant for a -> b.

```
ruleObj = addrule(modelObj, 'c = k*(a+b)')
```

- 3 Change the RuleType from default ('algebraic') to 'rate', and verify using the get command.

```
set(ruleObj, 'RuleType', 'rate');
get(ruleObj)
```

MATLAB returns all the properties for the rule object.

```
Active: 1
Annotation: ''
Name: ''
Notes: ''
Parent: [1x1 SimBiology.Model]
Rule: 'c = k*(a+b)'
RuleType: 'rate'
Tag: ''
Type: 'rule'
UserData: []
```

**See Also**      `copyobj`, `delete`, `sbiomodel`

# addspecies (model, compartment)

---

**Purpose** Create species object and add to compartment object within model object

**Syntax**

```
speciesObj = addspecies(compObj, 'NameValue')
speciesObj = addspecies(compObj, 'NameValue', InitialAmountValue)
speciesObj = addspecies(modelObj, 'NameValue')
speciesObj = addspecies(modelObj, 'NameValue',
InitialAmountValue)
speciesObj = addspecies(...'PropertyName', PropertyValue...)
```

**Arguments**

|                           |  |
|---------------------------|--|
| <i>compObj</i>            | Compartment object.  |
| <i>modelObj</i>           | Model object containing zero or one compartment.   |
| <i>NameValue</i>          | Name for a species object. Enter a character string unique among species within <i>modelObj</i> or <i>compObj</i> . Species objects are identified by name within Event, ReactionRate, and Rule property strings. For information on naming species, see Name.<br><br>You can use the function <code>sbiomselect</code> to find an object with a specific Name property value. |
| <i>InitialAmountValue</i> | Initial amount value for the species object. Enter double. Positive real number, default = 0.  |
| <i>PropertyName</i>       | Enter the name of a valid property. Valid property names are listed in “Property Summary” on page 2-65.  |
| <i>PropertyValue</i>      | Enter the value for the property specified in <i>PropertyName</i> . Valid property values are listed on each property reference page.  |



# addspecies (model, compartment)

---

## Description

`speciesObj = addspecies(compObj, 'NameValue')` creates `speciesObj`, a species object, and adds it to `compObj`, a compartment object. In the species object, this method assigns `NameValue` to the Name property, assigns `compObj` to the Parent property, and assigns 0 to the InitialAmount property. In the compartment object, this method adds the species object to the Species property.

`speciesObj = addspecies(compObj, 'NameValue', InitialAmountValue)`, in addition to the above, assigns `InitialAmountValue` to the InitialAmount property for the species object.

`speciesObj = addspecies(modelObj, 'NameValue')` creates `speciesObj`, a species object, and adds it to `compObj`, the compartment object in `modelObj`, a model object. If `modelObj` does not contain any compartments, it creates `compObj` with a Name property of 'unnamed'. In the species object, this method assigns `NameValue` to the Name property, assigns `compObj` to the Parent property, and assigns 0 to the InitialAmount property. In the compartment object, this method adds the species object to the Species property.

`speciesObj = addspecies(modelObj, 'NameValue', InitialAmountValue)`, in addition to the above, assigns `InitialAmountValue` to the InitialAmount property for the species object.

You can also add a species to a reaction using the methods `addreactant` and `addproduct`.

A species object must have a unique name at the level at which it is created. For example, a compartment object cannot contain two species objects named H2O. However, another compartment can have a species named H2O.

View properties for a species object with the `get` command, and modify properties for a species object with the `set` command. You can view a summary table of species objects in a compartment (`compObj`) with `get(compObj, 'Species')` or the properties of the first species with `get(compObj.Species(1))`.

# addspecies (model, compartment)

---

`speciesObj = addspecies(...'PropertyName', PropertyValue...)` defines optional properties. The property name/property value pairs can be in any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs). The property summary on this page shows the list of properties.

If there is more than one compartment object (`compObj`) in the model, you must qualify the species name with the compartment name. For example, `cell.glucose` denotes that you want to put the species named `glucose` into a compartment named `cell`. Additionally, if the compartment named `cell` does not exist, the process of adding the reaction creates the compartment and names it `cell`.

If you change the name of a species you must configure all applicable elements, such as events and rules that use the species, any user-specified `ReactionRate`, or the kinetic law object property `SpeciesVariableNames`. Use the method `setspecies` to configure `SpeciesVariableNames`.

To update species names in the SimBiology graphical user interface, access each appropriate pane through the **Project Explorer**. You can also use the **Find** feature to locate the names that you want to update. The **Output** pane opens with the results of **Find**. Double-click a result row to go to the location of the model component.

Species names are automatically updated for reactions that use `MassAction` kinetic law.

## Method Summary

Methods for species objects

|                                   |   |
|-----------------------------------|---|
| <code>copyobj</code> (any object) | Copy SimBiology object and its children |
| <code>delete</code> (any object)  | Delete SimBiology object                |
| <code>display</code> (any object) | Display summary of SimBiology object    |
| <code>get</code> (any object)     | Get object properties                   |

# addspecies (model, compartment)

---

|  |                                      |
|--|--------------------------------------|
| rename (compartment, parameter, species, reaction) | Rename object and update expressions |
| set (any object)                                   | Set object properties                |

## Property Summary

Properties for species objects

|                    |   |
|--------------------|---|
| BoundaryCondition  | Indicate species boundary condition         |
| ConstantAmount     | Specify variable or constant species amount |
| InitialAmount      | Species initial amount                      |
| InitialAmountUnits | Species initial amount units                |
| Name               | Specify name of object                      |
| Notes              | HTML text describing SimBiology object      |
| Parent             | Indicate parent object                      |
| Tag                | Specify label for SimBiology object         |
| Type               | Display SimBiology object type              |
| UserData           | Specify data to associate with object       |

## Examples

Add two species to a model, where one is a reactant and the other is the enzyme catalyzing the reaction.

- 1 Create a model object named `my_model` and add a compartment object.

```
modelObj = sbiomodel ('my_model');  
compObj = addcompartment(modelObj, 'comp1');
```

## addspecies (model, compartment)

---

- 2** Add two species objects named `glucose_6_phosphate` and `glucose_6_phosphate_dehydrogenase`.

```
speciesObj1 = addspecies (compObj, 'glucose_6_phosphate');
speciesObj2 = addspecies (compObj, ...
                        'glucose_6_phosphate_dehydrogenase');
```

- 3** Set the initial amount of `glucose_6_phosphate` to 100 and verify.

```
set (speciesObj1, 'InitialAmount',100);
get (speciesObj1, 'InitialAmount')
```

MATLAB returns:

```
ans =
    100
```

- 4** Use `get` to note that `modelObj` contains the species object array.

```
get(compObj, 'Species')
```

MATLAB returns:

SimBiology Species Array

| Index: | Name:                             | InitialAmount: | InitialAmountUnits: |
|--------|-----------------------------------|----------------|---------------------|
| 1      | glucose_6_phosphate               | 100            |                     |
| 2      | glucose_6_phosphate_dehydrogenase | 0              |                     |

- 5** Retrieve information about the first species in the array.

```
get(compObj.Species(1))
    Annotation: ''
    BoundaryCondition: 0
    ConstantAmount: 0
    InitialAmount: 100
    InitialAmountUnits: ''
    Name: 'glucose_6_phosphate'
```

## addspecies (model, compartment)

---

```
Notes: ''
Parent: [1x1 SimBiology.Compartment]
Tag: ''
Type: 'species'
UserData: []
```

**See Also**      addcompartment, addproduct, addreactant, addreaction, get, set

# addvariant (model)

---

**Purpose** Add variant to model

**Syntax**  
`variantObj = addvariant(modelObj, 'NameValue')`  
`variantObj2 = addvariant(modelObj, variantObj)`

## Arguments

|                   |  |
|-------------------|--|
| <i>modelObj</i>   | Specify the model object to which you want add a variant.  |
| <i>variantObj</i> | Variant object to create and add to the model object.  |
| <i>NameValue</i>  | Name of the variant object. <i>NameValue</i> is assigned to the Name property of the variant object. |

## Description

`variantObj = addvariant(modelObj, 'NameValue')` creates a SimBiology variant object (*variantObj*) with the name *NameValue* and adds the variant object to the SimBiology model object *modelObj*. The variant object Parent property is assigned the value of *modelObj*.

A SimBiology variant object stores alternate values for properties on a SimBiology model. For more information on variants, see [Variant object](#).

`variantObj2 = addvariant(modelObj, variantObj)` adds a SimBiology variant object (*variantObj*) to the SimBiology model object and returns another variant object *variantObj2*. The variant object *variantObj2* Parent property is assigned the value of *modelObj*.

View properties for a variant object with the `get` command, and modify properties for a variant object with the `set` command.

---

**Note** Remember to use the `addcontent` method instead of using the `set` method on the Content property, because the `set` method replaces the data in the Content property, whereas `addcontent` appends the data.

---

To view the variants stored on a model object, use the `getvariant` method. To copy a variant object to another model, use `copyobj`. To remove a variant object from a SimBiology model, use the `delete` method.

### Examples

- 1 Create a model containing one species.

```
modelObj = sbiomodel('mymodel');  
compObj = addcompartment(modelObj, 'comp1');  
speciesObj = addspecies(compObj, 'A');
```

- 2 Add a variant object that varies the `InitialAmount` property of a species named A.

```
variantObj = addvariant(modelObj, 'v1');  
addcontent(variantObj, {'species', 'A', 'InitialAmount', 5});
```

### See Also

`addcontent`, `commit`, `copyobj`, `delete`, `getvariant`

# boxplot(LeastSquaresResults,OptimResults,NLINResults)

---

**Purpose** Create box plot showing the variation of estimated SimBiology model parameters

**Syntax** `boxplot(resultsObj)`

**Description** `boxplot(resultsObj)` creates a box plot showing the variation of the estimated SimBiology model parameters.

**Input Arguments** **resultsObj - Estimation results**  
`OptimResults` object | `NLINResults` object | vector of results objects  
Estimation results, specified as an `OptimResults` object or `NLINResults` object, or vector of results objects which contains estimation results from running `sbiofit`.

**See Also** `NLINResults` object | `OptimResults` object | `sbiofit`



|                        |  |
|------------------------|--|
| <b>Purpose</b>         | Create box plot showing the variation of estimated SimBiology model parameters   |
| <b>Syntax</b>          | <code>boxplot(resultsObj)</code>   |
| <b>Description</b>     | <code>boxplot(resultsObj)</code> creates a box plot showing the variation of the estimated SimBiology model parameters.  |
| <b>Input Arguments</b> | <b>resultsObj - Estimation results</b><br>NLMEResults object<br><br>Estimation results, specified as an NLMEResults object, which contains estimation results from running <code>sbiofitmixed</code> . |
| <b>See Also</b>        | NLMEResults object   <code>sbiofitmixed</code>   |

# commit (variant)

---

**Purpose** Commit variant contents to model

**Syntax** `commit(variantObj, modelObj)`

## Arguments

|                   |   |
|-------------------|---|
| <i>modelObj</i>   | Specify the model object to which you want to commit a variant. |
| <i>variantObj</i> | Variant object to commit to the model object.                   |

## Description

`commit(variantObj, modelObj)` commits the `Contents` property of a SimBiology variant object (*variantObj*) to the model object *modelObj*. The property values stored in the variant object replace the values stored in the model.

A SimBiology variant object stores alternate values for properties on a SimBiology model. For more information on variants, see `Variant` object.

The `Contents` are set on the model object in order of occurrence, with duplicate entries overwriting. If the `commit` method finds an incorrectly specified entry, an error occurs and the remaining properties defined in the `Contents` property are not set.

## Examples

- 1 Create a model containing one species.

```
modelObj = sbiomodel('mymodel');  
compObj = addcompartment(modelObj, 'comp1');  
speciesObj = addspecies(compObj, 'A', 10);
```

- 2 Add a variant object that varies the `InitialAmount` property of a species named A.

```
variantObj = addvariant(modelObj, 'v1');  
addcontent(variantObj, {'species', 'A', 'InitialAmount', 5});
```

- 3 Commit the contents of the variant (*variantObj*).

```
commit (variantObj, modelObj);
```

**See Also**      addvariant, Variant object

# Compartment object

---

## Purpose

Options for compartments

## Description

The SimBiology compartment object represents a container for species in a model. Compartment size can vary or remain constant during a simulation. All models must have at least one compartment and all species in a model must be assigned to a compartment. Compartment names must be unique within a model.

Compartments allow you to define the size (**Capacity**) of physically isolated regions that may affect simulation, and associate pools of species within those regions. You can specify or change **Capacity** using rules, events, and variants, similar to species amounts or parameter values.

The model object stores compartments as a flat list. Each compartment stores information on its own organization; in other words a compartment has information on which compartment it lives within (**Owner**) and who it contains (**Compartments**).

The flat list of compartments in the model object lets you vary the way compartments are organized in your model without invalidating any expressions.

To add species that participate in reactions, add the reaction to the model using the `addreaction` method. When you define a reaction with a new species:

- If no compartment objects exist in the model, the `addreaction` method creates a compartment object (called '*unnamed*') in the model and adds the newly created species to that compartment.
- If only one compartment object exists in the model, the method creates a species object in that compartment.
- If there is more than one compartment object in the model, you must qualify the species name with the compartment name.

For example, `cell.glucose` denotes that you want to put the species named `glucose` into a compartment named `cell`. Additionally, if the compartment named `cell` does not exist, the process of adding the reaction creates the compartment and names it `cell`.

Alternatively, create and add a species object to a compartment object, using the `addspecies` method at the command line.

When you use the SimBiology desktop to create a new model, it adds an empty compartment (*unnamed*), to which you can add species.

You can specify reactions that cross compartments using the syntax `compartment1Name.species1Name -> compartment2Name.species2Name`. If you add a reaction that contains species from different compartments, and the reaction rate dimensions are concentration/time, all reactants should be from the same compartment.

In addition, if the reaction is reversible then there are two cases:

- If the kinetic law is `MassAction`, and the reaction rate dimensions are concentration/time, then the products must be from the same compartment.
- If the kinetic law is not `MassAction`, then both reactants and products must be in the same compartment.

See “Property Summary” on page 2-76 for links to compartment property reference pages. Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the graphical user interface.

## Constructor Summary

`addcompartment (model, compartment)`

Create compartment object

# Compartment object

---

## Method Summary

|  |   |
|--|---|
| Methods for compartment objects                    |   |
| addcompartment (model, compartment)                | Create compartment object   |
| addspecies (model, compartment)                    | Create species object and add to compartment object within model object |
| copyobj (any object)                               | Copy SimBiology object and its children                                 |
| delete (any object)                                | Delete SimBiology object  |
| display (any object)                               | Display summary of SimBiology object                                    |
| get (any object)                                   | Get object properties   |
| rename (compartment, parameter, species, reaction) | Rename object and update expressions                                    |
| reorder (model, compartment, kinetic law)          | Reorder component lists   |
| set (any object)                                   | Set object properties   |

## Property Summary

|                                    |   |
|------------------------------------|---|
| Properties for compartment objects |   |
| Capacity                           | Compartment capacity                              |
| CapacityUnits                      | Compartment capacity units                        |
| Compartments                       | Array of compartments in model or compartment     |
| ConstantCapacity                   | Specify variable or constant compartment capacity |
| Name                               | Specify name of object                            |
| Notes                              | HTML text describing SimBiology object            |

|          |  |
|----------|--|
| Owner    | Owning compartment                     |
| Parent   | Indicate parent object                 |
| Species  | Array of species in compartment object |
| Tag      | Specify label for SimBiology object    |
| Type     | Display SimBiology object type         |
| UserData | Specify data to associate with object  |

## See Also

AbstractKineticLaw object, Configset object, KineticLaw object, Model object, Parameter object, Reaction object, Root object, Rule object

# Configset object

---

**Purpose** Solver settings information for model simulation

**Description** The SimBiology configset object, also known as the configuration set object, contains the options that the solver uses during simulation of the model object. The configuration set object contains the following options for you to choose:

- Type of solver
- Stop time for the simulation
- Solver error tolerances, and for ode solvers — the maximum time step the solver should take
- Whether to perform sensitivity analysis during simulation
- Whether to perform dimensional analysis and unit conversion during simulation
- Species and parameter input factors for sensitivity analysis

A SimBiology model can contain multiple configsets with one being active at any given time. The active configset contains the settings that are used during the simulation. Use the method `setactiveconfigset` to define the active configset. Use the method `getconfigset` to return a list of configsets contained by a model. Use the method `addconfigset` to add a new configset to a model.

See “Property Summary” on page 2-79 for links to configset object property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the SimBiology desktop.

## Constructor Summary

`addconfigset (model)`

Create configuration set object and add to model object



## Method Summary

|                      |   |
|----------------------|---|
| copyobj (any object) | Copy SimBiology object and its children |
| delete (any object)  | Delete SimBiology object                |
| display (any object) | Display summary of SimBiology object    |
| set (any object)     | Set object properties                   |

## Property Summary

|                            |  |
|----------------------------|--|
| Active                     | Indicate object in use during simulation           |
| CompileOptions             | Dimensional analysis and unit conversion options   |
| MaximumNumberOfLogs        | Maximum number of logs criteria to stop simulation |
| MaximumWallClock           | Maximum elapsed wall clock time to stop simulation |
| Name                       | Specify name of object                             |
| Notes                      | HTML text describing SimBiology object             |
| RuntimeOptions             | Options for logged species                         |
| SensitivityAnalysisOptions | Specify sensitivity analysis options               |
| SolverOptions              | Specify model solver options                       |
| SolverType                 | Select solver type for simulation                  |
| StartTime                  | Start time for initial dose time                   |
| StopTime                   | Simulation time criteria to stop simulation        |

# Configset object

---

|           |   |
|-----------|---|
| TimeUnits | Show time units for dosing and simulation |
| Type      | Display SimBiology object type            |

## See Also

AbstractKineticLaw object, KineticLaw object, Model object, Parameter object, Reaction object, Root object, Rule object, Species object

## Purpose

Construct SimBiology model from PKModelDesign object

## Syntax

```
[modelObj, pkModelMapObject] =  
construct(pkModelDesignObject)  
[modelObj, pkModelMapObject,  
  CovModelObj] = construct(pkModelDesignObject)
```

## Arguments

|                         |   |
|-------------------------|---|
| <i>modelObj</i>         | SimBiology model object specifying a pharmacokinetic model.   |
| <i>pkModelMapObject</i> | Defines the roles of the components in <i>modelObj</i> . For details, see PKModelMap object.        |
| <i>CovModelObj</i>      | Defines the relationship between parameters and covariates. For details, see CovariateModel object. |

## Description

[*modelObj*, *pkModelMapObject*] = construct(*pkModelDesignObject*) constructs a SimBiology model object, *modelObj*, containing the model components (such as compartments, species, reactions, and rules) required to represent the pharmacokinetic model specified in *pkModelDesignObject*. It also constructs *pkModelMapObject*, a PKModelMap object, which defines the roles of the model components.

The newly constructed model object, *modelObj*, is named 'Generated Model' (which you can change). It contains one compartment for each compartment specified in the PKCompartment property of *pkModelDesignObject*. Each compartment contains a species that represents a drug concentration. The compartments are connected with reversible reactions that models flux between compartments.

[*modelObj*, *pkModelMapObject*, *CovModelObj*] = construct(*pkModelDesignObject*) constructs *CovModelObj*, a CovariateModel object, which defines the relationship between parameters and covariates. Within the Expression property of

## construct (PKModelDesign)

---

*CovModelObj*, each parameter being estimated has an expression of the form  $\text{parameterName} = \exp(\text{theta1} + \text{eta1})$  (without covariate dependencies), where *theta1* is a fixed effect, and *eta1* is a random effect. You can modify the expressions to add covariate dependencies. For details, see *CovariateModel* object.

### See Also

*PKModelDesign* object | *PKModelMap* object | *CovariateModel* object

### How To

- “Create a Pharmacokinetic Model Using the Command Line”
- Modeling the Population Pharmacokinetics of Phenobarbital in Neonates
- “Specify a Covariate Model”

# constructDefaultFixedEffectValues (covmodel)

---

**Purpose** Create initial estimate vector needed for fit

**Syntax** `FEInitEstimates =  
constructDefaultFixedEffectValues(CovModelObj)`

**Description** `FEInitEstimates =  
constructDefaultFixedEffectValues(CovModelObj)` creates `FEInitEstimates`, a structure containing the initial estimates for the fixed effects in `CovModelObj`, a `CovariateModel` object. These initial estimates are set to a default of zero, but you can edit these estimates. The number and names of the fields in the `FEInitEstimates` structure matches the number and names of fixed effects (theta values) in the `Expression` property of `CovModelObj`.

---

**Tip** After creating the `FEInitEstimates` structure, you can edit it and use it to change the `FixedEffectValues` property of `CovModelObj`, before using the object as an input argument to `sbionlmeFit` or `sbionlmeFitSA`.

---

**See Also** `CovariateModel` | `Expression` | `FixedEffectValues` | `sbionlmeFit` | `sbionlmeFitSA`

**How To**

- Modeling the Population Pharmacokinetics of Phenobarbital in Neonates
- “Specify a Covariate Model”

# copyobj (any object)

---

**Purpose** Copy SimBiology object and its children

**Syntax**  
`copiedObj = copyobj(Obj, parentObj)`  
`copiedObj = copyobj(modelObj)`

## Arguments

*Obj* Abstract kinetic law, compartment, configuration set, event, kinetic law, model, parameter, reaction, rule, species, or variant object.

*parentObj*

| If <i>copiedObj</i> is...                                   | <i>parentObj</i> must be... |
|---|-----------------------------|
| configuration set, event, reaction, rule, or variant object | model object                |
| compartment object  | compartment or model object |
| species object  | compartment object          |
| parameter object  | model or kinetic law object |
| kinetic law object  | reaction object             |
| model object or abstract kinetic law object                 | sbioroot                    |

*modelObj* Model object to be copied.

*copiedObj* Output returned by the copyobj method with the parent set as specified in input argument (*parentObj*).

## Description

`copiedObj = copyobj(Obj, parentObj)` makes a copy of a SimBiology object (*Obj*) and returns a pointer to the copy (*copiedObj*). In the copied object (*copiedObj*), this method assigns a value (*parentObj*) to the property Parent.

`copiedObj = copyobj(modelObj)` makes a copy of a model object (`modelObj`) and returns the copy (`copiedObj`). In the copied model object (`copiedObj`), this method assigns the root object to the property `Parent`.

---

**Note** When the `copyobj` method copies a model, it resets the `StatesToLog` property to the default value. Similarly, the `Inputs` and `Outputs` properties are not copied but rather left empty. Thus, when you simulate a copied model you see results for the default states, unless you manually update these properties.

---

## Examples

Create a reaction object separate from a model object, and then add it to a model.

- 1 Create a model object and add a reaction object.

```
modelObj1 = sbiomodel('cell1');  
reactionObj = addreaction(modelObj1, 'a -> b');
```

- 2 Create a copy of the reaction object and assign it to another model object.

```
modelObj2 = sbiomodel('cell2');  
reactionObjCopy = copyobj(reactionObj, modelObj2);  
modelObj2.Reactions
```

SimBiology Reaction Array

| Index: | Reaction: |
|--------|-----------|
| 1      | a -> b    |

## See Also

`sbiomodel`, `sbioroot`

# CovariateModel object

---

**Purpose** Define relationship between parameters and covariates

**Description** CovariateModel defines the relationship between estimated parameters and covariates.

---

**Tip** Use a CovariateModel object as an input argument to `sbionlmeFit` or `sbionlmeFitsa` to fit a model with covariate dependencies. Before using the CovariateModel object with either fitting function, set the `FixedEffectValues` property to specify the initial estimates for the fixed effects.

---

**Construction** `CovModelObj = CovariateModel` creates an empty CovariateModel object.

`CovModelObj = CovariateModel(Expression)` creates a CovariateModel object with its `Expression` property set to `Expression`, a string or cell array of strings, where each string represents the relationship between a parameter being estimated and one or more covariates. `Expression` must denote fixed effects with the prefix `theta`, and random effects with the prefix `eta`. Each string in `Expression` must be in the form:

```
parameterName = relationship
```

This example of an expression string defines the relationship between a parameter (`volume`) and a covariate (`weight`), with fixed effects, but no random effects:

```
Expression = {'volume = theta1 + theta2*weight'};
```

This table illustrates expression formats for some common parameter-covariate relationships.



| Parameter-Covariate Relationship                                  | Expression Format   |
|---|---|
| Linear with random effect   | $C1 = \text{theta1} + \text{theta2} * \text{WEIGHT} + \text{eta1}$  |
| Exponential without random effect                                 | $C1 = \exp(\text{theta\_C1} + \text{theta\_C1\_WT} * \text{WEIGHT})$  |
| Exponential, WEIGHT centered by mean, has random effect           | $C1 = \exp(\text{theta1} + \text{theta2} * (\text{WEIGHT} - \text{mean}(\text{WEIGHT})) + \text{eta1})$           |
| Exponential, log(WEIGHT), which is equivalent to power model      | $C1 = \exp(\text{theta1} + \text{theta2} * \log(\text{WEIGHT}) + \text{eta1})$                                    |
| Exponential, dependent on WEIGHT and AGE, has random effect       | $C1 = \exp(\text{theta1} + \text{theta2} * \text{WEIGHT} + \text{theta3} * \text{AGE} + \text{eta1})$             |
| Inverse of probit, dependent on WEIGHT and AGE, has random effect | $C1 = \text{probitinv}(\text{theta1} + \text{theta2} * \text{WEIGHT} + \text{theta3} * \text{AGE} + \text{eta1})$ |
| Inverse of logit, dependent on WEIGHT and AGE, has random effect  | $C1 = \text{logitinv}(\text{theta1} + \text{theta2} * \text{WEIGHT} + \text{theta3} * \text{AGE} + \text{eta1})$  |

**Tip** To simultaneously fit data from multiple dose levels, use a CovariateModel object as an input argument to `sbionlmeFit`, and omit the random effect (eta) from the Expression property in the CovariateModel object.

# CovariateModel object

---

---

**Tip** Use the `getCovariateData` method of a `PKData` object to view the covariate data when writing equations for the *Expression* input argument.

---

---

**Note** You can also construct a `CovariateModel` object using the `construct` method of a `PKModelDesign` object. However, the `Expression` property of the `CovariateModel` object does not include covariate dependencies. You can modify the expressions to add covariate dependencies. For details, see `Expression`.

---

## Method Summary

|   |   |
|---|---|
| <code>constructDefaultFixedEffectValues (covmodel)</code> | Create initial estimate vector needed for fit |
| <code>verify (covmodel)</code>                            | Check covariate model for errors              |

## Properties

|  |   |
|--|---|
| <code>CovariateLabels (CovariateModel)</code>        | Labels for covariates in <code>CovariateModel</code> object                         |
| <code>Expression (CovariateModel)</code>             | Define relationship between parameters and covariates                               |
| <code>FixedEffectDescription (CovariateModel)</code> | Descriptions of fixed effects in <code>CovariateModel</code> object                 |
| <code>FixedEffectNames (CovariateModel)</code>       | Names of fixed effects in <code>CovariateModel</code> object                        |
| <code>FixedEffectValues (CovariateModel)</code>      | Values for initial estimates of fixed effects in <code>CovariateModel</code> object |

|                                       |   |
|---------------------------------------|---|
| ParameterNames<br>(CovariateModel)    | Names of parameters in<br>CovariateModel object     |
| RandomEffectNames<br>(CovariateModel) | Names of random effects in<br>CovariateModel object |

## Examples

Create a `CovariateModel` object and set the `Expression` property to define the relationship between two parameters (clearance and volume) and two covariates (weight and age) using fixed effects (thetas) and random effects (etas):

```
covModelObj = CovariateModel  
covModelObj.Expression = {'CL = theta1 + theta2*WT + eta1', 'V = theta3 + eta2'}
```

## See Also

[construct](#) | [getCovariateData](#) | [PKData](#) object | [PKModelDesign](#) object | [sbionlmefit](#) | [sbionlmefitsa](#)

## How To

- [Modeling the Population Pharmacokinetics of Phenobarbital in Neonates](#)
- [“Specify a Covariate Model”](#)

# covariateModel(NLMEResults)

---

**Purpose** Return a copy of the covariate model that was used for the nonlinear mixed-effects estimation using `sbiofitmixed`

**Syntax** `covmodel = covariateModel(resultsObj)`

**Description** `covmodel = covariateModel(resultsObj)` returns a copy of the covariate model that was used for the nonlinear mixed-effects estimation using `sbiofitmixed`.

**Input Arguments** **resultsObj - Estimation results**  
NLMEResults object

Estimation results, specified as an NLMEResults object, which contains estimation results from running `sbiofitmixed`.

**Output Arguments** **covmodel - Covariate model**  
CovariateModel object

Covariate model, returned as a CovariateModel object, that was used for the nonlinear mixed-effects estimation using `sbiofitmixed`. The model describes the relationship between SimBiology model parameters, fixed effects, random effects, and covariates.

**See Also** NLMEResults object | `sbiofitmixed`

## Purpose

Create dose objects from groupedData object

## Syntax

```
doseArray = createDoses(grpData, amountVarNames)
doseArray =
createDoses(grpData, amountVarNames, rateVarNames)
doseArray =
createDoses(grpData, amountVarNames, rateVarNames,
tempDoses)
```

## Description

`doseArray = createDoses(grpData, amountVarNames)` creates an array of SimBiology dose objects using dose times and amount data specified in `grpData`, with one row per group and one column per dose amount variable.

`grpData.Properties.IndependentVariable` specifies which variable contains dose times, and `amountVarNames` specifies which variables contain valid dose amounts.

`doseArray = createDoses(grpData, amountVarNames, rateVarNames)` uses dose rate variables specified by `rateVarNames`.

`doseArray = createDoses(grpData, amountVarNames, rateVarNames, tempDoses)` uses template doses specified by `tempDoses` as templates for dose objects in `doseArray`. The string (nonnumeric) properties of template dose objects in `tempDoses` are copied into the dose objects in `doseArray`. This argument lets you initialize string properties of dose objects such as `Name`, `TargetName`, `AmountUnits`, and `TimeUnits`.

## Input Arguments

### **grpData - Grouped data**

`groupedData` object

Grouped data, specified as a `groupedData` object.

### **amountVarNames - Amount variable names**

## createDoses(groupedData)

---

string | cell array of strings

Amount variable names, specified as a string or a cell array of strings that specifies variables in `grpData` that define dose amounts. Each string must specify a valid amount variable.

An amount variable is valid if it is a real, nonnegative column vector containing no infinite values.

### **rateVarNames - Rate variable names**

string | cell array of strings

Rate variable names, specified as a string or a cell array of strings that specify variables in `grpData` that define dose rates. If it is empty `[]` or `{}`, it indicates there are no dose rates. If it is not empty, it must be a string or cell array of strings of the same length as `amountVarNames`. Individual strings must be empty strings `''` indicating no dose rates for the corresponding doses or valid rate variables in `grpData` specifying dose rates.

A rate variable is valid if it is a real, nonnegative column vector containing no infinite values. NaN rate values are allowed, but they are treated the same as the rate values of 0.

### **tempDoses - Template doses**

dose object | vector of dose object | []

Template doses, specified as a dose object (`ScheduleDose` object or `RepeatDose` object), vector of dose objects, or empty array `[]`. If `tempDoses` is a single dose object, the nonnumeric properties of this object are copied to all doses in `doseArray`. If it is a vector, it must have the same length as `amountVarNames`, and the nonnumeric properties from each element are copied to the corresponding column of doses in `doseArray`.

## **Output Arguments**

### **doseArray - SimBiology dose objects**

2-D matrix of dose objects

SimBiology dose objects, returned as a 2-D matrix of dose objects containing dose time and amount data from `grpData`. If dose times

for a particular dose in `grpData` are regularly spaced, then the corresponding dose object in `doseArray` is a `RepeatDose` object. Otherwise, the corresponding dose object is a `ScheduleDose` object.

---

**Note** If there is a single dose time, then the dose object is represented as a `ScheduleDose` object.

---

## Examples

### Create an Array of Dose Objects from GroupedData

This example uses data collected on 59 preterm infants given phenobarbital during the first 16 days after birth. Each infant received an initial dose followed by one or more sustaining doses by intravenous bolus administration. A total of between 1 and 6 concentration measurements were obtained from each infant at times other than dose times, for a total of 155 measurements. Infant weights and APGAR scores (a measure of newborn health) were also recorded. Data is described in [1], a study funded by the NIH/NIBIB grant P41-EB01975.

Load the sample data set.

```
load pheno.mat ds
```

Create a `groupedData` object from the data set `ds`.

```
grpData = groupedData(ds);
```

Display the properties.

```
grpData.Properties
```

```
ans =
```

```
    Description: 'This dataset was generated using sbionm
```

```
    FileName...'
```

```
    VariableDescriptions: {}
```

## createDoses(groupedData)

---

```
VariableUnits: {}  
DimensionNames: {'Observations' 'Variables'}  
  UserData: []  
  RowNames: {}  
VariableNames: {'ID' 'Time' 'Dose' 'Response'}  
GroupVariableName: 'ID'  
IndependentVariableName: 'Time'
```

GroupVariableName and IndependentVariableName have been automatically assigned to 'ID' and 'Time', respectively.

Create an array of dose objects using the dosing information from grpData specified by the DOSE variable.

```
doseArray = createDoses(grpData, 'DOSE');
```

## References

[1] Grasela Jr, T.H., Donn, S.M. (1985) Neonatal population pharmacokinetics of phenobarbital derived from routine clinical data. *Dev Pharmacol Ther.* 8(6), 374–83.

## See Also

table | groupedData object | ScheduleDose object | RepeatDose object



## Purpose

Create SimFunction object

## Syntax

```
F = createSimFunction(model,params,observables,dosed)
F =
createSimFunction(model,params,observables,dosed,'UseParallel',
tf)
```

## Arguments

|             |   |
|-------------|---|
| F           | SimFunction object that can be executed like a function handle  |
| model       | SimBiology model object   |
| params      | Cell array of strings containing names of model quantities (species, compartments, or parameters) that defines the inputs of F. To unambiguously name a model quantity, use the qualified name, which includes the name of the compartment. To name a reaction-scoped parameter, use the reaction name to qualify the parameter. If the name is not a valid MATLAB variable name, surround it by square brackets such as [reaction 1].[parameter 1] |
| observables | Cell array of strings containing names of model quantities (species, compartments, or parameters) that defines the output of F. The name of the model quantity must be qualified by including the name of the compartment if it is not scoped to the model.   |
| dosed       | <ul style="list-style-type: none"><li>• A string or cell array of strings with names of one or more species that are being dosed</li></ul>  |
| tf          | <ul style="list-style-type: none"><li>• true or false. If true, the function dose executes the function F in parallel after it is created if Parallel Computing Toolbox is available.</li><li>• 1-by-N vector of dose objects (ScheduleDose object or RepeatDose object), where N is the number of species being dosed</li></ul>  |

## Description

`F = createSimFunction(model,params,observables,dosed)` creates a SimFunction object F for time-varying inputs. The `params` and `observables` arguments define the inputs and outputs of the function F. The `dosed` argument finds the `LagParameterName` properties of each dose object are used.

- An empty array [].

## createSimFunction (model)

---

dosing information of species. See `SimFunction` object for details on how to execute `F`.

```
F =  
createSimFunction(model,params,observables,dosed,'UseParallel',  
tf) provides an option to execute the function F in parallel if  
Parallel Computing Toolbox is available.
```

---

### Note

- Active doses and variants of the model are ignored when `F` is executed.
  - `F` is immutable after it is created.
- 

## Examples

### Create a SimFunction Object

This example uses a radioactive decay model with the first-order

reaction  $\frac{dz}{dt} = c \cdot x$ , where `x` and `z` are species and `c` is the forward rate constant.

Load the sample project containing the radioactive decay model `m1`.

```
sbioloadproject radiodecay;
```

Create a `SimFunction` object, specifying the parameter `Reaction1.c` to be scanned, and species `x` as the output of the function with no dosed species.

```
f = createSimFunction(m1, {'Reaction1.c'}, {'x'}, [])
```

```
f =
```

```
SimFunction
```

```
Parameters:
```

## createSimFunction (model)

---

| Name          | Value | Type        | Units      |
|---------------|-------|-------------|------------|
| 'Reaction1.c' | 0.5   | 'parameter' | '1/second' |

Observables:

| Name | Type      | Units      |
|------|-----------|------------|
| 'x'  | 'species' | 'molecule' |

Dosed: None

If the `UnitConversion` option was set to `false` when the `SimFunction` object `f` was created, the table does not display the units of the model quantities.

To illustrate this, first set the `UnitConversion` option to `false`.

```
m1.getConfigset.CompileOptions.UnitConversion = false;
```

Create the `SimFunction` object as before and note that the variable named `Units` disappears.

```
f = createSimFunction(m1, {'Reaction1.c'}, {'x'}, [])
```

```
f =
```

```
SimFunction
```

Parameters:

| Name          | Value | Type        |
|---------------|-------|-------------|
| 'Reaction1.c' | 0.5   | 'parameter' |

## createSimFunction (model)

---

Observables:

| Name | Type      |
|------|-----------|
| 'x'  | 'species' |

Dosed: None

If any of the species in the model is being dosed, specify the names of dosed species as the last argument. For example, if the species x is being dosed, specify it as the last argument.

```
f = createSimFunction(m1, {'Reaction1.c'}, {'x'}, 'x')
```

f =

SimFunction

Parameters:

| Name          | Value | Type        |
|---------------|-------|-------------|
| 'Reaction1.c' | 0.5   | 'parameter' |

Observables:

| Name | Type      |
|------|-----------|
| 'x'  | 'species' |

Dosed:

TargetName

'x'

Once the `SimFunction` object is created, you can execute it like a function handle and perform parameter scans (in parallel if Parallel Computing Toolbox is available), Monte Carlo simulations, and scans with multiple or vectorized doses. See `SimFunction` object for more examples.

### Create a SimFunction Object with Dosing Information

This example creates a `SimFunction` object with dosing information using a `RepeatDose` or `ScheduleDose` object or a vector of these objects. However, if any dose object contains data such as `StartTime`, `Amount`, and `Rate`, such data are ignored, and a warning is issued. Only data, if available, used are `TargetName`, `LagParameterName`, and `DurationParameterName` of the dose object.

Load the sample project containing the radioactive decay model `m1`.

```
sbioloadproject radiodecay;
```

Create a `RepeatDose` object and specify its properties.

```
rdose = sbiodose('rd');  
rdose.TargetName = 'x';  
rdose.StartTime = 5;  
rdose.TimeUnits = 'second';  
rdose.Amount = 300;  
rdose.AmountUnits = 'molecule';  
rdose.Rate = 1;  
rdose.RateUnits = 'molecule/second';  
rdose.Interval = 100;  
rdose.RepeatCount = 2;
```

Add a lag parameter and duration parameter to the model.

```
lagPara = addparameter(m1, 'lp');
```

## createSimFunction (model)

---

```
lagPara.Value = 1;
lagPara.ValueUnits = 'second';
duraPara = addparameter(m1,'dp');
duraPara.Value = 1;
duraPara.ValueUnits = 'second';
```

Set these parameters to the dose object.

```
rdose.LagParameterName = 'lp';
rdose.DurationParameterName = 'dp';
```

Create a `SimFunction` object `f` using the `RepeatDose` object `rdose` that you just created.

```
f = createSimFunction(m1,{'Reaction1.c'},{'x','z'},rdose)
```

Warning: Some Dose objects in DOSED had data. This data will be ignored.

```
> In SimFunction>SimFunction.SimFunction at 847
  In SimFunction>SimFunction.createSimFunction at 374
```

```
f =
```

```
SimFunction
```

```
Parameters:
```

| Name          | Value | Type        | Units      |
|---------------|-------|-------------|------------|
| 'Reaction1.c' | 0.5   | 'parameter' | '1/second' |

```
Observables:
```

| Name | Type      | Units      |
|------|-----------|------------|
| 'x'  | 'species' | 'molecule' |

## createSimFunction (model)

---

```
      'z'      'species'  'molecule'

Dosed:

      TargetName      TargetDimension
-----
      'x'      'Amount(e.g. mole or molecule)'
```

A warning message appears because the `rdose` object contains data (`StartTime`, `Amount`, `Rate`) that are ignored by the `createSimFunction` method.

### See Also

`SimFunction` object

# delete (any object)

---

**Purpose** Delete SimBiology object

**Syntax** `delete(Obj)`

**Arguments**

*Obj* abstract kinetic law, configuration set, event, kinetic law, model, parameter, reaction, rule, SimData, species, unit, unit prefix, or variant object.

**Description**

`delete(Obj)` removes an object (*Obj*) from its parent.

- If *Obj* is a model object, the model is deleted from the root object. `delete` removes all references to the model both at the command line and in the SimBiology desktop.
- If *Obj* is a species object that is being used by a reaction object, this method returns an error and the species object is not deleted. You need to delete the reaction or remove the species from the reaction before you can delete the species object.
- If *Obj* is a parameter object being used by a kinetic law object, there is no warning when the object is deleted. However, when you try to simulate your model, an error occurs because the parameter cannot be found.
- If *Obj* is a reaction object, this method deletes the object, but the species objects that were being used by the reaction object are not deleted.
- If *Obj* is an abstract kinetic law object and there is a kinetic law object referencing it, this method returns an error.
- If *Obj* is a SimBiology configuration set object, and it is the active configuration set object, this method, after deleting the object, makes the default configuration set object active. Note that you cannot delete the default configuration set.
- You cannot delete the SimBiology root.



You can also delete all model objects from the root with one call to the `sbioreset` function.

## Examples

### Example 1

Delete a reaction from a model. Notice the species objects are not deleted with the reaction object.

```
modelObj = sbiomodel('cell');  
reactionObj = addreaction(modelObj, 'a -> b');  
delete(reactionObj)
```

### Example 2

Delete a single model from the root object.

```
modelObj1 = sbiomodel('cell');  
modelObj2 = sbiomodel('virus');  
delete(modelObj2)
```

## See Also

`sbiomodel`, `sbioreset`, `sbioroot`

# display (any object)

---

**Purpose** Display summary of SimBiology object

**Syntax** `display(Obj)`

**Arguments**

*Obj* SimBiology object: abstract kinetic law, configuration set, compartment, event, kinetic law, model, parameter, reaction, rule, species, or unit.

**Description**

Display the SimBiology object array. `display(Obj)` is called for the SimBiology object, *Obj* when the semicolon is not used to terminate a statement. The display of *Obj* gives a brief summary of the *Obj* configuration. You can view a complete list of *Obj* properties with the command `get`. You can modify all *Obj* properties that can be changed, with the command `set`.

**Examples**

```
modelObj = sbiomodel('cell')
reactionObj = addreaction(modelObj, 'A + B -> C')
```

|                        |   |
|------------------------|---|
| <b>Purpose</b>         | Object containing information about estimated model quantities  |
| <b>Description</b>     | The <code>estimatedInfo</code> object contains information about estimated model quantities (species, parameters, or compartments). Use this object to specify which quantities in a SimBiology model are estimated, what parameter transforms are used, and optionally, the initial estimates for these quantities. Use this object to specify what quantities in a SimBiology model are estimated when using <code>sbiofit</code> or <code>sbiofitmixed</code> .  |
| <b>Construction</b>    | <pre>estimInfo = estimatedInfo</pre> creates an empty <code>estimatedInfo</code> object.<br><pre>estimInfoArray = estimatedInfo(transformedNames)</pre> creates an array of <code>estimatedInfo</code> objects for quantities specified in <code>transformedNames</code> . The initial values for these quantities are obtained from the SimBiology model when you run <code>sbiofit</code> or <code>sbiofitmixed</code> .<br><pre>estimInfoArray = estimatedInfo(transformedNames,<br/>'InitialTransformedValue',itValues)</pre> defines the initial transformed values of model quantities specified by <code>itValues</code> .<br><pre>estimInfoArray =<br/>estimatedInfo(transformedNames,'InitialValue',<br/>iValues)</pre> defines the transformed values of model quantities specified by <code>iValues</code> . |
| <b>Input Arguments</b> | <b>transformedNames - Names of estimated model quantities</b><br><code>string</code>   cell array of strings<br>Names of estimated model quantities, specified as a string or cell array of strings. Each string must be in one of these formats: <ul style="list-style-type: none"><li>• Name or qualified name of a model quantity, such as <code>'C1'</code>, <code>'Reaction1.k'</code>, <code>'[c 1].[r 1]'</code></li><li>• Name of a supported parameter transform (<code>log</code>, <code>logit</code>, or <code>probit</code>) followed by a quantity name in parenthesis, such as <code>'log(C1)'</code>, <code>'logit(Reaction1.k)'</code>, <code>'probit([c 1].[r 1])'</code></li></ul>  |

# EstimatedInfo object

---

## **iValues - Initial transformed values of model quantities**

vector | cell array

Initial transformed values of model quantities, specified as a vector or cell array. It must have the same length as `transformedNames`. If it is a cell array, each element of the cell must be a scalar or the empty matrix `[]`.

## **iValues - Initial values of model quantities**

vector | cell array

Initial values of model quantities, specified as a vector or cell array. It must have the same length as `transformedNames`. If it is a cell array, each element of the cell must be a scalar or the empty matrix `[]`.

## **Output Arguments**

### **estimInfo - Estimated model quantity**

`estimatedInfo` object

Estimated model quantity, returned as an `estimatedInfo` object.

### **estimInfoArray - Estimated model quantities**

`estimatedInfo` object | vector

Estimated model quantities, returned as an `estimatedInfo` object or vector of `estimatedInfo` objects. If `transformedNames` is a single string, `estimInfoArray` is a scalar `estimatedInfo` object. Otherwise, `estimInfoArray` is a vector of `estimatedInfo` objects with the same length as the input argument `transformedNames`.

## Properties

|                 |   |
|-----------------|---|
| Name            | A string indicating the name of an estimated model quantity. Changing this property also updates the TransformedName property.  |
| Transform       | <p>A string indicating whether the quantity value is transformed during estimation. Valid names are '', 'log', 'logit', and 'probit'. An empty string '' indicates that no transform is applied. The probit function is the inverse cumulative distribution function associated with the standard normal distribution. For the probit transform, SimBiology uses the norminv function. Hence Statistics Toolbox is required for the transform.</p> <p>The logit function, which is the inverse of sigmoid function, is defined as <math>\text{logit}(x) = \frac{\log(x)}{\log(1-x)}</math>.</p> |
| TransformedName | A read-only string that combines the transform name (such as 'log') and the quantity name (such as 'Central') into an expression (such as 'log(Central)').  |
| InitialValue    | A matrix or scalar value specifying the initial values of model quantities used for estimation. If neither InitialValue nor InitialTransformedValue is specified when you created the object, the initial values for estimation are obtained from the relevant quantity property (Value for parameters, InitialAmount for species, and Capacity for compartments). Changing this property automatically updates the   |

# EstimatedInfo object

---

InitialTransformedValue property of corresponding model quantities.

InitialTransformedValue matrix or scalar value specifying the initial transformed values of model quantities used for estimation. If neither InitialValue nor InitialTransformedValue is specified when you created the object, the initial transformed values for estimation are obtained by transforming the relevant quantity property (Value for parameters, InitialAmount for species, and Capacity for compartments). Changing this property automatically updates the InitialValue property of corresponding model quantities.

## Examples

### Specify Estimated Parameters Using an EstimatedInfo Object

Create a one-compartment PK model with bolus dosing and linear clearance.

```
pkmd = PKModelDesign;  
pkmd.addCompartment('Central', 'DosingType', 'Bolus', 'EliminationType', 'lin');  
[model, map] = pkmd.construct;
```

Suppose you want to estimate two model quantities: the volume of the central compartment (Central) and clearance rate (Cl\_Central). You can specify such estimated model quantities as well as appropriate parameter transforms (log transform in this example) and initial values using the estimatedInfo object.

```
estimatedParams = estimatedInfo({'log(Central)', 'log(Cl_Central)'}, 'InitialValue');
```

```
estimatedParams =
```

1x2 estimatedInfo array with properties:

| Name | Transform | TransformedName | InitialValue |
|------|-----------|-----------------|--------------|
|------|-----------|-----------------|--------------|

|       |              |       |                   |     |
|-------|--------------|-------|-------------------|-----|
| (1,1) | 'Central'    | 'log' | 'log(Central)'    | [1] |
| (1,2) | 'Cl_Central' | 'log' | 'log(Cl_Central)' | [1] |

For more examples, see `sbiofit` or `sbiofitmixed` reference pages.

## See Also

`sbiofit` | `sbiofitmixed` | `groupedData` object | `CovariateModel` object

# Event object

---

**Purpose** Store event information

**Description** Events are used to describe sudden changes in model behavior. An event lets you specify discrete transitions in model component values that occur when a user-specified condition become true. You can specify that the event occurs at a particular time, or specify a time-independent condition.

For details on how events are handled during a simulation, see “Event Object”.

See “Property Summary” on page 2-111 for links to event property reference pages.

Properties define the characteristics of an object. For example, an event object includes properties that allow you to specify the conditions to trigger an event (Trigger), and what to do after the event is triggered (EventFcn). Use the get and set commands to list object properties and change their values at the command line. You can graphically change object properties in the SimBiology desktop.

|                            |                  |                                  |
|----------------------------|------------------|----------------------------------|
| <b>Constructor Summary</b> | addevent (model) | Add event object to model object |
|----------------------------|------------------|----------------------------------|

|                       |                      |   |
|-----------------------|----------------------|---|
| <b>Method Summary</b> | copyobj (any object) | Copy SimBiology object and its children |
|                       | delete (any object)  | Delete SimBiology object                |
|                       | display (any object) | Display summary of SimBiology object    |
|                       | get (any object)     | Get object properties                   |
|                       | set (any object)     | Set object properties                   |



## Property Summary

|           |  |
|-----------|--|
| Active    | Indicate object in use during simulation |
| EventFcns | Event expression                         |
| Name      | Specify name of object                   |
| Notes     | HTML text describing SimBiology object   |
| Parent    | Indicate parent object                   |
| Tag       | Specify label for SimBiology object      |
| Trigger   | Event trigger                            |
| Type      | Display SimBiology object type           |
| UserData  | Specify data to associate with object    |

## See Also

AbstractKineticLaw object, Configset object, KineticLaw object, Model object, Parameter object, Reaction object, Root object, Rule object, Species object

# export (model)

---

**Purpose** Export SimBiology models for deployment and standalone applications

**Syntax**

```
exportedModel = export(model)
exportedModel = export(model,editvals)
exportedModel = export(model,editvals,editdoses)
```

## Arguments

|                        |  |
|------------------------|--|
| <code>model</code>     | Model object.  |
| <code>editvals</code>  | Vector of species, parameter, and compartment objects that are editable in the exported model. |
| <code>editdoses</code> | Vector of dose objects that are editable in the exported model.                                |

## Description

`exportedModel = export(model)` returns a `SimBiology.export.Model` object, `exportedModel`, from a `SimBiology.Model` object, `model`, including all active doses. In addition, if the `model` has any active variants, they are automatically applied to the exported model. By default, all species, parameters, compartments, and doses are editable in the exported model.

The exported SimBiology model object is a subclass of `hgsetget`. Thus, you can view properties for an exported model object with the `get` command, and modify properties for an exported model object with the `set` command.

`exportedModel = export(model,editvals)` specifies a vector of species, parameter, and compartment objects that are editable in the exported model. All doses in the exported model are editable.

`exportedModel = export(model,editvals,editdoses)` additionally specifies a vector of dose objects that are editable in the exported model.

## Method Summary

Methods for exported model objects

|               |   |
|---------------|---|
| accelerate    | Prepare exported SimBiology model for acceleration            |
| getdose       | Return exported SimBiology model dose object                  |
| getIndex      | Get indices into ValueInfo and InitialValues properties       |
| isAccelerated | Determine whether an exported SimBiology model is accelerated |
| simulate      | Simulate exported SimBiology model                            |

## Examples

### Export a SimBiology Model

Export a SimBiology model object.

```
modelObj = sbmlimport('lotka');  
exportedModel = export(modelObj)
```

```
exportedModel =
```

```
Model with properties:
```

```
    Name: 'lotka'  
  ExportTime: '14-Mar-2013 09:41:31'  
  ExportNotes: ''
```

Display the editable values (compartments, species, and parameters) information for the exported model object.

```
{exportedModel.ValueInfo.Name}
```

```
ans =
```

```
'unnamed'    'x'    'y1'    'y2'    'z'    'c1'    'c2'    'c3'
```

## export (model)

---

There are 8 editable values in the exported model. Export the model again, allowing only the parameters (c1, c2, and c3) to be editable.

```
parameters = sbioselect(modelObj, 'Type', 'parameter');
exportedModelParam = export(modelObj, parameters);
{exportedModelParam.ValueInfo.Name}
```

```
ans =
```

```
    'c1'    'c2'    'c3'
```

Export the model a third time, allowing the parameters and species to be editable.

```
PS = sbioselect(modelObj, 'Type', {'species', 'parameter'});
exportedModelPS = export(modelObj, PS);
{exportedModelPS.ValueInfo.Name}
```

```
ans =
```

```
    'x'    'y1'    'y2'    'z'    'c1'    'c2'    'c3'
```

**See Also** [SimBiology.export.Model](#)

### Related Examples

- “PK/PD Modeling and Simulation to Guide Dosing Strategy for Antibiotics”
- “Deploy a SimBiology Model”

# fitted(LeastSquaresResults,OptimResults,NLINResults)

---

**Purpose** Return simulation results of SimBiology model fitted using least-squares regression

**Syntax** [yfit,parameterEstimates] = fitted(resultsObj)

**Description** [yfit,parameterEstimates] = fitted(resultsObj) returns simulation results `yfit` and parameter estimates `parameterEstimates` from a fitted SimBiology model.

---

**Tip** Use this method to retrieve simulation results from the fitted model if you did not specify the second optional output argument that corresponds to simulation results when you first ran `sbiofit`.

---

## Input Arguments

### **resultsObj - Estimation results**

OptimResults object | NLINResults object

Estimation results, specified as an `OptimResults` object or `NLINResults` object, which contains estimation results from running `sbiofit`.

## Output Arguments

### **yfit - Simulation results**

vector of SimData objects

Simulation results, returned as a vector of `SimData` objects. The states reported in `yfit` are the states that were included in the `responseMap` input argument of `sbiofit` as well as any other states listed in the `StatesToLog` property of the runtime options (`RuntimeOptions`) of the SimBiology model.

### **parameterEstimates - Estimated parameter values**

table

Estimated parameter values, returned as a table. This argument is identical to the `resultsObj.ParameterEstimates` property.

## Examples

### Estimate a Parameter from the Yeast G Protein Model

This example uses the yeast heterotrimeric G protein model and experimental data reported by [1]. For more background information and details about the model, see the **Background** section in “Parameter Scanning, Parameter Estimation, and Sensitivity Analysis in the Yeast Heterotrimeric G Protein Cycle”.

Load the G protein model.

```
sbioloadproject gprotein
```

Store the experimental data containing the time course for the fraction of active G protein, as reported in the reference paper [1].

```
time = [0 10 30 60 110 210 300 450 600]';  
GaFracExpt = [0 0.35 0.4 0.36 0.39 0.33 0.24 0.17 0.2]';
```

Create a groupedData object based on the experimental data.

```
tbl = table(time,GaFracExpt);  
grpData = groupedData(tbl);
```

Map the appropriate model component to the experimental data. In other words, indicate which species in the model corresponds to which response variable in the data. In this example, map the model parameter GaFrac to the experimental data variable GaFracExpt from grpData.

```
responseMap = 'GaFrac = GaFracExpt';
```

Use an estimatedInfo object to define the model parameter kGd as a parameter to be estimated.

```
estimatedParam = estimatedInfo('kGd');
```

Perform the parameter estimation.

```
fitResult = sbiofit(m1,grpData,responseMap,estimatedParam);
```

## fitted(LeastSquaresResults,OptimResults,NLINResults)

---

View the estimated parameter value of kGd.

```
fitResult.ParameterEstimates
```

```
ans =
```

| Name  | Estimate | StandardError |
|-------|----------|---------------|
| 'kGd' | 0.11     | 0.00037969    |

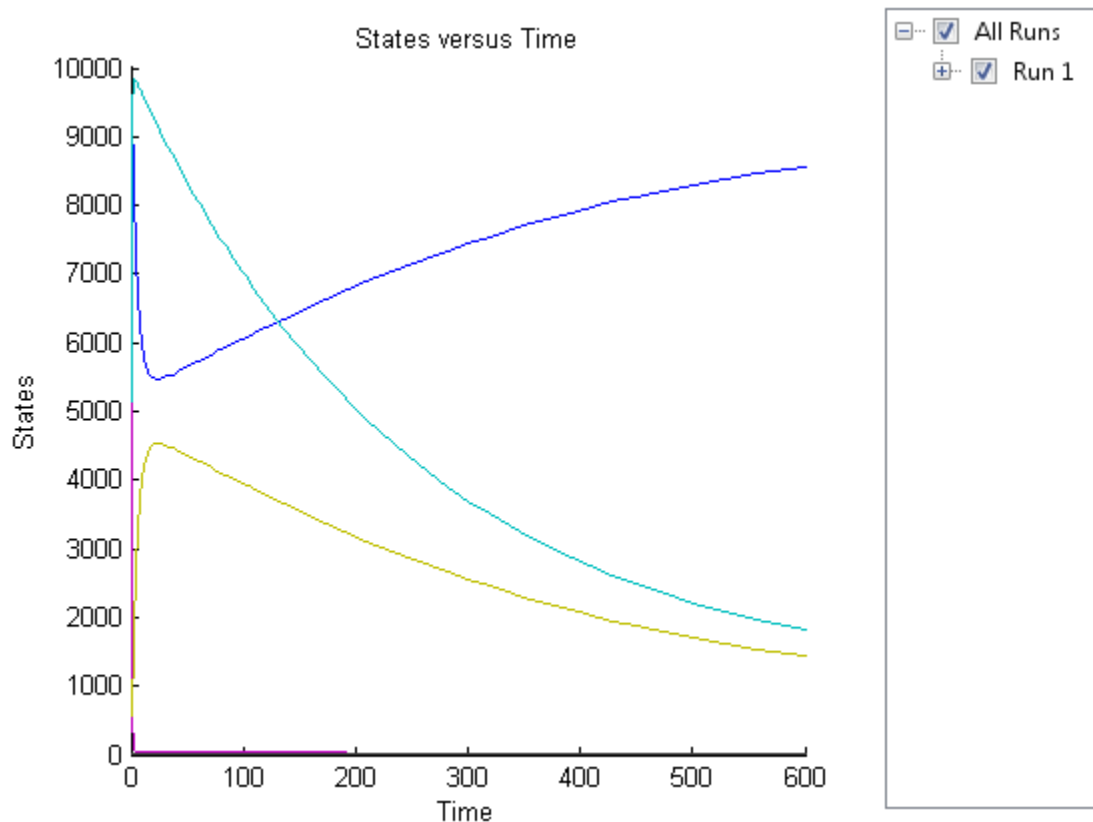
Suppose you want to plot the model simulation results using the estimated parameter value. You can either rerun the `sbiofit` function and specify to return the optional second output argument, which contains simulation results, or use the `fitted` method to retrieve the results without rerunning `sbiofit`.

```
[yfit,paramEstim] = fitted(fitResult);
```

Plot the simulation results.

```
sbioplot(yfit)
```

# fitted(LeastSquaresResults,OptimResults,NLINResults)



## References

[1] Yi, T-M., Kitano, H., and Simon, M. (2003). A quantitative characterization of the yeast heterotrimeric G protein cycle. PNAS. 100, 10764–10769.

## See Also

NLINResults object | OptimResults object | sbiofit



**Purpose** Return the simulation results of a fitted nonlinear mixed-effects model

**Syntax** `[yfit,parameterEstimates]= fitted(resultsObj)`  
`[yfit,parameterEstimates]=`  
`fitted(resultsObj, 'ParameterType', value)`

**Description** `[yfit,parameterEstimates]= fitted(resultsObj)` returns simulation results `yfit` and parameter estimates `parameterEstimates` from a fitted nonlinear mixed-effect model.

`[yfit,parameterEstimates]= fitted(resultsObj, 'ParameterType', value)` returns simulation results that are simulated using either individual or population parameter estimates. The two choices for `value` are 'population' or 'individual' (default).

---

**Tip** Use this method to retrieve simulation results from the fitted model if you did not specify the second or third optional output argument that corresponds to simulation results when you first ran `sbiofitmixed`.

---

## Input Arguments

### **resultsObj - Estimation results**

NLMEResults object

Estimation results, specified as an NLMEResults object, which contains estimation results returned by `sbiofitmixed`.

### **value - Parameter type**

string

Parameter type, specified as 'population' or 'individual' (default). If 'population', the method returns the model simulation results using the population parameter estimates. If 'individual', it returns simulation results using the individual-specific parameter estimates.

# fitted(NLMEResults)

---

## Output Arguments

### **yfit - Simulation results**

vector of SimData objects

Simulation results, returned as a vector of SimData objects. The states reported in `yfit` are the states that were included in the `responseMap` input argument of `sbiofitmixed` as well as any other states listed in the `StatesToLog` property of the runtime options (`RuntimeOptions`) of the SimBiology model.

### **parameterEstimates - Estimated parameter values**

table

Estimated parameter values, returned as a table. This is identical to `resultsObj.IndividualParameterEstimates` property when the `value` argument is 'individual' or `resultsObj.PopulationParameterEstimates` property when the value is 'population'.

## See Also

NLMEResults object | `sbiofitmixed`

## Purpose

Get object properties

## Syntax

```
PropertyValue = get(Obj, 'PropertyName')  
objProperties = get(Obj)
```

## Arguments

*PropertyValue* Value defined for '*PropertyName*'

*Obj* Abstract kinetic law, compartment, configuration set, event, kinetic law, model, parameter, PKCompartment, PKData, PKModelDesign, PKModelMap, reaction, rule, SimData, species, or variant object.

'*PropertyName*' Name of the property to get.

*objProperties* Struct containing properties and values for the object, *Obj*.

## Description

*PropertyValue* = get(*Obj*, '*PropertyName*') gets the value '*PropertyValue*' of the object, *Obj*'s *PropertyName* property.

*objProperties* = get(*Obj*) gets the properties for the object, *Obj*, and returns it to *objProperties*.

## Examples

1 Create a model object.

```
modelObj = sbiomodel ('my_model');
```

## get (any object)

---

**2** Add parameter object.

```
parameterObj = addparameter (modelObj, 'kf');
```

**3** Set the ConstantValue property of the parameter object to false and verify.

MATLAB returns 1 for true and 0 for false.

```
set (parameterObj, 'ConstantValue', false);  
get(parameterObj, 'ConstantValue')
```

MATLAB returns

```
ans =
```

```
0
```

### See Also

getadjacencymatrix, getconfigset, getdata, getparameters,  
getsensmatrix, getspecies, getstoichmatrix, set

**Purpose** Get adjacency matrix from model object

**Syntax**

```
M = getadjacencymatrix(modelObj)  
[M, Headings] = getadjacencymatrix(modelObj)  
[M, Headings, Mask] = getadjacencymatrix(modelObj)
```

## Arguments

|                 |   |
|-----------------|---|
| <i>M</i>        | Adjacency matrix for <i>modelObj</i> .  |
| <i>modelObj</i> | Specify the model object.   |
| <i>Headings</i> | Return row and column headings. If species are in multiple compartments, species names are qualified with the compartment name in the form <code>compartmentName.speciesName</code> . For example, <code>nucleus.DNA</code> , <code>cytoplasm.mRNA</code> . |
| <i>Mask</i>     | Return 1 for the species object and 0 for the reaction object to <i>Mask</i> .  |

## Description

`getadjacencymatrix` returns the adjacency matrix for a model object.

`M = getadjacencymatrix(modelObj)` returns an adjacency matrix for the model object (*modelObj*) to *M*.

An adjacency matrix is defined by listing all species contained by *modelObj* and all reactions contained by *modelObj* column-wise and row-wise in a matrix. The reactants of the reactions are represented in the matrix with a 1 at the location of [row of species, column of reaction]. The products of the reactions are represented in the matrix with a 1 at the location of [row of reaction, column of species]. All other locations in the matrix are 0.

`[M, Headings] = getadjacencymatrix(modelObj)` returns the adjacency matrix to *M* and the row and column headings to *Headings*. *Headings* is defined by listing all Name property values of species contained by *modelObj* and all Name property values of reactions contained by *modelObj*.

## getadjacencymatrix (model)

---

`[M, Headings, Mask] = getadjacencymatrix(modelObj)` returns an array of 1s and 0s to *Mask*, where a 1 represents a species object and a 0 represents a reaction object.

### Examples

**1** Read in *m1*, a model object, using `sbmlimport`:

```
m1 = sbmlimport('lotka.xml');
```

**2** Get the adjacency matrix for *m1*:

```
[M, Headings] = getadjacencymatrix(m1)
```

### See Also

`getstoichmatrix`

## Purpose

Get configuration set object from model object

## Syntax

```
configsetObj = getConfigset(modelObj, 'NameValue')  
configsetObj = getConfigset(modelObj)  
configsetObj = getConfigset(modelObj, 'active')
```

## Arguments

|                     |   |
|---------------------|---|
| <i>modelObj</i>     | Model object. Enter a variable name for a model object. |
| <i>NameValue</i>    | Name of the configset object.                           |
| <i>configsetObj</i> | Object holding the simulation-specific information.     |

## Description

*configsetObj* = getConfigset(*modelObj*, 'NameValue') returns the configuration set attached to *modelObj* that is named *NameValue*, to *configsetObj*.

*configsetObj* = getConfigset(*modelObj*) returns a vector of all attached configuration sets, to *configsetObj*.

*configsetObj* = getConfigset(*modelObj*, 'active') retrieves the active configuration set.

A configuration set object stores simulation-specific information. A SimBiology model can contain multiple configsets with one being active at any given time. The active configuration set contains the settings that are used during the simulation.

Use the `setactiveconfigset` function to define the active configset. *modelObj* always contains at least one configset object with the name configured to 'default'. Additional configset objects can be added to *modelObj* with the method `addconfigset`.

## Examples

1 Retrieve the default configset object from the *modelObj*.

```
modelObj = sbiomodel('cell');  
configsetObj = getConfigset(modelObj)
```

## getConfigset (model)

---

```
Configuration Settings - default (active)
  SolverType:          ode15s
  StopTime:           10

SolverOptions:
  AbsoluteTolerance:  1.000000e-06
  RelativeTolerance:  1.000000e-03
  SensitivityAnalysis: false

RuntimeOptions:
  StatesToLog:        all

CompileOptions:
  UnitConversion:     false
  DimensionalAnalysis: true

SensitivityAnalysisOptions:
  Inputs:             0
  Outputs:            0
```

### 2 Configure the SolverType to ssa.

```
set(configsetObj, 'SolverType', 'ssa')
get(configsetObj)
```

```
Active: 1
CompileOptions: [1x1 SimBiology.CompileOptions]
  Name: 'default'
  Notes: ''
RuntimeOptions: [1x1 SimBiology.RuntimeOptions]
SensitivityAnalysisOptions: [1x1 SimBiology.SensitivityAnalysisOptions]
SolverOptions: [1x1 SimBiology.SSASolverOptions]
  SolverType: 'ssa'
  StopTime: 10
MaximumNumberOfLogs: Inf
MaximumWallClock: Inf
TimeUnits: 'second'
```



Type: 'configset'

**See Also**      addconfigset, removeconfigset, setactiveconfigset

# getCovariateData (pkdata)

---

**Purpose** Create design matrix needed for fit

**Syntax** `CovData = getCovariateData(PKDataObj)`

**Description** `CovData = getCovariateData(PKDataObj)` creates `CovData`, a dataset array containing only the covariate data from the data set in `PKDataObj`, a `PKData` object. `CovData` contains one row for each individual and one column for each covariate.

---

**Tip** Use the `getCovariateData` method to view the covariate data when writing equations for the `Expression` property of a `CovariateModel` object.

---

**See Also** `CovariateModel` | `Expression` | `PKData`

**How To**

- Modeling the Population Pharmacokinetics of Phenobarbital in Neonates
- “Specify a Covariate Model”

**Purpose** Get data from SimData object array

**Syntax** `[t, x, names] = getdata(simDataObj)`  
`[Out] = getdata(simDataObj, 'FormatValue')`

## Arguments

## Output Arguments

- |                 |  |
|-----------------|--|
| <i>t</i>        | An n-by-1 vector of time points.   |
| <i>x</i>        | An n-by-m data array. <i>t</i> and <i>names</i> label the rows and columns of <i>x</i> respectively.   |
| <i>names</i>    | An m-by-1 cell array of names.   |
| <i>Metadata</i> | When used with the 'nummetadata' input argument, <i>Metadata</i> contains a cell array of metadata structures. The elements of <i>Metadata</i> label the columns of <i>x</i> .   |
| <i>Out</i>      | Data returned in the format specified in 'FormatValue', shown in “Input Arguments” on page 2-129. Depending on the specified 'FormatValue', <i>Out</i> contains one of the following: <ul style="list-style-type: none"><li>• Structure array</li><li>• SimData object</li><li>• Time series object</li><li>• Combined time series object from an array of SimData objects</li></ul> |

## Input Arguments

- |                    |   |
|--------------------|---|
| <i>simDataObj</i>  | SimData object. Enter a variable name for a SimData object. |
| <i>FormatValue</i> | Choose a format from the following table.                   |

## getdata (SimData)

---

| <b>FormatValue</b> | <b>Description</b>   |
|--------------------|--|
| 'num'              | Specifies the format that lets you return data in numeric arrays. This is the default when <code>getdata</code> is called with multiple output arguments.                          |
| 'nummetadata'      | Specifies the format that lets you return a cell array of metadata structures in <i>metadata</i> instead of names. The elements of <i>metadata</i> label the columns of <i>x</i> . |
| 'numqualnames'     | Specifies the format that lets you return qualified names in <i>names</i> to resolve ambiguities.  |
| 'struct'           | Specifies the format that lets you return a structure array holding both data and metadata. This is the default when you use a single output argument.                             |
| 'simdata'          | Specifies the format that lets you return data in a new <code>SimData</code> object. This format is more useful for <code>SimData</code> methods other than <code>getdata</code> . |

| <b>FormatValue</b> | <b>Description</b>   |
|--------------------|--|
| 'ts'               | Specifies the format that lets you return data in time series objects, creating an individual time series for each state or column and SimData object in <code>simDataObj</code> . |
| 'tslumped'         | Specifies the format that lets you return data in time series objects, combining data from each SimData object into a single time series.  |

## Description

`[t, x, names] = getdata(simDataObj)` gets simulation time and state data from the SimData object `simDataObj`. When `simDataObj` contains more than one element, the outputs `t`, `x`, `names` are cell arrays in which each cell contains data for the corresponding element of `simDataObj`.

`[Out] = getdata(simDataObj, 'FormatValue')` returns the data in the specified format. Valid formats are listed in “Input Arguments” on page 2-129.

## Examples

### Simulating and Retrieving Data

- 1 The project file, `radiodecay.sbproj`, contains a model stored in a variable called `m1`. Load `m1` into the MATLAB workspace and simulate the model.

```
sbioloadproject('radiodecay');
simDataObj = sbiosimulate(m1);
```

- 2 Get all the simulation data from the SimData object.

```
[t x names] = getdata(simDataObj);
```

# getdata (SimData)

---

## Retrieving Data for Ensemble Runs

- 1 The project file, `radiodecay.sbproj`, contains a model stored in a variable called `m1`. Load `m1` into the MATLAB workspace.

```
sbioloadproject('radiodecay');
```

- 2 Change the solver to use during the simulation and perform an ensemble run.

```
csObj = getconfigset(m1);  
set(csObj, 'SolverType', 'ssa');  
simDataObj = sbioenssemblerun(m1, 10);
```

- 3 Get all the simulation data from the `SimData` object.

```
tsObjs = getdata(simDataObj(1:5), 'ts');
```

## See Also

`display`, `get`, `resample`, `select`, `selectbyname`, `setactiveconfigset`  
MATLAB function struct

|                         |   |
|-------------------------|---|
| <b>Purpose</b>          | Return exported SimBiology model dose object  |
| <b>Syntax</b>           | <pre>doses = getdose(model) doses = getdose(model,doseName)</pre>   |
| <b>Description</b>      | <p><code>doses = getdose(model)</code> returns all the <code>SimBiology.export.Dose</code> objects associated with the exported model.</p> <p><code>doses = getdose(model,doseName)</code> returns the export dose object with the <code>Name</code> property matching <code>doseName</code>.</p>         |
| <b>Input Arguments</b>  | <p><b>model</b><br/>SimBiology.export.Model object.</p> <p><b>doseName</b><br/>String containing a dose name to match against the <code>Name</code> property of the export dose objects in <code>model</code>.</p> <p><b>Default:</b> All dose objects.</p>   |
| <b>Output Arguments</b> | <p><b>doses</b><br/>Export dose objects in <code>model</code>, or the export dose object with <code>Name</code> property <code>doseName</code>.</p>   |
| <b>Examples</b>         | <p><b>Retrieve SimBiology Model Dose Objects</b></p> <p>Open a sample SimBiology model project, and export the included model object.</p> <pre>sbioloadproject('AntibacterialPKPD') em = export(m1);</pre> <p>Display the editable doses in the exported model object.</p> <pre>doses = getdose(em)</pre> |

# SimBiology.export.Model.getdose

---

```
doses =  
  
    1x4 RepeatDose array with properties:  
  
        Interval  
        RepeatCount  
        StartTime  
        TimeUnits  
        Amount  
        AmountUnits  
        DurationParameterName  
        LagParameterName  
        Name  
        Notes  
        Parent  
        Rate  
        RateUnits  
        TargetName
```

The exported model has 4 repeated dose objects. Display the dose names.

```
{doses.Name}
```

```
ans =  
  
    '250 mg bid'    '250 mg tid'    '500 mg bid'    '500 mg tid'
```

Extract only the 3rd dose object from the exported model object.

```
dose3 = getdose(em,'500 mg bid')
```

```
dose3 =  
  
    RepeatDose with properties:  
  
                Interval: 12  
                RepeatCount: 27
```



```
StartTime: 0
TimeUnits: 'hour'
Amount: 500
AmountUnits: 'milligram'
DurationParameterName: 'TDose'
LagParameterName: ''
Name: '500 mg bid'
Notes: ''
Parent: 'Antibacterial'
Rate: 0
RateUnits: ''
TargetName: 'Central.Drug'
```

## See Also

[SimBiology.export.Model](#) | [SimBiology.export.Dose](#) | [export](#)

## Related Examples

- “PK/PD Modeling and Simulation to Guide Dosing Strategy for Antibiotics”
- “Deploy a SimBiology Model”

# getdose (model)

---

**Purpose** Return SimBiology dose object

**Syntax**  
`doseObj = getdose(modelObj)`  
`doseObj = getdose(modelObj, 'DoseName')`

**Arguments**

|                 |  |
|-----------------|--|
| <i>modelObj</i> | Selects a model object that contains a dose object.  |
| <i>DoseName</i> | Name of a dose object contained in a model object. <i>DoseName</i> is from the dose object property, Name. |

**Outputs**

|                |  |
|----------------|--|
| <i>doseObj</i> | ScheduleDose or RepeatDose object retrieved from a model object. A RepeatDose or ScheduleDose object defines an increase (dose) to a species amount during a simulation. |
|----------------|--|

**Description**

`doseObj = getdose(modelObj)` returns a Simbiology dose object (*doseObj*) contained in a Simbiology model object (*modelObj*).

`doseObj = getdose(modelObj, 'DoseName')` returns a SimBiology dose object (*modelObj*) with the name *DoseName*.

**Examples** Get a dose object from a model object.

**1** Create a model object, and then add a dose object to the model object.

```
modelObj = sbiomodel('mymodel');  
doseObj = adddose(modelObj, 'dose1');
```

**2** Get the dose object from a model object.

```
myModelDose = getdose(modelObj);
```

### See Also

Model object methods:

- `adddose` — add a dose object to a model object
- `getdose` — get dose information from a model object
- `removedose` — remove a dose object from a model object

Dose object constructor `sbiodose`.

`ScheduleDose` object and `RepeatDose` object methods:

- `copyobj` — copy a dose object from one model object to another model object
- `get` — view properties for a dose object
- `set` — define or modify properties for a dose object

# Model.getequations

---

**Purpose** Return system equations for model object

**Syntax**

```
equations = getequations(modelobj)
equations =
getequations(modelobj,configsetobj,variantobj,doseobj)
```

**Description** equations = getequations(modelobj) returns equations, a string containing the system of equations that represent modelobj, a Model object. The active Configset object is used to generate the equations and must specify a deterministic solver.

```
equations =
getequations(modelobj,configsetobj,variantobj,doseobj)
```

returns the system of equations that represent the model specified by a Model object, Variant objects, and dose objects (RepeatDose or ScheduleDose). The Configset object, configsetobj, is used to generate the equations and must specify a deterministic solver.

**Tips** Use getequations to see the system of equations that represent a model for:

- Publishing purposes
- Model debugging

## Input Arguments

**modelobj**  
Object of the Model class.

---

**Note** If using modelobj as the only input argument, the active Configset object must specify a deterministic solver.

---

**configsetobj**  
Object of the Configset class. This object must specify a deterministic solver.

**Default:** [ ] (Empty, which specifies the active Configset object for modelobj)

## **variantobj**

Object or array of objects of the Variant class.

**Default:** [ ] (Empty, which specifies no variant object)

## **doseobj**

Object or array of objects of the RepeatDose or ScheduleDose class.

**Default:** [ ] (Empty, which specifies no dose object)

## **Output Arguments**

### **equations**

String containing the system of equations that represent a model. This string includes equations for reactions, rules, events, variants, and doses.

## **Examples**

### **View System of Equations for Simple Model**

View system of equations that represent a simple model, containing only reactions.

Import the lotka model, included with SimBiology, into a variable named model1:

```
model1 = sbmlimport('lotka');
```

View all equations that represent the model1 model and its active configset:

```
m1equations = getequations(model1)
```

```
m1equations =
```

```
ODEs:
```

## Model.getequations

---

```
d(y1)/dt = 1/unnamed*(ReactionFlux1 - ReactionFlux2)
d(y2)/dt = 1/unnamed*(ReactionFlux2 - ReactionFlux3)
d(z)/dt = 1/unnamed*(ReactionFlux3)
```

Fluxes:

```
ReactionFlux1 = c1*y1*x
ReactionFlux2 = c2*y1*y2
ReactionFlux3 = c3*y2
```

Parameter Values:

```
c1 = 10
c2 = 0.01
c3 = 10
unnamed = 1
```

Initial Conditions:

```
x = 1
y1 = 900
y2 = 900
z = 0
```

MATLAB displays the ODEs, fluxes, parameter values, and initial conditions for the reactions in `model1`.

### View System of Equations for Model and Dose

View system of equations that represent a model, containing only reactions, and a repeated dose.

Import the `lotka` model, included with SimBiology, into a variable named `model1`:

```
model1 = sbmlimport('lotka');
```

Add a repeated dose to the model:

```
doseObj1 = adddose(model1, 'dose1', 'repeat');
```

Set the properties of the dose to administer 3 mg, at a rate of 10 mg/hour, 6 times, at an interval of every 24 hours, to species y1:

```
doseObj1.Amount = 0.003;
doseObj1.AmountUnits = 'gram';
doseObj1.Rate = 0.010;
doseObj1.RateUnits = 'gram/hour';
doseObj1.Repeat = 6;
doseObj1.Interval = 24;
doseObj1.TimeUnits = 'hour';
doseObj1.TargetName = 'y1';
```

View all equations that represent the model1 model, its active configset, and the repeated dose:

```
m1_with_dose_equations = getequations (model1,[],[],doseObj1)
```

```
m1_with_dose_equations =
```

ODEs:

```
d(y1)/dt = 1/unnamed*(ReactionFlux1 - ReactionFlux2) + dose1
d(y2)/dt = 1/unnamed*(ReactionFlux2 - ReactionFlux3)
d(z)/dt = 1/unnamed*(ReactionFlux3)
```

Fluxes:

```
ReactionFlux1 = c1*y1*x
ReactionFlux2 = c2*y1*y2
ReactionFlux3 = c3*y2
```

Parameter Values:

```
c1 = 10
c2 = 0.01
c3 = 10
unnamed = 1
```

Initial Conditions:

```
y1 = 900
y2 = 900
```

# Model.getequations

---

```
z = 0
x = 1
```

```
Doses:
Variable                Type                Units
dose1                   repeatdose        gram
```

MATLAB displays the ODEs, fluxes, parameter values, and initial conditions for the reactions and the dose in `model1`.

## See Also

`Model` object | `Configset` object | `Variant` object | `RepeatDose` object | `ScheduleDose` object



## Purpose

Get indices into ValueInfo and InitialValues properties

## Syntax

```
indices = getIndex(model,name)
indices = getIndex(model,name,type)
```

## Description

`indices = getIndex(model,name)` returns the indices of all ValueInfo objects in a SimBiology.export.Model object that have a QualifiedName or Name property that match the specified name input argument.

- `getIndex` first tries to match the QualifiedName property. If there are matches, then `getIndex` returns their indices.
- If there are no matches based on QualifiedName, then `getIndex` tries to match the Name property. If there are matches, then `getIndex` returns their indices.
- If there are no matches based on QualifiedName or Name, then `getIndex` returns [].

`indices = getIndex(model,name,type)` returns indices for only the ValueInfo objects with a Type property that matches the type input argument.

## Input Arguments

### model

SimBiology.export.Model object.

### name

String containing a name to match against the QualifiedName, then Name, properties of the ValueInfo objects in model.

### type

String containing a name to match against the Type property of the ValueInfo objects in model.

**Default:** All types.

# SimBiology.export.Model.getIndex

---

## Output Arguments

### indices

Vector of indices indicating which ValueInfo objects in a SimBiology.export.Model object match on the specified name and type.

## Examples

### Index Exported SimBiology Editable Values

Load a sample SimBiology model object, and export.

```
modelObj = sbmlimport('lotka');  
em = export(modelObj);
```

Get the index of the editable value with name y1.

```
ix = getIndex(em,'y1')
```

```
ix =
```

```
3
```

Display the type of value.

```
em.ValueInfo(ix).Type
```

```
ans =
```

```
species
```

The name y1 corresponds to an editable species.

## See Also

SimBiology.export.Model | SimBiology.export.ValueInfo | export

## Related Examples

- “PK/PD Modeling and Simulation to Guide Dosing Strategy for Antibiotics”
- “Deploy a SimBiology Model”

## Purpose

Get specific parameters in kinetic law object

## Syntax

```
parameterObj = getparameters(kineticlawObj)
parameterObj = getparameters(kineticlawObj,
    'ParameterVariablesValue')
```

## Arguments

|                                |   |
|--------------------------------|---|
| <i>kineticlawObj</i>           | Retrieve parameters used by the kinetic law object.   |
| <i>ParameterVariablesValue</i> | Retrieve parameters used by the kinetic law object corresponding to the specified parameter in the ParameterVariables property of the kinetic law object. |

## Description

*parameterObj* = `getparameters(kineticlawObj)` returns the parameters used by the kinetic law object *kineticlawObj* to *parameterObj*.

`parameterObj = getparameters(kineticlawObj, 'ParameterVariablesValue')` returns the parameter in the ParameterVariableNames property that corresponds to the parameter specified in the ParameterVariables property of *kineticlawObj*, to *parameterObj*. ParameterVariablesValue is the name of the parameter as it appears in the ParameterVariables property of *kineticlawObj*. ParameterVariablesValue can be a cell array of strings.

If you change the name of a parameter, you must configure all applicable elements such as rules that use the parameter, any user-specified ReactionRate, or the kinetic law object property ParameterVariableNames. Use the method `setParameter` to configure ParameterVariableNames.

## Examples

Create a model, add a reaction, and assign the ParameterVariableNames for the reaction rate equation.

## getparameters (kineticlaw)

---

- 1 Create the model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');  
reactionObj = addreaction(modelObj, 'a -> c + d');
```

- 2 Create a kinetic law object for the reaction object, of the type 'Henri-Michaelis-Menten'.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

- 3 Add two parameter objects.

```
parameterObj1 = addparameter(kineticlawObj, 'Va');  
parameterObj2 = addparameter(kineticlawObj, 'Ka');
```

- 4 The 'Henri-Michaelis-Menten' kinetic law has two parameter variables ( $V_m$  and  $K_m$ ) that should to be set. To set these variables:

```
setparameter(kineticlawObj, 'Vm', 'Va');  
setparameter(kineticlawObj, 'Km', 'Ka');
```

- 5 To retrieve a parameter variable:

```
parameterObj3 = getparameters(kineticlawObj, 'Vm')
```

MATLAB returns:

SimBiology Parameter Array

| Index: | Name: | Value: | ValueUnits: |
|--------|-------|--------|-------------|
| 1      | Va    | 1      |             |

```
parameterObj4 = getparameters (kineticlawObj, 'Km')
```

### See Also

addparameter, getspecies, setparameter

## Purpose

Get 3-D sensitivity matrix from SimData array

## Syntax

```
[T, R, Outputs, InputFactors] = getsensmatrix(simDataObj)
[T, R, Outputs, InputFactors] =
getsensmatrix(simDataObj, OutputNames,
InputFactorNames)
```

## Arguments

|                     |   |
|---------------------|---|
| <i>T</i>            | <i>T</i> is an m-by-1 array specifying time points for the sensitivity data in <i>R</i> .   |
| <i>R</i>            | <i>R</i> is an m-by-n-by-p array of sensitivity data with times, outputs, and input factors corresponding to its first, second, and third dimensions respectively. <i>R</i> (:, <i>i</i> , <i>j</i> ) is the time course for the sensitivity of state <i>Outputs</i> { <i>i</i> } to the input factor <i>InputFactors</i> { <i>j</i> }. |
| <i>Outputs</i>      | Name of the output factors, where output factors are the names of the states for which you want to calculate sensitivity.   |
| <i>InputFactors</i> | Name of the input factors, where input factors are the names of the states with respect to which you want to calculate sensitivity.   |

## Description

[*T*, *R*, *Outputs*, *InputFactors*] = getsensmatrix(*simDataObj*) gets time and sensitivity data from the SimData object (*simDataObj*).

When *simDataObj* contains more than one element, the output arguments are cell arrays in which each cell contains data for the corresponding element of *simDataObj*.

The getsensmatrix method can only return sensitivity data that is contained in the SimData object. The sensitivity data that is logged in a SimData object is set at simulation time by the configuration set used during the simulation. This is typically the model's active configuration set. For an explanation of how to set up a sensitivity calculation using the configuration set, see "Sensitivity Analysis". Note in particular that

## getsensmatrix (SimData)

---

the sensitivity data *R* returned by `getsensmatrix` may be normalized, as specified at simulation time.

```
[T, R, Outputs, InputFactors] =  
getsensmatrix(simDataObj, OutputNames, InputFactorNames) gets  
sensitivity data for the outputs specified by OutputNames and the input  
factors specified by InputFactorNames.
```

*OutputNames* and *InputFactorNames* can both be any one of the following:

- Empty array
- Single name
- Cell array of names

Pass an empty array for *OutputNames* or *InputFactorNames* to ask for sensitivity data on all output factors or input factors contained in *simDataObj*, respectively. You can also use qualified names such as '*CompartmentName.SpeciesName*' or '*ReactionName.ParameterName*' to resolve ambiguities.

### Examples

This example shows how to retrieve sensitivity data from a `SimData` object.

**1** Set up the simulation:

- a** Import the radio decay model from SimBiology examples.

```
modelObj = sbmlimport('radiodecay');
```

- b** Retrieve the configuration settings and the sensitivity analysis options from the `modelObj`.

```
configsetObj = getconfigset(modelObj);  
sensitivityObj = get(configsetObj, 'SensitivityAnalysisOptions');
```

- c** Specify the species for which you want sensitivity data in the `Outputs` property. All model species are selected in this example.

Use the `sbioselect` function to retrieve the species objects from the model.

```
allSpeciesObj = sbioselect(modelObj, 'Type', 'species');  
set(sensitivityObj, 'Outputs', allSpeciesObj);
```

- d** Specify species and parameters with respect to which you want to calculate the sensitivities in the `Inputs` property.
- e** Enable `SensitivityAnalysis`.

```
set(configsetObj.SolverOptions, 'SensitivityAnalysis', true)  
get(configsetObj.SolverOptions, 'SensitivityAnalysis')
```

```
ans =
```

```
1
```

- f** Simulate and return the results in a `SimData` object.

```
simDataObj = sbiosimulate(modelObj)
```

- 2** Extract and plot sensitivity data from the `SimData` object.

- a** Use `getsensmatrix` to retrieve sensitivity data.

```
[t R outs ifacs] = getsensmatrix(simDataObj);
```

- b** Plot sensitivity values.

```
plot(t, R(:, :, 2));  
legend(outs);  
title(['Sensitivities of species relative to ' ifacs{2}]);
```

### See Also

`display`, `get`, `getdata`, `resample`, `selectbyname`

MATLAB function struct

# getspecies (kineticlaw)

---

**Purpose** Get specific species in kinetic law object

**Syntax**  
`speciesObj = getspecies(kineticlawObj)`  
`speciesObj = getspecies(kineticlawObj, 'SpeciesVariablesValue')`

## Arguments

|                              |   |
|------------------------------|---|
| <i>kineticlawObj</i>         | Retrieve species used by the kinetic law object.  |
| <i>SpeciesVariablesValue</i> | Retrieve species used by the kinetic law object corresponding to the specified species in the <code>SpeciesVariables</code> property of the kinetic law object. |

## Description

`speciesObj = getspecies(kineticlawObj)` returns the species used by the kinetic law object `kineticlawObj` to `speciesObj`.

`speciesObj = getspecies(kineticlawObj, 'SpeciesVariablesValue')` returns the species in the `SpeciesVariableNames` property to `speciesObj`.

`SpeciesVariablesValue` is the name of the species as it appears in the `SpeciesVariables` property of `kineticlawObj`. `SpeciesVariablesValue` can be a cell array of strings.

Species names are referenced by reaction objects, kinetic law objects, and model objects. If you change the name of a species, the reaction updates to use the new name. You must, however, configure all other applicable elements such as rules that use the species, and the kinetic law object `SpeciesVariableNames`. Use the method `setspecies` to configure `SpeciesVariableNames`.

## Examples

Create a model, add a reaction, and then assign the `SpeciesVariableNames` for the reaction rate equation.

- 1 Create a model object, and then add a reaction object.



```
modelObj = sbiomodel('my_model');  
reactionObj = addreaction(modelObj, 'a -> c + d');
```

- 2 Create a kinetic law object for the reaction object, of the type 'Henri-Michaelis-Menten'.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

reactionObj KineticLaw property is configured to kineticlawObj.

- 3 The 'Henri-Michaelis-Menten' kinetic law has one species variable (S) that should to be set. To set this variable:

```
setspecies(kineticlawObj, 'S', 'a');
```

- 4 Retrieve the species variable using getspecies.

```
speciesObj = getspecies (kineticlawObj, 'S')
```

MATLAB returns:

SimBiology Species Array

| Index: | Compartment: | Name: | InitialAmount: | InitialAmountUnits: |
|--------|--------------|-------|----------------|---------------------|
| 1      | unnamed      | a     | 0              |                     |

### See Also

addspecies, getparameters, setparameter, setspecies

# getstoichmatrix (model)

---

**Purpose** Get stoichiometry matrix from model object

**Syntax**

```
M = getstoichmatrix(modelObj)
[M,objSpecies] = getstoichmatrix(modelObj)
[M,objSpecies,objReactions] = getstoichmatrix(modelObj)
```

## Arguments

|                     |   |
|---------------------|---|
| <i>M</i>            | Adjacency matrix for <i>modelObj</i> .  |
| <i>modelObj</i>     | Specify the model object <i>modelObj</i> .  |
| <i>objSpecies</i>   | Return the list of <i>modelObj</i> species by Name property of the species. If the species are in multiple compartments, species names are qualified with the compartment name in the form <code>compartmentName.speciesName</code> . For example, <code>nucleus.DNA</code> , <code>cytoplasm.mRNA</code> . |
| <i>objReactions</i> | Return the list of <i>modelObj</i> reactions by the Name property of reactions.   |

## Description

`getstoichmatrix` returns a stoichiometry matrix for a model object.

`M = getstoichmatrix(modelObj)` returns a stoichiometry matrix for a SimBiology model object (*modelObj*) to *M*.

A stoichiometry matrix is defined by listing all reactions contained by *modelObj* column-wise and all species contained by *modelObj* row-wise in a matrix. The species of the reaction are represented in the matrix with the stoichiometric value at the location of [row of species, column of reaction]. Reactants have negative values. Products have positive values. All other locations in the matrix are 0.

For example, if *modelObj* is a model object with two reactions with names R1 and R2 and Reaction values of  $2 A + B \rightarrow 3 C$  and  $B + 3 D \rightarrow 4 A$ , the stoichiometry matrix would be defined as:

|   | R1 | R2 |
|---|----|----|
| A | -2 | 4  |
| B | -1 | -1 |
| C | 3  | 0  |
| D | 0  | -3 |

`[M,objSpecies] = getstoichmatrix(modelObj)` returns the stoichiometry matrix to *M* and the species to *objSpecies*. *objSpecies* is defined by listing all Name property values of species contained by *Obj*. In the above example, *objSpecies* would be {'A', 'B', 'C', 'D'}.

`[M,objSpecies,objReactions] = getstoichmatrix(modelObj)` returns the stoichiometry matrix to *M* and the reactions to *objReactions*. *objReactions* is defined by listing all Name property values of reactions contained by *modelObj*. In the above example, *objReactions* would be {'R1', 'R2'}.

## Examples

**1** Read in *m1*, a model object, using `sbmlimport`:

```
m1 = sbmlimport('lotka.xml');
```

**2** Get the stoichiometry matrix for the *m1*:

```
[M,objSpecies,objReactions] = getstoichmatrix(m1)
```

## See Also

`getadjacencymatrix`, “Determining the Stoichiometry Matrix for a Model”

# getTable(ScheduleDose,RepeatDose)

---

|                         |   |
|-------------------------|---|
| <b>Purpose</b>          | Return data from SimBiology dose object as table  |
| <b>Syntax</b>           | <code>tbl = getTable(doseObj)</code>  |
| <b>Description</b>      | <code>tbl = getTable(doseObj)</code> returns dosing data from the dose object <code>doseObj</code> as a table <code>tbl</code> .  |
| <b>Input Arguments</b>  | <b>doseObj - Dose object</b><br>ScheduleDose object   RepeatDose object   array of dose objects<br><br>Dose object, specified as a ScheduleDose object or RepeatDose object or array of these objects.  |
| <b>Output Arguments</b> | <b>tbl - Dosing data</b><br>table   cell array of tables<br><br>Dosing data, returned as a table or cell array of tables. If <code>doseObj</code> is an array of dose objects, then <code>tbl</code> is a cell array of tables with the same size as <code>doseObj</code> .   |
| <b>Examples</b>         | <b>Retrieve a Table of Dosing Information from a RepeatDose Object</b><br><br>Create a RepeatDose object with some dosing information.<br><br><pre>rdose = sbiodose('rd','repeat'); rdose.TargetName = 'x'; rdose.StartTime = 5; rdose.TimeUnits = 'second'; rdose.Amount = 300; rdose.AmountUnits = 'molecule'; rdose.Rate = 1; rdose.RateUnits = 'molecule/second'; rdose.Interval = 100; rdose.RepeatCount = 2;</pre><br>Get a table of such dosing information. |

## getTable(ScheduleDose,RepeatDose)

```
tbl = getTable(rdose)
```

```
tbl =
```

| StartTime | Amount | Rate | Interval | RepeatCount |
|-----------|--------|------|----------|-------------|
| 5         | 300    | 1    | 100      | 2           |

Note that the units are also copied over and assigned to `tbl.Properties.VariableUnits` property.

```
tbl.Properties
```

```
ans =
```

```
      Description: ''
VariableDescriptions: {}
      VariableUnits: {'second' 'molecule' 'molecule/second' 's'}
      DimensionNames: {'Row' 'Variable'}
      UserData: []
      RowNames: {}
      VariableNames: {'StartTime' 'Amount' 'Rate' 'Interval'}
```

### Retrieve a Table of Dosing Information from a Schedule Object

Create a `ScheduleDose` object with some dosing information.

```
sdose = sbiodose('sdose','schedule');
sdose.Amount = [100 200 300];
sdose.Time = [5 10 15];
```

Get a table of such dosing information.

```
tbl = getTable(sdose)
```

```
tbl =
```

## getTable(ScheduleDose,RepeatDose)

---

| Time | Amount |
|------|--------|
| 5    | 100    |
| 10   | 200    |
| 15   | 300    |

### See Also

setTable | ScheduleDose object | RepeatDose object

**Purpose** Get variant from model

**Syntax**  
`variantObj = getvariant(modelObj)`  
`variantObj = getvariant(modelObj, 'NameValue')`

## Arguments

|                          |  |
|--------------------------|--|
| <code>variantObj</code>  | Variant object returned by the <code>getvariant</code> method.           |
| <code>modelObj</code>    | Model object from which to get the variant.                              |
| <code>'NameValue'</code> | Name of the variant to get from the model object <code>modelObj</code> . |

## Description

`variantObj = getvariant(modelObj)` returns SimBiology variant objects contained by the SimBiology model object `modelObj` to `variantObj`.

A SimBiology variant object stores alternate values for properties on a SimBiology model. For more information on variants, see `Variant` object.

`variantObj = getvariant(modelObj, 'NameValue')` returns the SimBiology variant object with the name `NameValue`, contained by the SimBiology model object, `modelObj`.

View properties for a variant object with the `get` command, and modify properties for a variant object with the `set` command.

---

**Note** Remember to use the `addcontent` method instead of using the `set` method on the `Content` property, because the `set` method replaces the data in the `Content` property whereas `addcontent` appends the data.

---

To copy a variant object to another model, use `copyobj`. To remove a variant object from a SimBiology model, use the `delete` method.

# getvariant (model)

---

## Examples

- 1 Create a model containing several variants.

```
modelObj = sbiomodel('mymodel');  
variantObj1 = addvariant(modelObj, 'v1');  
variantObj2 = addvariant(modelObj, 'v2');
```

- 2 Get all variants in the model.

```
vObjs = getvariant(modelObj)
```

SimBiology Variant Array

| Index: | Name: | Active: |
|--------|-------|---------|
| 1      | v1    | false   |
| 2      | v2    | false   |

- 3 Get the variant object named 'v2' from the model.

```
vObjv2 = getvariant(modelObj, 'v2');
```

## See Also

addvariant, removevariant



**Purpose** Table-like collection of data and metadata

**Description** The groupedData object is a table-like object that supports all the methods and properties of `table` and has two additional properties to identify the independent variable and an optional grouping variable. It also has additional methods that let you create doses from data set containing dosing data and convert a groupedData object to a table.

**Constructor Summary**

|                          |                              |
|--------------------------|------------------------------|
| <code>groupedData</code> | Create a groupedData object. |
|--------------------------|------------------------------|

**Method Summary** The groupedData object supports all methods of `table` and provides the following additional methods.

|   |   |
|---|---|
| <code>createDoses(groupedData)</code>       | Create dose objects from groupedData object |
| <code>groupedData2table(groupedData)</code> | Convert groupedData object to table         |

**Properties** The groupedData object supports all properties of `table` and provides the following additional properties.

|                                      |  |
|--------------------------------------|--|
| <code>GroupVariableName</code>       | A string indicating the name of grouping variable that indicates the groups in the data. To indicate there are no groups (or just one group), set the property to the empty string <code>''</code> . |
| <code>IndependentVariableName</code> | A string indicating the name of the independent variable in the data such as time. To indicate there is no independent variable, set the property to the empty string <code>''</code> .              |

# GroupedData object

---

## Examples

### Create a GroupedData Object from Data Set

This example uses data collected on 59 preterm infants given phenobarbital during the first 16 days after birth. Each infant received an initial dose followed by one or more sustaining doses by intravenous bolus administration. A total of between 1 and 6 concentration measurements were obtained from each infant at times other than dose times, for a total of 155 measurements. Infant weights and APGAR scores (a measure of newborn health) were also recorded. Data is described in [1], a study funded by the NIH/NIBIB grant P41-EB01975.

Load the sample dataset.

```
load pheno.mat ds
```

Create a groupedData object from the data set ds.

```
grpData = groupedData(ds);
```

Display the properties.

```
grpData.Properties
```

```
ans =
```

```
        Description: 'This dataset was generated using sbionmimp
```

```
FileNa...'
```

```
    VariableDescriptions: {}
```

```
    VariableUnits: {}
```

```
    DimensionNames: {'Observations' 'Variables'}
```

```
    UserData: []
```

```
    RowNames: {}
```

```
    VariableNames: {'ID' 'Time' 'Dose' 'Response'}
```

```
    GroupVariableName: 'ID'
```

```
    IndependentVariableName: 'Time'
```

GroupVariableName and IndependentVariableName have been automatically assigned to 'ID' and 'Time' respectively.

## References

[1] Grasela Jr, T.H., Donn, S.M. (1985) Neonatal population pharmacokinetics of phenobarbital derived from routine clinical data. *Dev Pharmacol Ther.* 8(6), 374–83.

## See Also

`table` | `groupedData` object | `sbiofit` | `sbiofitmixed`

# groupedData2table (groupedData)

---

|                         |  |
|-------------------------|--|
| <b>Purpose</b>          | Convert groupedData object to table  |
| <b>Syntax</b>           | <code>tbl = groupedData2table(grpData)</code>  |
| <b>Description</b>      | <code>tbl = groupedData2table(grpData)</code> converts a groupedData object <code>grpData</code> to a table. |
| <b>Input Arguments</b>  | <b>grpData - Grouped data</b><br>groupedData object<br>Grouped data, specified as a groupedData object.      |
| <b>Output Arguments</b> | <b>tbl - Data table</b><br>table<br>Data table, returned as a table.   |

## Examples **Convert a GroupedData Object to a Table**

This example uses data collected on 59 preterm infants given phenobarbital during the first 16 days after birth. Each infant received an initial dose followed by one or more sustaining doses by intravenous bolus administration. A total of between 1 and 6 concentration measurements were obtained from each infant at times other than dose times, for a total of 155 measurements. Infant weights and APGAR scores (a measure of newborn health) were also recorded. Data is described in [1], a study funded by the NIH/NIBIB grant P41-EB01975.

Load the sample dataset.

```
load pheno.mat ds
```

Create a groupedData object from the data set `ds`.

```
grpData = groupedData(ds);
```

Display the properties.

# groupedData2table (groupedData)

---

```
grpData.Properties
```

```
ans =
```

```
    Description: 'This dataset was generated using sbionm
```

```
FileNa...'
```

```
    VariableDescriptions: {}
    VariableUnits: {}
    DimensionNames: {'Observations' 'Variables'}
    UserData: []
    RowNames: {}
    VariableNames: {'ID' 'Time' 'Dose' 'Response'}
    GroupVariableName: 'ID'
    IndependentVariableName: 'Time'
```

GroupVariableName and IndependentVariableName have been automatically assigned to 'ID' and 'Time', respectively.

Convert grpData to a table.

```
tbl = groupedData2table(grpData);
```

## References

[1] Grasela Jr, T.H., Donn, S.M. (1985) Neonatal population pharmacokinetics of phenobarbital derived from routine clinical data. *Dev Pharmacol Ther.* 8(6), 374–83.

## See Also

table | groupedData object | sbiofit | sbiofitmixed

# isaccelerated(SimFunction)

---

**Purpose** Determine if SimFunction object is accelerated

**Syntax**  
`tf = isaccelerated(F)`  
`tf = isaccelerated(F,computerType)`

**Arguments**

|                           |  |
|---------------------------|--|
| <code>F</code>            | SimFunction object created by the <code>createSimFunction</code> method of a SimBiology model.           |
| <code>computerType</code> | String specifying a computer type. You can specify any valid archstr supported by the function computer. |

**Description**

`tf = isaccelerated(F)` returns true if SimFunction object F is accelerated for the current type of computer or false otherwise.

`tf = isaccelerated(F,computerType)` returns true if F is accelerated for the specified type of computer or false otherwise.

---

**Note** F is automatically accelerated at the first function execution. However, manually accelerate the object if you want it accelerated in your deployment applications.

---

## Examples **Simulate SimFunction Object**

This example uses the the Lotka-Volterra (predator-prey) model described by Gillespie [1].

Load the sample project containing the lotka model.

```
sbioloadproject lotka;
```

Create a SimFunction object f with c1 and c2 as input parameters to be scanned, and y1 and y2 as the output of the function with no dose.

```
f = createSimFunction(m1,{'Reaction1.c1', 'Reaction2.c2'},{'y1', 'y2'}, [
```

```
f =
```

```
SimFunction
```

```
Parameters:
```

| Name           | Value | Type        |
|----------------|-------|-------------|
| 'Reaction1.c1' | 10    | 'parameter' |
| 'Reaction2.c2' | 0.01  | 'parameter' |

```
Observables:
```

| Name | Type      |
|------|-----------|
| 'y1' | 'species' |
| 'y2' | 'species' |

```
Dosed: None
```

The `SimFunction` object `f` is not set for acceleration at the time of creation. But it will be automatically accelerated when executed.

```
f.isAccelerated
```

```
ans =
```

```
0
```

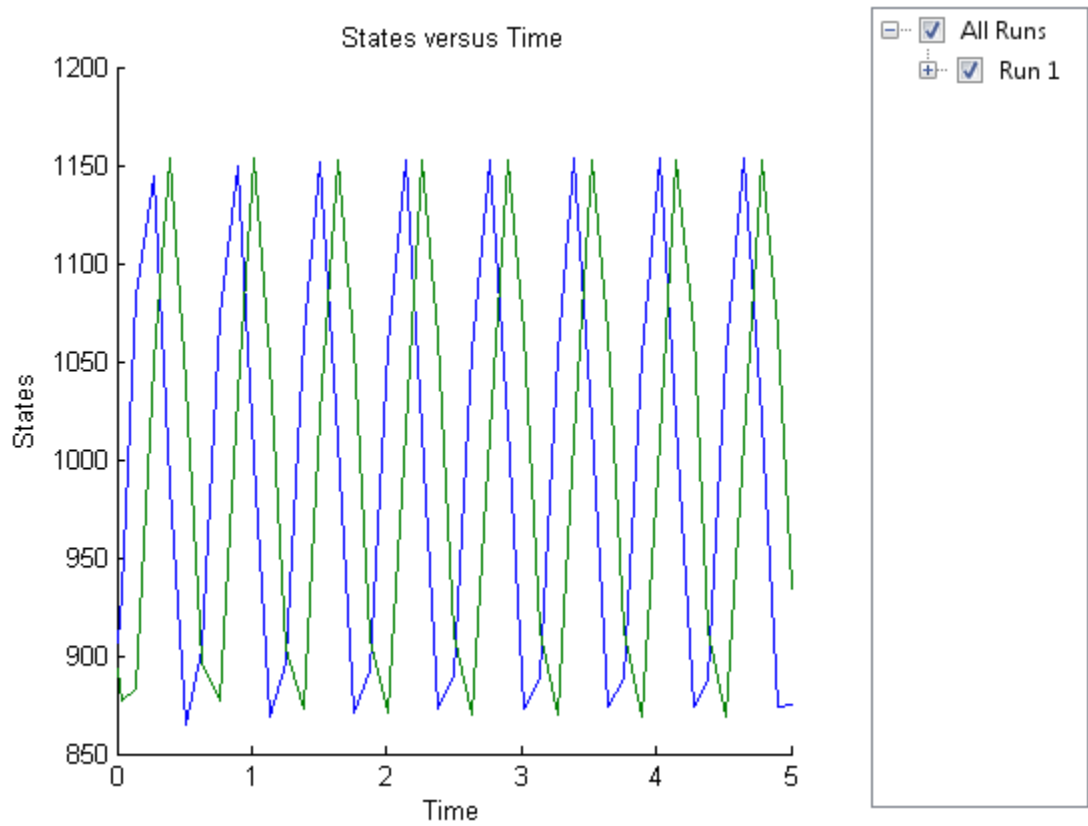
Define an input matrix that contains parameter values for `c1` and `c2`.

```
phi = [10 0.01];
```

Run simulations until the stop time is 5 and plot the simulation results.

```
sbioplot(f(phi,5))
```

# isaccelerated(SimFunction)



Confirm the SimFunction object `f` was accelerated during execution.

```
f.isAccelerated
```

```
ans =
```

```
1
```

## See Also

`createSimFunction`, SimFunction object



## References

- [1] Gillespie D.T. "Exact Stochastic Simulation of Coupled Chemical Reactions," (1977) The Journal of Physical Chemistry, 81(25), 2340-2361.

# SimBiology.export.Model.isAccelerated

---

**Purpose** Determine whether an exported SimBiology model is accelerated

**Syntax**  
`tf = isAccelerated(model)`  
`tf = isAccelerated(model,computerType)`

**Description**  
`tf = isAccelerated(model)` returns true if `model` is accelerated for the current type of computer, and false otherwise.  
`tf = isAccelerated(model,computerType)` returns true if `model` is accelerated for the specified computer type.

**Input Arguments**

**model**  
SimBiology.export.Model object.

**computerType**  
String specifying a computer type. You can specify any valid archstr supported by the function `computer`.

**Output Arguments**

**tf**  
Logical value true if `model` is accelerated for the current computer type, or computer type specified by `computerType`.  
Logical value false if the exported model is not accelerated for the specified computer type.

## Examples Accelerate Exported SimBiology Model

Load a sample SimBiology model object, and export.

```
modelObj = sbmlimport('lotka');  
em = export(modelObj)
```

```
em =
```

```
Model with properties:
```

```
    Name: 'lotka'
```

```
ExportTime: '12-Dec-2012 15:20:13'  
ExportNotes: ''
```

Accelerate the exported model.

```
accelerate(em);  
em.isAccelerated
```

```
ans =
```

```
1
```

The logical value 1 indicates that the exported model is accelerated.

## See Also

`SimBiology.export.Model` | `SimBiology.export.Model.accelerate`  
| `computer`

## Related Examples

- “PK/PD Modeling and Simulation to Guide Dosing Strategy for Antibiotics”
- “Deploy a SimBiology Model”

# KineticLaw object

---

**Purpose** Kinetic law information for reaction

**Description** The kinetic law object holds information about the abstract kinetic law applied to a reaction and provides a template for the reaction rate. In the model, the SimBiology software uses the information you provide in a fully defined kinetic law object to determine the `ReactionRate` property in the reaction object.

When you first create a kinetic law object, you must specify the name of the abstract kinetic law to use. The SimBiology software fills in the `KineticLawName` property and the `Expression` property in the kinetic law object with the name of the abstract kinetic law you specified and the mathematical expression respectively. The software also fills in the `ParameterVariables` property and the `SpeciesVariables` property of the kinetic law object with the values found in the corresponding properties of the abstract kinetic law object.

To obtain the reaction rate, you must fully define the kinetic law object:

- 1** In the `ParameterVariableNames` property, specify the parameters from the model that you want to substitute in the expression (`Expression` property).
- 2** In the `SpeciesVariableNames` property, specify the species from the model that you want to substitute in the expression.

The SimBiology software substitutes in the expression, the names of parameter variables and species variables in the order specified in the `ParameterVariables` and `SpeciesVariables` properties respectively.

The software then shows the substituted expression as the reaction rate in the `ReactionRate` property of the reaction object. If the kinetic law object is not fully defined, the `ReactionRate` property remains ' ' (empty).

For links to kinetic law object property reference pages, see “Property Summary” on page 2-175.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can interactively change object properties in the SimBiology desktop.

For an explanation of how relevant properties relate to one another, see “Command Line” on page 2-171.

The following sections use a kinetic law example to show how you can fully define your kinetic law object to obtain the reaction rate in the SimBiology desktop and at the command line.

The Henri-Michaelis-Menten kinetic law is expressed as follows:

$$V_m * S / (K_m + S)$$

In the SimBiology software Henri-Michaelis-Menten is a built-in abstract kinetic law, where  $V_m$  and  $K_m$  are defined in the `ParameterVariables` property of the abstract kinetic law object, and  $S$  is defined in the `SpeciesVariables` property of the abstract kinetic law object.

## SimBiology Desktop

To fully define a kinetic law in the SimBiology desktop, define the names of the species variables and parameter variables that participate in the reaction rate.

## Command Line

To fully define the kinetic law object at the command line, define the names of the parameters in the `ParameterVariableNames` property of the kinetic law object, and define the species names in the `SpeciesVariableNames` property of the kinetic law object. For example, to apply the Henri-Michaelis-Menten abstract kinetic law to a reaction

```
A -> B
where Vm = Va, Km = Ka
and S = A
```

## KineticLaw object

---

Define  $V_a$  and  $K_a$  in the `ParameterVariableNames` property to substitute the variables that are in the `ParameterVariables` property ( $V_m$  and  $K_m$ ). Define  $A$  in the `SpeciesVariableName` property to be used to substitute the species variable in the `SpeciesVariables` property ( $S$ ). Specify the order of the model parameters to be used for substitution in the same order that the parameter variables are listed in the `ParameterVariables` property. Similarly, specify species order if more than one species variable is represented.

```
% Find the order of the parameter variables
% in the kinetic law expression.

get(kineticlawObj, 'ParameterVariables')

ans =

    'Vm'    'Km'

% Find the species variable in the
% kinetic law expression

get(kineticlawObj, 'SpeciesVariables')
ans =

    'S'

% Specify the parameters and species variables
% to be used in the substitution.
% Remember to specify order, for example  $V_m = V_a$ 
%  $V_m$  is listed first in 'ParameterVariables',
% therefore list  $V_a$  first in 'ParameterVariableNames'.

set(kineticlawObj, 'ParameterVariableNames', {'Va' 'Ka'});
set(kineticlawObj, 'SpeciesVariableNames', {'A'});
```

The rate equation is assigned in the reaction object as follows:

$$V_a * A / (K_a + A)$$

For a detailed procedure, see “Examples” on page 2-176.

The following table summarizes the relationships between the properties in the abstract kinetic law object and the kinetic law object in the context of the above example.

| Property  | Property Purpose  | Abstract Kinetic Law Object  | Kinetic Law Object  |
|---|---|--|---|
| Name (abstract kinetic law object)<br>KineticLawName (kinetic law object) | Name of abstract kinetic law applied to a reaction. For example:<br><br>Henri-Michaelis-Menten                  | Read-only for built-in abstract kinetic law.<br>User-determined for user-defined abstract kinetic law. | Read-only   |
| Expression  | Mathematical expression used to determine the reaction rate equation. For example:<br><br>$V_m * S / (K_m + S)$ | Read-only for built-in abstract kinetic law.<br>User-determined for user-defined abstract kinetic law. | Read-only; depends on abstract kinetic law applied to reaction. |
| ParameterVariables  | Variables in Expression that are parameters. For example:<br><br>Vm and Km                                      | Read-only for built-in abstract kinetic law.<br>User-determined for user-defined abstract kinetic law. | Read-only; depends on abstract kinetic law applied to reaction. |

# KineticLaw object

| Property               | Property Purpose   | Abstract Kinetic Law Object   | Kinetic Law Object  |
|------------------------|--|---|---|
| SpeciesVariables       | Variables in Expression that are species. For example:<br><br>S              | Read-only for built-in abstract kinetic law. User-determined for user-defined abstract kinetic law. | Read-only; depends on abstract kinetic law applied to reaction. |
| ParameterVariableNames | Variables in ReactionRate that are parameters. For example:<br><br>Va and Ka | Not applicable  | Define these variables corresponding to ParameterVariables.     |
| SpeciesVariablesNames  | Variables in ReactionRate that are species. For example:<br><br>A            | Not applicable  | Define these variables corresponding to SpeciesVariables.       |

## Constructor Summary

addkineticlaw (reaction)

Create kinetic law object and add to reaction object

## Method Summary

addparameter (model, kineticlaw)

Create parameter object and add to model or kinetic law object

copyobj (any object)

Copy SimBiology object and its children

delete (any object)

Delete SimBiology object



|   |   |
|---|---|
| display (any object)                      | Display summary of SimBiology object              |
| get (any object)                          | Get object properties                             |
| getparameters (kineticlaw)                | Get specific parameters in kinetic law object     |
| getspecies (kineticlaw)                   | Get specific species in kinetic law object        |
| reorder (model, compartment, kinetic law) | Reorder component lists                           |
| set (any object)                          | Set object properties                             |
| setParameter (kineticlaw)                 | Specify specific parameters in kinetic law object |
| setspecies (kineticlaw)                   | Specify species in kinetic law object             |

## Property Summary

|   |  |
|---|--|
| Expression<br>(AbstractKineticLaw,<br>KineticLaw) | Expression to determine reaction rate equation |
| KineticLawName                                    | Name of kinetic law applied to reaction        |
| Name  | Specify name of object                         |
| Notes   | HTML text describing SimBiology object         |
| Parameters  | Array of parameter objects                     |
| ParameterVariableNames                            | Cell array of reaction rate parameters         |
| ParameterVariables                                | Parameters in kinetic law definition           |

# KineticLaw object

---

|                      |   |
|----------------------|---|
| Parent               | Indicate parent object                          |
| SpeciesVariableNames | Cell array of species in reaction rate equation |
| SpeciesVariables     | Species in abstract kinetic law                 |
| Tag                  | Specify label for SimBiology object             |
| Type                 | Display SimBiology object type                  |
| UserData             | Specify data to associate with object           |

## Examples

This example shows how to define the reaction rate for a reaction.

- 1 Create a model object, and add a reaction object to the model.

```
modelObj = sbiomodel ('my_model');  
reactionObj = addreaction (modelObj, 'A -> B');
```

- 2 Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

- 3 Query the parameters and species variables defined in the kinetic law.

```
get(kineticlawObj, 'ParameterVariables')
```

```
ans =
```

```
    'Vm'    'Km'
```

```
get(kineticlawObj, 'SpeciesVariables')
```

```
ans =
```

```
    'S'
```

- 4** Define  $V_a$  and  $K_a$  as `ParameterVariableNames`, which correspond to the `ParameterVariables`  $V_m$  and  $K_m$ . To set these variables, first create the parameter variables as parameter objects (`parameterObj1`, `parameterObj2`) with the names  $V_a$  and  $K_a$ , and then add them to `kineticlawObj`. The species object with Name  $A$  is created when `reactionObj` is created and need not be redefined.

```
parameterObj1 = addparameter(kineticlawObj, 'Va');  
parameterObj2 = addparameter(kineticlawObj, 'Ka');
```

- 5** Set the variable names for the kinetic law object.

```
set(kineticlawObj, 'ParameterVariableNames', {'Va' 'Ka'});  
set(kineticlawObj, 'SpeciesVariableNames', {'A'});
```

- 6** Verify that the reaction rate is expressed correctly in the reaction object `ReactionRate` property.

```
get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =
```

```
Va*A/(Ka+A)
```

### See Also

`AbstractKineticLaw` object, `Configset` object, `Model` object, `Parameter` object, `Reaction` object, `Root` object, `Rule` object, `Species` object

`SimBiology` property `Expression`(`AbstractKineticLaw`, `KineticLaw`)

# LeastSquaresResults object

---

## Purpose

Results object containing estimation results from least-squares regression

## Description

The `LeastSquaresResults` object is a superclass of two results objects: `NLINResults` object and `OptimResults` object. These objects contain estimation results from fitting a SimBiology model to data using `sbiofit` with any supported algorithm.

If `sbiofit` uses the `nlinfit` estimation algorithm, the results object is the `NLINResults` object. If `sbiofit` uses any other supporting algorithm, then the results object is an `OptimResults` object. See the `sbiofit` function for the list of supported algorithms.

## Method Summary

`boxplot(LeastSquaresResults,OptimResults,NLINResults)` showing the variation of estimated SimBiology model parameters

`fitted(LeastSquaresResults,OptimResults,NLINResults)` results of SimBiology model fitted using least-squares regression

`plot(LeastSquaresResults,OptimResults,NLINResults)` on results to the training data, creating a time-course subplot for each group

`plotActualVersusPredicted(LeastSquaresResults,OptimResults,NLINResults)` data, creating a subplot for each response

`plotResidualDistribution(LeastSquaresResults,OptimResults,NLINResults)` residuals

`plotResiduals(LeastSquaresResults,OptimResults,NLINResults)` response, using time, group, or prediction as *x*-axis

`predict(LeastSquaresResults,OptimResults,NLINResults)` fitted SimBiology model

random(LeastSquaresResults, OptimizationResults, NonRobust) model, adding variations by sampling error model

summary(LeastSquaresResults, OptimizationResults, NonRobust) that contains estimated values and estimation statistics

## Properties

|                    |   |
|--------------------|---|
| GroupName          | Categorical variable representing the name of the group associated with the results, or [] if the 'Pooled' name-value pair argument was set to true when you ran sbiofit.                                   |
| Beta               | Table of estimated parameters where the $j$ th row represents the $j$ th estimated parameter $\beta_j$ . It contains the transformed values of parameter estimates if any parameter transform is specified. |
| ParameterEstimates | Table of estimated parameters where the $j$ th row represents the $j$ th estimated parameter  |
| J                  | Jacobian matrix of the model, with respect to an estimated parameter, that is,  |
| COVB               | Estimated covariance matrix for Beta.   |
| CovarianceMatrix   | Estimated covariance matrix for ParameterEstimates.   |
| R                  | Residuals matrix where $R_i$ is the residual for the $i$ th time point and the $j$ th response in the space, and $y_{jt}$ is the $j$ th response in the group of data.                                      |
| MSE                | Weighted mean squared error.  |
| SSE                | Sum of squared (weighted) errors (residuals).   |

# LeastSquaresResults object

---

|                         |  |
|-------------------------|--|
| Weights                 | Matrix of weights with one column per response and one row per observation.  |
| EstimatedParameterNames | Cell array of strings specifying estimated parameter names.  |
| ErrorModel              | Name of the error model. There are four built-in error models. Each model defines the error using a standard mean-zero and unit-variance (Gaussian) variable $e$ , the function value $f$ , and one or two parameters $a$ and $b$ . In SimBiology, the function $f$ represents simulation results from a SimBiology model. |
| ErrorParameters         | Table of error model parameters.   |
| EstimationFunction      | Name of the estimation function.   |

## See Also

|                                   |                               |
|-----------------------------------|-------------------------------|
| NLINResults object   sbiofitmixed | OptimResults object   sbiofit |
|-----------------------------------|-------------------------------|

- 'constant':  $y = f + ae$
- 'proportional':  $y = f + b|f|e$
- 'combined':  $y = f + (a + b|f|)e$
- 'exponential':  $y = f * \exp(ae)$

## Purpose

Model and component information

## Description

The SimBiology model object represents a *model*, which is a collection of interrelated reactions and rules that transform, transport, and bind species. The model includes model components such as compartments, reactions, parameters, rules, and events. Each of the components is represented as a property of the model object. A model object also has a default configuration set object to define simulation settings. You can also add more configuration set objects to a model object.

See “Property Summary” on page 2-183 for links to model property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the SimBiology desktop.

You can retrieve SimBiology model objects from the SimBiology root object. A SimBiology model object has its `Parent` property set to the SimBiology root object. The root object contains a list of model objects that are accessible from the MATLAB command line and from the SimBiology desktop. Because both the command line and the desktop point to the same model object in the `Root` object, any changes you make to the model at the command line are reflected in the desktop, and vice versa.

## Constructor Summary

|                        |                        |
|------------------------|------------------------|
| <code>sbiomodel</code> | Construct model object |
|------------------------|------------------------|

## Method Summary

|  |                           |
|--|---------------------------|
| <code>addcompartment (model, compartment)</code> | Create compartment object |
|--|---------------------------|

|                                   |   |
|-----------------------------------|---|
| <code>addconfigset (model)</code> | Create configuration set object and add to model object |
|-----------------------------------|---|

|                              |                          |
|------------------------------|--------------------------|
| <code>adddose (model)</code> | Add dose object to model |
|------------------------------|--------------------------|

# Model object

---

|   |   |
|---|---|
| <code>addevent (model)</code>                 | Add event object to model object  |
| <code>addparameter (model, kineticlaw)</code> | Create parameter object and add to model or kinetic law object          |
| <code>addreaction (model)</code>              | Create reaction object and add to model object                          |
| <code>addrule (model)</code>                  | Create rule object and add to model object                              |
| <code>addspecies (model, compartment)</code>  | Create species object and add to compartment object within model object |
| <code>addvariant (model)</code>               | Add variant to model  |
| <code>copyobj (any object)</code>             | Copy SimBiology object and its children                                 |
| <code>createSimFunction (model)</code>        | Create SimFunction object   |
| <code>delete (any object)</code>              | Delete SimBiology object  |
| <code>display (any object)</code>             | Display summary of SimBiology object                                    |
| <code>export (model)</code>                   | Export SimBiology models for deployment and standalone applications     |
| <code>get (any object)</code>                 | Get object properties   |
| <code>getadjacencymatrix (model)</code>       | Get adjacency matrix from model object                                  |
| <code>getconfigset (model)</code>             | Get configuration set object from model object                          |
| <code>getdose (model)</code>                  | Return SimBiology dose object   |
| <code>getequations</code>                     | Return system equations for model object                                |



|   |   |
|---|---|
| getstoichmatrix (model)                   | Get stoichiometry matrix from model object    |
| getvariant (model)                        | Get variant from model                        |
| removeconfigset (model)                   | Remove configuration set from model           |
| removedose (model)                        | Add dose object to model                      |
| removevariant (model)                     | Remove variant from model                     |
| reorder (model, compartment, kinetic law) | Reorder component lists                       |
| set (any object)                          | Set object properties                         |
| setactiveconfigset (model)                | Set active configuration set for model object |
| verify (model, variant)                   | Validate and verify SimBiology model          |

## Property Summary

|              |   |
|--------------|---|
| Compartments | Array of compartments in model or compartment |
| Events       | Contain all event objects                     |
| Name         | Specify name of object                        |
| Notes        | HTML text describing SimBiology object        |
| Parameters   | Array of parameter objects                    |
| Parent       | Indicate parent object                        |
| Reactions    | Array of reaction objects                     |
| Rules        | Array of rules in model object                |
| Tag          | Specify label for SimBiology object           |

# Model object

---

|          |                                       |
|----------|---------------------------------------|
| Type     | Display SimBiology object type        |
| UserData | Specify data to associate with object |

## See Also

AbstractKineticLaw object, Configset object, KineticLaw object, Parameter object, Reaction object, Root object, Rule object, Species object

|                       |   |
|-----------------------|---|
| <b>Purpose</b>        | Estimation results object, subclass of LeastSquaresResults  |
| <b>Description</b>    | The NLINResults object contains estimation results from fitting a SimBiology model to data using sbiofit with nlinfit as a choice of estimation algorithm. See the sbiofit function for a list of other supported algorithms. |
| <b>Method Summary</b> | The NLINResults object shares all methods of the LeastSquaresResults object.  |
| <b>Properties</b>     | The NLINResults object shares all properties of the LeastSquaresResults object .  |
| <b>See Also</b>       | LeastSquaresResults object   OptimResults object   sbiofit   sbiofitmixed   |

# NLMEResults object

---

|                       |  |   |
|-----------------------|--|---|
| <b>Purpose</b>        | Results object containing estimation results from nonlinear mixed-effects modeling   |   |
| <b>Description</b>    | The <code>NLMEResults</code> object contains estimation results from fitting a nonlinear mixed-effects model using <code>sbiofitmixed</code> . |   |
| <b>Method Summary</b> | <code>boxplot(NLMEResults)</code>  | Create box plot showing the variation of estimated SimBiology model parameters  |
|                       | <code>covariateModel(NLMEResults)</code>   | Return a copy of the covariate model that was used for the nonlinear mixed-effects estimation using <code>sbiofitmixed</code> |
|                       | <code>fitted(NLMEResults)</code>   | Return the simulation results of a fitted nonlinear mixed-effects model   |
|                       | <code>plot(NLMEResults)</code>   | Compare simulation results to the training data, creating a time-course subplot for each group                                |
|                       | <code>plotActualVersusPredicted(NLMEResults)</code>  | Compare predictions to actual data, creating a subplot for each response  |
|                       | <code>plotResidualDistribution(NLMEResults)</code>   | Plot the distribution of the residuals  |
|                       | <code>plotResiduals(NLMEResults)</code>  | Plot the residuals for each response, using the time, group, or prediction as the x-axis                                      |

|                      |  |
|----------------------|--|
| predict(NLMEResults) | Simulate and evaluate fitted SimBiology model                              |
| random(NLMEResults)  | Simulate a SimBiology model, adding variations by sampling the error model |

## Properties

|                              |  |
|------------------------------|--|
| FixedEffects                 | Table of the estimated fixed effects and their standard errors.  |
| RandomEffects                | Table of the estimated random effects for each group.  |
| IndividualParameterEstimates | Table of estimated parameter values, including fixed and random effects.   |
| PopulationParameterEstimates | Table of estimated parameter values, including only fixed effects.   |
| RandomEffectCovarianceMatrix | Cell array of the covariance matrix of the random effects.   |
| stats                        | Struct of statistics returned by the <code>nlmefit</code> and <code>nlmefitsa</code> algorithm, such as AIC, BIC, rmse, and weighted residuals.  |
| CovariateNames               | Cell array of strings specifying covariate names.  |
| EstimatedParameterNames      | Cell array of strings specifying estimated parameter names.  |
| ErrorModel                   | Name of the error model. There are four built-in error models. Each model defines the error using a standard mean-zero and unit-variance (Gaussian) variable $e$ , the function value $f$ , and one or two parameters $a$ and $b$ . In SimBiology, the function represents simulation results from a SimBiology model. |
| ErrorParameters              | Table of error model parameters.   |
| EstimationFunction           | Name of the estimation function which must be either 'nlmefit' or 'nlmefitsa'.   |

## See Also

`sbiofitmixed` | `sbiofit` | `nlmefit` | `nlmefitsa`

- 'constant':  $y = f + ae$
- 'proportional':  $y = f + b|f|e$
- 'combined':  $y = f + (a + b|f|)e$
- 'exponential':  $y = f * \exp(ae)$

# OptimResults object

---

|                       |   |                       |   |                     |   |
|-----------------------|---|-----------------------|---|---------------------|---|
| <b>Purpose</b>        | Estimation results object, subclass of <code>LeastSquaresResults</code>   |                       |   |                     |   |
| <b>Description</b>    | The <code>OptimResults</code> object contains estimation results from fitting a SimBiology model to data using the <code>sbiofit</code> function with any supported algorithm except <code>nlinfit</code> . See the <code>sbiofit</code> function for a list of supported algorithms.   |                       |   |                     |   |
| <b>Method Summary</b> | The <code>OptimResults</code> object shares all methods of the <code>LeastSquaresResults</code> object.   |                       |   |                     |   |
| <b>Properties</b>     | The <code>OptimResults</code> object shares all properties of the <code>LeastSquaresResults</code> object and has the following additional properties.<br><br><table><tr><td><code>ExitFlag</code></td><td>An exit flag specific to the estimation function.</td></tr><tr><td><code>Output</code></td><td>A struct of additional outputs specific to the estimation function.</td></tr></table> | <code>ExitFlag</code> | An exit flag specific to the estimation function. | <code>Output</code> | A struct of additional outputs specific to the estimation function. |
| <code>ExitFlag</code> | An exit flag specific to the estimation function.   |                       |   |                     |   |
| <code>Output</code>   | A struct of additional outputs specific to the estimation function.   |                       |   |                     |   |

---

**Note** See the reference page of the specific algorithm to get more information on the values of `ExitFlag` and the `Output` structure.

---

**See Also** `LeastSquaresResults` object | `NLINResults` object | `sbiofit` | `sbiofitmixed`

## Purpose

Parameter and scope information

## Description

The parameter object represents a *parameter*, which is a quantity that can change or can be constant. SimBiology parameters are generally used to define rate constants. You can add parameter objects to a model object or a kinetic law object. The scope of a parameter depends on where you add the parameter object: If you add the parameter object to a model object, the parameter is available to all reactions in the model and the Parent property of the parameter object is `SimBiology.Model`. If you add the parameter object to a kinetic law object, the parameter is available only to the reaction for which you are using the kinetic law object and the Parent property of the parameter object is `SimBiology.KineticLaw`.

See “Property Summary” on page 2-190 for links to parameter object property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the graphical user interface.

## Constructor Summary

|  |  |
|--|--|
| <code>addparameter(model, kineticlaw)</code> | Create parameter object and add to model or kinetic law object |
|--|--|

## Method Summary

|                                  |   |
|----------------------------------|---|
| <code>copyobj(any object)</code> | Copy SimBiology object and its children |
| <code>delete(any object)</code>  | Delete SimBiology object                |
| <code>display(any object)</code> | Display summary of SimBiology object    |
| <code>get(any object)</code>     | Get object properties                   |

# Parameter object

---

|  |                                      |
|--|--------------------------------------|
| rename (compartment, parameter, species, reaction) | Rename object and update expressions |
| set (any object)                                   | Set object properties                |

## Property Summary

|               |  |
|---------------|--|
| ConstantValue | Specify variable or constant parameter value |
| Name          | Specify name of object                       |
| Notes         | HTML text describing SimBiology object       |
| Parent        | Indicate parent object                       |
| Tag           | Specify label for SimBiology object          |
| Type          | Display SimBiology object type               |
| UserData      | Specify data to associate with object        |
| Value         | Assign value to parameter object             |
| ValueUnits    | Parameter value units                        |

## See Also

AbstractKineticLaw object, Configset object, KineticLaw object, Model object, Reaction object, Root object, Rule object, Species object



## Purpose

Used by PKModelDesign to create SimBiology model

## Description

The PKCompartment object is used by the PKModelDesign object to construct a SimBiology model for pharmacokinetic modeling. PKCompartment holds the following information:

- Name of the compartment
- Dosing type
- Elimination type
- Whether the drug concentration in this compartment is reported

The PKCompartment class is a subclass of the hgsetget class which is a subclass of the handle class. For more information on the inherited methods, see hgsetget, and handle.

## Construction

|                                   |  |
|-----------------------------------|--|
| addCompartment<br>(PKModelDesign) | Add compartment to<br>PKModelDesign object |
|-----------------------------------|--|

## Method Summary

|                  |                       |
|------------------|-----------------------|
| get (any object) | Get object properties |
| set (any object) | Set object properties |

## Property Summary

|                     |   |
|---------------------|---|
| DosingType          | Drug dosing type in compartment                   |
| EliminationType     | Drug elimination type from<br>compartment         |
| HasLag              | Lag associated with dose<br>targeting compartment |
| HasResponseVariable | Compartment drug concentration<br>reported        |
| Name                | Specify name of object                            |

## PKCompartment object

---

### **See Also**

“Create Pharmacokinetic Models” in the SimBiology User’s Guide,  
PKModelDesign object

**Purpose** Define roles of data set columns

**Compatibility** PKData object will be removed in a future release. Use groupedData object instead.

**Description** The properties of the PKData object specify what each column in the data represents. The PKData object specifies which columns in the data set represent the following:

- The grouping variable
- The independent and dependent variables
- The dose
- The rate (only if infusion is the dosing type)
- The covariates

This information is used by the fitting functions, `sbionlmeFit` and `sbionlinfit`.

To create the PKData object specify:

```
pkDataObject = PKData(data);
```

Where `data` is the imported data set.

The PKData class is a subclass of the `hgsetget` class, which is a subclass of the `handle` class. For more information on the inherited methods, see `hgsetget` and `handle`.

**Construction** PKData Create PKData object

# PKData object

---

## Method Summary

|                           |                                     |
|---------------------------|-------------------------------------|
| get (any object)          | Get object properties               |
| getCovariateData (pkdata) | Create design matrix needed for fit |
| set (any object)          | Set object properties               |

## Property Summary

|                     |  |
|---------------------|--|
| CovariateLabels     | Identify covariate columns in data set           |
| DataSet             | Dataset object containing imported data          |
| DependentVarLabel   | Identify dependent variable column in data set   |
| DependentVarUnits   | Response units in PKData object                  |
| DoseLabel           | Dose column in data set                          |
| DoseUnits           | Dose units in PKData object                      |
| GroupID             | Integer identifying each group in data set       |
| GroupLabel          | Identify group column in data set                |
| GroupNames          | Unique values from GroupLabel in data set        |
| IndependentVarLabel | Identify independent variable column in data set |
| IndependentVarUnits | Time units in PKData object                      |
| RateLabel           | Rate of infusion column in data set              |
| RateUnits           | Units for dose rate                              |

**See Also**

“Specify and Classify the Data to Fit” in the SimBiology User’s Guide,  
PKModelDesign object

# PKModelDesign object

---

**Purpose** Helper object to construct pharmacokinetic model

**Description** Use the PKModelDesign object to construct a SimBiology model for PK modeling. The PKModelDesign object lets you specify the number of compartments, the type of dosing, and method of elimination which you then use to construct the SimBiology model object with the necessary compartments, species, reactions, rules, and events.

```
pkm = PKModelDesign;
```

Use the `addCompartment` method to add a compartment with a specified dosing and elimination. `addCompartment` adds each subsequent compartment and connects it to the previous compartment using a reversible reaction. This reaction models the flux between compartments in a PK model.

The `construct` method uses the PKModelDesign object to create a SimBiology model object.

The PKModelDesign class is a subclass of the `hgsetget` class, which is a subclass of the `handle` class. For more information on the inherited methods see `hgsetget` and `handle`.

**Construction** PKModelDesign Create PKModelDesign object

|                       |  |  |
|-----------------------|--|--|
| <b>Method Summary</b> | <code>addCompartment</code><br>(PKModelDesign) | Add compartment to PKModelDesign object              |
|                       | <code>construct</code> (PKModelDesign)         | Construct SimBiology model from PKModelDesign object |
|                       | <code>get</code> (any object)                  | Get object properties                                |
|                       | <code>set</code> (any object)                  | Set object properties                                |

## Property Summary

PKCompartments

Hold compartments in PK model

## See Also

“Create Pharmacokinetic Models” in the SimBiology User’s Guide,  
PKCompartment object

# PKModelMap object

---

**Purpose** Define SimBiology model components' roles

**Compatibility** PKModelMap object will be removed in a future release. Use a combination of estimatedInfo object, CovariateModel object, cell array of strings, and sbiodose. See sbiofit and sbiofitmixed for illustrated examples.

**Description** The PKModelMap object holds information about the dosing type, and defines which components of a SimBiology model represent the observed response, the dose, and the estimated parameters.

The PKModelMap class is a subclass of the hgsetget class which is a subclass of the handle class. For more information on the inherited methods see, hgsetget, and handle.

**Construction** PKModelMap Create PKModelMap object

**Method Summary**

|                  |                       |
|------------------|-----------------------|
| get (any object) | Get object properties |
| set (any object) | Set object properties |

**Property Summary**

|                            |   |
|----------------------------|---|
| Dosed                      | Dosed object name                       |
| DosingType                 | Drug dosing type in compartment         |
| Estimated                  | Names of parameters to estimate         |
| LagParameter               | Parameter specifying time lag for doses |
| Observed                   | Measured response object name           |
| ZeroOrderDurationParameter | Zero-order dose absorption duration     |



**See Also**

“Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters” in the SimBiology User’s Guide, PKModelDesign object

# plot(LeastSquaresResults, OptimResults, NLINResults)

---

|                        |   |
|------------------------|---|
| <b>Purpose</b>         | Compare simulation results to the training data, creating a time-course subplot for each group  |
| <b>Syntax</b>          | <code>plot(resultsObj)</code>   |
| <b>Description</b>     | <code>plot(resultsObj)</code> returns a figure displaying the comparison between simulation results to the training data, with a time-course subplot for each group.  |
| <b>Input Arguments</b> | <b>resultsObj - Estimation results</b><br><code>OptimResults</code> object   <code>NLINResults</code> object   vector of results objects<br>Estimation results, specified as an <code>OptimResults</code> object or <code>NLINResults</code> object, or vector of results objects which contains estimation results from running <code>sbiofit</code> . |
| <b>See Also</b>        | <code>NLINResults</code> object   <code>OptimResults</code> object   <code>sbiofit</code>   |

|                        |   |
|------------------------|---|
| <b>Purpose</b>         | Compare simulation results to the training data, creating a time-course subplot for each group  |
| <b>Syntax</b>          | <pre>plot(resultsObj) plot(resultsObj, 'ParameterType', value)</pre>  |
| <b>Description</b>     | <p><code>plot(resultsObj)</code> compares simulation results to the training data, creating a time-course subplot for each group.</p> <p><code>plot(resultsObj, 'ParameterType', value)</code> uses the individual or population parameter estimates as specified by <code>value</code>. The two choices for <code>value</code> are 'population' or 'individual' (default).</p>       |
| <b>Input Arguments</b> | <p><b>resultsObj - Estimation results</b><br/>NLMEResults object</p> <p>Estimation results, specified as an NLMEResults object, which contains estimation results returned by <code>sbiofitmixed</code>.</p> <p><b>value - Parameter type</b><br/>string</p> <p>Parameter type, specified as a string which must be one of the following: 'individual' (default) or 'population'.</p> |
| <b>See Also</b>        | NLMEResults object   sbiofitmixed   |

# plotActualVersusPredicted(LeastSquaresResults,OptimResults,N

---

**Purpose** Compare predictions to actual data, creating a subplot for each response

**Syntax** `plotActualVersusPredicted(resultsObj)`

**Description** `plotActualVersusPredicted(resultsObj)` returns a figure displaying the comparison between predictions to the actual data, with a subplot for each response.

**Input Arguments** **resultsObj - Estimation results**  
`OptimResults` object | `NLINResults` object | vector of results objects  
Estimation results, specified as an `OptimResults` object or `NLINResults` object, or vector of results objects which contains estimation results from running `sbiofit`.

**See Also** `NLINResults` object | `OptimResults` object | `sbiofit`

# plotActualVersusPredicted(NLMEResults)

---

|                        |   |
|------------------------|---|
| <b>Purpose</b>         | Compare predictions to actual data, creating a subplot for each response  |
| <b>Syntax</b>          | <code>plotActualVersusPredicted(resultsObj)</code>  |
| <b>Description</b>     | <code>plotActualVersusPredicted(resultsObj)</code> returns a figure displaying the comparison between predictions to the actual data, with a subplot for each response.                               |
| <b>Input Arguments</b> | <b>resultsObj - Estimation results</b><br>NLMEResults object<br><br>Estimation results, specified as an NLMEResults object, which contains estimation results returned by <code>sbiofitmixed</code> . |
| <b>See Also</b>        | NLMEResults object   <code>sbiofitmixed</code>  |

# plotResiduals(LeastSquaresResults,OptimResults,NLINResults)

---

|                        |   |
|------------------------|---|
| <b>Purpose</b>         | Plot residuals for each response, using time, group, or prediction as x-axis  |
| <b>Syntax</b>          | <code>plotResiduals(resultsObj,type)</code>   |
| <b>Description</b>     | <code>plotResiduals(resultsObj,type)</code> plots the residuals for each response, using the time, group, or predictions as the x-axis as specified by the argument <code>type</code> .   |
| <b>Input Arguments</b> | <p><b>resultsObj - Estimation results</b><br/>OptimResults object   NLINResults object</p> <p>Estimation results, specified as an OptimResults object or NLINResults object, which contains estimation results returned by sbiofit.</p> <p><b>type - x-axis option for residual plot</b><br/>string</p> <p>X-axis option for the residual plot, specified as a string that must be one of the following: 'Time', 'Group', or 'Predictions'.</p> |
| <b>See Also</b>        | NLINResults object   OptimResults object   sbiofit  |

|                        |   |
|------------------------|---|
| <b>Purpose</b>         | Plot the residuals for each response, using the time, group, or prediction as the x-axis  |
| <b>Syntax</b>          | <code>plotResiduals(resultsObj, type)</code>  |
| <b>Description</b>     | <code>plotResiduals(resultsObj, type)</code> plots the residuals for each response, using the time, group, or predictions as the x-axis as specified by the argument <code>type</code> .  |
| <b>Input Arguments</b> | <p><b>resultsObj - Estimation results</b><br/>NLMEResults object</p> <p>Estimation results, specified as an NLMEResults object, which contains estimation results returned by <code>sbiofitmixed</code>.</p> <p><b>type - X-axis option for residual plot</b><br/>string</p> <p>X-axis option for the residual plot, specified as a string which must be one of the following: 'Time', 'Group', or 'Predictions'.</p> |
| <b>See Also</b>        | NLMEResults object   <code>sbiofitmixed</code>  |

# plotResidualDistribution(LeastSquaresResults, OptimResults, NLINResults)

---

|                        |   |
|------------------------|---|
| <b>Purpose</b>         | Plot the distribution of the residuals  |
| <b>Syntax</b>          | <code>plotResidualDistribution(resultsObj)</code>   |
| <b>Description</b>     | <code>plotResidualDistribution(resultsObj)</code> plots the distribution of the residuals.  |
| <b>Input Arguments</b> | <b>resultsObj - Estimation results</b><br><code>OptimResults</code> object   <code>NLINResults</code> object   vector of results objects<br>Estimation results, specified as an <code>OptimResults</code> object or <code>NLINResults</code> object, or vector of results objects which contains estimation results from running <code>sbiofit</code> . |
| <b>See Also</b>        | <code>NLINResults</code> object   <code>OptimResults</code> object   <code>sbiofit</code>   |



# plotResidualDistribution(NLMEResults)

---

|                        |  |
|------------------------|--|
| <b>Purpose</b>         | Plot the distribution of the residuals   |
| <b>Syntax</b>          | <code>plotResidualDistribution(resultsObj)</code>  |
| <b>Description</b>     | <code>plotResidualDistribution(resultsObj)</code> plots the distribution of the residuals.   |
| <b>Input Arguments</b> | <b>resultsObj - Estimation results</b><br>NLMEResults object<br><br>Estimation results, specified as an NLMEResults object, which contains estimation results from running <code>sbiofitmixed</code> . |
| <b>See Also</b>        | NLMEResults object   <code>sbiofitmixed</code>   |

# predict(LeastSquaresResults, OptimResults, NLINResults)

---

|                        |  |
|------------------------|--|
| <b>Purpose</b>         | Simulate and evaluate fitted SimBiology model  |
| <b>Syntax</b>          | <pre>[ypred,parameterEstimates]= predict(resultsObj) [ypred,parameterEstimates]= predict(resultsObj,data,dosing)</pre>   |
| <b>Description</b>     | <p><code>[ypred,parameterEstimates]= predict(resultsObj)</code><br/>returns simulation results <code>ypred</code> and parameter estimates <code>parameterEstimates</code> of a fitted SimBiology model.</p> <p><code>[ypred,parameterEstimates]= predict(resultsObj,data,dosing)</code><br/>returns simulation results <code>ypred</code> and estimated parameter values <code>parameterEstimates</code> from evaluating the fitted SimBiology model using the specified <code>data</code> and <code>dosing</code> information.</p> <p>Use this method when you want to evaluate the fitted model and predict responses using new data and/or dosing information.</p>  |
| <b>Input Arguments</b> | <p><b>resultsObj - Estimation results</b><br/>OptimResults object   NLINResults object</p> <p>Estimation results, specified as an OptimResults object or NLINResults object, which contains estimation results returned by sbiofit. It must be a scalar object.</p> <p><b>data - Grouped data or output times</b><br/>groupedData object   vector   cell array of vectors</p> <p>Grouped data or output times, specified as a groupedData object, vector, or cell array of vectors of output times.</p> <p><b>dosing - Dosing information</b><br/>[]   2-D matrix of SimBiology dose objects</p> <p>Dosing information, specified as an empty array [] or 2-D matrix of SimBiology dose objects (ScheduleDose object or RepeatDose object). It must be consistent with the original dosing data used with sbiofit, i.e., dose objects in dosing must have the same TargetName, LagParameterName, or DurationParameterName properties as those of the original dosing data.</p> |

# predict(LeastSquaresResults, OptimResults, NLINResults)

If empty, no doses are applied during simulation. If not empty, the matrix must have a single row or one row per group in the data `data`. If it has a single row, the same doses are applied to all groups during simulation. If it has multiple rows, each row is applied to a separate group, in the same order as the groups appear in `data`. Multiple columns are allowed so that you can apply multiple dose objects to each group. Each column of doses must reference the same components in the model. Specifically, all doses (except for empty doses) in a column must have the same values for `TargetName`, `DurationParameterName`, and `LagParameterName`. If some groups require more doses than others, then fill in the matrix with dummy doses that are either default doses or empty doses.

## Output Arguments

### **ypred - Simulation results**

vector of `SimData` objects

Simulation results, returned as a vector of `SimData` objects. The states reported in `ypred` are the states that were included in the `responseMap` input argument of `sbiofit` as well as any other states listed in the `StatesToLog` property of the runtime options (`RuntimeOptions`) of the `SimBiology` model.

### **parameterEstimates - Estimated parameter values**

table

Estimated parameter values, returned as a table. This is identical to `resultsObj.ParameterEstimates` property.

## Examples

### **Estimate a Parameter from the Yeast G Protein Model**

This example uses the yeast heterotrimeric G protein model and experimental data reported by [1]. For details about the model, see the **Background** section in “Parameter Scanning, Parameter Estimation, and Sensitivity Analysis in the Yeast Heterotrimeric G Protein Cycle”.

Load the G protein model.

```
sbioloadproject gprotein
```

# predict(LeastSquaresResults, OptimResults, NLINResults)

---

Enter the experimental data containing the time course for the fraction of active G protein, as reported in the reference paper [1].

```
time = [0 10 30 60 110 210 300 450 600]';  
GaFracExpt = [0 0.35 0.4 0.36 0.39 0.33 0.24 0.17 0.2]';
```

Create a groupedData object based on the experimental data.

```
tbl = table(time, GaFracExpt);  
grpData = groupedData(tbl);
```

Map the appropriate model component to the experimental data. In other words, indicate which species in the model corresponds to which response variable in the data. In this example, map the model parameter GaFrac to the experimental data variable GaFracExpt from grpData.

```
responseMap = 'GaFrac = GaFracExpt';
```

Use an estimatedInfo object to define the model parameter kGd as a parameter to be estimated.

```
estimatedParam = estimatedInfo('kGd');
```

Perform the parameter estimation.

```
fitResult = sbiofit(m1, grpData, responseMap, estimatedParam);
```

View the estimated parameter value of kGd.

```
fitResult.ParameterEstimates
```

```
ans =
```

| Name  | Estimate | StandardError |
|-------|----------|---------------|
| 'kGd' | 0.11     | 0.00037969    |

## **predict(LeastSquaresResults,OptimResults,NLINResults)**

---

Suppose you have obtained three more experimental data points, and want to see how the fitted model performs on these data points. You can create a new `groupedData` object containing these data points, and use the `predict` method to compare experimental data with predicted results.

Create a new `groupedData` object with three new experimental data points.

```
TIME = [0 10 30 60 110 210 300 450 600 700 850 900]';  
GaFracExptNew = [0 0.35 0.4 0.36 0.39 0.33 0.24 0.17 0.2 0.18 0.14 0.1];  
newgrpData = groupedData(table(TIME,GaFracExptNew));
```

Use the `predict` method to simulate the fitted model on the new data. Note that no dosing was specified when you first ran `sbiofit`. Hence, you cannot use any dosing information with the `predict` method, and an empty array must be specified as the third input argument.

```
[ypred,paramestim] = predict(fitResult,newgrpData,[]);
```

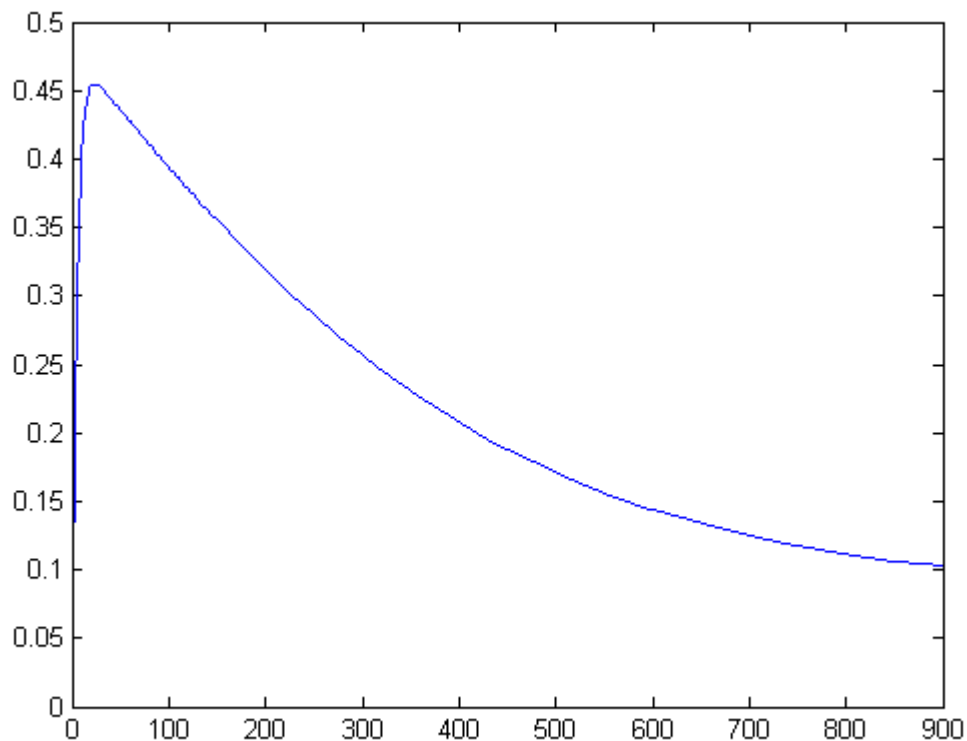
Select simulation results for the `GaFrac` parameter only.

```
GaFracPred = select(ypred,{'Name','GaFrac'});
```

Plot the simulated data.

```
plot(GaFracPred.Time,GaFracPred.Data);  
hold on
```

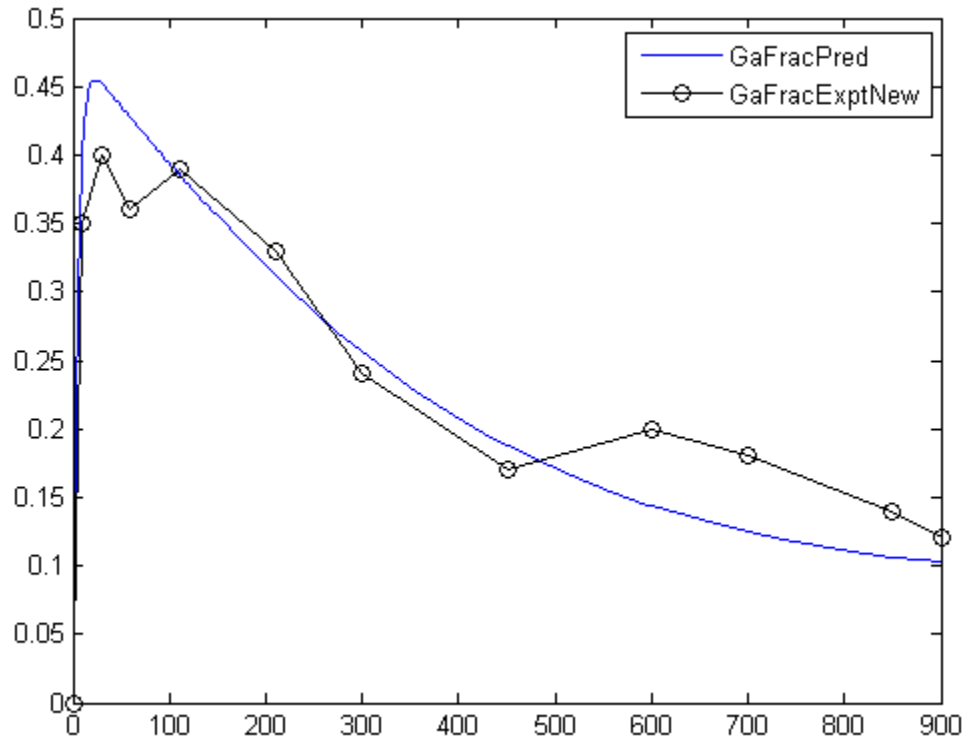
## predict(LeastSquaresResults,OptimResults,NLINResults)



Plot the experimental data to compare it with the simulated data.

```
plot(TIME,GaFracExptNew,'Color','k','Marker','o');  
legend('GaFracPred','GaFracExptNew')
```

# predict(LeastSquaresResults,OptimResults,NLINResults)



## References

- [1] Yi, T-M., Kitano, H., and Simon, M. (2003). A quantitative characterization of the yeast heterotrimeric G protein cycle. PNAS. 100, 10764–10769.

## See Also

NLINResults object | OptimResults object | sbiofit

# predict(NLMEResults)

---

**Purpose** Simulate and evaluate fitted SimBiology model

**Syntax**

```
[ypred,parameterEstimates] = predict(resultsObj)
[ypred,parameterEstimates] =
predict(resultsObj,data,dosing)
[ypred,parameterEstimates] =
predict(_, 'ParameterType',value)
```

**Description**

[ypred,parameterEstimates] = predict(resultsObj) returns simulation results ypred of the fitted SimBiology model using estimated parameter values parameterEstimates, which are returned by sbiofitmixed.

[ypred,parameterEstimates] = predict(resultsObj,data,dosing) returns simulation results ypred from evaluating the fitted SimBiology model using the specified data and dosing information. The data can be different from the training data that were used to fit the model.

[ypred,parameterEstimates] = predict(\_, 'ParameterType',value) returns simulation results that are simulated using either individual or population parameter estimates. The two choices for value are 'population' or 'individual' (default).

---

**Tip** Use this method to evaluate the fitted model and predict responses using additional (new) data and/or dosing information.

---

## Input Arguments

### **resultsObj - Estimation results**

NLMEResults object

Estimation results, specified as an NLMEResults object, which contains nonlinear mixed-effects estimation results returned by sbiofitmixed. It must be a scalar object.

### **data - Grouped data or output times**



groupedData object | vector | cell array of vectors

Grouped data or output times, specified as a `groupedData` object, vector, or cell array of vectors of output times.

If the mixed-effects model from the original fit (using `sbiofitmixed`) uses a covariate model with covariates, the `data` must be a `groupedData` object containing covariate data with the same labels for the covariates (`CovariateLabels` property) specified in the original covariate model.

If data contains any new groups, their parameter values are calculated using the covariate model with fixed effects only, without any random effects. Specifying data as a vector or cell array of vectors is treated as specifying different groups than in the original data that was used with `sbiofitmixed`.

## **dosing - Dosing information**

[] | 2-D matrix of SimBiology dose objects

Dosing information, specified as an empty array [] or 2-D matrix of SimBiology dose objects (`ScheduleDose` object or `RepeatDose` object). It must be consistent with the original dosing data used with `sbiofitmixed`, i.e., dose objects in `dosing` must have the same `TargetName`, `LagParameterName`, or `DurationParameterName` properties as those of the original dosing data.

If empty, no doses are applied during simulation. If not empty, the matrix must have a single row or one row per group in the data `data`. If it has a single row, the same doses are applied to all groups during simulation. If it has multiple rows, each row is applied to a separate group, in the same order as the groups appear in `data`. Multiple columns are allowed so that you can apply multiple dose objects to each group. Each column of doses must reference the same components in the model. Specifically, all doses (except for empty doses) in a column must have the same values for `TargetName`, `DurationParameterName`, and `LagParameterName`. If some groups require more doses than others, then fill in the matrix with dummy doses that are either default doses or empty doses.

# predict(NLMEResults)

---

## **value - Parameter type**

string

Parameter type, specified as 'population' or 'individual' (default). If `value` is 'population', the method returns the simulation results using the population parameter estimates. The estimated parameter values used in simulation are identical to `resultsObj.PopulationParameterEstimates` property unless you specify a new `groupedData` object `data` with new covariate data. In this case, the method will reevaluate the covariate model and this could change the parameter estimates.

If `value` is 'individual', the method returns the simulation results using the corresponding group's parameter values in the `resultsObj.IndividualParameterEstimates` property, which includes both fixed and random effect estimates. If data contains new groups, a warning is issued that no random effect estimates are used, and only fixed effects (population parameter estimates) are used for these groups.

## **Output Arguments**

### **ypred - Simulation results**

vector of `SimData` objects

Simulation results, returned as a vector of `SimData` objects. The states reported in `ypred` are the states that were included in the `responseMap` input argument of `sbiofitmixed` as well as any other states listed in the `StatesToLog` property of the runtime options (`RuntimeOptions`) of the `SimBiology` model.

### **parameterEstimates - Estimated parameter values**

table

Estimated parameter values, returned as a table.

These estimated values will differ from those values from the original fit if you specify the `value` argument as 'individual' since parameter values are recalculated using `sbiosampleparameters`.

If 'ParameterType' is 'population', the estimated parameter values are identical to the `resultsObj.PopulationParameterEstimates`

property unless you specify a new groupedData object data with new covariate data.

### **See Also**

NLMEResults object | sbiofitmixed | sbiosampleparameters | sbiosampleerror

# random(LeastSquaresResults, OptimResults, NLINResults)

---

**Purpose** Simulate SimBiology model, adding variations by sampling error model

**Syntax**  
[ynew,parameterEstimates] = random(resultsObj)  
[ynew,parameterEstimates] = random(resultsObj,data,dosing)

**Description** [ynew,parameterEstimates] = random(resultsObj) returns simulation results ynew with added noise using the error model specified by the resultsObj.ErrorModel property and estimated parameter values parameterEstimates.  
[ynew,parameterEstimates] = random(resultsObj,data,dosing) uses the specified data and dosing information.

---

**Note** The noise is only added to states that are responses which are the states included in the responseMap input argument when you called sbiofit.

---

## Input Arguments

### resultsObj - Estimation results

OptimResults object | NLINResults object

Estimation results, specified as an OptimResults object or NLINResults object, which contains estimation results returned by sbiofit. It must be a scalar object.

### data - Grouped data or output times

groupedData object | vector | cell array of vectors

Grouped data or output times, specified as a groupedData object, vector, or cell array of vectors of output times.

### dosing - Dosing information

[] | 2-D matrix of SimBiology dose objects

Dosing information, specified as an empty array [] or 2-D matrix of SimBiology dose objects (ScheduleDose object or RepeatDose object). It must be consistent with the original dosing data used with

# random(LeastSquaresResults, OptimResults, NLINResults)

---

sbiofit, i.e., dose objects in `dosing` must have the same `TargetName`, `LagParameterName`, or `DurationParameterName` properties as those of the original dosing data.

If empty, no doses are applied during simulation. If not empty, the matrix must have a single row or one row per group in the data `data`. If it has a single row, the same doses are applied to all groups during simulation. If it has multiple rows, each row is applied to a separate group, in the same order as the groups appear in `data`. Multiple columns are allowed so that you can apply multiple dose objects to each group. Each column of doses must reference the same components in the model. Specifically, all doses (except for empty doses) in a column must have the same values for `TargetName`, `DurationParameterName`, and `LagParameterName`. If some groups require more doses than others, then fill in the matrix with dummy doses that are either default doses or empty doses.

## Output Arguments

### **ynew - Simulation results with noise**

vector of `SimData` objects

Simulation results with noise, returned as a vector of `SimData` objects. The states reported in `ynew` are the states that were included in the `responseMap` input argument of `sbiofit` as well as any other states listed in the `StatesToLog` property of the runtime options (`RuntimeOptions`) of the `SimBiology` model.

### **parameterEstimates - Estimated parameter values**

table

Estimated parameter values, returned as a table. This is identical to `resultsObj.ParameterEstimates` property.

## Examples

### **Add Noise to Simulation Results of a Fitted SimBiology Model**

This example uses the yeast heterotrimeric G protein model and experimental data reported by [1]. For more details about the model, see the **Background** section in “Parameter Scanning, Parameter Estimation, and Sensitivity Analysis in the Yeast Heterotrimeric G Protein Cycle”.

# random(LeastSquaresResults, OptimResults, NLINResults)

---

Load the G protein model.

```
sbioloadproject gprotein
```

Enter the experimental data containing the time course for the fraction of active G protein, as reported in the reference paper [1].

```
time = [0 10 30 60 110 210 300 450 600]';  
GaFracExpt = [0 0.35 0.4 0.36 0.39 0.33 0.24 0.17 0.2]';
```

Create a groupedData object based on the experimental data.

```
tbl = table(time, GaFracExpt);  
grpData = groupedData(tbl);
```

Map the appropriate model component to the experimental data. In other words, indicate which species in the model corresponds to which response variable in the data. In this example, map the model parameter GaFrac to the experimental data variable GaFracExpt from grpData.

```
responseMap = 'GaFrac = GaFracExpt';
```

Use an estimatedInfo object to define the model parameter kGd as a parameter to be estimated.

```
estimatedParam = estimatedInfo('kGd');
```

Perform the parameter estimation. Use the name-value pair argument 'ErrorModel' to specify the error model that adds error to simulation data.

```
fitResult = sbiofit(m1, grpData, responseMap, estimatedParam, 'ErrorModel', 'p
```

View the estimated parameter value of kGd.

```
fitResult.ParameterEstimates
```

```
ans =
```

## random(LeastSquaresResults,OptimResults,NLINResults)

---

| Name  | Estimate | StandardError |
|-------|----------|---------------|
| 'kGd' | 0.11     | 0.00064116    |

Use the `random` method to retrieve the simulation data with added noise using the proportional error model which was specified by `sbiofit`. Note that the noise is added only to the response state, that is the `GaFrac` parameter.

```
[ynew,paramEstim] = random(fitResult);
```

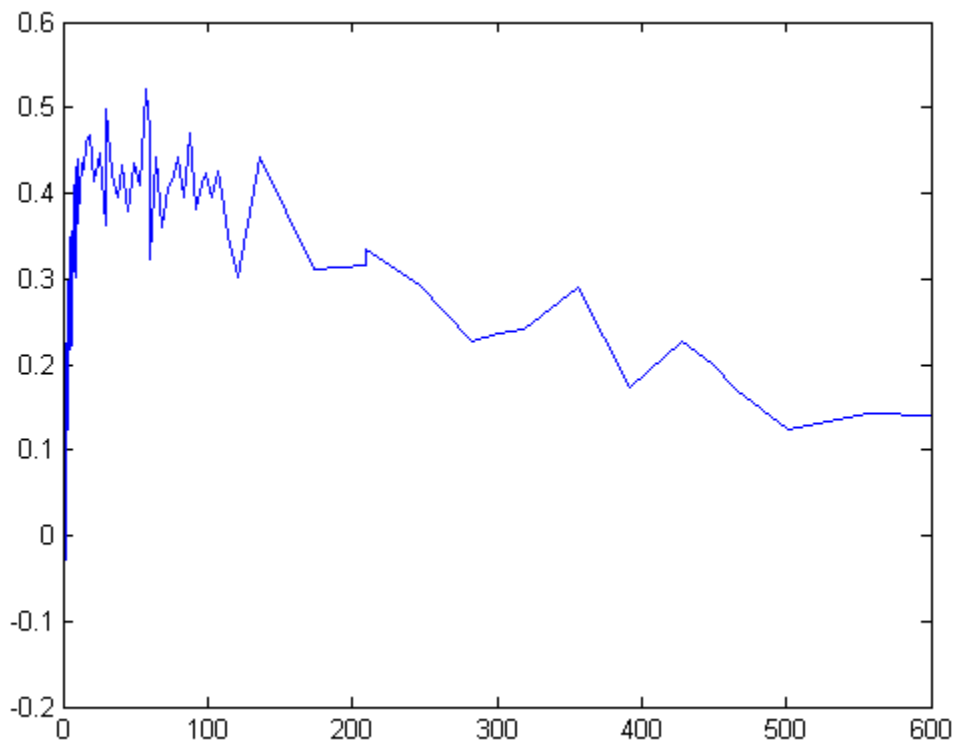
Select the simulation data for the `GaFrac` parameter.

```
GaFracNew = select(ynew,{'Name','GaFrac'});
```

Plot the simulation results.

```
plot(GaFracNew.Time,GaFracNew.Data)  
hold on
```

## random(LeastSquaresResults,OptimResults,NLINResults)

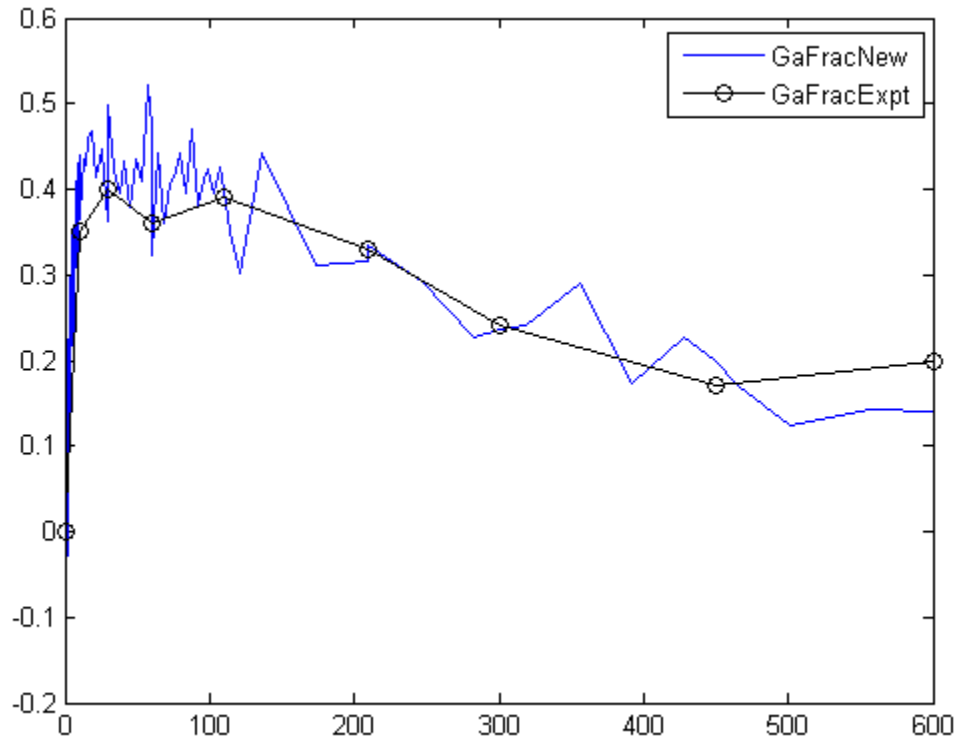


Plot the experimental data to compare it with the simulated data.

```
plot(time,GaFracExpt,'Color','k','Marker','o')  
legend('GaFracNew','GaFracExpt')
```



# random(LeastSquaresResults,OptimResults,NLINResults)



## References

- [1] Yi, T-M., Kitano, H., and Simon, M. (2003). A quantitative characterization of the yeast heterotrimeric G protein cycle. PNAS. 100, 10764–10769.

## See Also

NLINResults object | OptimResults object | sbiofit

# random(NLMEResults)

---

**Purpose** Simulate a SimBiology model, adding variations by sampling the error model

**Syntax**

```
[ynew,parameterEstimates,randomEffects]= random(resultsObj)
[ynew,parameterEstimates]= random(resultsObj,data,dosing)
[ynew,parameterEstimates,randomEffects]=
random(_, 'ParameterType',
value)
```

**Description** [ynew,parameterEstimates,randomEffects]= random(resultsObj) returns simulation results ynew with added noise using the error model specified by the resultsObj.ErrorModel property and estimated parameter values parameterEstimates which are returned by sbiofitmixed.

[ynew,parameterEstimates]= random(resultsObj,data,dosing) uses the specified data and dosing information.

[ynew,parameterEstimates,randomEffects]= random(\_, 'ParameterType', value) adds noise to simulation results that are simulated using either individual or population parameter estimates. The two choices for value are 'population' or 'individual' (default).

---

**Note** The noise is only added to states that are responses which are the states included in the responseMap input argument when you called sbiofitmixed.

---

## Input Arguments

### resultsObj - Estimation results

NLMEResults object

Estimation results, specified as an NLMEResults object, which contains estimation results returned by sbiofitmixed. It must be a scalar object.

### data - Grouped data or output times

groupedData object | vector | cell array of vectors

Grouped data or output times, specified as a `groupedData` object, vector, or cell array of vectors of output times.

If the mixed-effects model from the original fit (using `sbiofitmixed`) uses a covariate model with covariates, the `data` must be a `groupedData` object containing covariate data with the same labels for the covariates (`CovariateLabels` property) specified in the original covariate model.

If data contains any new groups, these groups' parameter values are calculated using the covariate model with fixed effects only, without any random effects. Specifying data as a vector or cell array of vectors is treated as specifying different groups than in the original data that was used with `sbiofitmixed`.

## **dosing - Dosing information**

[] | 2-D matrix of SimBiology dose objects

Dosing information, specified as an empty array [] or 2-D matrix of SimBiology dose objects (`ScheduleDose` object or `RepeatDose` object). It must be consistent with the original dosing data used with `sbiofit`, i.e., dose objects in `dosing` must have the same `TargetName`, `LagParameterName`, or `DurationParameterName` properties as those of the original dosing data.

If empty, no doses are applied during simulation. If not empty, the matrix must have a single row or one row per group in the data `data`. If it has a single row, the same doses are applied to all groups during simulation. If it has multiple rows, each row is applied to a separate group, in the same order as the groups appear in `data`. Multiple columns are allowed so that you can apply multiple dose objects to each group. Each column of doses must reference the same components in the model. Specifically, all doses (except for empty doses) in a column must have the same values for `TargetName`, `DurationParameterName`, and `LagParameterName`. If some groups require more doses than others, then fill in the matrix with dummy doses that are either default doses or empty doses.

# random(NLMEResults)

---

## **value - Parameter type**

string

Parameter type, specified as 'population' or 'individual' (default). If `value` is 'population', the method returns the simulation results with noise using the population parameter estimates. The estimated parameter values used in simulation are identical to `resultsObj.PopulationParameterEstimates` property unless you specify a new `groupedData` object `data` with new covariate data. In this case, the method will reevaluate the covariate model and this could change the parameter estimates.

If `value` is 'individual', estimated parameter values and random effect values are resampled by calling `sbiosampleparameters` with the covariate model (specified by the `data` argument or returned by the `covariateModel` method of `resultsObj`), the fixed effect estimates (`resultsObj.FixedEffects`), and random effect covariance matrix (`resultsObj.RandomEffectCovarianceMatrix`). Parameter estimates and random effects are resampled for all groups.

## **Output Arguments**

### **ynew - Simulation results with noise**

vector of `SimData` objects

Simulation results with noise, returned as a vector of `SimData` objects. The states reported in `ynew` are the states that were included in the `responseMap` input argument of `sbiofitmixed` as well as any other states listed in the `StatesToLog` property of the runtime options (`RuntimeOptions`) of the `SimBiology` model.

### **parameterEstimates - Estimated parameter values**

table

Estimated parameter values, returned as a table.

These estimated values will differ from those values from the original fit if you specify the `value` argument as 'individual' since parameter values are recalculated using `sbiosampleparameters`.

If 'ParameterType' is 'population', the estimated parameter values are identical to `resultsObj.PopulationParameterEstimates` property

unless you specify a new groupedData object data with new covariate data.

### **randomEffects - Random effect values**

table

Random effect values, specified as a table.

### **See Also**

NLMEResults object | sbiofitmixed | sbiosampleparameters | sbiosampleerror

# Reaction object

---

**Purpose** Options for model reactions

**Description** The reaction object represents a *reaction*, which describes a transformation, transport, or binding process that changes one or more species. Typically, the change is the amount of a species. For example:

```
Creatine + ATP <-> ADP + phosphocreatine
```

```
glucose + 2 ADP + 2 Pi -> 2 lactic acid + 2 ATP + 2 H2O
```

Spaces are required before and after species names and stoichiometric values.

See “Property Summary” on page 2-229 for links to reaction object property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the graphical user interface.

## Constructor Summary

|                                  |  |
|----------------------------------|--|
| <code>addreaction (model)</code> | Create reaction object and add to model object |
|----------------------------------|--|

## Method Summary

|                                       |  |
|---------------------------------------|--|
| <code>addkineticlaw (reaction)</code> | Create kinetic law object and add to reaction object |
| <code>addproduct (reaction)</code>    | Add product species object to reaction object        |
| <code>addreactant (reaction)</code>   | Add species object as reactant to reaction object    |
| <code>copyobj (any object)</code>     | Copy SimBiology object and its children              |
| <code>delete (any object)</code>      | Delete SimBiology object                             |

|  |  |
|--|--|
| display (any object)                               | Display summary of SimBiology object                 |
| get (any object)                                   | Get object properties                                |
| rename (compartment, parameter, species, reaction) | Rename object and update expressions                 |
| rmproduct (reaction)                               | Remove species object from reaction object products  |
| rmreactant (reaction)                              | Remove species object from reaction object reactants |
| set (any object)                                   | Set object properties                                |

## Property Summary

|               |  |
|---------------|--|
| Active        | Indicate object in use during simulation               |
| KineticLaw    | Show kinetic law used for ReactionRate                 |
| Name          | Specify name of object                                 |
| Notes         | HTML text describing SimBiology object                 |
| Parent        | Indicate parent object                                 |
| Products      | Array of reaction products                             |
| Reactants     | Array of reaction reactants                            |
| Reaction      | Reaction object reaction                               |
| ReactionRate  | Reaction rate equation in reaction object              |
| Reversible    | Specify whether reaction is reversible or irreversible |
| Stoichiometry | Species coefficients in reaction                       |

# Reaction object

---

|          |                                       |
|----------|---------------------------------------|
| Tag      | Specify label for SimBiology object   |
| Type     | Display SimBiology object type        |
| UserData | Specify data to associate with object |

## See Also

“Definitions and Evaluations of Reactions”, AbstractKineticLaw object, Configset object, KineticLaw object, Model object, Parameter object, Root object, Rule object, Species object



**Purpose** Remove configuration set from model

**Syntax**  
`removeconfigset(modelObj, 'NameValue')`  
`removeconfigset(modelObj, configsetObj)`

## Arguments

|                     |   |
|---------------------|---|
| <i>modelObj</i>     | Model object from which to remove the configuration set.              |
| <i>NameValue</i>    | Name of the configuration set.  |
| <i>configsetObj</i> | Configuration set object that is to be removed from the model object. |

## Description

`removeconfigset(modelObj, 'NameValue')` removes the configset object with the name *NameValue* from the SimBiology model object *modelObj*. A configuration set object stores simulation-specific information. A SimBiology model can contain multiple configuration sets with one being active at any given time. The active configuration set contains the settings that are used during the simulation. *modelObj* always contains at least one configuration set object with name configured to 'default'. You cannot remove the default configuration set from *modelObj*. If the active configuration set is removed from *modelObj*, then the default configuration set will be made active.

`removeconfigset(modelObj, configsetObj)` removes the configuration set object, *configsetObj*, from the SimBiology model, *modelObj*. The configuration set is not deleted; if you want to delete *configsetObj*, use the `delete` method.

If however, there is no MATLAB variable holding the configset, `removeconfigset(modelObj, 'NameValue')` removes the configset from the model and deletes it.

## Examples

- 1 Create a model object by importing the file `oscillator.xml` and add a configset.

```
modelObj = sbmlimport('oscillator');
```

## removeconfigset (model)

---

```
configsetObj = addconfigset(modelObj, 'myset');
```

**2** Remove the configset from modelObj by name or alternatively by indexing.

```
% Remove the configset with name 'myset'.  
removeconfigset(modelObj, 'myset');
```

```
% Get all configset objects and remove the second.  
configsetObj = getconfigset(modelObj);  
removeconfigset(modelObj, configsetObj(2));
```

### See Also

addconfigset, getconfigset, setactiveconfigset

**Purpose** Add dose object to model

**Syntax**  
`doseObj2 = removedose(modelObj, 'DoseName')`  
`doseObj2 = removedose(modelObj, doseObj)`

## Arguments

|                 |   |
|-----------------|---|
| <i>modelObj</i> | Model object from which you remove a dose object.   |
| <i>DoseName</i> | Name of the dose object to remove from a model object. <i>DoseName</i> is the value of the dose object property Name. |
| <i>doseObj</i>  | Dose object to remove from a model object.  |

## Outputs

|                 |                                    |
|-----------------|------------------------------------|
| <i>doseObj2</i> | ScheduleDose or RepeatDose object. |
|-----------------|------------------------------------|

## Description

`doseObj2 = removedose(modelObj, 'DoseName')` removes a SimBiology ScheduleDose or RepeatDose object with the name *DoseName* from a model object (*modelObj*). returns the dose object (*doseObj*), and assigns [] to the dose object property Parent.

You can add a removed dose object back to a model object using the method `adddose`.

`doseObj2 = removedose(modelObj, doseObj)` removes a SimBiology ScheduleDose or RepeatDose object *doseObj*.

## Examples

Remove a dose object from a model object.

- 1 Create model and dose objects, and then add dose to model.

```
modelObj = sbiomodel('mymodel');  
doseObj = adddose(modelObj, 'dose1');
```

- 2 Remove dose object from model object.

## removedose (model)

---

```
removedose(mymodel, 'dose1');
```

Get all dose objects from a model object, and then remove the second dose object.

```
AllDoseObjects = getdose(mymodel);  
removedose(mymodel, AllDoseObjects(2));
```

### See Also

Model object methods:

- `adddose` — add a dose object to a model object
- `getdose` — get dose information from a model object
- `removedose` — remove a dose object from a model object

Dose object constructor `sbiodose`.

`ScheduleDose` object and `RepeatDose` object methods:

- `copyobj` — copy a dose object from one model object to another model object
- `get` — view properties for a dose object
- `set` — define or modify properties for a dose object

**Purpose** Remove variant from model

**Syntax**  
`variantObj = removevariant(modelObj, 'NameValue')`  
`variantObj = removevariant(modelObj, variantObj)`

## Arguments

*modelObj* Specify the model object from which you want to remove the variant.

*variantObj* Specify the variant object to return from the model object.

## Description

`variantObj = removevariant(modelObj, 'NameValue')` removes a SimBiology variant object with the name *NameValue* from the model object *modelObj* and returns the variant object to *variantObj*. The variant object `Parent` property is assigned [] (empty).

A SimBiology variant object stores alternate values for properties on a SimBiology model. For more information on variants, see `Variant` object.

`variantObj = removevariant(modelObj, variantObj)` removes a SimBiology variant object (*variantObj*) and returns the variant object *variantObj*.

To view the variants stored on a model object, use the `getvariant` method. To copy a variant object to another model, use `copyobj`. To add a variant object to a SimBiology model, use the `addvariant` method.

## Examples

- 1 Create a model containing several variants.

```
modelObj = sbiomodel('mymodel');  
variantObj1 = addvariant(modelObj, 'v1');  
variantObj2 = addvariant(modelObj, 'v2');  
variantObj3 = addvariant(modelObj, 'v3');
```

- 2 Remove a variant object using its name.

## removevariant (model)

---

```
removevariant(modelObj, 'v1');
```

**3** Remove a variant object using its index number.

**a** Get the index number of the variant in the model.

```
vObjs = getvariant(modelObj)
```

SimBiology Variant Array

| Index: | Name: | Active: |
|--------|-------|---------|
| 1      | v2    | false   |
| 2      | v3    | false   |

**b** Remove the variant object.

```
removevariant(modelObj, vObjs(2));
```

### See Also

addvariant, getvariant

# rename (compartment, parameter, species, reaction)

---

**Purpose** Rename object and update expressions

**Syntax** `rename(Obj, 'NewNameValue')`

## Arguments

*Obj* Compartment, parameter, species, or reaction object.

'*NewNameValue*' Specify the new name.

## Description

`rename(Obj, 'NewNameValue')`, changes the Name property of the object, *Obj* to *NewNameValue* and updates any uses of it in the model such as rules, events, reactions, variants, and doses to use the new name.

If the new name is already being used by another model component, the new name will be qualified to ensure that it is unique. For example if you change a species named *A* to *K*, and a parameter with the name *K* exists, the species will be qualified as *CompartmentName.K* to indicate that the reference is to the species. If you are referring to an object by its qualified name, for example *CompartmentName.A* and you change the species name, the reference will contain the qualified name in its updated form, for example, *CompartmentName.K*

When you want to change the name of a compartment, parameter, species, or reaction object, use this method instead of `set`.

---

**Note** The `set` method only changes the Name property of the object, except for species and compartments. The method updates the species or compartment object's Name property and any reaction strings referring to the species or compartment to use the new name.

---

## Examples

1 Create a model object that contains a species *A* in a rule.

```
m = sbiomodel('cell');
s = addspecies(m, 'A');
r = addrule(m, 'A = 4');
```

## rename (compartment, parameter, species, reaction)

---

**2** Rename the species to Y

```
rename(s, 'Y');
```

**3** See that the rule expression is now updated.

```
r
```

```
SimBiology Rule Array
```

| Index: | RuleType:         | Rule: |
|--------|-------------------|-------|
| 1      | initialAssignment | Y = 4 |

### See Also

set



# reorder (model, compartment, kinetic law)

---

## Purpose

Reorder component lists

## Syntax

```
modelObj = reorder(Obj,NewOrder)
```

## Input Arguments

|          |  |
|----------|--|
| Obj      | Model, compartment, or kinetic law object.   |
| NewOrder | Object vector in the new order. If Obj is a model object, NewOrder can be an array of compartment, event, parameter, reaction, rule, variant, or dose objects. If Obj is a compartment object, NewOrder must be an array of species objects. If Obj is a kinetic law object, NewOrder must be an array of parameter objects. |

## Description

`modelObj = reorder(Obj,NewOrder)` reorders the component vector `modelObj` to be in the order specified.

Use this method to reorder any of the component vectors, such as compartments, events, parameters, rules, species, doses, and variants. When reordered, the vector of components must contain the same objects as the original list of objects, though they can be in a different order.

## Examples

### Reorder Reactions in SimBiology Model

Import a model.

```
modelObj = sbmlimport('lotka');
```

Display reactions in the model.

```
modelObj.Reactions
```

```
SimBiology Reaction Array
```

## reorder (model, compartment, kinetic law)

---

```
Index:   Reaction:
1        x + y1 -> 2 y1 + x
2        y1 + y2 -> 2 y2
3        y2 -> z
```

Reverse the order of reactions in the model.

```
reorder(modelObj,modelObj.Reactions([3 2 1]));
```

Display the new order of reactions.

```
modelObj.Reactions
```

```
SimBiology Reaction Array
```

```
Index:   Reaction:
1        y2 -> z
2        y1 + y2 -> 2 y2
3        x + y1 -> 2 y1 + x
```

### See Also

[Model object](#) | [Compartment object](#) | [KineticLaw object](#)

**Purpose** Define drug dosing protocol

**Description** A RepeatDose object defines a series of doses to the amount of a species during a simulation. The TargetName property of a dose object defines the species that receives the dose.

Each dose is the same amount, as defined by the Amount property, and given at equally spaced times, as defined by the Interval property. The RepeatCount property defines the number of injections in the series, excluding the initial injection. The Rate property defines how fast each dose is given.

To use a dose object in a simulation you must add the dose object to a model object and set the Active property of the dose object to true. Set the Active property to true if you always want the dose to be applied before simulating the model.

When there are multiple active RepeatDose objects on a model and if there are duplicate specifications for a property value, the last occurrence for the property value in the array of dose, is used during simulation. You can find out which dose is applied last by looking at the indices of the variant objects stored on the model.

See “Property Summary” on page 2-323 for links to species property reference pages. Properties define the characteristics of an object. Use the get and set commands to list object properties and change their values at the command line. You can graphically change object properties in the graphical user interface.

## Constructor Summary

sbiodose Construct dose object

## Method Summary

Methods for RepeatDose objects

copyobj (any object) Copy SimBiology object and its children

get (any object) Get object properties

# RepeatDose object

---

getTable(ScheduleDose,RepeatDose)Return data from SimBiology dose object as table  
set (any object) Set object properties  
setTable(ScheduleDose,RepeatDose)Set dosing information from table to dose object

## Property Summary

Properties for RepeatDose objects

|                       |  |
|-----------------------|--|
| Active                | Indicate object in use during simulation     |
| Amount                | Amount of dose                               |
| AmountUnits           | Dose amount units                            |
| DurationParameterName | arameter length of time to administer a dose |
| Interval              | Time between doses                           |
| Name                  | Specify name of object                       |
| Notes                 | HTML text describing SimBiology object       |
| ParameterName         | arameter                                     |
| Parent                | Indicate parent object                       |
| Rate                  | of dose                                      |
| RateUnits             | Units for dose rate                          |
| RepeatCount           | Dose repetitions                             |
| StartTime             | Start time for initial dose time             |
| Tag                   | Specify label for SimBiology object          |
| TargetName            | Species receiving dose                       |

|           |   |
|-----------|---|
| TimeUnits | Show time units for dosing and simulation |
| Type      | Display SimBiology object type            |
| UserData  | Specify data to associate with object     |

## See Also

Model object, ScheduleDose object, sbiodose, sbiosimulate

# resample (SimData)

---

**Purpose** Resample SimData object array onto new time vector

**Syntax**

```
newSimDataObj = resample(simDataObj)  
newSimDataObj = resample(simDataObj, timevector)  
newSimDataObj = resample(simDataObj, timevector, method)
```

## Arguments

|                      |   |
|----------------------|---|
| <i>newSimDataObj</i> | Resampled SimData object array.   |
| <i>simDataObj</i>    | SimData object array that you want to resample.   |
| <i>timevector</i>    | Real numeric array of time points onto which you want to resample the data.   |
| <i>method</i>        | Method to use during resampling. Can be one of the following: <ul style="list-style-type: none"><li>• 'interp1q' — Uses the MATLAB function interp1q.</li><li>• — To use the MATLAB function interp1, specify one of the following methods:<ul style="list-style-type: none"><li>▪ 'nearest'</li><li>▪ 'linear'</li><li>▪ 'spline'</li><li>▪ 'pchip'</li><li>▪ 'cubic'</li><li>▪ 'v5cubic'</li></ul></li><li>• 'zoh' — specifies zero-order hold.</li></ul> |

---

**Warning:** Note that 'cubic' method will change in a future release. Use 'pchip' instead.

---

## Description

`newSimDataObj = resample(simDataObj)` resamples the simulation data contained in every element of the SimData object array `simDataObj` onto a common time vector, producing a new SimData array `newSimDataObj`. By default, the common time vector is taken from the element of `simDataObj` with the earliest stopping time.

`newSimDataObj = resample(simDataObj, timevector)` resamples the SimData array `simDataObj` onto the time vector `timevector`. `timevector` must either be a real numeric array or the empty array `[]`. If you use an empty array, `resample` uses the default time vector as described above.

`newSimDataObj = resample(simDataObj, timevector, method)` uses the interpolation method specified in `method`.

If the specified `timevector` includes time points outside the time interval encompassed by one or more SimData objects in `simDataObj`, the resampling will involve extrapolation and you will see a warning. See the help for the MATLAB function corresponding to the interpolation method in use for information on how the function performs the extrapolation.

## Examples

### Simulating and Resampling Data

- 1 The project file, `radiodecay.sbproj` contains a model stored in a variable called `m1`. Load `m1` into the MATLAB workspace.

```
sbioloadproject('radiodecay');  
simDataObj = sbiosimulate(m1);
```

- 2 Resample the data.

```
newSimDataObj = resample(simDataObj, [1:5], 'linear');
```

### Resampling Data for Ensemble Runs

- 1 The project file, `radiodecay.sbproj`, contains a model stored in a variable called `m1`. Load `m1` into the MATLAB workspace.

## resample (SimData)

---

```
sbioloadproject('radiodecay');
```

- 2 Change the solver to use during the simulation and perform an ensemble run.

```
csObj = getconfigset(m1);  
set(csObj, 'SolverType', 'ssa');  
simDataObj = sbioensemblerun(m1, 10);
```

- 3 Interpolate the time steps.

```
newSimDataObj = resample(simDataObj, [1:10], 'linear');
```

- 4 View the time steps in the SimData object arrays.

```
newSimDataObj(1).Time  
simDataObj(1).Time
```

### See Also

`sbioensemblerun`, `sbioensemblestats`, `sbiosimulate`, `SimData` object

MATLAB functions `interp1`, `interp1q`



**Purpose** Delete all model objects from root object

**Syntax** `reset(sbioroot)`

**Description** `reset(sbioroot)` deletes all SimBiology model objects contained by the root object. This call is equivalent to `sbioreset`.

The root object contains a list of model objects, available units, unit prefixes, and kinetic laws.

To add a kinetic law to the user-defined library, use the `sbioaddtolibrary` function. To add a unit to the user-defined library, use `sbiounit` followed by `sbioaddtolibrary`. To add a unit prefix to the user-defined library, use `sbiounitprefix` followed by `sbioaddtolibrary`.

**Examples** **1** Query `sbioroot`, which has two model objects.

```
sbioroot
```

```
SimBiology Root Contains:
```

```
Models: 2
Builtin Abstract Kinetic Laws: 3
User Abstract Kinetic Laws: 1
Builtin Units: 54
User Units: 0
Builtin Unit Prefixes: 13
User Unit Prefixes: 0
```

**2** Call `reset`.

```
sbioroot
```

```
SimBiology Root Contains:
```

```
Models: 0
Builtin Abstract Kinetic Laws: 3
```

## reset (root)

---

|                             |    |
|-----------------------------|----|
| User Abstract Kinetic Laws: | 1  |
| Builtin Units:              | 54 |
| User Units:                 | 0  |
| Builtin Unit Prefixes:      | 13 |
| User Unit Prefixes:         | 0  |

### **See Also**

`sbioaddtolibrary`, `sbioreset`, `sbioroot`, `sbiounit`, `sbiounitprefix`

**Purpose** Remove contents from variant object

**Syntax** `rmcontent(variantObj, contents)`  
`rmcontent(variantObj, idx)`

## Arguments

*variantObj* Specify the variant object from which you want to remove data. The Content property is modified to remove the new data.

*contents* Specify the data you want to remove from a variant object. Contents can either be a cell array or an array of cell arrays. A valid cell array should have the form {'Type', 'Name', 'PropertyName', PropertyValue}, where PropertyValue is the new value to be applied for the PropertyName. Valid Type, Name, and PropertyName values are as follows.

| 'Type'        | 'Name'   | 'PropertyName'  |
|---------------|--|-----------------|
| 'species'     | Name of the species. If there are multiple species in the model with the same name, specify the species as [compartmentName.speciesName], where compartmentName is the name of the compartment containing the species. | 'InitialAmount' |
| 'parameter'   | If the parameter scope is a model, specify the parameter name. If the parameter scope is a kinetic law, specify [reactionName.parameterName].  | 'Value'         |
| 'compartment' | Name of the compartment.   | 'Capacity'      |

*idx* Specify the ContentIndex or indices of the data to be removed. To display the ContentIndex, enter the object name and press **Enter**.

# rmcontent (variant)

---

## Description

`rmcontent(variantObj, contents)` removes the data stored in the variable `contents` from the variant object (`variantObj`).

`rmcontent(variantObj, idx)` removes the data specified by the indices `idx` (also called ContentIndex) from the Content property of the variant object.

## Examples

- 1 Create a model containing three species in one compartment.

```
modelObj = sbiomodel('mymodel');
compObj = addcompartment(modelObj, 'comp1');
A = addspecies(compObj, 'A');
B = addspecies(compObj, 'B');
C = addspecies(compObj, 'C');
```

- 2 Add a variant object that varies the species' InitialAmount property.

```
variantObj = addvariant(modelObj, 'v1');
addcontent(variantObj, {'species','A', 'InitialAmount', 5}, ...
{'species', 'B', 'InitialAmount', 10}, ...
{'species', 'C', 'InitialAmount', 15});% Display the variant
variantObj
```

```
SimBiology Variant - v1 (inactive)
```

| ContentIndex: | Type:   | Name: | Property:     | Value: |
|---------------|---------|-------|---------------|--------|
| 1             | species | A     | InitialAmount | 5      |
| 2             | species | B     | InitialAmount | 10     |
| 3             | species | C     | InitialAmount | 15     |

- 3 Use the ContentIndex number to remove a species from the Content property of the variant object.

```
rmcontent(variantObj, 2);
variantObj
```

```
SimBiology Variant - v1 (inactive)
```

| ContentIndex: | Type:   | Name: | Property:     | Value: |
|---------------|---------|-------|---------------|--------|
| 1             | species | A     | InitialAmount | 5      |
| 2             | species | C     | InitialAmount | 15     |

- 4** (Alternatively) Remove a species from the contents of the variant object using detailed reference to the species.

```
rmcontent(variantObj, {'species','A', 'InitialAmount', 5});  
% Display variant object  
variantObj  
SimBiology Variant - v1 (inactive)
```

| ContentIndex: | Type:   | Name: | Property:     | Value: |
|---------------|---------|-------|---------------|--------|
| 1             | species | C     | InitialAmount | 15     |

### See Also

addvariant, rmcontent, sbiovariant

# rmproduct (reaction)

---

**Purpose** Remove species object from reaction object products

**Syntax** `rmproduct(reactionObj, SpeciesName)`  
`rmproduct(reactionObj, speciesObj)`

## Arguments

|                    |   |
|--------------------|---|
| <i>reactionObj</i> | Reaction object.  |
| <i>SpeciesName</i> | Name for a model object. Enter a species name or a cell array of species names. |
| <i>speciesObj</i>  | Species object. Enter a species object or an array of species objects.          |

## Description

`rmproduct(reactionObj, SpeciesName)`, in a reaction object (`reactionObj`), removes a species object with a specified name (`SpeciesName`) from the property `Products`, removes the species name from the property `Reaction`, and updates the property `Stoichiometry` to exclude the species coefficient.

`rmproduct(reactionObj, speciesObj)` removes a species object as described above using a MATLAB variable for a species object.

The species object is not removed from the parent model property `Species`. If the species object is no longer used by any reaction, you can use the function `delete` to remove it from the parent object.

If one of the species specified does not exist as a product, a warning is returned.

## Examples

### Example 1

This example shows how to remove a product that was previously added to a reaction. You can remove the species object using the species name.

```
modelObj = sbiomodel('cell');  
reactionObj = addreaction(modelObj, 'Phosphocreatine + ADP -> creatine + ATP + Pi');  
rmproduct(reactionObj, 'Pi')
```

SimBiology Reaction Array

```
Index: Reaction:  
1      Phosphocreatine + ADP -> creatine + ATP
```

## Example 2

Remove a species object using a model index to a species object.

```
modelObj = sbiomodel('cell');  
reactionObj = addreaction(modelObj, 'A -> B + C');  
reactionObj.Reaction  
ans =  
      A -> B + C  
  
rmproduct(reactionObj, modelObj.Species(2));  
reactionObj.Reaction  
ans =  
      A -> C
```

## See Also

`rmreactant`

# rmreactant (reaction)

---

**Purpose** Remove species object from reaction object reactants

**Syntax** `rmreactant(reactionObj, SpeciesName)`  
`rmreactant(reactionObj, speciesObj)`

## Arguments

|                    |   |
|--------------------|---|
| <i>reactionObj</i> | Reaction object.  |
| <i>SpeciesName</i> | Name for a species object. Enter a species name or a cell array of species names. |
| <i>speciesObj</i>  | Species object. Enter a species object or an array of species objects.            |

## Description

`rmreactant(reactionObj, SpeciesName)`, in a reaction object (`reactionObj`), removes a species object with a specified name (`SpeciesName`) from the property `Reactants`, removes the species name from the property `Reaction`, and updates the property `Stoichiometry` to exclude the species coefficient.

`rmreactant(reactionObj, speciesObj)` removes a species object as described above using a MATLAB variable for a species object, or a model index for a species object.

The species object is not removed from the parent model property `Species`. If the species object is no longer used by any reaction, you can use the method `delete` to remove it from the parent object.

If one of the species specified does not exist as a reactant, a warning is returned.

## Examples

### Example 1

This example shows how to remove a reactant that was added to a reaction by mistake. You can remove the species object using the species name.

```
modelObj = sbiomodel('cell');  
reactionObj = addreaction(modelObj, 'Phosphocreatine + ADP + Pi -> creatine + ATP');
```



```
rmreactant(reactionObj, 'Pi')
```

SimBiology Reaction Array

```
Index:    Reaction:
  1      Phosphocreatine + ADP -> creatine + ATP
```

### Example 2

Remove a species object using a model index to a species object.

```
modelObj = sbiomodel('cell');
reactionObj = addreaction(modelObj, 'A -> B + C');
```

```
reactionObj.Reaction
ans =
    A + B -> C
```

```
rmreactant(reactionObj, modelObj.Species(1));
reactionObj.Reaction
```

```
ans =
    A -> C
```

### See Also

`delete`, `rmproduct`

# Root object

---

**Purpose** Hold models, unit libraries, and abstract kinetic law libraries

**Description** The SimBiology root object contains a list of the SimBiology model objects and SimBiology libraries. The components that the libraries contain are: all available units, unit prefixes, and available abstract kinetic law objects. There are two types of libraries: one contains components that are built in (`BuiltinLibrary`), and the other contains components that are user defined (`UserdefinedLibrary`).

You can retrieve SimBiology model objects from the SimBiology root object. A SimBiology model object has its `Parent` property set to the SimBiology root object.

See “Property Summary” on page 2-257 for links to root object property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can interactively change object properties in the SimBiology desktop.

|                            |                       |                               |
|----------------------------|-----------------------|-------------------------------|
| <b>Constructor Summary</b> | <code>sbioroot</code> | Return SimBiology root object |
|----------------------------|-----------------------|-------------------------------|

|                       |                                   |   |
|-----------------------|-----------------------------------|---|
| <b>Method Summary</b> | <code>copyobj</code> (any object) | Copy SimBiology object and its children   |
|                       | <code>get</code> (any object)     | Get object properties                     |
|                       | <code>reset</code> (root)         | Delete all model objects from root object |
|                       | <code>set</code> (any object)     | Set object properties                     |

## Property Summary

|                    |                                    |
|--------------------|------------------------------------|
| BuiltInLibrary     | Library of built-in components     |
| Models             | Contain all model objects          |
| Type               | Display SimBiology object type     |
| UserDefinedLibrary | Library of user-defined components |

## See Also

AbstractKineticLaw object, Configset object, KineticLaw object, Model object, Parameter object, Reaction object, Rule object, Species object

# Rule object

---

**Purpose** Hold rule for species and parameters

**Description** The SimBiology rule object represents a *rule*, which is a mathematical expression that modifies a species amount or a parameter value. For a description of the types of SimBiology rules, see `RuleType`.

See “Property Summary” on page 2-258 for links to rule property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the graphical user interface.

## Constructor Summary

|                              |  |
|------------------------------|--|
| <code>addrule (model)</code> | Create rule object and add to model object |
|------------------------------|--|

## Method Summary

|                                   |   |
|-----------------------------------|---|
| <code>copyobj (any object)</code> | Copy SimBiology object and its children |
| <code>delete (any object)</code>  | Delete SimBiology object                |
| <code>display (any object)</code> | Display summary of SimBiology object    |
| <code>get (any object)</code>     | Get object properties                   |
| <code>set (any object)</code>     | Set object properties                   |

## Property Summary

|                     |  |
|---------------------|--|
| <code>Active</code> | Indicate object in use during simulation |
| <code>Name</code>   | Specify name of object                   |
| <code>Notes</code>  | HTML text describing SimBiology object   |

|          |  |
|----------|--|
| Parent   | Indicate parent object                     |
| Rule     | Specify species and parameter interactions |
| RuleType | Specify type of rule for rule object       |
| Tag      | Specify label for SimBiology object        |
| Type     | Display SimBiology object type             |
| UserData | Specify data to associate with object      |

## See Also

“Definitions and Evaluations of Rules”, AbstractKineticLaw object, Configset object, KineticLaw object, Model object, Parameter object, Reaction object, Root object, Species object

# summary(LeastSquaresResults, OptimResults, NLINResults)

---

**Purpose** Plot a summary figure that contains estimated values and estimation statistics

**Syntax** `summary(resultsObj)`

**Description** `summary(resultsObj)` plots a summary figure that contains estimated values and estimation statistics.

**Input Arguments** **resultsObj - Estimation results**  
`OptimResults` object | `NLINResults` object | vector of results objects  
Estimation results, specified as an `OptimResults` object or `NLINResults` object, or vector of results objects which contains estimation results from running `sbiofit`.

**See Also** `NLINResults` object | `OptimResults` object | `sbiofit`

## Purpose

Define drug dosing protocol

## Description

A `ScheduleDose` object defines a series of doses to the amount of a species during a simulation. The `TargetName` property defines the species that receives the dose.

Each dose can have a different amount, as defined by an amount array in the `Amount` property. Each dose can be given at specified times, as defined by a time array in the `Time` property. A rate array in the `Rate` property defines how fast each dose is given. At each time point in the time array, a dose is given with the corresponding amount and rate.

To use a dose object in a simulation, you must add the dose object to a model object and set the `Active` property of the dose object to `true`.

When there are multiple active `ScheduleDose` objects on a model, and there are duplicate specifications for a property value, the simulation uses the last occurrence for the property value in the array of doses. You can find out which dose you applied last by looking at the indices of the dose objects stored on the model.

See the “Property Summary” on page 2-323 for links to species property reference pages. Properties define the characteristics of an object. Use the `get` command to list object properties and the `set` command to change their values at the command line. Use can graphically change object properties in the graphical user interface

## Constructor Summary

|                       |                       |
|-----------------------|-----------------------|
| <code>sbiodose</code> | Construct dose object |
|-----------------------|-----------------------|

## Method Summary

Methods for variant objects

|                                   |   |
|-----------------------------------|---|
| <code>copyobj</code> (any object) | Copy SimBiology object and its children |
|-----------------------------------|---|

|                               |                       |
|-------------------------------|-----------------------|
| <code>get</code> (any object) | Get object properties |
|-------------------------------|-----------------------|

# ScheduleDose object

---

getTable(ScheduleDose,RepeatDose)Return data from SimBiology dose object as table

set (any object) Set object properties

setTable(ScheduleDose,RepeatDose)Set dosing information from table to dose object

## Property Summary

Properties for variant objects

|                       |  |
|-----------------------|--|
| Active                | Indicate object in use during simulation     |
| Amount                | Amount of dose                               |
| AmountUnits           | Dose amount units                            |
| DurationParameterName | arameter length of time to administer a dose |
| Name                  | Specify name of object                       |
| Notes                 | HTML text describing SimBiology object       |
| ParameterName         | arameter                                     |
| Parent                | Indicate parent object                       |
| Rate                  | of dose                                      |
| RateUnits             | Units for dose rate                          |
| Tag                   | Specify label for SimBiology object          |
| TargetName            | Species receiving dose                       |
| Time                  | Simulation time steps or schedule dose times |
| TimeUnits             | Show time units for dosing and simulation    |



|          |                                       |
|----------|---------------------------------------|
| Type     | Display SimBiology object type        |
| UserData | Specify data to associate with object |

**See Also** Model object, RepeatDose object, sbiodose, sbiosimulate

# select (SimData)

---

**Purpose** Select data from SimData object

**Syntax** `[t,x, names] = select(simDataObj, Query)`  
`[Out] = select(simDataObj, Query, 'Format', 'FormatValue')`

## Arguments

## Output Arguments

- t* An n-by-1 vector of time points.
- x* An n-by-m data array. *t* and *names* label the rows and columns of *x* respectively.
- names* An m-by-1 cell array of names.
- Out* Data returned in the format specified in 'FormatValue', shown in "Input Arguments" on page 2-264. Depending on the specified 'FormatValue', *Out* contains one of the following:
- Structure array
  - SimData object
  - Time series object
  - Combined time series object from an array of SimData objects

## Input Arguments

*simDataObj* SimData object array. Enter a variable name for a SimData object.

**Query** A cell array of arguments consisting of some combination of property name/property value pairs and/or 'Where' clauses. For a more complete description of the query syntax, including 'Where' clauses and their supported condition types, see `sbiiselect`. You can use any of the metadata fields available in the cells of the `DataInfo` property of a `SimData` object in a query. These include 'Type', 'Name', 'Units', 'Compartment' (species only), or 'Reaction' (parameter only).

**FormatValue** Choose a format from the following table.

| <b>FormatValue</b> | <b>Description</b>  |
|--------------------|---|
| 'num'              | Specifies the format that lets you return data in numeric arrays. This is the default when <code>select</code> is called with multiple output arguments.  |
| 'nummetadata'      | Specifies the format that lets you return a cell array of metadata structures in <i>metadata</i> instead of names. The elements of <i>metadata</i> label the columns of <i>x</i> .              |
| 'numqualnames'     | Specifies the format that lets you return qualified names in <i>names</i> to resolve ambiguities.   |
| 'struct'           | Specifies the format that lets you return a structure array holding both data and metadata. This is the default when you use a single output argument.  |
| 'simdata'          | Specifies the format that lets you return data in a new <code>SimData</code> object. This is the default format when <code>select</code> is called with zero or one output argument.            |
| 'ts'               | Specifies the format that lets you return data in time series objects, creating an individual time series for each state or column and <code>SimData</code> object in <code>simDataObj</code> . |
| 'tslumped'         | Specifies the format that lets you return data in time series objects, combining data from each <code>SimData</code> object into a single time series.  |

# select (SimData)

---

## Description

`[t,x, names] = select(simDataObj, Query)` returns simulation time and state data from the SimData object (`simDataObj`) that matches the query argument `Query`.

In a SimData object `simDataObj`, the columns of the data matrix `simDataObj.Data` are labeled by the cell array of metadata structures given by `simDataObj.DataInfo`. The `select` method enables you to pick out columns of the data matrix based on their metadata labels. For example, to extract data for all parameters logged in a SimData object `simDataObj`, use the syntax `[t, x, names] = select (simDataObj, {'Type', 'parameter'})`.

`[Out] = select(simDataObj, Query, 'Format', 'FormatValue')` returns the data in the specified format. Valid formats are listed in “Input Arguments” on page 2-264.

## Examples

This example shows how to extract data of interest from your simulation data with the `select` method.

- 1 The project file `radiodecay.sproj` contains a model stored in a variable called `m1`. Load `m1` into the MATLAB workspace.

```
sbioloadproject gprotein_norules m1
```

- 2 Change the solver to use during the simulation and perform an ensemble run.

```
csObj = getconfigset(m1);  
set(csObj, 'SolverType', 'ssa');  
simDataObj = sbioenssemblerun(m1, 10);
```

- 3 Select all species data logged in the SimData array `sdata`.

```
[t x n] = select(simDataObj, {'Type','species'});
```

- 4 Select data for the parameters with name 'Kd' and return the results in a new SimData object array.

```
newsd = select(simDataObj, {'Type','parameter','name', 'Kd'});
```

**5** This selects all data from `simDataObj` with a name that matches the pattern 'G' and returns time series objects.

```
ts = select(simDataObj, {'Where','Name','regex','G'}, ...  
            'Format','ts');
```

### See Also

`getdata`, `sbioselect`, `sbiosimulate`, `selectbyname`, `Simdata` object

# selectbyname (SimData)

---

**Purpose** Select data by name from SimData object array

**Syntax**  
`[t,x,n] = selectbyname(simDataObj, 'NameValue')`  
`Out = selectbyname(simDataObj, NameValue, 'Format', Format)`

## Arguments

## Output Arguments

*t* An n-by-1 vector of time points.

*x* An n-by-m data array. *t* and *names* label the rows and columns of *x* respectively.

*n* An m-by-1 cell array of names.

*Out* Data returned in the format as specified in '*FormatValue*', shown in "Input Arguments" on page 2-268. Depending on the specified '*FormatValue*', *Out* contains one of the following:

- Structure array
- SimData object
- Time series object
- Combined time series object from an array of SimData objects

## Input Arguments

*simDataObj* SimData object array. Enter a variable name for a SimData object.

*NameValue* Names of the states for which you want to select data from *simDataObj*. Must be either a string or a cell array of strings.

**Query** A cell array of arguments consisting of some combination of property name/property value pairs and/or 'Where' clauses. For a more complete description of the query syntax, including 'Where' clauses and their supported condition types, see `sbiiselect`. You can use any of the metadata fields available in the cells of the `DataInfo` property of a `SimData` object. These include 'Type', 'Name', 'Units', 'Compartment' (species only), or 'Reaction' (parameter only).

**FormatValue** Choose a format from the following table.

| <b>FormatValue</b> | <b>Description</b>   |
|--------------------|--|
| 'num'              | Specifies the format that lets you return data in numeric arrays. This is the default when <code>selectbyname</code> is called with multiple output arguments.                             |
| 'nummetadata'      | Specifies the format that lets you return a cell array of metadata structures in <i>metadata</i> instead of names. The elements of <i>metadata</i> label the columns of <i>x</i> .         |
| 'numqualnames'     | Specifies the format that lets you return qualified names in <i>names</i> to resolve ambiguities.  |
| 'struct'           | Specifies the format that lets you return a structure array holding both data and metadata. This is the default when you use a single output argument.                                     |
| 'simdata'          | Specifies the format that lets you return data in a new <code>SimData</code> object. This is the default format when <code>selectbyname</code> is called with zero or one output argument. |

## selectbyname (SimData)

---

| <b>FormatValue</b> | <b>Description</b>   |
|--------------------|--|
| 'ts'               | Specifies the format that lets you return data in time series objects, creating an individual time series for each state or column and SimData object in <code>simDataObj</code> . |
| 'tslumped'         | Specifies the format that lets you return data in time series objects, combining data from each SimData object into a single time series.  |

### Description

The `selectbyname` method allows you to select data from a SimData object array by name. `[t,x,n] = selectbyname(simDataObj, 'NameValue')` returns time and state data from the SimData object `simDataObj` for states with names `'NameValue'`.

In a SimData object `simDataObj`, the names labeling the columns of the data matrix `simDataObj.Data` are given by `simDataObj.DataNames`. A name specified in `'NameValue'` can match more than one data column, for example, when `simDataObj` contains data for a species and parameter both named `'k'`. To resolve ambiguities, use qualified names in `'NameValue'`, such as `'CompartmentName.SpeciesName'` or `'ReactionName.ParameterName'`. `selectbyname` returns qualified names in the output argument `names` when there are ambiguities.

`Out = selectbyname(simDataObj, NameValue, 'Format', Format)` returns the data in the specified format. Valid formats are listed in “Input Arguments” on page 2-268.

### Examples

#### Select Specific Species Data from Simulation Results

This example uses the Lotka-Volterra (predator-prey) model described by Gillespie [1].

Load the sample project containing the Lotka-Volterra model `m1`.

```
sbioloadproject lotka;
```

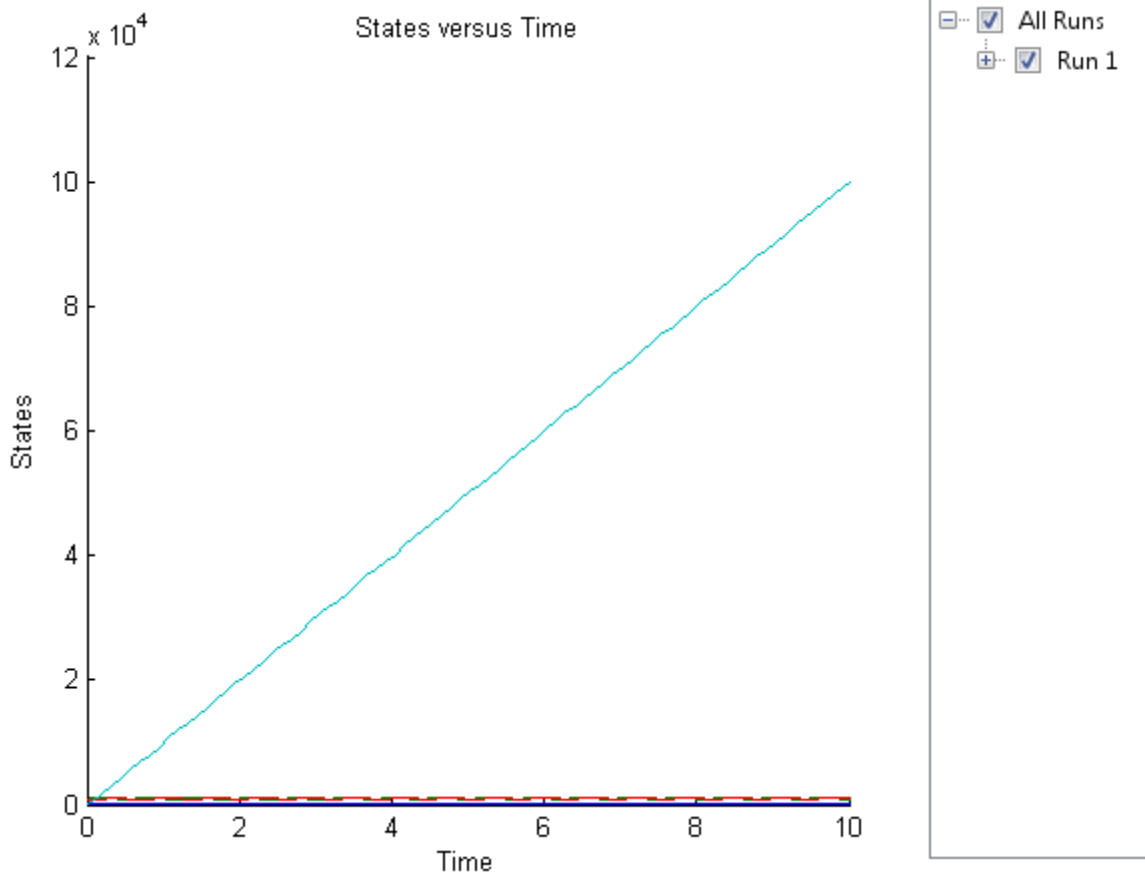
Simulate the model `m1`.



```
sd = sbiosimulate(m1);
```

Plot the simulation results which show the states of all species in the model.

```
sbioplot(sd);
```



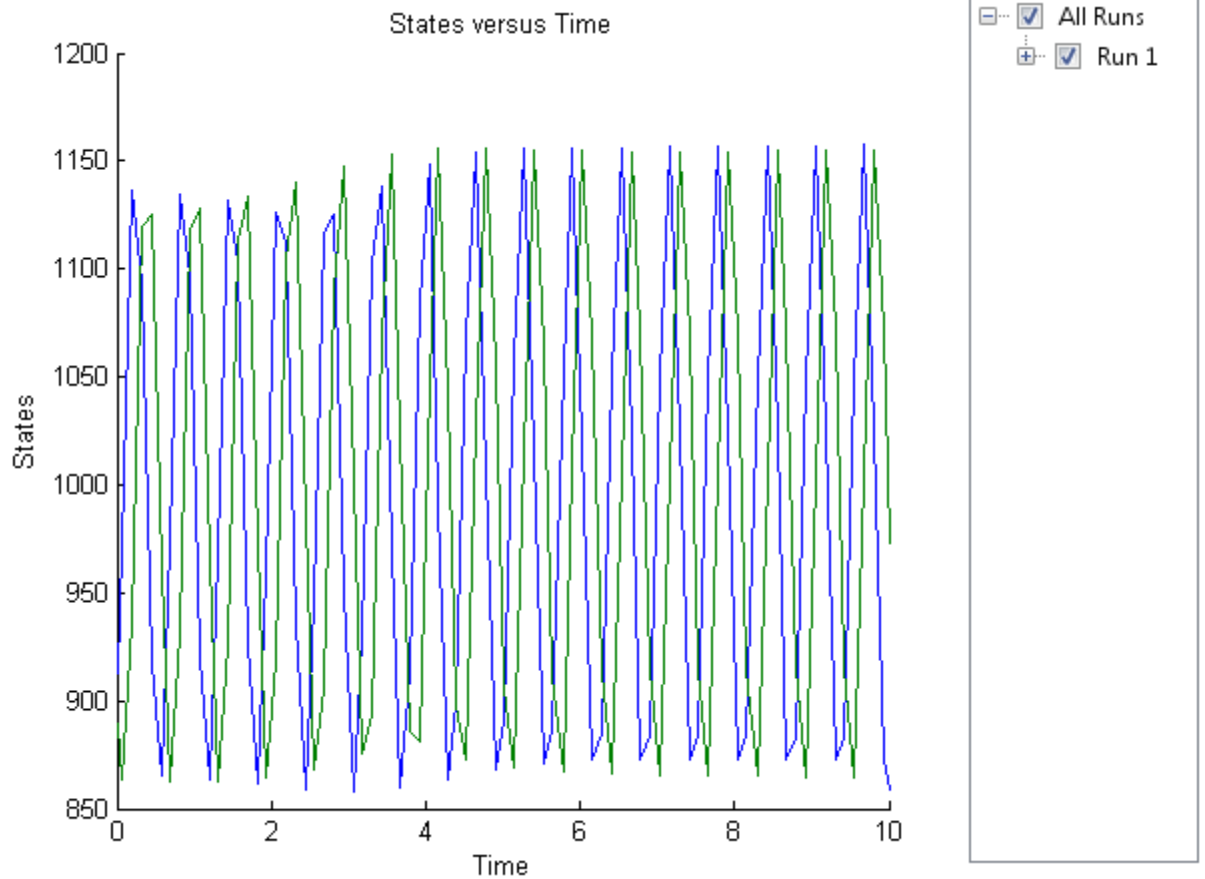
Select the simulation data of y1 and y2 species only.

# selectbyname (SimData)

```
sd2 = selectbyname(sd,{'y1','y2'});
```

Plot the selected data.

```
sbioplot(sd2);
```



### See Also

[1] Gillespie D.T. "Exact Stochastic Simulation of Coupled Chemical Reactions," (1977) The Journal of Physical Chemistry, 81(25), 2340-2361.

getdata, sbioselect, sbiosimulate

# set (any object)

---

**Purpose** Set object properties

**Syntax**

```
set(Obj, 'PropertyName', PropertyValue)  
set(Obj, 'PropertyName1', PropertyValue1, 'PropertyName2',  
    PropertyValue2...)
```

## Arguments

*Obj* Abstract kinetic law, compartment, configuration set, event, kinetic law, model, parameter, PKCompartment, PKData, PKModelDesign, PKModelMap, reaction, rule, SimData, species, or variant object.

'*PropertyName*' Name of the property to set.

*PropertyValue* Specify the value to set. Property values depend on the property being set. See the reference page for an object property for values that can be specified.

## Description

`set(Obj, 'PropertyName', PropertyValue)` sets the property '*PropertyName*' of the object *Obj*, to *PropertyValue*.

`set(Obj, 'PropertyName1', PropertyValue1, 'PropertyName2', PropertyValue2...)` sets the properties '*PropertyName1*' and '*PropertyName2*' to *PropertyValue1* and *PropertyValue2* respectively, and so on in sequence. You can specify multiple *PropertyName*, *PropertyValue* pairs.

When you want to change the name of a compartment, parameter, or species object, use the `rename` method instead of `set`. The `rename` method allows you to change the name and update the expressions in which these components are used.

## Examples

**1** Create a model object.

```
modelObj = sbiomodel ('my_model');
```

**2** Add parameter object.

```
parameterObj = addparameter (modelObj, 'kf');
```

- 3 Set the ConstantValue property of the parameter object to false and verify.

MATLAB returns 1 for true and 0 for false.

```
set (parameterObj, 'ConstantValue', false);  
get(parameterObj, 'ConstantValue')
```

MATLAB returns

```
ans =
```

```
0
```

### See Also

get , rename, setactiveconfigset

# setactiveconfigset (model)

---

**Purpose** Set active configuration set for model object

**Syntax**  
`configsetObj = setactiveconfigset(modelObj, 'NameValue')`  
`configsetObj2 = setactiveconfigset(modelObj, configsetObj1)`

**Description**  
`configsetObj = setactiveconfigset(modelObj, 'NameValue')` sets the configuration set *NameValue* to be the active configuration set for the model *modelObj* and returns to *configsetObj*.  
`configsetObj2 = setactiveconfigset(modelObj, configsetObj1)` sets the configset *configsetObj1* to be the active configset for *modelObj* and returns to *configsetObj2*. Any change in one of these two configset objects *configsetObj1* and *configsetObj2* is reflected in the other. To copy over a configset object from one model object to another, use the `copyobj` method.

The active configuration set contains the settings that are be used during a simulation. A default configuration set is attached to any new model.

**Examples**  
**1** Create a model object by importing the `oscillator.xml` file, and add a Configset object to the model.

```
modelObj = sbmlimport('oscillator');  
configsetObj = addconfigset(modelObj, 'myset');
```

**2** Configure the simulation stop criteria by setting the `StopTime`, `MaximumNumberOfLogs`, and `MaximumWallClock` properties of the Configset object. Set the stop criteria to a simulation time of 3000 seconds, 50 logs, or a wall clock time of 10 seconds, whichever comes first.

```
set(configsetObj, 'StopTime', 3000, 'MaximumNumberOfLogs', 50,...  
    'MaximumWallClock', 10)  
get(configsetObj)
```

```
Active: 0  
CompileOptions: [1x1 SimBiology.CompileOptions]
```

## setactiveconfigset (model)

---

```
Name: 'myset'  
Notes: ''  
RuntimeOptions: [1x1 SimBiology.RuntimeOptions]  
SensitivityAnalysisOptions: [1x1 SimBiology.SensitivityAnalysisOptions]  
SolverOptions: [1x1 SimBiology.ODESolverOptions]  
SolverType: 'ode15s'  
StopTime: 3000  
MaximumNumberOfLogs: 50  
MaximumWallClock: 10  
TimeUnits: 'second'  
Type: 'configset'
```

- 3 Set the new Configset object to be active, simulate the model using the new Configset object, and plot the result.

```
setactiveconfigset(modelObj, configsetObj);  
[t,x] = sbiosimulate(modelObj);  
plot (t,x)
```

### See Also

addconfigset, getconfigset, removeconfigset

# setparameter (kineticlaw)

---

**Purpose** Specify specific parameters in kinetic law object

**Syntax** `setparameter(kineticlawObj, 'ParameterVariablesValue',  
'ParameterVariableNamesValue')`

## Arguments

|                                    |  |
|------------------------------------|--|
| <i>ParameterVariableValue</i>      | Specify the value of the parameter variable in the kinetic law object.   |
| <i>ParameterVariableNamesValue</i> | Specify the parameter name with which to configure the parameter variable in the kinetic law object. Determines parameters in the ReactionRate equation. |

## Description

Configure `ParameterVariableNames` in the kinetic law object.

`setparameter(kineticlawObj, 'ParameterVariablesValue', 'ParameterVariableNamesValue')` configures the `ParameterVariableNames` property of the kinetic law object (`kineticlawObj`). `ParameterVariableValue` corresponds to one of the strings in `kineticlawObj` `ParameterVariables` property. The corresponding element in the `kineticlawObj` `ParameterVariableNames` property is configured to `ParameterVariableNamesValue`. For example, if `ParameterVariables` is `{'Vm', 'Km'}` and `ParameterVariablesValue` is specified as `Vm`, then the first element of the `ParameterVariableNames` cell array is configured to `ParameterVariableNamesValue`.

## Examples

Create a model, add a reaction, and then assign the `ParameterVariableNames` for the reaction rate equation.

1 Create the model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
```



```
reactionObj = addreaction(modelObj, 'a -> c + d');
```

- 2 Create a kinetic law object for the reaction object of the type 'Henri-Michaelis-Menten'.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

reactionObj.KineticLaw property is configured to kineticlawObj.

- 3 The 'Henri-Michaelis-Menten' kinetic law has two parameter variables (Vm and Km) that should be set. To set these variables:

```
setparameter(kineticlawObj, 'Vm', 'Va');  
setparameter(kineticlawObj, 'Km', 'Ka');
```

- 4 Verify that the parameter variables are correct.

```
get (kineticlawObj, 'ParameterVariableNames')
```

MATLAB returns:

```
ans =
```

```
    'Va'    'Ka'
```

### See Also

addparameter, getspecies, setspecies

# setspecies (kineticlaw)

---

**Purpose** Specify species in kinetic law object

**Syntax** `setspecies(kineticlawObj, 'SpeciesVariablesValue',  
'SpeciesVariableNamesValue')`

## Arguments

|                                  |   |
|----------------------------------|---|
| <i>SpeciesVariablesValue</i>     | Specify the species variable in the kinetic law object.   |
| <i>SpeciesVariableNamesValue</i> | Specify the species name with which to configure the species variable in the kinetic law object. Determines the species in the ReactionRate equation. |

## Description

setspecies configures the kinetic law object SpeciesVariableNames property.

setspecies(kineticlawObj, 'SpeciesVariablesValue', 'SpeciesVariableNamesValue') configures the SpeciesVariableNames property of the kinetic law object, kineticlawObj. SpeciesVariablesValue corresponds to one of the strings in the SpeciesVariables property of kineticlawObj. The corresponding element in kineticlawObj SpeciesVariableNames property is configured to SpeciesVariableNamesValue.

For example, if SpeciesVariables are {'S', 'S1'} and SpeciesVariablesValue is specified as S1, the first element of the SpeciesVariableNames cell array is configured to SpeciesVariableNamesValue.

## Examples

Create a model, add a reaction, and assign the SpeciesVariableNames for the reaction rate equation.

**1** Create the model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');  
reactionObj = addreaction(modelObj, 'a -> c + d');
```

- 2 Create a kinetic law object for the reaction object of the type 'Henri-Michaelis-Menten'.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

reactionObj.KineticLaw property is configured to kineticlawObj.

- 3 The 'Henri-Michaelis-Menten' kinetic law has one species variable (S) that should be set. To set this variable:

```
setspecies(kineticlawObj, 'S', 'a');
```

- 4 Verify that the species variable is correct.

```
get(kineticlawObj, 'SpeciesVariableNames')
```

MATLAB returns:

```
ans =
```

```
'a'
```

### See Also

addparameter, getspecies, setparameter

# setTable(ScheduleDose,RepeatDose)

---

**Purpose** Set dosing information from table to dose object

**Syntax** `setTable(doseObj,tbl)`

**Description** `setTable(doseObj,tbl)` sets the dosing data from a table `tbl` to a dose object `doseObj`.

**Input Arguments** **doseObj - Dose object**  
`ScheduleDose` object | `RepeatDose` object | array of dose objects

Dose object, specified as a `ScheduleDose` object or `RepeatDose` object or array of these objects.

**tbl - Dosing data**  
`table` | cell array of tables

Dosing data, specified as a table or cell array of tables. If `doseObj` is an array of dose objects, then `tbl` must be a cell array of tables of the same size as `doseObj`.

If `doseObj` is a `ScheduleDose` object, `tbl` must have 2 or 3 variables (columns) representing dose time, amount, and rate (optional). The variable names (`tbl.Properties.VariableNames`) must be 'Time', 'Amount', and 'Rate' (optional), respectively.

If `doseObj` is a `RepeatDose` object, `tbl` must have only one row with 4 or 5 variables (columns) representing dose start time, amount, interval, repeat count, and rate (optional). The variable names (`tbl.Properties.VariableNames`) must be 'StartTime', 'Amount', 'Interval', 'RepeatCount', and 'Rate' (optional), respectively.

The units of `tbl` variables (`tbl.Properties.VariableUnits`), if any, are copied over to the corresponding property units of `doseObj`.

**Examples** **Set a Table of Dosing Data to a RepeatDose Object**

Create a table containing dose start time, amount, interval, repeat count, and rate.

## setTable(ScheduleDose,RepeatDose)

---

```
StartTime = 5;
Amount = 500;
Interval = 1;
RepeatCount = 3;
Rate = 1;
tbl = table(StartTime,Amount,Interval,RepeatCount,Rate);
```

Create a RepeatDose object, and set the dosing information from the table.

```
rdose = sbiodose('rdose','repeat');
setTable(rdose,tbl);
```

### **Set a Table of Dosing Data to a ScheduleDose Object**

Create a table containing dose time and amount.

```
Time = [1 2 3 4 5]';
Amount = [10 15 20 25 30]';
tbl = table(Time,Amount);
```

Create a ScheduleDose object, and set the dosing information from the table.

```
sdose = sbiodose('sdose','schedule');
setTable(sdose,tbl);
```

### **Set an Array of Dosing Tables to an Array of Dose Objects**

Create a table containing dose time and amount.

```
Time = [1 2 3 4 5]';
Amount = [10 15 20 25 30]';
tbl1 = table(Time,Amount);
```

Create a table containing dose start time, amount, interval, repeat count, and rate.

```
StartTime = 5;
Amount = 500;
```

## setTable(ScheduleDose,RepeatDose)

---

```
Interval = 1;  
RepeatCount = 3;  
Rate = 1;  
tbl2 = table(StartTime,Amount,Interval,RepeatCount,Rate);
```

Create a cell array of dose tables.

```
tblArray = {tbl1,tbl2};
```

Create ScheduleDose and RepeatDose objects

```
sdose = sbiodose('sdose','schedule');  
rdose = sbiodose('rdose','repeat');  
doseArray = [sdose,rdose];
```

Set the cell array of dose tables to dose objects.

```
setTable(doseArray,tblArray);
```

### See Also

getTable | ScheduleDose object | RepeatDose object

## Purpose

Function-like interface to execute SimBiology models

## Description

The `SimFunction` object provides an interface that allows you to execute a SimBiology model like a function and a workflow to perform parameter scans (in parallel if Parallel Computing Toolbox is available), Monte Carlo simulations, and scans with multiple or vectorized doses. Since a `SimFunction` object can be executed like a function handle, you can customize it to integrate SimBiology models with other MATLAB products and other custom analyses (such as visual predictive checks).

Use the `createSimFunction` method to construct the `SimFunction` object. `SimFunction` objects are immutable once created.

## Syntax

If you specified any dosing information when you called `createSimFunction` to construct the `SimFunction` object `F`, then `F` has these signatures.

`simdata = F(phi,t_stop,u,t_output)` returns a `SimData` object `simdata` after simulating a SimBiology model using `phi`, a matrix of parameter values, `t_stop`, simulation stop time, `u`, dosing information, and `t_output`, output time.

`simdata = F(phi,t_stop,u)` uses the input arguments `phi`, `t_stop`, and `u`.

If you did not specify any dosing information when you called `createSimFunction`, then `F` has these signatures:

`simdata = F(phi,t_stop)` returns a `SimData` object `simdata` using `phi` and `t_stop`.

`simdata = F(phi,t_stop,[],t_output)` uses the input arguments `phi`, `t_stop`, empty dosed argument `[]`, and `t_output`. You must specify `u`, the dosing information, as an empty array `[]` for this signature.

# SimFunction object

---

---

**Note** When `t_output` is empty and `t_stop` is specified, the simulations report the solver time points until `t_stop`. When `t_output` is specified and `t_stop` is empty, only the time points in `t_output` are reported. When both are specified, the reported time points are the union of solver time points and the time points in `t_output`. If the last `t_output` is greater than the corresponding `t_stop`, then simulation proceeds until the last time point in `t_output`.

---

`simdata = F(phi, tbl)` uses the input arguments `phi` and `tbl`. Using this signature only lets you specify output times as one of the variables of `tbl`.

`[T,Y] = F(_)` returns `T`, a cell array of numeric vector, and `Y`, a cell array of 2-D numeric matrices, using any of the input arguments in the preceding syntaxes.

## Input Arguments

### **phi**

Matrix of size S-by-P, where S is the number of simulations to perform and P is the number of parameters specified in the `params` argument when you called `createSimFunction` to construct `F`. Each simulation is performed with the parameters specified in the corresponding row of `phi`.

When `phi` is specified as a 1-by-P matrix, then all simulations use the same parameters, and the number of simulations is determined from the `t_stop`, `u`, or `t_output` argument in that order.

For example, if `phi` and `t_stop` have a single row and `u` is a matrix of size N-by-DoseTargets, the number of simulations is determined as N.

### **t\_stop**

- Scalar specifying the same stop time for all simulations
- Vector of size N specifying a stop time for each simulation for all N simulations



## u

- table of dosing information with two or three variables representing dose time, dose amount, and dose rate (optional). You must name the table variables as follows.

```
u.Properties.VariableNames = {'Time', 'Amount', 'Rate'};
```

If `UnitConversion` is on, specify units for each variable. For instance, you can specify units as follows.

```
u.Properties.VariableUnits = {'second', 'molecule', 'molecule/second'};
```

This table can have multiple rows, where each row represents a dose applied to the dose target at a specified dose time with a specified amount and rate if available.

---

**Note** If you already have a `ScheduleDose` object, you can get this table by using the `getTable` method of the object.

---

- table with one row and five variables containing `RepeatDose` data. Dose rate variable is optional. Name the variables as follows.

```
u.Properties.VariableNames = {'StartTime', 'Amount', 'Rate', 'Interval'};
```

If `UnitConversion` is on, specify units for each variable. Units for 'RepeatCount' variable can be empty '' or 'dimensionless'. The unit of the 'Amount' variable must be dimensionally consistent with that of the target species. For example, if the unit of target species is in an *amount* unit (such as mole or molecule), then the 'Amount' variable unit must have the same dimension, i.e., its unit must be an *amount* unit and cannot be a *mass* unit (such as gram or kilogram). The unit for the 'Rate' variable must be dimensionally consistent as well.

```
u.Properties.VariableUnits = {'second', 'molecule', 'molecule/second'};
```

# SimFunction object

---

---

**Note** If you already have a `RepeatDose` object, you can get this table by using the `gettable` method of the object.

---

- Cell array of tables of size 1-by-N, where N is the number of dose targets. Each cell represents a table as described previously.
- Cell array of tables of size S-by-N, where S is the number of simulations and N is the number of dose targets. Each cell represents a table. S is equal to the number of rows in `phi`.

## **t\_output**

- Vector of monotonically increasing output times that is applied to all simulations
- Cell array containing a single time vector that is applied to all simulations
- Cell array of vectors representing output times. The *i*th cell element provides the output times for the *i*th simulation. The number of elements in the cell array must match the number of rows (simulations) in `phi`.

## **tbl**

table or dataset that has time and dosing information such as group labels, independent variable, dependent variable(s), amount(s), and rate(s). You must name the variables of the table or data set as 'GROUP', 'TIME', 'DEPENDENTVAR1', 'DEPENDENTVAR2', ..., 'AMOUNT1', 'RATE1', 'AMOUNT2', 'RATE2', ... The rate variable is optional for each dose.

If the `dosed` argument was empty when creating `F`, then amount- and rate-related variables are not required. If it is not empty, the number of amount and rate variables must match the number of dosed targets or species in `dosed`. The number of dependent variables must match the number of columns in `phi`.

If `UnitConversion` is on, specify a unit for each variable. The unit of 'Amount' variable must be dimensionally consistent with that of the target species. See the description of the input argument `u` for details.

## Output Arguments

### **simdata**

Array of `SimData` objects that contains results from executing the `SimFunction` `F`. The number of elements in the `simdata` array is the same as the number of rows in `phi`. The number of columns in each element of the `simdata` array, that is, `simdata(i).Data`, is equal to the number of elements in the observed cell array which was specified when creating `F`.

### **T**

Cell array containing a numeric vector of size  $S \times 1$ .  $S$  is the number of simulations. The  $i$ th element of `T` contains the time point from the  $i$ th simulation.

### **Y**

Cell array of 2-D numeric matrices. The  $i$ th element of `Y` contains data from the  $i$ th simulation. The number of rows in `T{i}` is equal to the number of rows in `Y{i}`.

## Constructor Summary

|                                       |  |
|---------------------------------------|--|
| <code>createSimFunction(model)</code> | Create <code>SimFunction</code> object |
|---------------------------------------|--|

## Method Summary

|   |   |
|---|---|
| <code>accelerate(SimFunction)</code>    | Prepare <code>SimFunction</code> object for accelerated simulations |
| <code>isaccelerated(SimFunction)</code> | Determine if <code>SimFunction</code> object is accelerated         |

# SimFunction object

---

## Property Summary

Parameters table with variables named:

- 'Name'
- 'Value'
- 'Type'
- 'Units' (only if UnitConversion is turned on)

The table contains information about model quantities (species, compartments, or parameters) that define the inputs of a `SimFunction` object. For instance, this table can contain parameters or species whose values are being scanned by the `SimFunction` object. This property is read only.

Observables table with variables named:

- 'Name'
- 'Type'
- 'Units' (only if UnitConversion is turned on)

This table contains information about model quantities (species, compartments, or parameters) that define the output of a `SimFunction` object. This property is read only.

**Dosed** table containing dosing information with variables named:

- 'TargetName'
- 'TargetDimension' (only if `UnitConversion` is turned on)
- 'DurationParameterName'
- 'DurationParameterValue'
- 'DurationParameterUnits' (only if `UnitConversion` is turned on)
- 'LagParameterName'
- 'LagParameterValue'
- 'LagParameterUnits' (only if `UnitConversion` is turned on)

Variables related to the lag and duration parameters are included only when 'LagParameterName' and 'DurationParameterName' are not empty. This property is read only.

**UseParallel** Logical. If true and Parallel Computing Toolbox is available, `SimFunction` is executed in parallel. This property is read-only.

**UnitConversion** Logical. If true,

- During the execution of the `SimFunction` object, `phi` is assumed to be in the same units as units for corresponding model quantities specified in the `params` argument when the object was created using the `createSimFunction` method.
- Time (`t_output` or `t_stop`) is assumed to be in the same unit as the `TimeUnits` property of the active `configset` object of the `SimBiology` model from which `F` was created.

# SimFunction object

---

- Variables of dose tables (u) must have units specified by setting `u.Properties.VariableUnits` to a cell array of appropriate units. The dimension of the dose target such as an amount (molecule, mole, etc.) or mass (gram, kilogram, etc.), is stored on the `Dosed` property of `F`.
- The simulation result is in the same units as those specified on the corresponding quantities in the `SimBiology` model from which `F` was created.

This property is read only.

`DependentFiles` Cell array of strings containing the names of files that the model depends on. This property is used for deployment. This property is read only.

## Examples

### Scan Parameters of the Lotka-Volterra Model

This example shows how to execute different signatures of the `SimFunction` object to simulate and scan parameters of the Lotka-Volterra (predator-prey) model described by Gillespie [1].

Load the sample project containing the model `m1`.

```
sbioloadproject lotka;
```

Create a `SimFunction` object `f` with `c1` and `c2` as input parameters to be scanned, and `y1` and `y2` as the output of the function with no dose.

```
f = createSimFunction(m1,{'Reaction1.c1', 'Reaction2.c2'},{'y1', 'y2'}, [
```

```
f =
```

```
SimFunction
```

```
Parameters:
```

| Name | Value | Type |
|------|-------|------|
|------|-------|------|

|                |      |             |
|----------------|------|-------------|
| 'Reaction1.c1' | 10   | 'parameter' |
| 'Reaction2.c2' | 0.01 | 'parameter' |

Observables:

| Name | Type      |
|------|-----------|
| 'y1' | 'species' |
| 'y2' | 'species' |

Dosed: None

Define an input matrix that contains values for each parameter (c1 and c2) for each simulation. The number of rows indicates the total number of simulations, and each simulation uses the parameter values specified in each row.

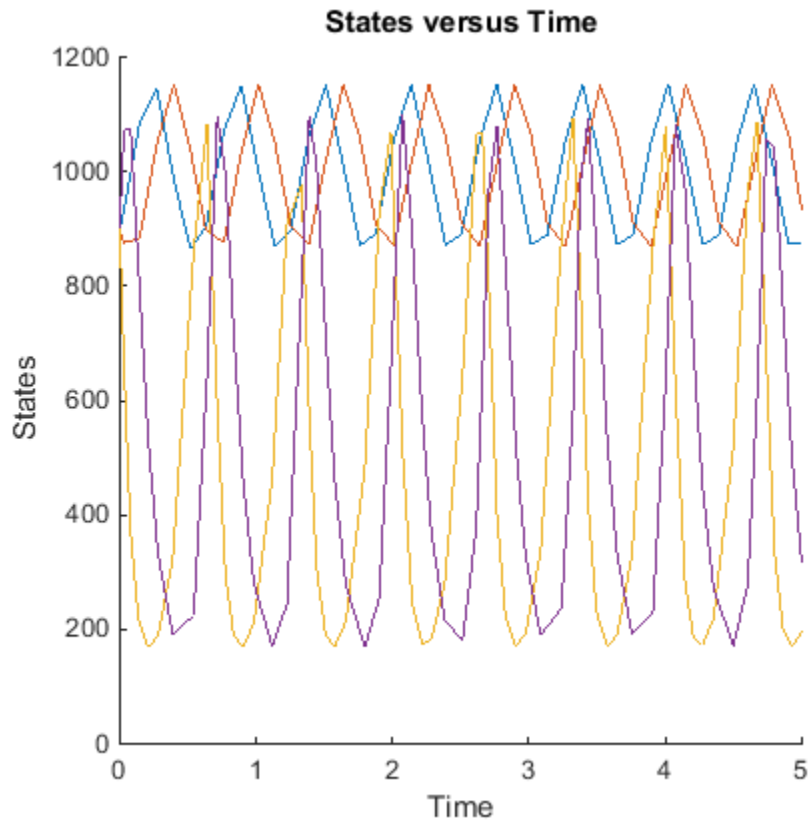
```
phi = [10 0.01; 10 0.02];
```

Run simulations until the stop time is 5 and plot the simulation results.

```
sbioplot(f(phi, 5));
```

# SimFunction object

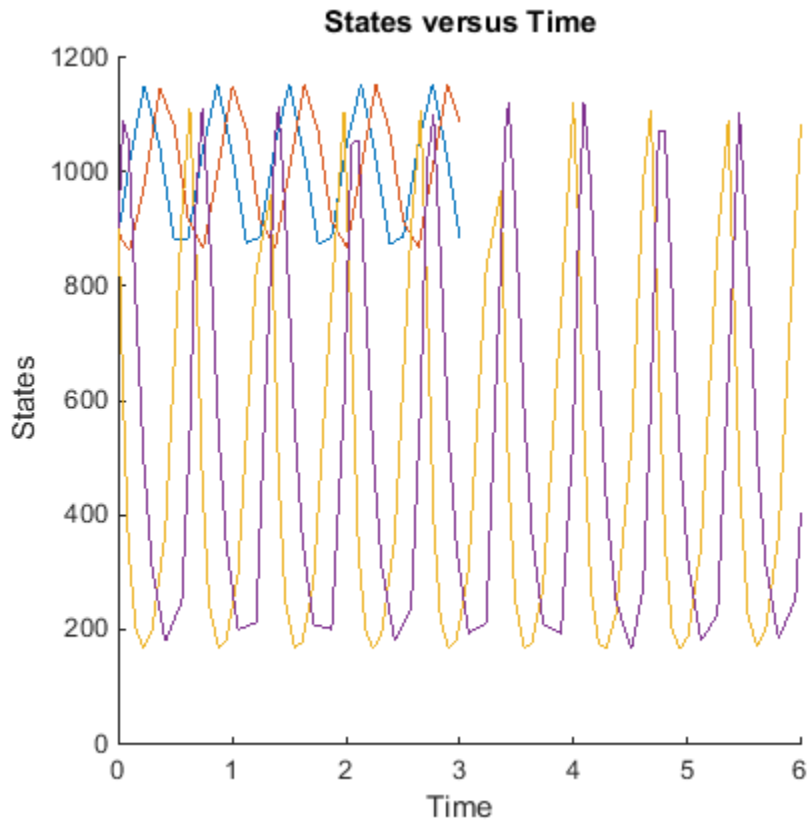
---



You can also specify a vector of different stop times for each simulation.

```
t_stop = [3;6];  
sbioplot(f(phi, t_stop));
```



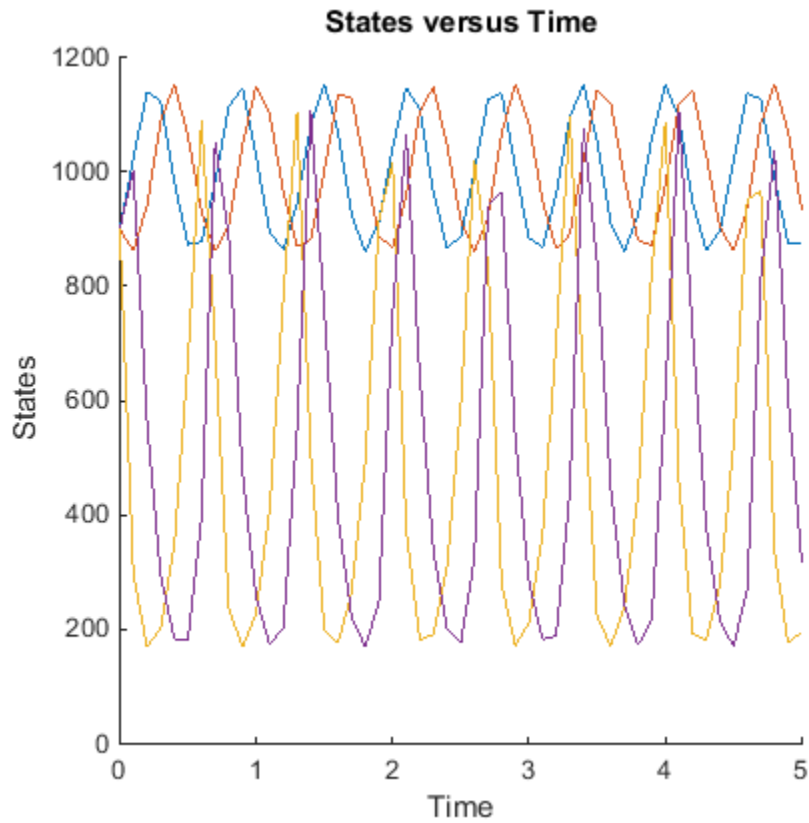


Next, specify the output times as a vector.

```
t_output = 0:0.1:5;  
sbioplot(f(phi,[],[],t_output));
```

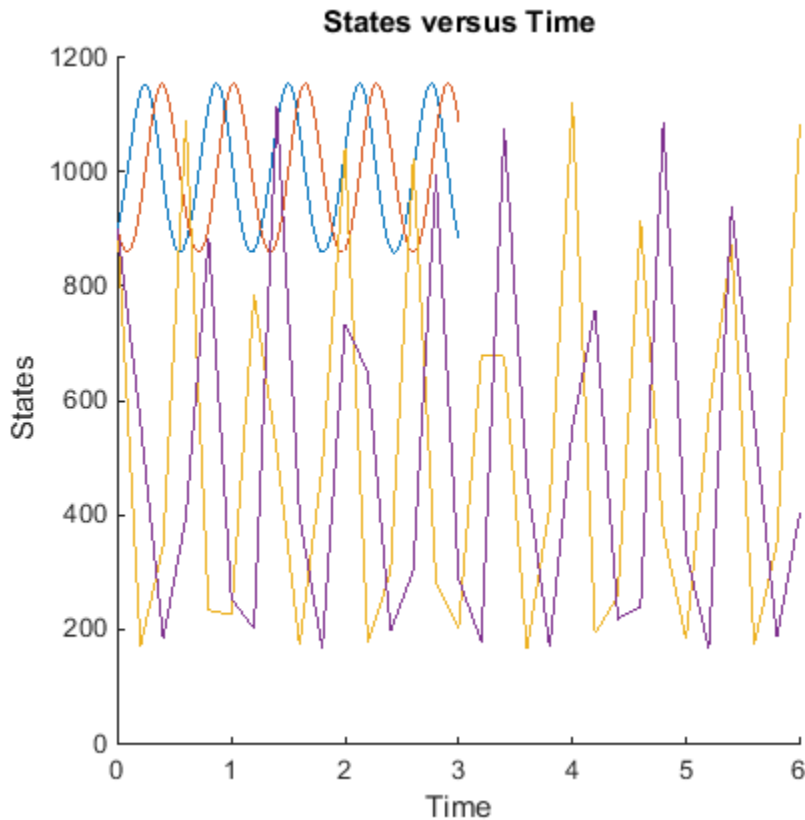
# SimFunction object

---



Specify output times as a cell array of vectors.

```
t_output = {0:0.01:3, 0:0.2:6};  
sbioplot(f(phi, []), [], t_output);
```



## Scan Initial Amounts of a Species from a Radioactive Decay Model

This example shows how to scan initial amounts of a species from a

radioactive decay model with the first-order reaction:  $\frac{dz}{dt} = c \cdot x$ , where  $x$  and  $z$  are species and  $c$  is the forward rate constant.

Load the sample project containing the radiodecay model `m1`.

## SimFunction object

---

```
sbioloadproject radiodecay;
```

Create a SimFunction object `f` to scan initial amounts of species `x`.

```
f = createSimFunction(m1,{'x'},{'x','z'},[])
```

```
f =
```

```
SimFunction
```

```
Parameters:
```

| Name | Value | Type      | Units      |
|------|-------|-----------|------------|
| 'x'  | 1000  | 'species' | 'molecule' |

```
Observables:
```

| Name | Type      | Units      |
|------|-----------|------------|
| 'x'  | 'species' | 'molecule' |
| 'z'  | 'species' | 'molecule' |

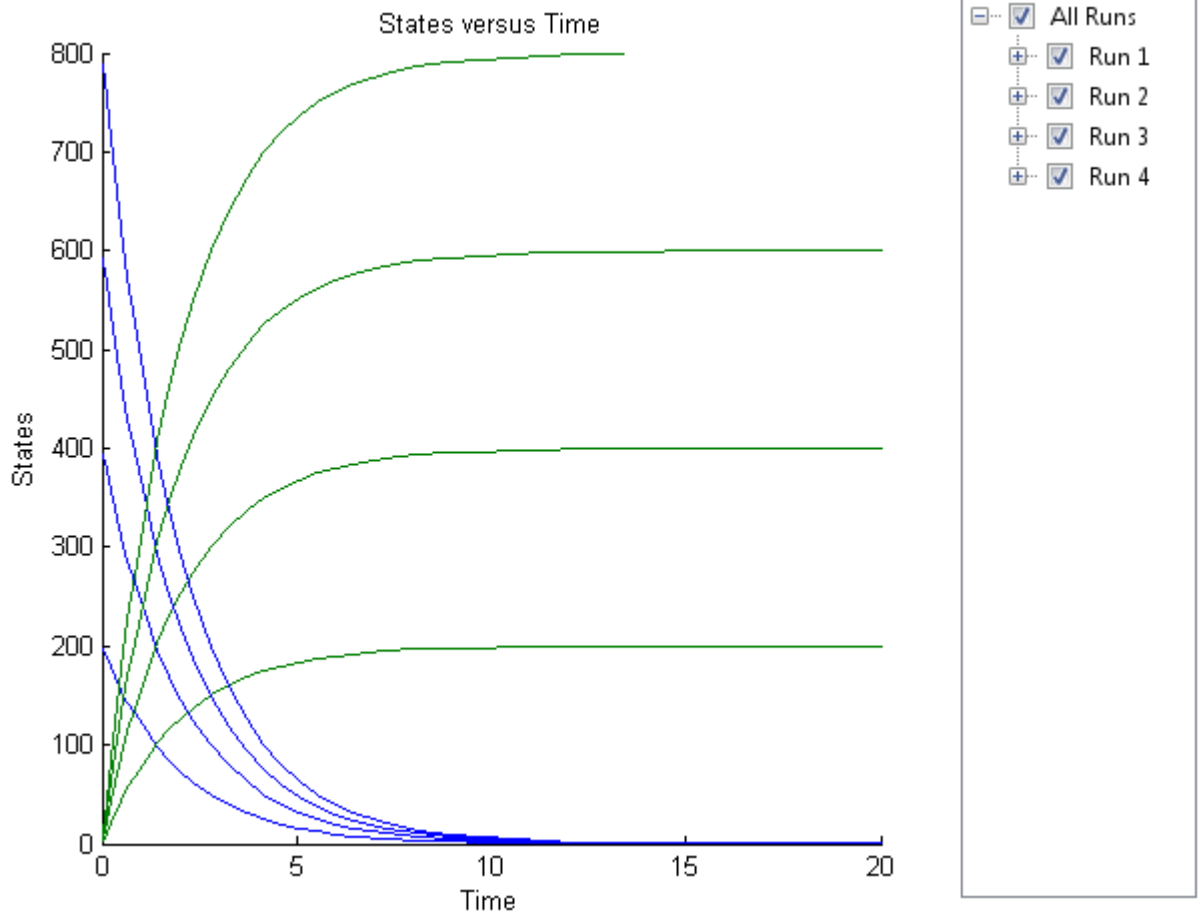
```
Dosed: None
```

Define four different initial amounts of species `x` for scanning. The number of rows indicates the total number of simulations, and each simulation uses the parameter value specified in each row of the vector.

```
phi = [200; 400; 600; 800];
```

Run simulations until the stop time is 20 and plot the simulation results.

```
sbioplot(f(phi, 20))
```



## Simulate a Model and Scan Parameters with Doses

This example shows how to simulate and scan a parameter of a radiodecay model while a species is being dosed.

Load the sample project containing the radiodecay model m1.

```
sbioloadproject radiodecay;
```

## SimFunction object

---

Create a `SimFunction` object `f` specifying parameter `Reaction1.c` to be scanned and species `x` as a dosed target.

```
f = createSimFunction(m1,{'Reaction1.c'},{'x','z'},{'x'});
```

Define a scalar dose of amount 200 molecules given at three time points (5, 10, and 15 seconds).

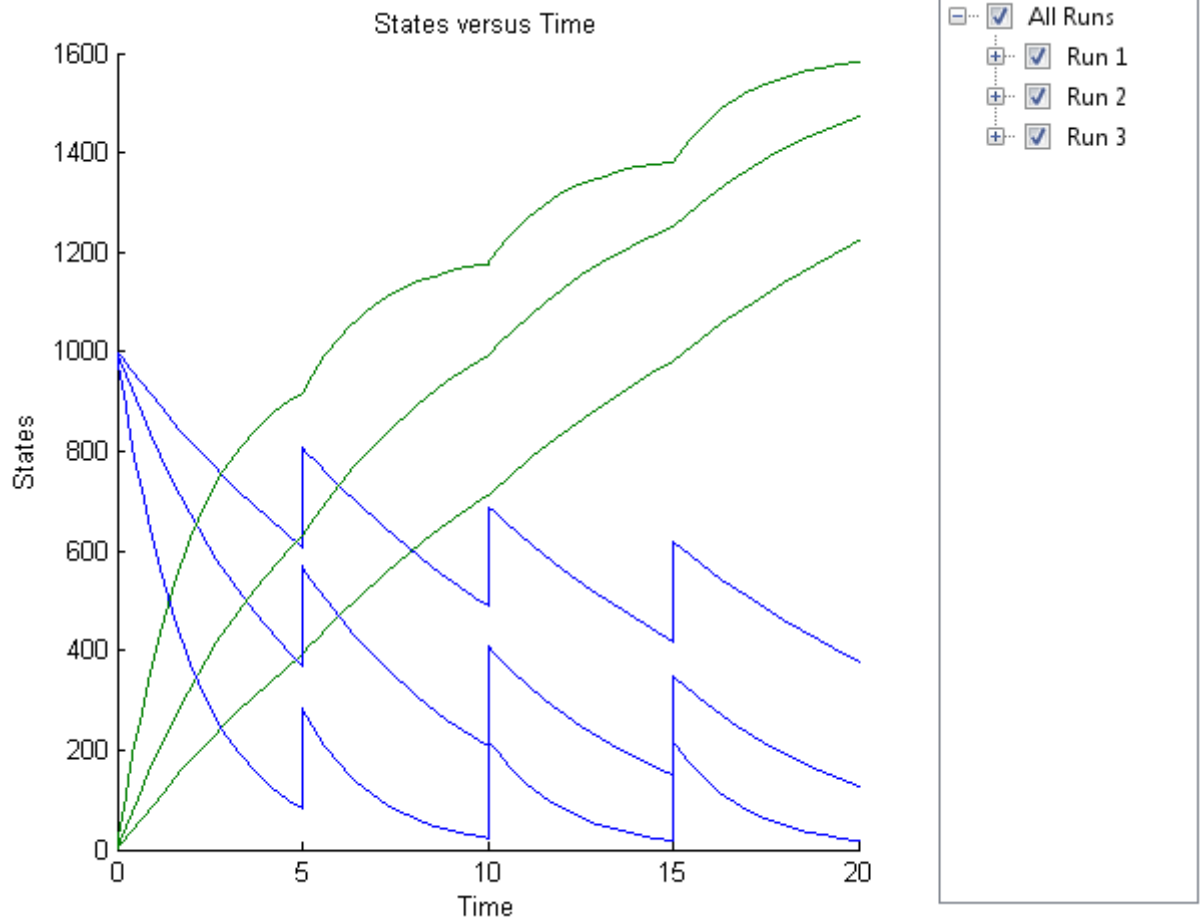
```
dosetime = [5 10 15];  
dose = [200 200 200];  
u = table(dosetime', dose');  
u.Properties.VariableNames = {'Time','Amount'};  
u.Properties.VariableUnits = {'second','molecule'};
```

Define the parameter values for `Reaction1.c` to scan.

```
phi = [0.1 0.2 0.5]';
```

Simulate the model for 20 seconds and plot the results.

```
sbioplot(f(phi,20,u));
```



You can also specify different dose amounts at different times.

```
d1 = table(5,100);  
d1.Properties.VariableNames = {'Time', 'Amount'};  
d1.Properties.VariableUnits = {'second', 'molecule'};  
d2 = table(10,300);
```

## SimFunction object

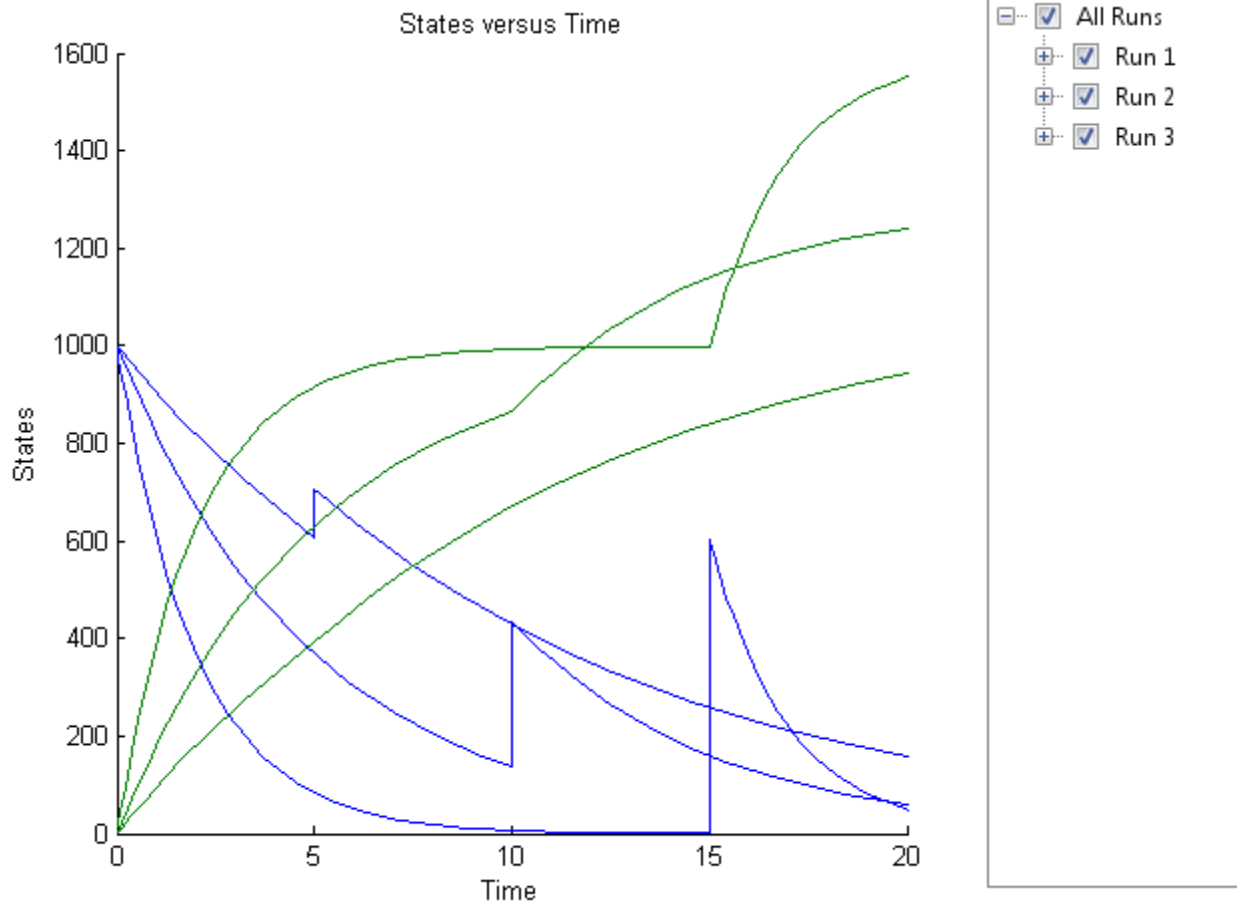
---

```
d2.Properties.VariableNames = {'Time', 'Amount'};
d2.Properties.VariableUnits = {'second', 'molecule'};
d3 = table(15,600);
d3.Properties.VariableNames = {'Time', 'Amount'};
d3.Properties.VariableUnits = {'second', 'molecule'};
```

Simulate the model using these doses and plot the results.

```
sbioplot(f(phi,20,{d1;d2;d3}));
```





You can also define a cell array of dose tables.

```
u = cell(3,1);  
dosetime = [5 10 15];  
dose = [200 200 200];  
u{1} = table(dosetime',dose');  
u{1}.Properties.VariableNames = {'Time','Amount'};
```

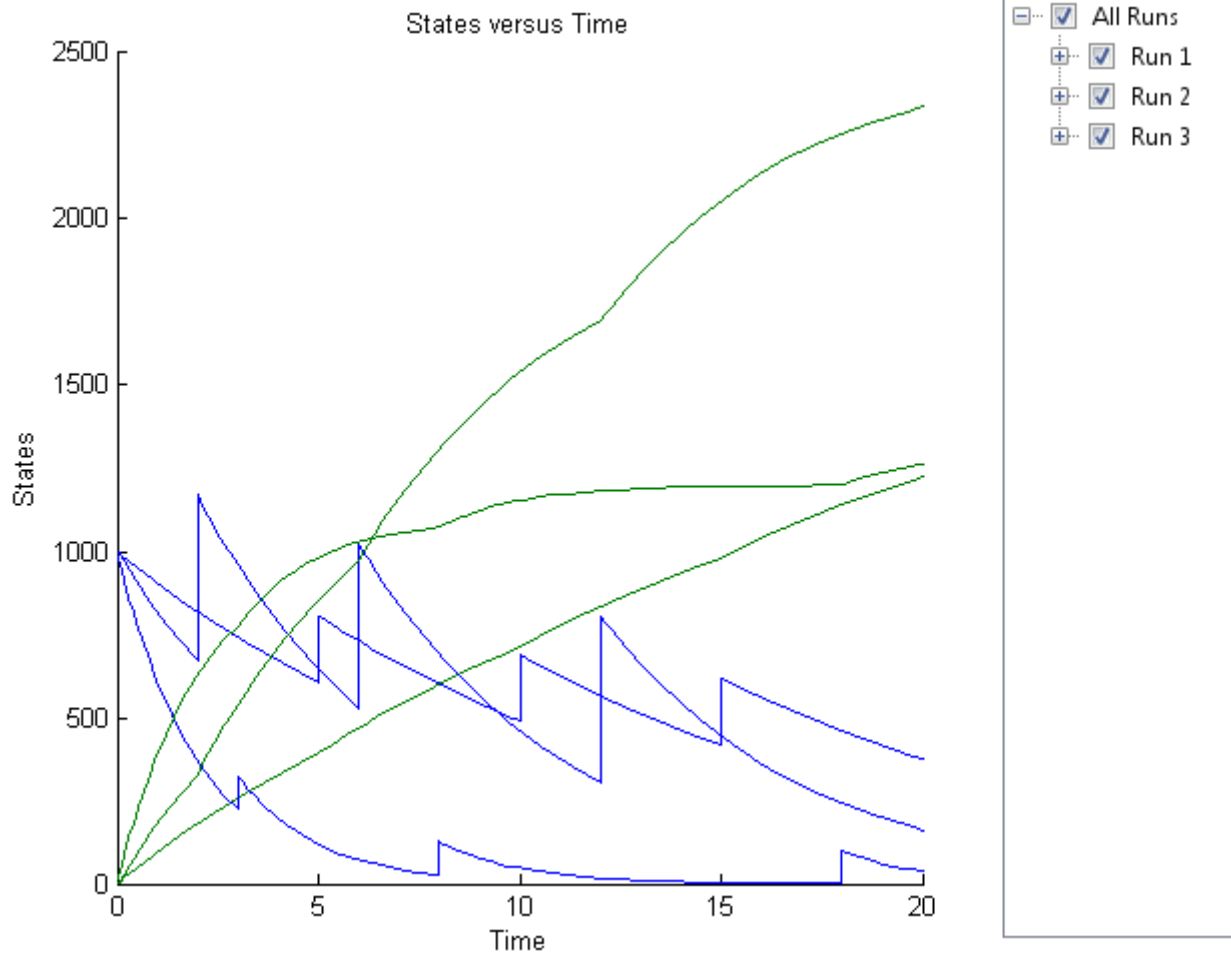
## SimFunction object

---

```
u{1}.Properties.VariableUnits = {'second','molecule'};
dosetime2 = [2 6 12];
dose2 = [500 500 500];
u{2} = table(dosetime2', dose2');
u{2}.Properties.VariableNames = {'Time','Amount'};
u{2}.Properties.VariableUnits = {'second','molecule'};
dosetime3 = [3 8 18];
dose3 = [100 100 100];
u{3} = table(dosetime3', dose3');
u{3}.Properties.VariableNames = {'Time','Amount'};
u{3}.Properties.VariableUnits = {'second','molecule'};
```

Simulate the model using the dose tables and plot results.

```
sbioplot(f(phi,20,u));
```



## See Also

[createSimFunction](#), [sbiosampleerror](#), [sbiosampleparameters](#)

## References

- [1] Gillespie, D.T. (1977). Exact Stochastic Simulation of Coupled Chemical Reactions. *The Journal of Physical Chemistry*. 81(25), 2340–2361.

**Purpose** Simulate exported SimBiology model

**Syntax**

```
[t,x,names] = simulate(model)
[t,x,names] = simulate(model,initialValues)
[t,x,names] = simulate(model,initialValues,doses)
simDataObj = simulate( ___ )
```

**Description** `[t,x,names] = simulate(model)` simulates a model, using the default initial values specified by `model.InitialValues` (which are always equal to the `InitialValue` property on the corresponding `ValueInfo` object). `simulate` returns:

- `t`, time samples.
- `x`, simulation data that contain variation in the quantity of states over time.
- `names`, column labels of simulation data `x`.

You can set additional simulation options using the property `SimBiology.export.Model.SimulationOptions`.

`[t,x,names] = simulate(model,initialValues)` simulates a model, using the values specified in `initialValues` as the initial values of the simulation.

`[t,x,names] = simulate(model,initialValues,doses)` simulates the model, using the specified initial values and doses.

`simDataObj = simulate( ___ )` returns simulation data in a `SimData` object `simDataObj` using any of the input arguments in the previous syntaxes. The `simDataObj` contains time and state data, as well as metadata, such as the types and names for the reported states. You can access the time, data, and names stores in `simDataObj` using the properties `simDataObj.Time`, `simDataObj.Data`, and `simDataObj.DataNames`, respectively.

# SimBiology.export.Model.simulate

---

## Input Arguments

### **model**

SimBiology.export.Model object.

### **initialValues**

Vector of values for `simulate` to use as the initial values of the simulation. `initialValues` must have the same number of elements as `model.InitialValues`.

**Default:** Values specified in `model.InitialValues`.

### **doses**

Vector of dose objects specifying the doses used for simulation. The input dose objects must be a subset of the doses in the exported model, as returned by `getdose`.

**Default:** All dose objects in the exported model.

## Output Arguments

### **t**

$n$ -by-1 vector of time samples from the simulation, where  $n$  is the number of time samples.

### **x**

$n$ -by- $m$  matrix of simulation data, where  $n$  is the number of time samples and  $m$  is the number of states logged during the simulation. Each column of `x` describes the variation in the quantity of a state over time.

### **names**

$m$ -by-1 cell array of strings with names labeling the rows and columns of `x`, respectively.

### **simDataObj**

SimData object containing simulation time and state data, as well as metadata, such as the types and names for the reported states.

## Examples

### Simulate an Exported SimBiology Model

Load a sample SimBiology model object, and select the species y1 and y2 for simulation.

```
modelObj = sbmlimport('lotka');  
modelObj.getConfigset.RuntimeOptions.StatesToLog = ...  
    sbioselect(modelObj, 'Name', {'y1', 'y2'});
```

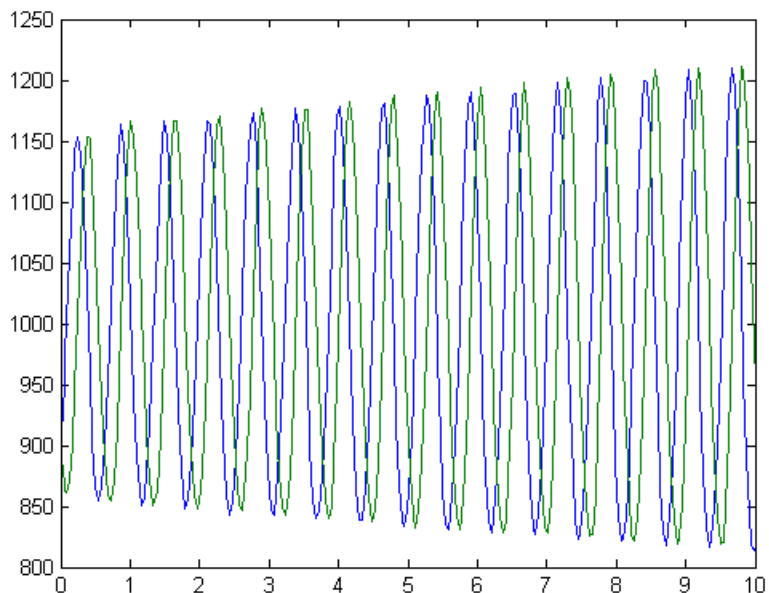
Export the model object.

```
em = export(modelObj);
```

Simulate the exported model.

```
[t,y] = simulate(em);
```

```
figure()  
plot(t,y)
```

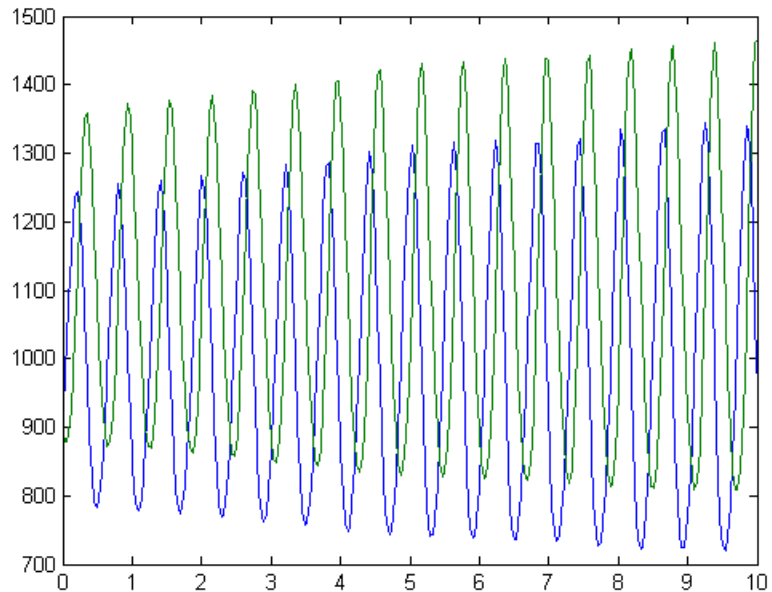


Modify the initial conditions, and simulate again.

```
xIndex = em.getIndex('x');  
em.InitialValues(xIndex) = em.InitialValues(xIndex)*1.1;  
[t,y] = simulate(em);
```

```
figure()  
plot(t,y)
```





## See Also

`SimBiology.export.Model` | `SimBiology.export.Model.getdose`  
| `SimBiology.export.ValueInfo` |  
`SimBiology.export.SimulationOptions` | `SimData` object | `export`

## Related Examples

- “PK/PD Modeling and Simulation to Guide Dosing Strategy for Antibiotics”
- “Deploy a SimBiology Model”

# SimData object

---

**Purpose** Simulation data storage

**Description** The SimBiology `SimData` object contains simulation data. The output from the `sbiosimulate` function, is stored in the `SimData` object which holds time and state data as well as metadata, such as the types and names for the logged states or the configuration set used during simulation.

You can also store data from multiple simulation runs as an array of `SimData` objects. Thus, the output of `sbioensemblerun` is an array of `SimData` objects. You can use any `SimData` method on an array of `SimData` objects.

You can access the time, data, and metadata stored in the `SimData` object through the properties in “Property Summary” on page 2-313. Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line.

Methods you can use to query the `SimData` object are listed in “Method Summary” on page 2-312.

## Constructor Summary

|                              |   |
|------------------------------|---|
| <code>sbioensemblerun</code> | Multiple stochastic ensemble runs of SimBiology model |
| <code>sbiosimulate</code>    | Simulate SimBiology model                             |

## Method Summary

|   |   |
|---|---|
| <code>delete</code> (any object)              | Delete SimBiology object                        |
| <code>display</code> (any object)             | Display summary of SimBiology object            |
| <code>get</code> (any object)                 | Get object properties                           |
| <code>getdata</code> ( <code>SimData</code> ) | Get data from <code>SimData</code> object array |

|                         |  |
|-------------------------|--|
| getsensmatrix (SimData) | Get 3-D sensitivity matrix from SimData array      |
| resample (SimData)      | Resample SimData object array onto new time vector |
| select (SimData)        | Select data from SimData object                    |
| selectbyname (SimData)  | Select data by name from SimData object array      |
| set (any object)        | Set object properties                              |

## Property Summary

|           |   |
|-----------|---|
| Data      | Store simulation data                         |
| DataCount | Numbers of species, parameters, sensitivities |
| DataInfo  | Metadata labels for simulation data           |
| DataNames | Show names in SimData object                  |
| ModelName | Name of model simulated                       |
| Name      | Specify name of object                        |
| Notes     | HTML text describing SimBiology object        |
| RunInfo   | Information about simulation                  |
| Time      | Simulation time steps or schedule dose times  |
| TimeUnits | Show time units for dosing and simulation     |
| UserData  | Specify data to associate with object         |

# SimData object

---

## Examples

Return simulation results to a SimData object and plot the results.

```
sbioloadproject('radiodecay', 'm1');
simDataObj = sbiosimulate(m1);
sbioplot(simDataObj)
```

Get simulation data at specific time points using the resample method.

```
% Load 'radiodecay' and set the initial amount of species 'x'.
sbioloadproject('radiodecay', 'm1');
x = sbioselect(m1, 'Type', 'species', 'Name', 'x');
x.InitialAmount = 100;
% Change the solver type to a stochastic solver.
cs = m1.getconfigset;
cs.SolverType = 'ssa';
% Simulate the model.
simDataObj = sbiosimulate(m1);
% This result could be misinterpreted as containing fractional molecules.
sbioplot(simDataObj);
title('Simulation Results Before Resampling');
%Resample the data using the zero-order hold method to obtain the correct
%number of molecules at intermediate time steps.
newsimDataObj = resample(simDataObj, linspace(0, 10, 1e4), 'zoh');
sbioplot(newsimDataObj);
title('Simulation Results After Resampling');
```

Initialize a simulation using results from a previous simulation.

```
% Load 'radiodecay'.
sbioloadproject('radiodecay', 'm1');
m1.Species
simDataObj = sbiosimulate(m1);
% Use the Data property to get the states at the final time point.
% Data is an m x n array, where m is the number of time steps in
% the simulation and n is the number of quantities logged.
finaldata = simDataObj.Data(end,:);
% Use the DataInfo property to get names of states.
info = simDataObj.DataInfo;
```

```
% Loop through the states (species) and set their initial amounts.
numSpecies = length(info);
for c = 1:numSpecies
    compObj = sbioselect(m1,'type','compartment','Name',info{c}.CompartmentName);
    speciesObj = sbioselect(compObj,'type','species','Name',info{c}.Name);
    speciesObj.InitialAmount = finaldata(c);
end
% Verify species initial amounts.
m1.Species
```

### See Also

Model object, Parameter object, Reaction object, Root object,  
Species object

# Species object

---

**Purpose** Options for compartment species

**Description** The SimBiology species object represents a *species*, which is a chemical or entity that participates in reactions, for example, DNA, ATP, Pi, creatine, G-Protein, or Mitogen-Activated Protein Kinase (MAPK). Species amounts can vary or remain constant during a simulation.

To add species that participate in reactions, add the reaction to the model. The process of adding the reaction to the model creates a compartment object (*unnamed*) and the necessary species objects.

Alternatively, create and add a species object to a compartment object, using the `addspecies` method at the command line.

When you use the SimBiology desktop to create a new model, it adds an empty compartment (*unnamed*), to which you can add species.

See “Property Summary” on page 2-317 for links to species property reference pages. Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change object properties in the graphical user interface.

**Constructor Summary**

|  |   |
|--|---|
| <code>addspecies (model, compartment)</code> | Create species object and add to compartment object within model object |
|--|---|

**Method Summary** Methods for species objects

|                                   |   |
|-----------------------------------|---|
| <code>copyobj (any object)</code> | Copy SimBiology object and its children |
| <code>delete (any object)</code>  | Delete SimBiology object                |
| <code>display (any object)</code> | Display summary of SimBiology object    |

|  |                                      |
|--|--------------------------------------|
| get (any object)                                   | Get object properties                |
| rename (compartment, parameter, species, reaction) | Rename object and update expressions |
| set (any object)                                   | Set object properties                |

## Property Summary

Properties for species objects

|                    |   |
|--------------------|---|
| BoundaryCondition  | Indicate species boundary condition         |
| ConstantAmount     | Specify variable or constant species amount |
| InitialAmount      | Species initial amount                      |
| InitialAmountUnits | Species initial amount units                |
| Name               | Specify name of object                      |
| Notes              | HTML text describing SimBiology object      |
| Parent             | Indicate parent object                      |
| Tag                | Specify label for SimBiology object         |
| Type               | Display SimBiology object type              |
| UserData           | Specify data to associate with object       |

## See Also

Compartment object, Configset object, KineticLaw object, Model object, Parameter object, Reaction object, Root object, Rule object

# Unit object

---

**Purpose** Hold information about user-defined unit

**Description** The SimBiology unit object holds information about user-defined units. To create a unit, create the unit object and add the unit to the library using the `sbiaddtolibrary` function.

Use the unit object property `Composition` to specify the composition of your units. See “Property Summary” on page 2-318 for links to unit object property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change unit object properties in the SimBiology desktop, using the Library Explorer. For more information, see “What Are SimBiology Libraries?”.

## Constructor Summary

|                      |                          |
|----------------------|--------------------------|
| <code>sbiunit</code> | Create user-defined unit |
|----------------------|--------------------------|

## Method Summary

|                                   |                                      |
|-----------------------------------|--------------------------------------|
| <code>delete (any object)</code>  | Delete SimBiology object             |
| <code>display (any object)</code> | Display summary of SimBiology object |
| <code>get (any object)</code>     | Get object properties                |
| <code>set (any object)</code>     | Set object properties                |

## Property Summary

|                          |   |
|--------------------------|---|
| <code>Composition</code> | Unit composition                                |
| <code>Multiplier</code>  | Relationship between defined unit and base unit |
| <code>Name</code>        | Specify name of object                          |
| <code>Notes</code>       | HTML text describing SimBiology object          |



|          |                                       |
|----------|---------------------------------------|
| Offset   | Unit composition modifier             |
| Parent   | Indicate parent object                |
| Tag      | Specify label for SimBiology object   |
| Type     | Display SimBiology object type        |
| UserData | Specify data to associate with object |

## See Also

AbstractKineticLaw object, KineticLaw object, Model object, Parameter object, Reaction object, Root object, Rule object, Species object, UnitPrefix object

# UnitPrefix object

---

**Purpose** Hold information about user-defined unit prefix

**Description** The SimBiology unit prefix object holds information about user-defined unit prefixes. To create a unit prefix, create the unit prefix object and add the unit prefix to the library using the `sbiaddtolibrary` function.

Use the unit prefix object property `Exponent`, to specify the exponent of your unit prefix. See “Property Summary” on page 2-320 for links to unit prefix object property reference pages.

Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change their values at the command line. You can graphically change unit prefix object properties in the SimBiology desktop, using the Library Explorer. For more information, see “What Are SimBiology Libraries?”.

|                            |                            |                                 |
|----------------------------|----------------------------|---------------------------------|
| <b>Constructor Summary</b> | <code>sbiunitprefix</code> | Create user-defined unit prefix |
|----------------------------|----------------------------|---------------------------------|

|                       |                                   |                                      |
|-----------------------|-----------------------------------|--------------------------------------|
| <b>Method Summary</b> | <code>delete (any object)</code>  | Delete SimBiology object             |
|                       | <code>display (any object)</code> | Display summary of SimBiology object |
|                       | <code>get (any object)</code>     | Get object properties                |
|                       | <code>set (any object)</code>     | Set object properties                |

|                         |                       |  |
|-------------------------|-----------------------|--|
| <b>Property Summary</b> | <code>Exponent</code> | Exponent value of unit prefix          |
|                         | <code>Name</code>     | Specify name of object                 |
|                         | <code>Notes</code>    | HTML text describing SimBiology object |
|                         | <code>Parent</code>   | Indicate parent object                 |

|          |                                       |
|----------|---------------------------------------|
| Tag      | Specify label for SimBiology object   |
| Type     | Display SimBiology object type        |
| UserData | Specify data to associate with object |

## See Also

AbstractKineticLaw object, KineticLaw object, Model object, Parameter object, Reaction object, Root object, Rule object, Species object, Unit object

# Variant object

---

**Purpose** Store alternate component values

**Description** The SimBiology variant object stores the names and values of model components and allows you to use the values stored in a variant object as the alternate value to be applied during a simulation. You can store values for species `InitialAmount`, parameter `Value`, and compartment `Capacity` in a variant object. Simulating using a variant does not alter the model component values. The values specified in the variant temporarily apply during simulation.

Using one or more variant objects associated with a model allows you to evaluate model behavior during simulation, with different values for the various model components without having to search and replace these values, or having to create additional models with these values. If you determine that the values in a variant object accurately define your model, you can permanently replace the values in your model with the values stored in the variant object, using the `commit` method.

To use a variant in a simulation you must add the variant object to the model object and set the `Active` property of the variant to true. Set the `Active` property to true if you always want the variant to be applied before simulating the model. You can also enter the variant object as an argument to `sbiosimulate`; this applies the variant only for the current simulation and supersedes any active variant objects on the model.

When there are multiple active variant objects on a model, if there are duplicate specifications for a property's value, the last occurrence for the property value in the array of variants, is used during simulation. You can find out which variant is applied last by looking at the indices of the variant objects stored on the model. Similarly, in the `Content` property, if there are duplicate specifications for a property's value, the last occurrence for the property in the `Content` property, is used during simulation.

Use the `addcontent` method to append contents to a variant object.

See “Property Summary” on page 2-323 for links to species property reference pages. Properties define the characteristics of an object. Use the `get` and `set` commands to list object properties and change

their values at the command line. You can graphically change object properties in the graphical user interface.

## Constructor Summary

`sbiovariant` Construct variant object

## Method Summary

Methods for variant objects

`addcontent (variant)` Append content to variant object  
`commit (variant)` Commit variant contents to model  
`copyobj (any object)` Copy SimBiology object and its children  
`delete (any object)` Delete SimBiology object  
`display (any object)` Display summary of SimBiology object  
`get (any object)` Get object properties  
`rmcontent (variant)` Remove contents from variant object  
`set (any object)` Set object properties  
`verify (model, variant)` Validate and verify SimBiology model

## Property Summary

Properties for variant objects

`Active` Indicate object in use during simulation  
`Content` Contents of variant object  
`Name` Specify name of object  
`Notes` HTML text describing SimBiology object

## Variant object

---

|          |                                       |
|----------|---------------------------------------|
| Parent   | Indicate parent object                |
| Tag      | Specify label for SimBiology object   |
| Type     | Display SimBiology object type        |
| UserData | Specify data to associate with object |

### See Also

Compartment object, Configset object, Model object, Parameter object, Species object

`sbiosimulate`

**Purpose**

Validate and verify SimBiology model

**Syntax**

```
verify(modelObj)
```

```
verify(modelObj,optionObj)
```

```
verify(modelObj,csObj,dvObj)
```

```
verify(modelObj,csObj,variantObj,doseObj)
```

**Description**

`verify(modelObj)` performs checks on a model object `modelObj` to verify that you can simulate the model. This method generates stacked errors and warnings if it finds any problems. To see the entire list of errors and warnings, use `sbiolasterror` and `sbiolastwarning`. The `verify` method uses the active configuration set for verification.

`verify(modelObj,optionObj)` verifies a model object `modelObj` while using an option object specified by `optionObj` as one of the following:

- Configset object
- Variant object
- ScheduleDose object
- RepeatDose object
- Array of doses or variants

`verify(modelObj,csObj,dvObj)` verifies a model object `modelObj` while using a configset object `csObj` and dose, variant, or an array of doses or variants specified by `dvObj`.

`verify(modelObj,csObj,variantObj,doseObj)` verifies a model object `modelObj` while using a configset object `csObj`, variant object or variant array specified by `variantObj` and dose object or dose array specified by `doseObj`.

# verify (model, variant)

---

## Input Arguments

### **modelObj - SimBiology model**

SimBiology model object

SimBiology model, specified as a SimBiology model object.

### **optionObj - Option object**

configset object | variant object or array of variant objects | dose object or array of dose objects

Option object, specified as a configset object, variant object, an array of variant objects, ScheduleDose object, RepeatDose object, or an array of dose objects.

### **csObj - Configuration set object**

configset object

Configuration set object, specified as a configset object that stores simulation-specific information.

### **dvObj - Dose or variant object**

dose object or array of dose objects | variant object or array of variant objects

Dose or variant object, specified as a ScheduleDose object, RepeatDose object, an array of dose objects, variant object, or an array of variant objects.

- When dvObj is a dose object, verify uses the specified dose object as well as any active variant objects if available.
- When dvObj is a variant object, verify uses the specified variant object as well as any active dose objects if available.

### **variantObj - Variant object**

variant object or array of variant objects

Variant object, specified as a variant object or an array of variant objects.

### **doseObj - Dose object**



dose object or array of dose objects

Dose object, specified as a `ScheduleDose` object, `RepeatDose` object, or an array of dose objects. A dose object defines additions that are made to species amounts or parameter values.

### Examples

#### Verify a SimBiology Model while Using a User-Defined Configset Object

Load a sample SimBiology model.

```
sbioloadproject radiodecay.sbproj
```

Add a new configuration set using a different stop time of 15 seconds.

```
csObj = addconfigset(m1,'newStopTimeConfigSet');  
csObj.StopTime = 15;
```

Verify the model while using the configset object.

```
verify(m1,csObj);
```

After verification, check the latest errors and warnings if there is any.

```
sbiolasterror
```

```
ans =
```

```
    []
```

```
sbiolastwarning
```

```
ans =
```

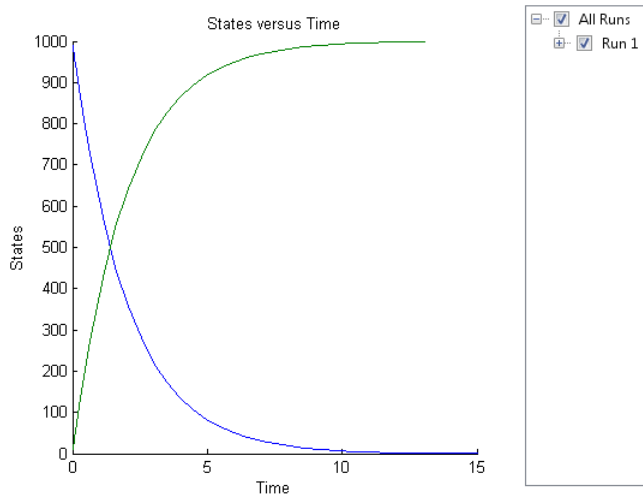
```
    []
```

Simulate the model.

```
sim = sbiosimulate(m1,csObj);  
sbioplot(sim)
```

# verify (model, variant)

---



## Verify a SimBiology Model while Using Configset and Dose Objects

Load a sample SimBiology model.

```
sbioloadproject radiodecay.sbproj
```

Get the default configuration set from the model.

```
defaultConfigSet = getconfigset(m1,'default');
```

Add a scheduled dose of 100 molecules at 2 seconds for species x.

```
dObj = adddose(m1,'d1','schedule');  
dObj.Amount = 100;  
dObj.AmountUnits = 'molecule';  
dObj.TimeUnits = 'second';  
dObj.Time = 2;  
dObj.TargetName = 'unnamed.x';
```

Verify the model while using the default configset object and added dose object.

```
verify(m1,defaultConfigSet,dObj);
```

After verification, check the latest errors and warnings if there is any.

```
sbiolasterror
```

```
ans =
```

```
    []
```

```
sbiolastwarning
```

```
ans =
```

```
    []
```

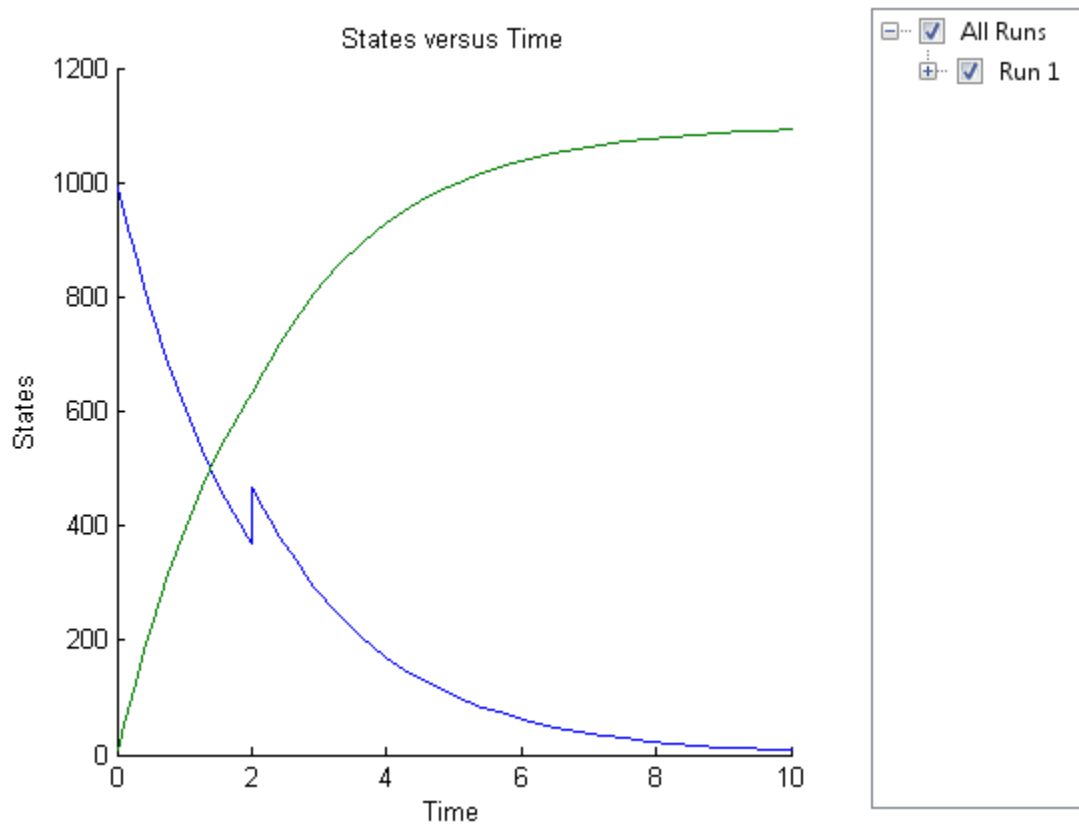
Simulate the model using the same configset and dose objects.

```
sim = sbiosimulate(m1,defaultConfigSet,dObj);
```

Plot the result.

```
sbioplot(sim);
```

## verify (model, variant)



### Verify a SimBiology Model while Using Configset, Dose, and Variant Objects

Load a sample SimBiology model.

```
sbioloadproject radiodecay.sbproj
```

Add a new configuration set using a different stop time of 15 seconds.

```
csObj = m1.addconfigset('newStopTimeConfigSet');  
csObj.StopTime = 15;
```

Add a scheduled dose of 100 molecules at 2 seconds for species x.

```
dObj = adddose(m1,'d1','schedule');  
dObj.Amount = 100;  
dObj.AmountUnits = 'molecule';  
dObj.TimeUnits = 'second';  
dObj.Time = 2;  
dObj.TargetName = 'unnamed.x';
```

Add a variant of species x using a different initial amount of 500 molecules.

```
vObj = addvariant(m1,'v1');  
addcontent(vObj,{'species','x','InitialAmount',500});
```

Verify the model while using the configset, dose, and variant objects. Note that the order of arguments should be as described.

```
verify(m1,csObj,vObj,dObj);
```

After verification, check the latest errors and warnings if there is any.

```
sbiolasterror
```

```
ans =
```

```
    []
```

```
sbiolastwarning
```

```
ans =
```

```
    []
```

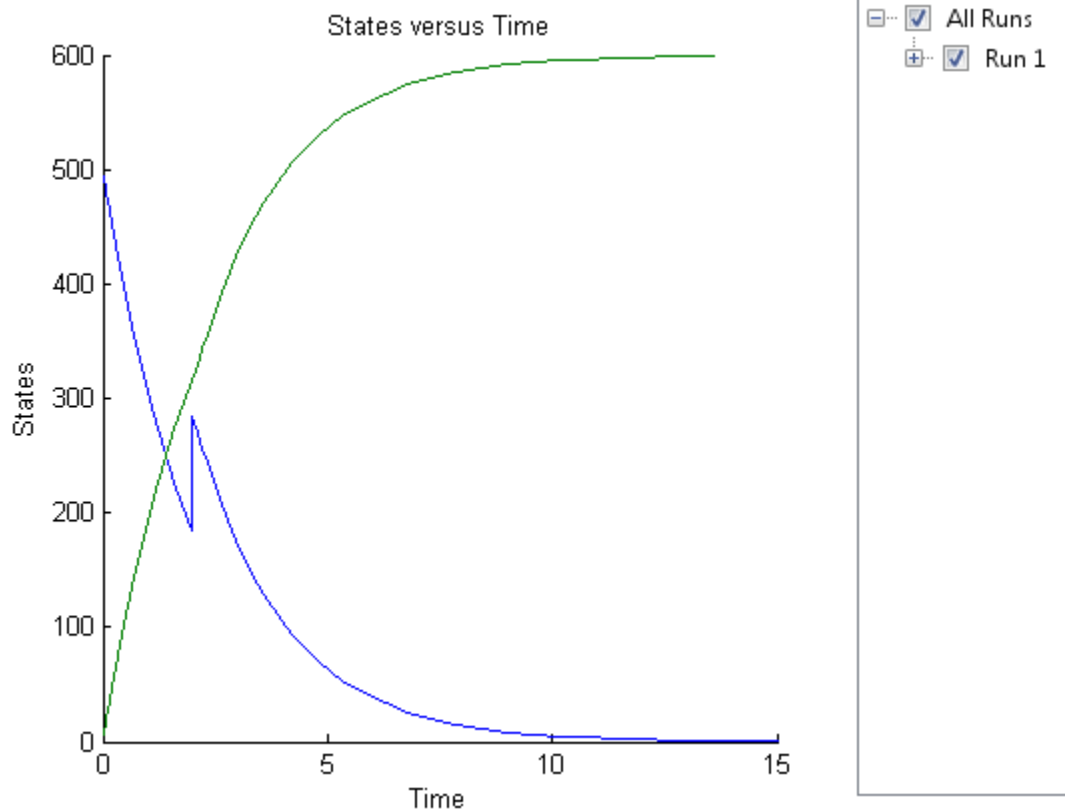
Simulate the model using the same configset, variant, and dose objects.

```
sim = sbiosimulate(m1,csObj,vObj,dObj);
```

Plot the result.

## verify (model, variant)

```
sbioplot(sim);
```



### See Also

`sbiolasterror`, `sbiolastwarning`

**Purpose** Check covariate model for errors

**Syntax** `verify(CovModelObj)`

**Description** `verify(CovModelObj)` verifies that the following are true about the Expression property of `CovModelObj`, a `CovariateModel` object:

- The expression strings are valid MATLAB code.
- Each expression string is linear with a transformation.
- There is exactly one expression string for each parameter.
- In each expression string, a covariate is used in at most one term.
- In each expression string, there is at most one random effect (`eta`)
- Fixed effect (`theta`) and random effect (`eta`) names are unique within and across expression strings. That is, each covariate has its own fixed effect.

If the previous requirements are true, then `verify` returns nothing.

**See Also** `construct` | `CovariateModel` | `Expression` | `PKModelDesign` object

- How To**
- Modeling the Population Pharmacokinetics of Phenobarbital in Neonates
  - “Specify a Covariate Model”

## **verify (covmodel)**

---



# Properties — Alphabetical List

---

# AbsoluteTolerance property

---

**Purpose** Absolute error tolerance applied to state value during simulation

**Description** AbsoluteTolerance is a property of a SolverOptions object, which is a property of a Configset object. It is available for the ode solvers (ode15s, ode23t, ode45, and sundials).

The software uses AbsoluteTolerance to determine the largest allowable absolute error at any step in a simulation. How the software uses AbsoluteTolerance to determine this error depends on whether the AbsoluteToleranceScaling property is enabled.

## When AbsoluteToleranceScaling Is Enabled

When the AbsoluteToleranceScaling property is enabled, the software uses the AbsoluteTolerance value as the absolute error tolerance for all state values whose size is of order 1. For all other state values, it scales the absolute error tolerance for each state value individually, based on that state value's maximum absolute value during simulation and the value of AbsoluteTolerance.

## When AbsoluteToleranceScaling Is Disabled

When the AbsoluteToleranceScaling property is disabled, the software uses the AbsoluteTolerance value as the absolute error tolerance for all state values, for example, amounts for all species.

**Algorithm** At each simulation step, the solver estimates the local error  $e_i$  in the  $i$ th state vector  $y$ . Simulation converges at that time step if  $e_i$  satisfies the following equation:

$$|e_i| \leq \max(\text{RelativeTolerance} * |y_i|, \text{AbsoluteTolerance})$$

Thus at higher state values, convergence is determined by RelativeTolerance. As the state values approach zero, convergence is controlled by AbsoluteTolerance. The choice of values for RelativeTolerance and AbsoluteTolerance varies depending on the problem. The default values should work for first trials of the simulation. However if you want to optimize the solution, consider that there is a tradeoff between speed and accuracy:

- If the simulation takes too long, you can increase the values of `RelativeTolerance` and `AbsoluteTolerance` at the cost of some accuracy.
- If the results seem inaccurate, you can decrease the tolerance values, but this will slow down the solver.
- If the magnitude of the state values is high, you can try to decrease the relative tolerance to get more accurate results.

## Characteristics

|             |                                   |
|-------------|-----------------------------------|
| Applies to  | Object: SolverOptions             |
| Data type   | double                            |
| Data values | Positive scalar. Default is 1e-6. |
| Access      | Read/write                        |

## Examples

This example shows how to change `AbsoluteTolerance`.

- 1 Retrieve the `configset` object from the `modelObj`.

```
modelObj = sbiomodel('cell');  
configsetObj = getconfigset(modelObj)
```

- 2 Change the `AbsoluteTolerance` to 1e-8.

```
set(configsetObj.SolverOptions, 'AbsoluteTolerance', 1.0e-8);  
get(configsetObj.SolverOptions, 'AbsoluteTolerance')
```

```
ans =
```

```
1.0000e-008
```

## See Also

`AbsoluteToleranceScaling`, `AbsoluteToleranceStepSize`,  
`RelativeTolerance`

# AbsoluteToleranceScaling property

---

**Purpose** Control scaling of absolute error tolerance during simulation

**Description** AbsoluteToleranceScaling is a property of a SolverOptions object, which is a property of a Configset object. It is available for the ode solvers (ode15s, ode23t, ode45, and sundials).

AbsoluteToleranceScaling controls how the software determines the largest allowable absolute error at any step in a simulation.

## When AbsoluteToleranceScaling Is Enabled

When the AbsoluteToleranceScaling property is enabled, the software uses the AbsoluteTolerance value as the absolute error tolerance for all state values whose size is of order 1. For all other state values, it scales the absolute error tolerance for each state value individually, based on that state value's maximum absolute value during simulation and the value of AbsoluteTolerance.

## When AbsoluteToleranceScaling Is Disabled

When the AbsoluteToleranceScaling property is disabled, the software uses the AbsoluteTolerance value as the absolute error tolerance for all state values, for example, amounts for all species.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: SolverOptions                  |
| Data type   | logical                                |
| Data values | 1, 0, true, or false. Default is true. |
| Access      | Read/write                             |

**See Also** AbsoluteTolerance, AbsoluteToleranceStepSize, RelativeTolerance

# AbsoluteToleranceStepSize property

---

**Purpose** Initial guess for time step size for scaling of absolute error tolerance

**Description** AbsoluteToleranceStepSize is a property of a SolverOptions object, which is a property of a Configset object. It is available for the ode solvers (ode15s, ode23t, ode45, and sundials).

When the AbsoluteToleranceScaling property is enabled, you can set the AbsoluteToleranceStepSize property to specify the initial guess for time step size for scaling. Then, for all state values whose size is of order 1, the software scales the absolute error tolerance for each state value individually, based on that state value's maximum absolute value during simulation and the value of AbsoluteTolerance.

---

**Tip** Use AbsoluteToleranceStepSize when a simulation is unsuccessful and generates numerically unstable solutions, and other corrective actions such as checking the model's kinetics do not work. You might encounter unstable solutions if you have very stiff systems in which state values change rapidly at the beginning of a simulation. To solve this, iteratively decrease AbsoluteToleranceStepSize and simulate to find the optimal setting. As a starting point, try setting this property to  $\text{AbsoluteTolerance} * \text{StopTime} * 0.1$ .

---

## Characteristics

Applies to Object: SolverOptions

Data type double

Data values Scalar in units specified by TimeUnits property. Default is [].

Access Read/write

**See Also** AbsoluteTolerance, AbsoluteToleranceScaling, RelativeTolerance

# Active property

---

**Purpose** Indicate object in use during simulation

**Description** The Active property indicates whether a simulation is using a SimBiology object. A SimBiology model is organized into a hierarchical group of objects. Use the Active property to include or exclude objects during a simulation.

- **Configuration set** — For the configset object, use the method `setactiveconfigset` to set the object Active property to true.
- **Event, Reaction, or Rule** — When an event, a reaction, or rule object Active property is set to false, the simulation does not include the event, reaction, or rule. This is a convenient way to test a model with and without a reaction or rule.
- **Variant** — Set the Active property to true if you always want the variant to be applied before simulating the model. You can also pass the variant object as an argument to `sbiosimulate`; this applies the variant only for the current simulation. For more information on using the Active property for variants, see `Variant` object.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Objects: configset, event, reaction, RepeatDose, rule, ScheduleDose, variant  |
| Data type   | boolean   |
| Data values | true or false. The default value for events, reactions, and rules is true. For the configset object, default is true. For added configset object, the default is false. For variants, the default is false. |
| Access      | Read/write  |

## Examples

1 Create a model object.

```
modelObj = sbiomodel ('my_model');
```

- 2 Add a reaction object and verify that the Active property setting is 'true' or 1.

```
reactionObj = addreaction (modelObj, 'a + b -> c + d');  
get (reactionObj, 'Active')
```

MATLAB returns:

```
ans =
```

```
1
```

- 3 Set the Active property to 'false' and verify.

```
set (reactionObj, 'Active', false);  
get (reactionObj, 'Active')
```

MATLAB returns:

```
ans =
```

```
0
```

### See Also

addconfigset, addreaction, addrule, Event object, Reaction object, RepeatDose object, Rule object, ScheduleDose object, Variant object,

# Amount property

---

**Purpose** Amount of dose

**Description** Amount is a property of a RepeatDose or ScheduleDose object. It defines an increase in the amount of a SimBiology species that receives a dose.

A RepeatDose object defines a series of doses. Each dose is the same amount, as defined by the Amount property, and given at equally spaced times, as defined by the Interval property. The number of injections in the series, excluding the initial injection, is defined by the RepeatCount property, and the Rate property defines how fast each dose is given.

A ScheduleDose object defines a series of doses. Each dose can have a different amount, as defined by an amount array in the Amount property, and given at specified times, as defined by a time array in the Time property. A rate array in the Rate property defines how fast each dose is given. At each time point in the time array, a dose is given with the corresponding amount and rate.

|                        |             |   |
|------------------------|-------------|---|
| <b>Characteristics</b> | Applies to  | Object: RepeatDose, ScheduleDose                                  |
|                        | Data type   | double (RepeatDose) or double array (ScheduleDose)                |
|                        | Data values | Nonnegative value. Default is 0 (RepeatDose) or [] (ScheduleDose) |
|                        | Access      | Read/write  |

**See Also** ScheduleDose object and RepeatDose object



**Purpose** Dose amount units

**Description** AmountUnits is a property of a RepeatDose or ScheduleDose object. This property defines units for the Amount property.

If the TargetName property defines a species, then AmountUnits for a dose must be a chemical amount (for example, milligram, mole, or molecule), not a concentration. To get a list of the defined units in the library, use the sbioshowunits function. To add a user-defined unit to the list, see sbioaddtolibrary.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Objects: RepeatDose, ScheduleDose                                     |
| Data type   | string  |
| Data values | Units from library with dimensions of amount.<br>Default = "" (empty) |
| Access      | Read/write  |

# AmountUnits property

---

---

**Note** SimBiology uses unit strings including empty units in association with DimensionalAnalysis and UnitConversion features.

- When DimensionalAnalysis and UnitConversion are both false, unit strings are not used. However, SimBiology still performs a minimum level of dimensional analysis to decide whether a reaction rate is in dimensions of amount/time or concentration/time.
- When DimensionalAnalysis is true and UnitConversion is false, units (if not empty) must have consistent dimensions so that SimBiology can perform dimensional analysis. However, the units are not converted.
- When UnitConversion is set to true (which requires DimensionalAnalysis to be true), SimBiology performs a dimensional analysis and converts everything to consistent units. Hence, you must specify consistent units, and no unit strings can be empty. If you have a dimensionless parameter, you must still set its unit string to dimensionless.

---

## See Also

ScheduleDose object and RepeatDose object

**Purpose** Store link to URL or file

---

**Note** The Annotation property will be removed in a future release. Use the Notes property instead.

---

**Description** The Annotation property stores the URL or file name linking to information about a model.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | SimBiology objects: abstract kinetic law, configuration set, compartment, event, kinetic law, model, parameter, reaction, RepeatDose, rule, ScheduleDose, species, or unit |
| Data type   | char string, URL   |
| Data values | Character string with a directory path and filename or a URL   |
| Access      | Read/write   |

## Examples

**1** Create a model object.

```
modelObj = sbiomodel ('my_model');
```

**2** Set the annotation for a model object.

```
set (modelObj, 'annotation', 'www.reactome.org')
```

**3** Verify the assignment.

```
get (modelObj, 'annotation')
```

MATLAB returns:

```
ans =
```

## Annotation property

---

[www.reactome.org](http://www.reactome.org)

### **See Also**

`addkineticlaw`, `addparameter`, `addreaction`, `addrule`, `addspecies`,  
`sbiomodel`, `sbioroot`, `sbiunit`, `sbiunitprefix`, `RepeatDose`  
object, `ScheduleDose` object

## Purpose

Indicate species boundary condition

## Description

The BoundaryCondition property indicates whether a species object has a boundary condition.

When the BoundaryCondition of a species is `false` (default), the species quantity is modified by reactions, rules, events, and doses. If the BoundaryCondition is `true`, the species quantity is modified by rules, events, and doses, but not by reactions.

Set the BoundaryCondition to `true` when you want the species to participate in a reaction, but do not want any reactions to modify its quantity.

All SimBiology species are state variables regardless of the BoundaryCondition or ConstantAmount property.

## More Information

Consider the following two use cases of boundary conditions:

- Modeling receptor-ligand interactions that affect the rate of change of the receptor but not the ligand. For example, in response to hormone, steroid receptors such as the glucocorticoid receptor (GR) translocate from the cytoplasm (`cyt`) to the nucleus (`nuc`). The `hsp90/hsp70` chaperone complex directs this nuclear translocation [Pratt 2004]. The natural ligand for GR is cortisol; the synthetic hormone dexamethasone (`dex`) is used in place of cortisol in experimental systems. In this system dexamethasone participates in the reaction but the quantity of dexamethasone in the cell is regulated using a rule. To simply model translocation of GR you could use the following reactions:

Formation of the chaperone-receptor complex,

```
Hsp90_complex + GR_cyt -> Hsp90_complex:GR_cyt
```

In response to the synthetic hormone dexamethasone (`dex`), GR moves from the cytoplasm to the nucleus.

# BoundaryCondition property

---

```
Hsp90_complex:GR_cyt + dex -> Hsp90_complex + GR_nuc + dex
```

For dex,

```
BoundaryCondition = true; ConstantAmount = false
```

In this example `dex` is modeled as a boundary condition with a rule to regulate the rate of change of `dex` in the system. Here, the quantity of `dex` is not determined by the rate of the second reaction but by a rate rule such as

```
ddex/dt = 0.001
```

which is specified in the SimBiology software as

```
dex = 0.001
```

- Modeling the role of nucleotides (for example, GTP, ATP, cAMP) and cofactors (for example, Ca<sup>++</sup>, NAD<sup>+</sup>, coenzyme A). Consider the role of GTP in the activation of Ras by receptor tyrosine kinases.

```
Ras-GDP + GTP -> Ras-GTP + GDP
```

For GTP, `BoundaryCondition = true; ConstantAmount = true`

Model GTP and GDP with boundary conditions, thus making them *boundary species*. In addition, you can set the `ConstantAmount` property of these species to `true` to indicate that their quantity does not vary during a simulation.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: species                            |
| Data type   | boolean                                    |
| Data values | true or false. The default value is false. |
| Access      | Read/write                                 |

## Examples

### Simulate a Model with a Boundary Condition for a Species

This example illustrates how to use the `BoundaryCondition` property of a species so that the species amount is not modified by the reaction it participates in, but by a user-defined dose object.

Load a sample project.

```
sbioloadproject radiodecay.sbproj
```

A SimBiology model named `m1` is loaded to the MATLAB Workspace. The model is a simple radioactive decay model in which two species (`x` and `z`) are modified by the following reaction.

```
m1.Reactions
```

```
SimBiology Reaction Array
```

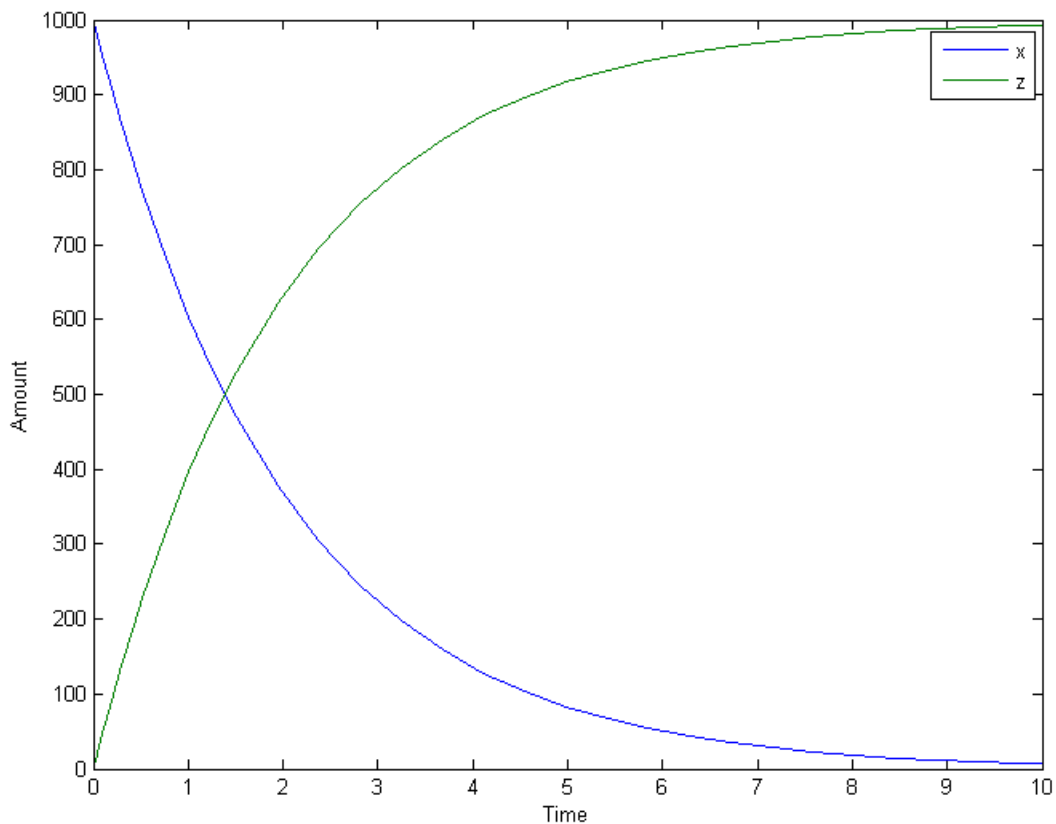
```
Index:    Reaction:
1         x -> z
```

Simulate the model and view results before adding any boundary conditions.

```
[t,x,names] = sbiosimulate(m1);
plot(t,x);
legend(names)
xlabel('Time');
ylabel('Amount');
```

# BoundaryCondition property

---



Add a RepeatDose object to the model and specify the species to be dosed, dose amount, dose schedule, and units.

```
d1 = adddose(m1,'d1','repeat');  
set(d1,'TargetName','z','Amount',100.0,'Interval',1.0,'RepeatCount',8);  
set(d1,'TimeUnits','second','AmountUnits','molecule');
```



Set the BoundaryCondition of species z to be true so that the species will be modified by the dose object d1, but not by the reaction.

```
set(m1.species(2), 'BoundaryCondition', true);
```

Simulate the model by applying the dose object.

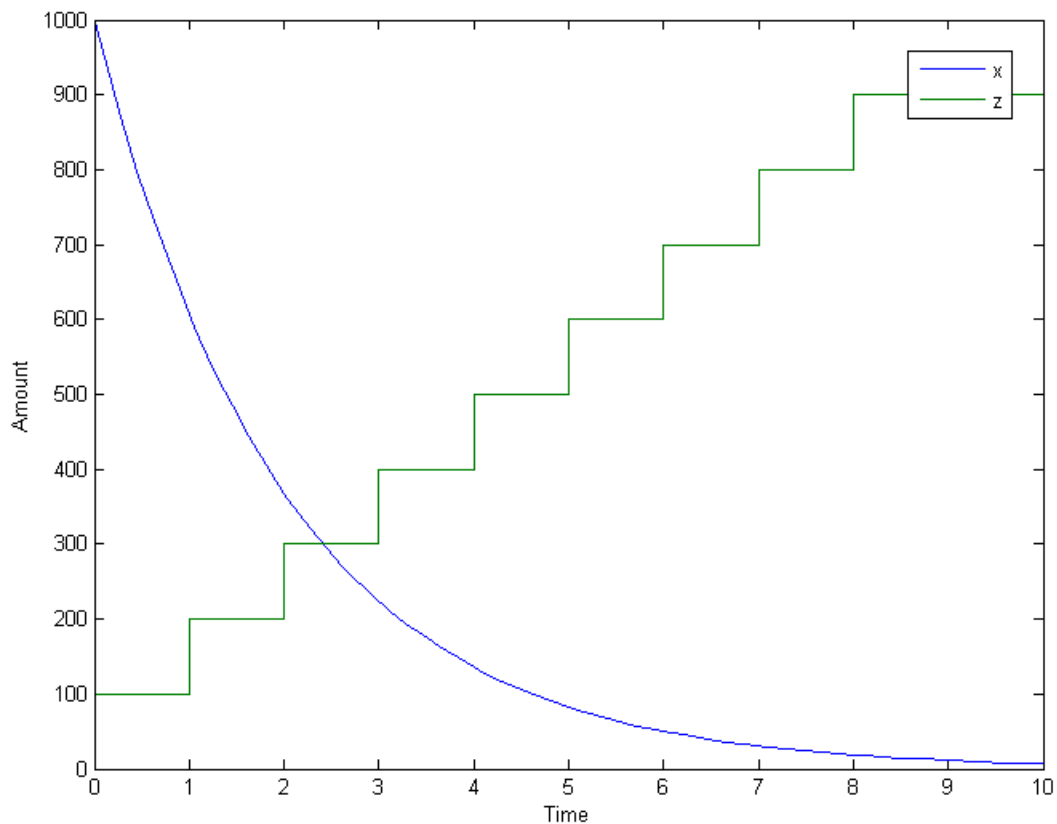
```
[t2,x2,names] = sbiosimulate(m1,d1);
```

Plot the results. Notice that the amount of species z is now modified by the repeated dose object, but not by the reaction.

```
[t2,x2,names] = sbiosimulate(m1,d1);  
plot(t2,x2);  
legend(names);  
xlabel('Time');  
ylabel('Amount');
```

# BoundaryCondition property

---



## References

Pratt, W.B., Galigniana, M.D., Morishima, Y., Murphy, P.J. (2004), Role of molecular chaperones in steroid receptor action, *Essays Biochem*, 40:41-58.

## See Also

addrule, addspecies, ConstantAmount, InitialAmount

**Purpose** Library of built-in components

**Description** `BuiltInLibrary` is a SimBiology root object property containing all built-in components namely, unit, unit-prefixes, and kinetic laws that are shipped with the SimBiology product. You cannot add, modify, or delete components in the built-in library. The `BuiltInLibrary` property is an object that contains the following properties:

- `Units` — contains all units that are shipped with the SimBiology product. You can specify units for compartment capacity, species amounts and parameter values, to do dimensional analysis and unit conversion during simulation. You can display the built-in units either by using the command `sbiowhos -builtin -unit`, or by accessing the root object.
- `UnitPrefixes` — contains all unit-prefixes that are shipped with the SimBiology product. You can specify unit—prefixes in combination with a valid unit for compartment capacity, species amounts and parameter values, to do dimensional analysis and unit conversion during simulation. You can display the built-in unit-prefixes either by using the command `sbiowhos -builtin -unitprefix`, or by accessing the root object.
- `KineticLaws` — contains all kinetic laws that are shipped with the SimBiology product. Use the command `sbiowhos -builtin -kineticlaw` to see the list of built-in kinetic laws. You can use built-in kinetic laws when you use the command `addkineticlaw` to create a kinetic law object for a reaction object. Refer to the kinetic law by name when you create the kinetic law object, for example, `kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');`

See “Kinetic Law Definition” on page 3-71 for a definition and more information.

**Characteristics** `BuiltInLibrary`

# BuiltInLibrary property

---

|             |   |
|-------------|---|
| Applies to  | Object: root  |
| Data type   | object  |
| Data values | Unit, unit-prefix, and abstract kinetic law objects |
| Access      | Read-only   |

Characteristics for BuiltInLibrary properties:

- Units

|             |                         |
|-------------|-------------------------|
| Applies to  | BuiltInLibrary property |
| Data type   | unit objects            |
| Data values | units                   |
| Access      | Read-only               |

- UnitPrefixes

|             |                         |
|-------------|-------------------------|
| Applies to  | BuiltInLibrary property |
| Data type   | unit prefix objects     |
| Data values | unit prefixes           |
| Access      | Read-only               |

- KineticLaws

|             |                             |
|-------------|-----------------------------|
| Applies to  | BuiltInLibrary property     |
| Data type   | Abstract kinetic law object |
| Data values | kinetic laws                |
| Access      | Read-only                   |

## Examples

### Example 1

This example uses the command `sbiowhos` to show the current list of built-in components.

```
sbiowhos -builtin -kineticlaw
sbiowhos -builtin -unit
sbiowhos -builtin -unitprefix
```

### Example 2

This example shows the current list of built-in components by accessing the root object.

```
rootObj = sbioroot;
get(rootObj.BuiltinLibrary, 'KineticLaws')
get(rootObj.BuiltinLibrary, 'Units')
get(rootObj.BuiltinLibrary, 'UnitPrefixes')
```

## See Also

Functions — `sbioaddtolibrary`, `sbioremovefromlibrary` `sbioroot`, `sbiounit`, `sbiounitprefix`

Properties — `UserDefinedLibrary`

# Capacity property

---

**Purpose**                    Compartment capacity

**Description**            The Capacity property indicates the size of the SimBiology compartment object. If the size of the compartment does not vary during simulation, set the property `ConstantCapacity` to `true`.

You can vary compartment capacity using rules or events.

---

**Note** Remember to set the `ConstantCapacity` property to `false` for varying capacity.

---

Events cannot result in the capacity having a negative value. Rules could result in the capacity having a negative value.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Object: compartment                           |
| Data type   | double  |
| Data values | Positive real number. The default value is 1. |
| Access      | Read/write                                    |

## Examples

Add a compartment to a model and set the compartment capacity.

**1** Create a model object named `my_model`.

```
modelObj = sbiomodel ('comp_model');
```

**2** Add the compartment object named `nucleus` with a capacity of 0.5.

```
compartmentObj = addcompartment(modelObj, 'nucleus', 0.5);
```

## See Also

`addcompartment`, `addspecies`, `CapacityUnits`, `ConstantCapacity`

**Purpose**                    Compartment capacity units

**Description**            The CapacityUnits property indicates the unit definition for the Capacity property of a compartment object. CapacityUnits can be any unit from the units library. To get a list of the defined units in the library, use the sbioshowunits function. If CapacityUnits changes from one unit definition to another, the Capacity does not automatically convert to the new units. The sbioconvertunits function does this conversion. To add a user-defined unit to the list, see sbioaddtolibrary.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: compartment  |
| Data type   | char string  |
| Data values | Units from library with dimensions of length, area, or volume. Default = '' (empty). |
| Access      | Read/write   |

# CapacityUnits property

---

---

**Note** SimBiology uses unit strings including empty units in association with DimensionalAnalysis and UnitConversion features.

- When DimensionalAnalysis and UnitConversion are both false, unit strings are not used. However, SimBiology still performs a minimum level of dimensional analysis to decide whether a reaction rate is in dimensions of amount/time or concentration/time.
- When DimensionalAnalysis is true and UnitConversion is false, units (if not empty) must have consistent dimensions so that SimBiology can perform dimensional analysis. However, the units are not converted.
- When UnitConversion is set to true (which requires DimensionalAnalysis to be true), SimBiology performs a dimensional analysis and converts everything to consistent units. Hence, you must specify consistent units, and no unit strings can be empty. If you have a dimensionless parameter, you must still set its unit string to dimensionless.

## Examples

- 1 Create a model object named my\_model.

```
modelObj = sbiomodel ('my_model');
```

- 2 Add a compartment object named cytoplasm with a capacity of 0.5.

```
compObj = addcompartment (modelObj, 'cytoplasm', 0.5);
```

- 3 Set the CapacityUnits to femtoliter, and verify.

```
set (compObj, 'CapacityUnits', 'femtoliter');  
get (compObj, 'CapacityUnits')
```

MATLAB returns:

```
ans =
```



femtoliter

### **See Also**

InitialAmount, sbioaddtolibrary, sbioconvertunits,  
sbioshowunits

# Compartments property

---

**Purpose** Array of compartments in model or compartment

**Description** Compartments shows you a read-only array of SimBiology compartment objects in the model object and the compartment object. In the model object, the Compartments property indicates all the compartments in a Model object as a flat list. In the compartment object, the Compartments property indicates other compartments that are referenced within the compartment. The two instances of Compartments are illustrated in “Examples” on page 3-26.

You can add a compartment object using the method `addcompartment`.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Objects: compartment, model                |
| Data type   | Array of compartment objects               |
| Data values | Compartment object. Default is [] (empty). |
| Access      | Read-only                                  |

## Examples

**1** Create a model object named `modelObj`.

```
modelObj = sbiomodel('cell');
```

**2** Add two compartments to the model object.

```
compartmentObj1 = addcompartment(modelObj, 'nucleus');  
compartmentObj2 = addcompartment(modelObj, 'mitochondrion');
```

**3** Add a compartment to one of the compartment objects.

```
compartmentObj3 = addcompartment(compartmentObj2, 'matrix');
```

**4** Display the Compartments property in the model object.

```
get(modelObj, 'Compartments')
```

```
SimBiology Compartment Array
```

| Index: | Name:         | Capacity: | CapacityUnits: |
|--------|---------------|-----------|----------------|
| 1      | nucleus       | 1         |                |
| 2      | mitochondrion | 1         |                |
| 3      | matrix        | 1         |                |

- 5 Display the Compartments property in the compartment object.

```
get(compartmentObj2, 'Compartments')
```

```
SimBiology Compartment - matrix
```

```
Compartment Components:  
Capacity:          1  
CapacityUnits:  
Compartments:     0  
ConstantCapacity: true  
Owner:            mitochondrion  
Species:          0
```

### See Also

`addcompartment`, `addreaction`, `addspecies`, `Compartment` object

# CompileOptions property

---

**Purpose** Dimensional analysis and unit conversion options

**Description** The SimBiology `CompileOptions` property is an object that defines the compile options available for simulation; you can specify whether dimensional analysis and unit conversion is necessary for simulation. Compile options are checked during compile time. The compile options object can be accessed through the `CompileOptions` property of the `configset` object. Retrieve `CompileOptions` object properties with the `get` function and configure the properties with the `set` function.

|                         |                                      |   |
|-------------------------|--------------------------------------|---|
| <b>Property Summary</b> | <code>DefaultSpeciesDimension</code> | Dimension of species name in expression |
|                         | <code>DimensionalAnalysis</code>     | Perform dimensional analysis on model   |
|                         | <code>Type</code>                    | Display SimBiology object type          |
|                         | <code>UnitConversion</code>          | Perform unit conversion                 |

## Characteristics

|             |                                |
|-------------|--------------------------------|
| Applies to  | Object: <code>configset</code> |
| Data type   | Object                         |
| Data values | Compile-time options           |
| Access      | Read-only                      |

## Examples

**1** Retrieve the `configset` object of `modelObj`.

```
modelObj = sbiomodel('cell');  
configsetObj = getconfigset(modelObj);
```

**2** Retrieve the `CompileOptions` object (`optionsObj`) from the `configsetObj`.

```
optionsObj = get(configsetObj, 'CompileOptions');
```

Compile Settings:

```
UnitConversion:    false
DimensionalAnalysis: true
```

**See Also**      [get, set](#)

# Composition property

---

**Purpose** Unit composition

**Description** The Composition property holds the composition of a unit object. The Composition property shows the combination of base and derived units that defines the unit. For example, molarity is the name of the unit and the composition is mole/liter. Base units are the set of units used to define all unit quantity equations. Derived units are defined using base units or mixtures of base and derived units.

Valid physical quantities for reaction rates are amount/time, mass/time, or concentration/time.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: Unit   |
| Data type   | char string  |
| Data values | Valid combination of units and prefixes from the library. Default is '' (empty). |
| Access      | Read/write   |

## Examples

This example shows you how to create a user-defined unit, add it to the user-defined library, and query the Composition property.

**1** Create a unit for the rate constants of a second-order reaction.

```
unitObj = sbiounit('secondconstant', '1/molarity*second', 1);
```

**2** Query the Composition property.

```
get(unitObj, 'Composition')
```

```
ans =
```

```
1/molarity*second
```

**3** Change the Composition property.

```
set(unitObj, 'Composition', 'liter/mole*second'))  
  
ans =  
  
liter/mole*second
```

**4** Add the unit to the user-defined library.

```
sbioaddtolibrary(unitObj);
```

### See Also

get, Multiplier, Offset, sbiounit, set

# ConstantAmount property

---

**Purpose** Specify variable or constant species amount

**Description** The ConstantAmount property indicates whether the quantity of the species object can vary during the simulation. ConstantAmount can be either true or false. If ConstantAmount is true, the quantity of the species cannot vary during the simulation. By default, ConstantAmount is false and the quantity of the species can vary during the simulation. If ConstantAmount is false, the quantity of the species can be determined by reactions and rules.

The property ConstantAmount is for species objects; the property ConstantValue is for parameter objects.

---

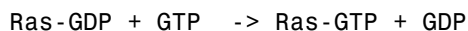
**Note** When you want the species to participate in a reaction, but do not want any reactions to modify its quantity, set its BoundaryCondition to true, and ConstantAmount to false.

---

## More Information

The following is an example of modeling species as constant amounts:

Modeling the role of nucleotides (GTP, ATP, cAMP) and cofactors (Ca<sup>++</sup>, NAD<sup>+</sup>, coenzyme A). Consider the role of GTP in the activation of Ras by receptor tyrosine kinases.



Model GTP and GDP with constant amount set to true. In addition, you can set the BoundaryCondition of these species to true, thus making them *boundary species*.

## Characteristics

|            |                 |
|------------|-----------------|
| Applies to | Object: species |
| Data type  | boolean         |



|             |  |
|-------------|--|
| Data values | true or false. The default value is false. |
| Access      | Read/write                                 |

## Examples

- 1 Create a model object named `my_model`.

```
modelObj = sbiomodel ('my_model');
```

- 2 Add a species object and verify that the `ConstantAmount` property setting is 'false' or 0.

```
speciesObj = addspecies (modelObj, 'glucose');  
get (speciesObj, 'ConstantAmount')
```

MATLAB returns:

```
ans =  
  
0
```

- 3 Set the constant amount to 'true' and verify.

```
set (speciesObj, 'ConstantAmount', true);  
get (speciesObj, 'ConstantAmount')
```

MATLAB returns:

```
ans =  
  
1
```

## See Also

`addspecies`, `BoundaryCondition`

# ConstantCapacity property

---

**Purpose** Specify variable or constant compartment capacity

**Description** The ConstantCapacity property indicates whether the capacity of the compartment object can vary during the simulation. ConstantCapacity can be either true (1) or false (0). If ConstantCapacity is true, the quantity of the compartment cannot vary during the simulation. By default, ConstantCapacity is true and the quantity of the compartment cannot vary during the simulation. If ConstantCapacity is false, the quantity of the compartment can be determined by rules and events.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Object: compartment                       |
| Data type   | boolean                                   |
| Data values | true or false. The default value is true. |
| Access      | Read/write                                |

## Examples

Add a compartment to a model and check the ConstantCapacity property of the compartment.

**1** Create a model object named comp\_model.

```
modelObj = sbiomodel ('comp_model');
```

**2** Add the compartment object named nucleus with a capacity of 0.5.

```
compartmentObj = addcompartment(modelObj, 'nucleus', 0.5);
```

**3** Display the ConstantCapacity property.

```
get(compartmentObj, 'ConstantCapacity')
```

```
ans =
```

```
1
```

## See Also

addcompartment, ConstantAmount, ConstantValue

**Purpose** Specify variable or constant parameter value

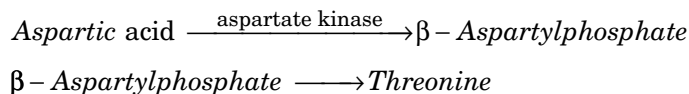
**Description** The ConstantValue property indicates whether the value of a parameter can change during a simulation. Enter either true (value is constant) or false (value can change).

You can allow the value of the parameter to change during a simulation by specifying a rule that changes the Value property of the parameter object.

The property ConstantValue is for parameter objects; the property ConstantAmount is for species objects.

### More Information

As an example, consider feedback inhibition of an enzyme such as aspartate kinase by threonine. Aspartate kinase has three isozymes that are independently inhibited by the products of downstream reactions (threonine, homoserine, and lysine). Although threonine is made through a series of reactions in the synthesis pathway, for illustration, the reactions are simplified as follows:



To model inhibition of aspartate kinase by threonine, you could use a rule like the algebraic rule below to vary the rate of the above reaction and simulate inhibition. In the rule, the rate constant for the above reaction is denoted by k\_aspartate\_kinase and the quantity of threonine is threonine.

$$\text{k\_aspartate\_kinase} - (1/\text{threonine})$$

### Characteristics

|            |                   |
|------------|-------------------|
| Applies to | Object: parameter |
| Data type  | boolean           |

# ConstantValue property

---

|             |   |
|-------------|---|
| Data values | true or false. The default value is 'true'. |
| Access      | Read/write                                  |

## Examples

**1** Create a model object.

```
modelObj = sbiomodel ('my_model');
```

**2** Add a parameter object.

```
parameterObj = addparameter (modelObj, 'kf');
```

**3** Change the ConstantValue property of the parameter object from default (true) to false and verify.

MATLAB returns 1 for true and 0 for false.

```
set (parameterObj, 'ConstantValue', false);  
get(parameterObj, 'ConstantValue')
```

MATLAB returns:

```
ans =
```

```
0
```

## See Also

addparameter

**Purpose** Contents of variant object

**Description** The Content property contains the data for the variant object. Content is a cell array with the structure {'Type', 'Name', 'PropertyName', 'PropertyValue'}. You can store values for species InitialAmount, parameter Value, and compartment Capacity, in a variant object.

For more information about variants, see Variant object.

## Characteristics

|             |                              |
|-------------|------------------------------|
| Applies to  | Object: variant              |
| Data type   | cell array                   |
| Data values | Default value is [] (empty). |
| Access      | Read/write                   |

## Examples

**1** Create a model containing three species in one compartment.

```
modelObj = sbiomodel('mymodel');  
compObj = addcompartment(modelObj, 'comp1');  
A = addspecies(compObj, 'A');  
B = addspecies(compObj, 'B');  
C = addspecies(compObj, 'C');
```

**2** Add a variant object that varies the species' InitialAmount property.

```
variantObj = addvariant(modelObj, 'v1');  
addcontent(variantObj, {'species','A', 'InitialAmount', 5}, ...  
{'species', 'B', 'InitialAmount', 10});  
% Display the variant  
variantObj
```

```
SimBiology Variant - v1 (inactive)
```

| ContentIndex: | Type:   | Name: | Property:     | Value: |
|---------------|---------|-------|---------------|--------|
| 1             | species | A     | InitialAmount | 5      |
| 2             | species | B     | InitialAmount | 10     |

# Content property

---

### 3 Append data to the Content property.

```
addcontent(variantObj, {'species', 'C', 'InitialAmount', 15});
```

```
SimBiology Variant - v1 (inactive)
```

| ContentIndex: | Type:   | Name: | Property:     | Value: |
|---------------|---------|-------|---------------|--------|
| 1             | species | A     | InitialAmount | 5      |
| 2             | species | B     | InitialAmount | 10     |
| 3             | species | C     | InitialAmount | 15     |

### 4 Remove a species from the Content property.

```
rmcontent(variantObj, 3);
```

### 5 Replace the data in the Content property.

```
set(variantObj, 'Content', {'species', 'C', 'InitialAmount', 15});
```

## See Also

addcontent, rmcontent, sbiovariant

**Purpose** Identify covariate columns in data set

**Description** CovariateLabels is a property of the PKData object. It specifies the column in DataSet that contains the covariate data.

## Characteristics

|             |                                       |
|-------------|---------------------------------------|
| Applies to  | Object: PKData                        |
| Data type   | char string or cell array of strings  |
| Data values | Column headers from imported data set |
| Access      | Read/write                            |

**See Also** PKData object

**How To**

- “Specify and Classify the Data to Fit”
- “Specify a Covariate Model”

# CovariateLabels (CovariateModel) property

---

**Purpose** Labels for covariates in CovariateModel object

**Description** The CovariateLabels property is a cell array of strings specifying the labels for the covariates in the Expression property of a CovariateModel object.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: CovariateModel                               |
| Data type   | Cell array of strings                                |
| Data values | Labels for the covariates in the Expression property |
| Access      | Read only  |

**See Also** CovariateModel | Expression



**Purpose** Store simulation data

**Description** The Data property contains the simulation data stored in the SimData object.

This property contains all data logged during a simulation, including species amounts, parameter values, and sensitivities. The property is an  $m \times n$  array, where  $m$  is the number of time steps in the simulation and  $n$  is the number of quantities logged. The rows of the array are labeled by the time points in the Time property, and the columns are labeled by the metadata in the DataInfo property.

## Characteristics

|             |                              |
|-------------|------------------------------|
| Applies to  | Object: SimData              |
| Data type   | double                       |
| Data values | Default value is [] (empty). |
| Access      | Read-only                    |

**See Also** DataInfo, ModelName

# DataCount property

---

**Purpose** Numbers of species, parameters, sensitivities

**Description** The DataCount property shows how many species, parameters, and sensitivities are logged in a SimData object. It is a MATLAB structure with the fields Species, Parameter, and Sensitivity. The information in this property is redundant with the DataInfo property and is there to give you a convenient means to access the information.

## Characteristics

|             |                                    |
|-------------|------------------------------------|
| Applies to  | Object: SimData                    |
| Data type   | struct                             |
| Data values | Default value for each field is 0. |
| Access      | Read-only                          |

**See Also** StopTime, StopTimeType

**Purpose** Metadata labels for simulation data

**Description** The DataInfo property contains the metadata that label the columns of the SimData object array. It is an  $n \times 1$  cell array of structures. The  $i$ th cell contains metadata labeling the  $i$ th column of the SimData object array.

The possible types of structures are as follows.

| Type        | Fields  |
|-------------|---|
| Species     | Type: species<br>Name:<br>Compartment:<br>Units:  |
| Parameter   | Type: parameter<br>Name:<br>Reaction: <name of reaction that a parameter is scoped to,<br>or '' if parameter is scoped to model><br>Units:  |
| Sensitivity | Type: sensitivity<br>Name: <for example: d[x]/d[y]_0><br>OutputType: <The type of the sensitivity output,<br>either 'species' or 'parameter'><br>OutputName: <The name of the sensitivity output><br>OutputQualifier: <The compartment or reaction for<br>the sensitivity output, for<br>species or parameters, respectively><br>InputType: <The type of the sensitivity input,<br>either 'species' or 'parameter'><br>InputName: <The name of the sensitivity input><br>InputQualifier: <The compartment or reaction for |

# DataInfo property

---

| Type | Fields  |
|------|---|
|      | the sensitivity input, for species or parameters, respectively><br>Units: |

## Characteristics

|             |                                  |
|-------------|----------------------------------|
| Applies to  | Object: SimData                  |
| Data type   | n x 1 cell array of structs      |
| Data values | Default value is 0x1 cell array. |
| Access      | Read-only                        |

**See Also**      StopTime, StopTimeType

**Purpose** Show names in SimData object

**Description** The DataNames property holds the names that label the columns of the data matrix in the Data property. The property contains an nx1 array of strings. The software provides this information for your convenience.

## Characteristics

|             |                                  |
|-------------|----------------------------------|
| Applies to  | Object: SimData                  |
| Data type   | string array                     |
| Data values | Default value is 0x1 cell array. |
| Access      | Read-only                        |

**See Also** StopTime, StopTimeType

# DataSet property

---

**Purpose**                      DataSet object containing imported data

**Description**                DataSet is a property of the PKData object. It contains the imported data set. The PKData object constructor (PKData) assigns the specified data set to its DataSet property during construction.

## Characteristics

|             |                                    |
|-------------|------------------------------------|
| Applies to  | Object: PKData                     |
| Data type   | dataset object                     |
| Data values | Variable containing dataset object |
| Access      | Read-only                          |

**See Also**                      “Specify and Classify the Data to Fit” in the SimBiology User’s Guide, PKData object

# DefaultSpeciesDimension property

---

**Purpose** Dimension of species name in expression

**Description** The DefaultSpeciesDimension property specifies how SimBiology interprets species names in expressions (namely reaction rate, rule, or event expressions). The valid property values are `substance` or `concentration`. If you specify `InitialAmountUnits`, SimBiology interprets species names appearing in expressions as concentration or substance amount according to the units specified, regardless of the value in `DefaultSpeciesDimension`. Thus, if `DefaultSpeciesDimension` is `concentration` and you specify species units as `molecule`, SimBiology interprets species names in expressions as `substance`. This interpretation applies even when `DimensionalAnalysis` is off.

You can find `DefaultSpeciesDimension` in the `CompileOptions` property.

When you set `DefaultSpeciesDimension` to `substance`, if you do not specify units, SimBiology interprets species names appearing in expressions as substance amounts, and does not scale by compartment capacity. To include a species concentration in an expression, divide by the appropriate compartment capacity in the expression. To specify compartment capacity in an expression enter the compartment name.

When you set `DefaultSpeciesDimension` to `concentration`, SimBiology interprets species names appearing in expressions as concentrations, and scales by compartment capacity in the expressions. To include a species amount in an expression, multiply by the species name by the appropriate compartment name in the expression.

For information on dimensional analysis for reaction rates, see “How Reaction Rates Are Evaluated”.

## Characteristics

|            |  |
|------------|--|
| Applies to | Object: <code>CompileOptions</code> (in <code>configset</code> object) |
| Data type  | <code>char</code> string   |

# DefaultSpeciesDimension property

---

|             |   |
|-------------|---|
| Data values | concentration or substance. Default value is concentration. |
| Access      | Read/write  |

## See Also

CompileOptions, DimensionalAnalysis, get, getconfigset, sbiosimulate, set



# DependentVarLabel property

---

**Purpose** Identify dependent variable column in data set

**Description** `DependentVarLabel` is a property of a `PKData` object. It specifies the column(s) in `DataSet` that contain the dependent variable(s), for example, measured response(s). The column must contain numeric values, and cannot contain `Inf` or `-Inf`.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Object: <code>PKData</code>                       |
| Data type   | <code>char</code> string or cell array of strings |
| Data values | Column header from an imported data set           |
| Access      | Read/write  |

**See Also** “Specify and Classify the Data to Fit” in the SimBiology User’s Guide, `PKData` object

# DependentVarUnits property

---

**Purpose** Response units in PKData object

**Description** `DependentVarUnits` is a property of a `PKData` object. It specifies the units for the column(s) containing the dependent variable(s) (responses) in the imported data set. If unit conversion is on, plot results in the SimBiology desktop show the units specified in `DependentVarUnits`.

To get a list of units, use the `sbioshowunits` function.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Object: <code>PKData</code>                                   |
| Data type   | <code>char</code> string or cell array of strings             |
| Data values | Units from the units library. Default is an empty cell array. |

---

**Tip** If there are no units associated with the dependent variable(s) in your data set, you can set this property to a cell array of empty strings, or simply an empty cell array.

---

|        |            |
|--------|------------|
| Access | Read/write |
|--------|------------|

**See Also** `DependentVarLabel`, `PKData` Object

**Purpose** Perform dimensional analysis on model

**Description** The `DimensionalAnalysis` property specifies whether to perform dimensional analysis on the model before simulation. It is a property of the `CompileOptions` object. `CompileOptions` holds the model's compile time options and is the object property of the `configset` object. When `DimensionalAnalysis` is set to `true`, the SimBiology software checks whether the physical quantities of the units involved in reactions and rules, match and are applicable.

For example, consider a reaction  $a + b \rightarrow c$ . Using mass action kinetics, the reaction rate is defined as  $a \cdot b \cdot k$ , where  $k$  is the rate constant of the reaction. If you specify that initial amounts of  $a$  and  $b$  are 0.01M and 0.005M respectively, then units of  $k$  are  $1 / (M \cdot \text{second})$ . If you specify  $k$  with another equivalent unit definition, for example,  $1 / [(\text{moles/liter}) \cdot \text{second}]$ , `DimensionalAnalysis` checks whether the physical quantities match. If the physical quantities do not match, you see an error and the model is not simulated.

Unit conversion requires dimensional analysis. If `DimensionalAnalysis` is off, and you turn `UnitConversion` on, then `DimensionalAnalysis` is turned on automatically. If `UnitConversion` is on and you turn off `DimensionalAnalysis`, then `UnitConversion` is turned off automatically.

If you have MATLAB function calls in your model, dimensional analysis ignores any expressions containing function calls and generates a warning.

Valid physical quantities for reaction rates are amount/time, mass/time, or concentration/time.

## Characteristics

|            |  |
|------------|--|
| Applies to | Object: <code>CompileOptions</code> (in <code>configset</code> object) |
| Data type  | <code>boolean</code>   |

# DimensionalAnalysis property

---

|             |                                       |
|-------------|---------------------------------------|
| Data values | true or false. Default value is true. |
| Access      | Read/write                            |

---

**Note** SimBiology allows exponentiation of any dimensionless quantity to any dimensionless power. For example, you can write the following expression if both  $x$  and  $a$  are dimensionless:  $(x + 3)^{(a + 0.5)}$

---

---

**Note** SimBiology uses unit strings including empty units in association with `DimensionalAnalysis` and `UnitConversion` features.

- When `DimensionalAnalysis` and `UnitConversion` are both false, unit strings are not used. However, SimBiology still performs a minimum level of dimensional analysis to decide whether a reaction rate is in dimensions of amount/time or concentration/time.
  - When `DimensionalAnalysis` is true and `UnitConversion` is false, units (if not empty) must have consistent dimensions so that SimBiology can perform dimensional analysis. However, the units are not converted.
  - When `UnitConversion` is set to true (which requires `DimensionalAnalysis` to be true), SimBiology performs a dimensional analysis and converts everything to consistent units. Hence, you must specify consistent units, and no unit strings can be empty. If you have a dimensionless parameter, you must still set its unit string to dimensionless.
- 

## Examples

This example shows how to retrieve and set `DimensionalAnalysis` from the default true to false in the default configuration set in a model object.

- 1 Import a model.

```
modelObj = sbmlimport('oscillator')
```

```
SimBiology Model - Oscillator
```

```
Model Components:
```

```
Models:          0
Parameters:      0
Reactions:       42
Rules:           0
Species:         23
```

**2** Retrieve the configset object of the model object.

```
configsetObj = getconfigset(modelObj)
```

```
Configuration Settings - default (active)
```

```
SolverType:      ode15s
StopTime:        10.000000
```

```
SolverOptions:
```

```
AbsoluteTolerance: 1.000000e-006
RelativeTolerance:  1.000000e-003
```

```
RuntimeOptions:
```

```
StatesToLog:     all
```

```
CompileOptions:
```

```
UnitConversion:  true
DimensionalAnalysis: true
```

**3** Retrieve the CompileOptions object.

```
optionsObj = get(configsetObj, 'CompileOptions')
```

```
Compile Settings:
```

```
UnitConversion:  true
```

## DimensionalAnalysis property

---

```
DimensionalAnalysis: true
```

**4** Assign a value of false to DimensionalAnalysis.

```
set(optionsObj, 'DimensionalAnalysis', false)
```

### **See Also**

get, getconfigset, sbiosimulate, set

**Purpose** Dosed object name

**Description** Dosed is a property of the PKModelMap object. It specifies the name(s) of species object(s) that receive an input, such as a drug in a compartment or a ligand.

When dosing multiple compartments, a one-to-one relationship must exist between the number and order of elements in the Dosed property and the DosingType property.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: PKModelMap   |
| Data type   | char string or cell array of strings                               |
| Data values | Name of a species object or empty. Default is an empty cell array. |
| Access      | Read/write   |

**See Also** “Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters” in the SimBiology User’s Guide, DosingType, Estimated, Observed, PKModelMap object

# DoseLabel property

---

**Purpose** Dose column in data set

**Description** DoseLabel is a property of the PKData object. DoseLabel specifies the column that contains that contains the dosing information, in DataSet. The column must contain positive values, and cannot contain Inf or Inf.

## Characteristics

|             |                                       |
|-------------|---------------------------------------|
| Applies to  | Object: PKData                        |
| Data type   | string or array of strings            |
| Data values | Column headers from imported data set |
| Access      | Read/write                            |

**See Also** PKData object, sbionmimport, sbionmfiledef, “Specify and Classify the Data to Fit” in the SimBiology documentation



**Purpose** Dose units in PKData object

**Description** The DoseUnits property indicates the units for dose values in the PKData object. Dose units must have dimensions of amount or mass. The length of DoseUnits must be the same as DoseLabel. For example, if the DoseLabel property defines two columns containing dosing information, DoseUnits must also define units for both columns. If unit conversion is on, dose and rate units must be consistent with each other (that is in terms of amount or mass) and must be consistent with the species object that is being dosed.

To get a list of units, use the sbioshowunits function.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: PKData                                   |
| Data type   | string or array of strings                       |
| Data values | Units from units library. Default is '' (empty). |
| Access      | Read/write                                       |

**See Also** DoseLabel, PKData Object

# DosingType property

---

**Purpose** Drug dosing type in compartment

**Description** DosingType is a property of the PKCompartment and PKModelMap objects. It specifies the type of dosing of a drug in a compartment. You can only dose one compartment in the model at any given time. For a description of the types of dosing supported, the model components created for each type of dosing, and the parameters to estimate, see “Dosing Types”.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Objects: PKCompartment, PKModelMap                  |
| Data type   | char string or cell array of strings                |
| Data values | ' ', 'Bolus', 'Infusion', 'ZeroOrder', 'FirstOrder' |
| Access      | Read/write  |

**See Also** EliminationType, PKCompartment object, PKModelMap object

# DurationParameterName property

---

**Purpose** Parameter specifying length of time to administer a dose

**Description** DurationParameterName is a property of a RepeatDose or ScheduleDose object.

Specify the name of a parameter object that is:

- Scoped to a model
- Constant, that is, its ConstantValue property is true

This property specifies the length of time it takes to administer a dose.

---

**Note** If you set the DurationParameterName property of a dose, you must also specify the Amount property of the dose, and set the Rate property to 0. This is because the rate is calculated from the amount and duration.

---

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Objects: RepeatDose, ScheduleDose  |
| Data type   | char string  |
| Data values | Name of a parameter object or empty. Default is an empty string.<br>The parameter object must be: <ul style="list-style-type: none"><li>• Scoped to a model</li><li>• Constant, that is, have a ConstantValue property set to true</li></ul> |
| Access      | Read/write   |

**See Also** RepeatDose object, ScheduleDose object

# EliminationType property

---

**Purpose** Drug elimination type from compartment

**Description** EliminationType is a property of the PKCompartment object. It specifies the type of elimination of a drug from a compartment. For a description of the types of elimination supported, the model components created for each type of elimination, and the parameters to estimate, see “Elimination Types”.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Object: PKCompartment                             |
| Data type   | char string                                       |
| Data values | 'Linear', 'Linear-Clearance', 'Enzymatic', and '' |
| Access      | Read/write  |

**See Also** addCompartment, DosingType, PKCompartment object

**Purpose** Specify explicit or implicit tau error tolerance

**Description** The ErrorTolerance property specifies the error tolerance for the explicit tau and implicit tau stochastic solvers. It is a property of the SolverOptions object. SolverOptions is a property of the configset object. The explicit and implicit tau solvers automatically chooses a time interval ( $\tau$ ) such that the relative change in the propensity function for each reaction is less than the user-specified error tolerance. A propensity function describes the probability that the reaction will occur in the next smallest time interval, given the conditions and constraints.

If the error tolerance is too large, there may not be a solution to the problem and that could lead to an error. If the error tolerance is small, the solver will take more steps than when the error tolerance is large leading to longer simulation times. The error tolerance should be adjusted depending upon the problem, but a good value for the error tolerance is between 1 % to 5 %.

## Characteristics

|             |                              |
|-------------|------------------------------|
| Applies to  | Object: SolverOptions        |
| Data type   | double                       |
| Data values | >0, <1. The default is 3e-2. |
| Access      | Read/write                   |

**Examples** This example shows how to change ErrorTolerance settings.

- 1 Retrieve the configset object from the modelObj and change the SolverType to expltau.

```
modelObj = sbiomodel('cell');  
configsetObj = getConfigset(modelObj);  
set(configsetObj, 'SolverType', 'expltau')
```

- 2 Change the ErrorTolerance to 1e-8.

# ErrorTolerance property

---

```
set(configsetObj.SolverOptions, 'ErrorTolerance', 5.0e-2);  
get(configsetObj.SolverOptions, 'ErrorTolerance')
```

```
ans =
```

```
5.000000e-002
```

## See Also

LogDecimation, RandomState

**Purpose** Names of parameters to estimate

**Description** Estimated is a property of the PKModelMap object. It specifies the name(s) of the object(s) to estimate. Specify the name(s) of species, compartment, or parameter object(s) that are scoped to a model.

---

**Note** If you specify a species object, you are estimating the InitialAmount property of the species object.

---

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Object: PKModelMap  |
| Data type   | char string or cell array of strings  |
| Data values | Name of a species, compartment, or parameter object or empty. Default is an empty cell array. |
| Access      | Read/write  |

**See Also** “Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters” in the SimBiology User’s Guide, Dosed, InitialAmount, Observed, PKModelMap object

# EventFcns property

---

**Purpose** Event expression

**Description** Property of the event object that defines what occurs when the event is triggered. Specify a cell array of strings.

EventFcns can be any MATLAB assignment or expression that defines what is executed when the event is triggered. All EventFcn expressions are assignments of the form '*objectname* = *expression*', where *objectname* is the name of a valid SimBiology object.

For more information about how SimBiology handles events, see “How Events Are Evaluated”. For examples of event functions, see “Specifying Event Functions”.

## Characteristics

|             |                             |
|-------------|-----------------------------|
| Applies to  | Object: event               |
| Data type   | Cell array of strings       |
| Data values | EventFcn strings '' (empty) |
| Access      | Read/write                  |

## Examples

**1** Create a model object, and then add an event object.

```
modelObj = sbmlimport('oscillator');  
eventObj = addevent(modelObj, 'time>= 5', 'OpC = 200');
```

**2** Set the EventFcns property of the event object.

```
set(eventObj, 'EventFcns', {'pA = 0pA', 'mA = pol'});
```

**3** Get the EventFcns property.

```
get(eventObj, 'EventFcns')
```

## See Also

Event object, Trigger



**Purpose** Contain all event objects

**Description** Property to indicate events in a model object. Read-only array of Event objects.

An event defines an action when a defined condition is met. For example, the quantity of a species may double when the quantity of species B is 100. An event is triggered when the conditions specified in the event are met by the model. For more information, see “Events” and “Event Object”.

Add an event to a Model object with the method `addevent` method and remove an event with the `delete` method. See `Event` object for more information.

You can view event object properties with the `get` command and modify the properties with the `set` command.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: model                            |
| Data type   | Array of event objects                   |
| Data values | Event object. The default is [] (empty). |
| Access      | Read-only                                |

## Examples

**1** Create a model object, and then add an event object.

```
modelObj = sbmlimport('oscillator')
eventObj = addevent(modelObj, 'time>= 5', 'OpC = 200');
```

**2** Get a list of properties for an event object.

```
get(modelObj.Events(1));
```

Or

```
get(eventObj)
```

## Events property

---

MATLAB displays a list of event properties.

```
Active: 1
Annotation: ''
EventFcns: {'OpC = 200'}
Name: ''
Notes: ''
Parent: [1x1 SimBiology.Model]
Tag: ''
Trigger: 'time >= 5'
TriggerDelay: 0
TriggerDelayUnits: 'second'
Type: 'event'
UserData: []
```

### See Also

EventFcns, Event object, Model object, Trigger

**Purpose** Exponent value of unit prefix

**Description** *Exponent* shows the value of  $10^{\text{Exponent}}$  that defines the numerical value of the unit prefix *Name*. You can use the unit prefix in conjunction with any built-in or user-defined units. For example, for the unit `mole`, specify as `picomole` to use the *Exponent*, `-12`.

## Characteristics

|             |                            |
|-------------|----------------------------|
| Applies to  | Object: Unit prefix        |
| Data type   | <code>double</code>        |
| Data values | Real number. Default is 0. |
| Access      | Read/write                 |

**Examples** This example shows you how to create a user-defined unit prefix, add it to the user-defined library, and query the *Exponent* property.

**1** Create a unit prefix.

```
unitprefixObj1 = sbiounitprefix('peta', 15);
```

**2** Add the unit prefix to the user-defined library.

```
sbioaddtolibrary(unitprefixObj1);
```

**3** Query the *Exponent* property.

```
get(unitprefixObj1, 'Exponent')
```

```
ans =
```

```
15
```

**See Also** `get`, `sbioaddtolibrary`, `sbiounitprefix`, `set`, `UnitPrefix` object

# Expression (CovariateModel) property

## Purpose

Define relationship between parameters and covariates

## Description

The Expression property is a string or cell array of strings, where each string represents the relationship between a parameter and one or more covariates. The Expression property denotes fixed effects with the prefix `theta`, and random effects with the prefix `eta`.

Each expression string must be in the form:

```
parameterName = relationship
```

This example of an expression string defines the relationship between a parameter (`volume`) and a covariate (`weight`), with fixed effects, but no random effects:

```
CovModelObj.Expression = {'volume = theta1 +  
theta2*weight'};
```

This table illustrates expression formats for some common parameter-covariate relationships.

| Parameter-Covariate Relationship  | Expression Format   |
|---|---|
| Linear with random effect   | $C1 = \text{theta1} + \text{theta2} * \text{WEIGHT} + \text{eta1}$                                      |
| Exponential without random effect                                       | $C1 = \exp(\text{theta\_C1} + \text{theta\_C1\_WT} * \text{WEIGHT})$                                    |
| Exponential, WEIGHT centered by mean, has random effect                 | $C1 = \exp(\text{theta1} + \text{theta2} * (\text{WEIGHT} - \text{mean}(\text{WEIGHT}))) + \text{eta1}$ |
| Exponential, $\log(\text{WEIGHT})$ , which is equivalent to power model | $C1 = \exp(\text{theta1} + \text{theta2} * \log(\text{WEIGHT})) + \text{eta1}$                          |

## Expression (CovariateModel) property

| Parameter-Covariate Relationship                                  | Expression Format   |
|---|---|
| Exponential, dependent on WEIGHT and AGE, has random effect       | $C1 = \exp(\text{theta1} + \text{theta2} * \text{WEIGHT} + \text{theta3} * \text{AGE} + \text{eta1})$             |
| Inverse of probit, dependent on WEIGHT and AGE, has random effect | $C1 = \text{probitinv}(\text{theta1} + \text{theta2} * \text{WEIGHT} + \text{theta3} * \text{AGE} + \text{eta1})$ |
| Inverse of logit, dependent on WEIGHT and AGE, has random effect  | $C1 = \text{logitinv}(\text{theta1} + \text{theta2} * \text{WEIGHT} + \text{theta3} * \text{AGE} + \text{eta1})$  |

**Tip** To simultaneously fit data from multiple dose levels, use a CovariateModel object as an input argument to sbionlmeFit, and omit the random effect (eta) from the Expression property in the CovariateModel object.

The Expression property must meet the following requirements:

- The expression strings are valid MATLAB code.
- Each expression string is linear with a transformation.
- There is exactly one expression string for each parameter.
- In each expression string, a covariate is used in at most one term.
- In each expression string, there is at most one random effect (eta)
- Fixed effect (theta) and random effect (eta) names are unique within and across expression strings. That is, each covariate has its own fixed effect.

# Expression (CovariateModel) property

---

---

**Tip** Use the `getCovariateData` method to view the covariate data when writing equations for the Expression property of a `CovariateModel` object.

---

---

**Tip** Use the `verify` method to check that the Expression property of a `CovariateModel` object meets the conditions described previously.

---

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Object: <code>CovariateModel</code>       |
| Data type   | String or cell array of strings           |
| Data values | <code>parameterName = relationship</code> |
| Access      | Read/write                                |

## See Also

`CovariateModel` | `getCovariateData` | `verify`

## How To

- Modeling the Population Pharmacokinetics of Phenobarbital in Neonates
- “Specify a Covariate Model”

# Expression (AbstractKineticLaw, KineticLaw) property

---

**Purpose** Expression to determine reaction rate equation

**Description** The Expression property indicates the mathematical expression that is used to determine the ReactionRate property of the reaction object. Expression is a reaction rate expression assigned by the kinetic law definition used by the reaction. The kinetic law being used is indicated by the property KineticLawName. You can configure Expression for user-defined kinetic laws, but not for built-in kinetic laws. Expression is read only for kinetic law objects.

---

**Note** If you set the Expression property to a reaction rate expression that is not continuous and differentiable, see “Using Events to Address Discontinuities in Rule and Reaction Rate Expressions” before simulating your model.

---

## Kinetic Law Definition

The *kinetic law definition* provides a mechanism for applying a specific rate law to multiple reactions. It acts as a mapping template for the reaction rate. The kinetic law is defined by a mathematical expression, (defined in the property Expression), and includes the species and parameter variables used in the expression. The species variables are defined in the SpeciesVariables property, and the parameter variables are defined in the ParameterVariables property of the kinetic law object.

If a reaction is using a kinetic law definition, the ReactionRate property of the reaction object shows the result of a mapping from the kinetic law definition. To determine ReactionRate, the species variables and parameter variables that participate in the reaction rate should be mapped in the kinetic law for the reaction. In this case, SimBiology software determines the ReactionRate by using the Expression property of the abstract kinetic law object, and by mapping SpeciesVariableNames to SpeciesVariables and ParameterVariableNames to ParameterVariables.

# Expression (AbstractKineticLaw, KineticLaw) property

---

For example, the kinetic law definition Henri-Michaelis-Menten has the Expression  $V_m \cdot S / (K_m + S)$ , where  $V_m$  and  $K_m$  are defined as parameters in the ParameterVariables property of the abstract kinetic law object, and  $S$  is defined as a species in the SpeciesVariable property of the abstract kinetic law object.

By applying the Henri-Michaelis-Menten kinetic law to a reaction  $A \rightarrow B$  with  $V_a$  mapping to  $V_m$ ,  $A$  mapping to  $S$ , and  $K_a$  mapping to  $K_m$ , the rate equation for the reaction becomes  $V_a \cdot A / (K_a + A)$ .

The exact expression of a reaction using MassAction kinetic law varies depending upon the number of reactants. Thus, for mass action kinetics the Expression property is set to MassAction because in general for mass action kinetics the reaction rate is defined as

$$r = k \prod_{i=1}^{n_r} [S_i]^{m_i}$$

where  $[S_i]$  is the concentration of the  $i$ th reactant,  $m_i$  is the stoichiometric coefficient of  $[S_i]$ ,  $n_r$  is the number of reactants, and  $k$  is the mass action reaction rate constant.

SimBiology software contains some built-in kinetic laws. You can also define your own kinetic laws. To find the list of available kinetic laws, use the `sbiowhos -kineticlaw` command (`sbiowhos`). You can create a kinetic law definition with the function `sbioabstractkineticlaw` and add it to the library using `sbioaddtolibrary`.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Objects: abstract kinetic law, kinetic law                                  |
| Data type   | char string   |
| Data values | Defined by kinetic law definition   |
| Access      | Read-only in kinetic law object.<br>Read/write in user-defined kinetic law. |



# Expression (**AbstractKineticLaw**, **KineticLaw**) property

## Examples

### Example 1

Example with Henri-Michaelis-Menten kinetics

- 1 Create a model object, and add a reaction object to the model.

```
modelObj = sbiomodel ('my_model');  
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

- 2 Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

- 3 Verify that the Expression property for the kinetic law object is Henri-Michaelis-Menten.

```
get (kineticlawObj, 'Expression')
```

MATLAB returns:

```
ans =
```

```
Vm*S/(Km + S)
```

- 4 The 'Henri-Michaelis-Menten' kinetic law has two parameter variables (Vm and Km) and one species variable (S) that you should set. To set these variables, first create the parameter variables as parameter objects (parameterObj1, parameterObj2) with names Vm\_d, Km\_d, and assign the objects' Parent property value to the kineticlawObj. The species object with Name a is created when reactionObj is created and need not be redefined.

```
parameterObj1 = addparameter(kineticlawObj, 'Vm_d');  
parameterObj2 = addparameter(kineticlawObj, 'Km_d');
```

- 5 Set the variable names for the kinetic law object.

```
set(kineticlawObj, 'ParameterVariableNames', {'Vm_d' 'Km_d'});  
set(kineticlawObj, 'SpeciesVariableNames', {'a'});
```

## Expression (AbstractKineticLaw, KineticLaw) property

---

- 6 Verify that the reaction rate is expressed correctly in the reaction object ReactionRate property.

```
get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =
```

```
Vm_d*a/(Km_d+a)
```

### Example 2

Example with Mass Action kinetics.

- 1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');  
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

- 2 Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');  
get(kineticlawObj, 'Expression')
```

MATLAB returns:

```
ans =
```

```
MassAction
```

- 3 Assign the rate constant for the reaction.

```
set (kineticlawObj, 'ParameterVariablenames', 'k');  
get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =
```

## Expression (**AbstractKineticLaw**, **KineticLaw**) property

---

$k*a*b$

### **See Also**

KineticLawName, Parameters, ParameterVariableNames,  
ParameterVariables, ReactionRate, sbioaddtolibrary, sbiowhos,  
SpeciesVariables, SpeciesVariableNames

# FixedEffectDescription (CovariateModel) property

---

**Purpose** Descriptions of fixed effects in CovariateModel object

**Description** The FixedEffectDescription property is a cell array of strings describing the fixed effects in the Expression property of a CovariateModel object. Each string describes the role of a fixed effect in the expression equation. For example, in the following expression equation:

$$C1 = \exp(\text{theta1} + \text{theta2} * \text{WEIGHT} + \text{theta3} * \text{AGE} + \text{eta1})$$

The description for the fixed effect theta1 is 'C1', which indicates it is the intercept for the parameter C1. Also, the description for the fixed effect theta2 is 'C1/WEIGHT', which indicates it is the slope of the line that defines the relationship between the parameter C1 and the covariate WEIGHT.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: CovariateModel   |
| Data type   | Cell array of strings  |
| Data values | Description of the roles of the fixed effects in the Expression property |
| Access      | Read-only  |

**See Also** CovariateModel | Expression | FixedEffectNames | FixedEffectValues

# FixedEffectNames (CovariateModel) property

---

**Purpose** Names of fixed effects in CovariateModel object

**Description** The FixedEffectNames property is a cell array of strings specifying the names of the fixed effects in the Expression property of a CovariateModel object. Names of fixed effects are denoted with the prefix theta.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Object: CovariateModel  |
| Data type   | Cell array of strings   |
| Data values | Names of the fixed effects in the Expression property. These name are denoted with the prefix theta |
| Access      | Read only   |

**See Also** CovariateModel | Expression | FixedEffectDescription | FixedEffectValues

# FixedEffectValues (CovariateModel) property

---

**Purpose** Values for initial estimates of fixed effects in CovariateModel object

**Description** The FixedEffectValues property is a structure containing one field for each fixed effect in the Expression property of a CovariateModel object. Each field contains the value of the initial estimate for a fixed effect.

---

**Tip** You must set this property before using the CovariateModel object as input to sbionlmeFit or sbionlmeFitsa. Use the constructDefaultFixedEffectValues method to create a structure of fixed-effect initial estimate values, set to a default of zero. Then edit the structure and use it to modify this property.

---

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Object: CovariateModel  |
| Data type   | Structure with one field for each fixed effect  |
| Data values | Each field contains a double specifying the value of the initial estimate for a fixed effect in the CovariateModel object |
| Access      | Read/write  |

**See Also** CovariateModel | constructDefaultFixedEffectValues | Expression | FixedEffectDescription | FixedEffectNames

**How To**

- Modeling the Population Pharmacokinetics of Phenobarbital in Neonates
- “Specify a Covariate Model”

**Purpose** Integer identifying each group in data set

**Description** GroupID is a property of the PKData object. It is an array of the same length as the DataSet property containing an integer to identify each group. PKData sets this property during construction of the PKData object.

## Characteristics

|             |                            |
|-------------|----------------------------|
| Applies to  | Object: PKData             |
| Data type   | double                     |
| Data values | Index value for each group |
| Access      | Read-only                  |

**See Also** “Specify and Classify the Data to Fit” in the SimBiology User’s Guide, PKData object

# GroupLabel property

---

**Purpose** Identify group column in data set

**Description** GroupLabel is a property of the PKData object. It specifies the column in DataSet that contains the group identification labels.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Object: PKData                              |
| Data type   | char string                                 |
| Data values | Column header string from imported data set |
| Access      | Read/write                                  |

**See Also** “Specify and Classify the Data to Fit” in the SimBiology User’s Guide, PKData object, GroupNames



**Purpose** Unique values from `GroupLabel` in data set

**Description** `GroupNames` is a property of the `PKData` object. It contains unique values from the data column specified by the `GroupLabel` property. `PKData` sets this property during construction of the `PKData` object.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Object: <code>PKData</code>                       |
| Data type   | <code>char</code> string or cell array of strings |
| Data values | Unique values in <code>GroupLabel</code>          |
| Access      | Read-only   |

**See Also** “Specify and Classify the Data to Fit” in the SimBiology User’s Guide, `PKData` object, `GroupLabel`

# HasLag property

---

**Purpose** Lag associated with dose targeting compartment

**Description** HasLag is a property of the PKCompartment object. It is a logical indicating if the dose targeting the compartment has a time lag or not.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: PKCompartment                        |
| Data type   | logical                                      |
| Data values | 1 (true) or 0 (false). Default is 0 (false). |
| Access      | Read/write                                   |

**See Also** addCompartment, DosingType, EliminationType, PKCompartment object

# HasResponseVariable property

---

**Purpose** Compartment drug concentration reported

**Description** HasResponseVariable is a property of the PKCompartment object. It is a logical indicating if the drug concentration in this compartment is reported.

---

**Note** The HasResponseVariable property can be true for more than one PKCompartment object in the model. If you perform a parameter fit on a model, at least one PKCompartment object in the model must have a HasResponseVariable property set to true.

---

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: PKCompartment                        |
| Data type   | Logical                                      |
| Data values | 1 (true) or 0 (false). Default is 0 (false). |
| Access      | Read/write                                   |

**See Also** addCompartment, DosingType, EliminationType, PKCompartment object

# IndependentVarLabel property

---

**Purpose** Identify independent variable column in data set

**Description** IndependentVarLabel is a property of the PKData object. It specifies the column in DataSet that contains the independent variable (for example, time).

The column must contain positive values, and cannot contain, NaN, Inf or -Inf.

## Characteristics

|             |                                      |
|-------------|--------------------------------------|
| Applies to  | Object: PKData                       |
| Data type   | char string                          |
| Data values | Column header from imported data set |

|        |            |
|--------|------------|
| Access | Read/write |
|--------|------------|

**See Also** “Specify and Classify the Data to Fit” in the SimBiology User’s Guide, PKData object

**Purpose** Time units in PKData object

**Description** The IndependentVarUnits property indicates the units for the column containing the independent variable (time) in the PKData object. If unit conversion is on, plot results in the SimBiology desktop show the units specified in IndependentVarUnits.

To get a list of units, use the sbioshowunits function.

## Characteristics

|             |                                    |
|-------------|------------------------------------|
| Applies to  | Object: PKData                     |
| Data type   | string                             |
| Data values | Time units. Default is '' (empty). |
| Access      | Read/write                         |

**See Also** DependentVarLabel, PKData Object

# InitialAmount property

---

**Purpose** Species initial amount

**Description** The `InitialAmount` property indicates the initial quantity of the SimBiology species object. `InitialAmount` is the quantity of the species before the simulation starts.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Object: species                           |
| Data type   | double                                    |
| Data values | Positive real number. Default value is 0. |
| Access      | Read/write                                |

## Examples

Add a species to a model and set the initial amount of the species.

**1** Create a model object named `my_model`.

```
modelObj = sbiomodel ('my_model');
```

**2** Add the species object named `glucose`.

```
speciesObj = addspecies (modelObj, 'glucose');
```

**3** Set the initial amount to 100 and verify.

```
set (speciesObj, 'InitialAmount',100);  
get (speciesObj, 'InitialAmount')
```

MATLAB returns:

```
ans =  
  
100
```

## See Also

`addspecies`, `InitialAmountUnits`

**Purpose** Species initial amount units

**Description** The InitialAmountUnits property indicates the unit definition for the InitialAmount property of a species object. InitialAmountUnits can be one of the built-in units. To get a list of the defined units, use the sbioshowunits function. If InitialAmountUnits changes from one unit definition to another, InitialAmount does not automatically convert to the new units. The sbioconvertunits function does this conversion. To add a user-defined unit to the list, use sbiounit followed by sbioaddtolibrary.

See DefaultSpeciesDimension for more information on specifying dimensions for species quantities. InitialAmountUnits must have corresponding dimensions to CapacityUnits. For example, if the CapacityUnits are meter<sup>2</sup>, then species must be amount/meter<sup>2</sup> or amount.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: species  |
| Data type   | char string  |
| Data values | Units from library with dimensions of amount, amount/length, amount/area, or amount/volume. Default is '' (empty). |
| Access      | Read/write   |

# InitialAmountUnits property

---

---

**Note** SimBiology uses unit strings including empty units in association with `DimensionalAnalysis` and `UnitConversion` features.

- When `DimensionalAnalysis` and `UnitConversion` are both false, unit strings are not used. However, SimBiology still performs a minimum level of dimensional analysis to decide whether a reaction rate is in dimensions of amount/time or concentration/time.
- When `DimensionalAnalysis` is true and `UnitConversion` is false, units (if not empty) must have consistent dimensions so that SimBiology can perform dimensional analysis. However, the units are not converted.
- When `UnitConversion` is set to true (which requires `DimensionalAnalysis` to be true), SimBiology performs a dimensional analysis and converts everything to consistent units. Hence, you must specify consistent units, and no unit strings can be empty. If you have a dimensionless parameter, you must still set its unit string to dimensionless.

## Examples

- 1 Create a model object named `my_model`.

```
modelObj = sbiomodel ('my_model');  
compObj = addcompartment(modelObj, 'cell');
```

- 2 Add a species object named `glucose`.

```
speciesObj = addspecies (compObj, 'glucose');
```

- 3 Set the initial amount to 100, `InitialAmountUnits` to `molecule`, and verify.

```
set (speciesObj, 'InitialAmountUnits', 'molecule');  
get (speciesObj, 'InitialAmountUnits')
```

MATLAB returns:



ans =

molecule

### See Also

DefaultSpeciesDimension, InitialAmount, sbioaddtolibrary, sbioconvertunits, sbioshowunits, sbiounit,

# Inputs property

---

**Purpose** Specify species and parameter input factors for sensitivity analysis

**Description** Inputs is a property of the SensitivityAnalysisOptions object. SensitivityAnalysisOptions is a property of the configuration set object.

Use Inputs to specify the species and parameters with respect to which you want to compute the sensitivities of the species or parameter states in your model.

The SimBiology software calculates sensitivities with respect to the values of the parameters and the initial amounts of the species specified in the Inputs property. When you simulate a model with SensitivityAnalysis enabled in the active configuration set object, sensitivity analysis returns the computed sensitivities of the species and parameters specified in the Outputs property. For a description of the output, see the SensitivityAnalysisOptions property description.

## Characteristics

|            |  |
|------------|--|
| Applies to | Object: SensitivityAnalysisOptions                 |
| Data type  | Species or parameter object or an array of objects |

---

**Note** If this object is determined by a repeated assignment rule, then you cannot use it as an Inputs property.

---

|             |   |
|-------------|---|
| Data values | Species or parameter object, or an array of objects. Default is [] (empty array). |
| Access      | Read/write  |

**Examples** This example shows how to set Inputs for sensitivity analysis.

- 1 Import the radio decay model from the SimBiology demos.

```
modelObj = sbmlimport('radiodecay');
```

- 2 Retrieve the configuration set object from modelObj.

```
configsetObj = getConfigset(modelObj);
```

- 3 Add a parameter to the Inputs property and display it. Use the sbioselect function to retrieve the parameter object from the model.

SimBiology Parameter Array

| Index: | Name: | Value: | ValueUnits: |
|--------|-------|--------|-------------|
| 1      | c     | 0.5    | 1/second    |

## See Also

Outputs, sbioselect, SensitivityAnalysis, SensitivityAnalysisOptions

# Interval property

---

**Purpose** Time between doses

**Description** Interval is a property of a RepeatDose object. This property defines the equally spaced times between repeated doses.

---

**Note** When the Interval property is 0, RepeatDose ignores the RepeatCount property, that is, it treats it as though it is set to 0.

---

|                        |             |                                       |
|------------------------|-------------|---------------------------------------|
| <b>Characteristics</b> | Applies to  | Object: RepeatDose                    |
|                        | Data type   | double                                |
|                        | Data values | Nonnegative real number. Default is 0 |
|                        | Access      | Read/Write                            |

**See Also** RepeatDose object, ScheduleDose object

**Purpose** Show kinetic law used for ReactionRate

**Description** The KineticLaw property shows the kinetic law that determines the reaction rate specified in the ReactionRate property of the reaction object. This property shows the kinetic law used to define ReactionRate.

KineticLaw can be configured with the addkineticlaw method. The addkineticlaw function configures the ReactionRate based on the KineticLaw and the species and parameters specified in the kinetic law object properties SpeciesVariableNames and ParameterVariableNames. SpeciesVariableNames are determined automatically for mass action kinetics.

If you update the reaction, the ReactionRate property automatically updates only for mass action kinetics. For all other kinetics, you must set the SpeciesVariableNames property of the kinetic law object.

For information on dimensional analysis for reaction rates, see “How Reaction Rates Are Evaluated”.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: reaction                           |
| Data type   | Kinetic law object                         |
| Data values | Kinetic law object. Default is [] (empty). |
| Access      | Read-only                                  |

## Examples

Example with Henri-Michaelis-Menten kinetics

- 1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');  
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

- 2 Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

## KineticLaw property

---

- 3 Verify that the KineticLaw property for the reaction object is Henri-Michaelis-Menten.

```
get (reactionObj, 'KineticLaw')
```

MATLAB returns:

```
SimBiology Kinetic Law Array
```

```
Index:    KineticLawName:  
    1      Henri-Michaelis-Menten
```

### See Also

KineticLawName, Parameters, ParameterVariableNames, ReactionRate, SpeciesVariableNames

**Purpose** Name of kinetic law applied to reaction

**Description** The `KineticLawName` property of the kinetic law object indicates the name of the kinetic law definition applied to the reaction. `KineticLawName` can be any valid name from the built-in or user-defined kinetic law library. See “Kinetic Law Definition” on page 3-71 for more information.

You can find the `KineticLawName` list in the kinetic law library by using the command `sbiowhos -kineticlaw (sbiowhos)`. You can create a kinetic law definition with the function `sbioabstractkineticlaw` and add it to the library using `sbioaddtolibrary`.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: <code>kineticlaw</code>                              |
| Data type   | <code>char string</code>                                     |
| Data values | <code>char string</code> specified by kinetic law definition |
| Access      | Read-only  |

## Examples

- 1 Create a model object, add a reaction object, and define a kinetic law for the reaction object.

```
modelObj = sbiomodel ('my_model');  
reactionObj = addreaction (modelObj, 'a + b -> c + d');  
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

- 2 Verify the `KineticLawName` of `kineticlawObj`.

```
get (kineticlawObj, 'KineticLawName')
```

MATLAB returns:

```
ans =
```

```
Henri-Michaelis-Menten
```

## KineticLawName property

---

### See Also

Expression(AbstractKineticLaw, KineticLaw), Parameters,  
ParameterVariableNames, ParameterVariables, ReactionRate,  
sbioaddtolibrary, sbiowhos, SpeciesVariables,  
SpeciesVariableNames



**Purpose** Parameter specifying time lag for doses

**Description** LagParameter is a property of the PKModelMap object. It specifies the name(s) of parameter object(s) that represent the time lag(s) of doses associated with the PKModelMap object.

Specify the name(s) of parameter object(s) that are:

- Scoped to a model
- Constant, that is, their ConstantValue property is true

When dosing multiple compartments, a one-to-one relationship must exist between the number and order of elements in the LagParameter property and the DosingType property. For a dose that has no lag, use ' ' (an empty string). For an example, see “Dosing Multiple Compartments in a Model”.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Object: PKModelMap  |
| Data type   | char string or cell array of strings  |
| Data values | <u>Name(s) of parameter object(s) or empty.</u><br><b>Tip</b> If you are not using any doses with time lags, you can set this property to a cell array of empty strings, or simply an empty cell array.<br>The parameter objects must be: <ul style="list-style-type: none"><li>• Scoped to a model</li><li>• Constant, that is, have a ConstantValue property set to true.</li></ul> |
| Access      | Read/write  |

**See Also** “Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters” in the SimBiology User’s Guide, DosingType, PKModelMap object

# LagParameterName property

---

**Purpose** Parameter specifying time lag for dose

**Description** LagParameterName is a property of a RepeatDose or ScheduleDose object.

Specify the name of a parameter object that is:

- Scoped to a model
- Constant, that is, its ConstantValue property is true

The parameter specifies the length of time it takes for the dose to reach its target after being introduced.

## Characteristics

Applies to Objects: RepeatDose, ScheduleDose

Data type char string

Data values Name of a parameter object or empty. Default is an empty string.

Access Read/write  
The parameter object must be:

**See Also** RepeatDose object, ScheduleDose object, Scoped model object

- Constant, that is, have a ConstantValue property set to true

**Purpose** Specify frequency to log stochastic simulation output

**Description** LogDecimation is a property of the SolverOptions property, which is a property of a configset object. This property defines how often stochastic simulation data is recorded. LogDecimation is available only for stochastic solvers (ssa, expltau, and impltau).

Use LogDecimation to specify how frequently you want to record the output of the simulation. For example, if you set LogDecimation to 1, for the command `[t,x] = sbiosimulate(modelObj)`, at each simulation step the time will be logged in `t` and the quantity of each logged species will be logged as a row in `x`. If LogDecimation is 10, then every 10th simulation step will be logged in `t` and `x`.

## Characteristics

|             |                       |
|-------------|-----------------------|
| Applies to  | Object: SolverOptions |
| Data type   | int                   |
| Data values | >0. Default is 1.     |
| Access      | Read/write            |

**Examples** This example shows how to change LogDecimation settings.

- 1 Retrieve the configset object from the modelObj, and change the SolverType to expltau.

```
modelObj = sbiomodel('cell');  
configsetObj = getconfigset(modelObj);  
set(configsetObj, 'SolverType', 'expltau')
```

- 2 Change the LogDecimation to 10.

```
set(configsetObj.SolverOptions, 'LogDecimation', 10);  
get(configsetObj.SolverOptions, 'LogDecimation')
```

```
ans =
```

## LogDecimation property

---

10

### **See Also**

ErrorTolerance, RandomState

# MaximumNumberOfLogs property

---

**Purpose** Maximum number of logs criteria to stop simulation

**Description** MaximumNumberOfLogs is a property of a Configset object. This property sets the maximum number of logs criteria to stop a simulation. A simulation stops when it meets any of the criteria specified by StopTime, MaximumNumberOfLogs, or MaximumWallClock. However, if you specify the OutputTimes property of the SolverOptions property of the Configset object, then StopTime and MaximumNumberOfLogs are ignored. Instead, the last value in OutputTimes is used as the StopTime criteria, and the length of OutputTimes is used as the MaximumNumberOfLogs criteria.

|                        |             |                                 |
|------------------------|-------------|---------------------------------|
| <b>Characteristics</b> | Applies to  | Object: Configset               |
|                        | Data type   | double                          |
|                        | Data values | Positive value. Default is Inf. |
|                        | Access      | Read/write                      |

## **Examples** **Set Maximum Number of Logs Criteria to Stop Simulation**

Set the maximum number of logs that triggers a simulation to stop.

Create a model object named cell and save it in a variable named modelObj.

```
modelObj = sbiomodel('cell');
```

Retrieve the configuration set from modelObj and save it in a variable named configsetObj.

```
configsetObj = getconfigset(modelObj);
```

Configure the simulation stop criteria by setting the MaximumNumberOfLogs property to 50. Leave the StopTime and MaximumWallClock properties at their default values of 10 seconds and Inf, respectively.

# MaximumNumberOfLogs property

---

```
set(configsetObj, 'MaximumNumberOfLogs', 50)
```

View the properties of configsetObj.

```
get(configsetObj)
```

```
Active: 1
CompileOptions: [1x1 SimBiology.CompileOptions]
    Name: 'default'
    Notes: ''
RuntimeOptions: [1x1 SimBiology.RuntimeOptions]
SensitivityAnalysisOptions: [1x1 SimBiology.SensitivityAnalysisOptions]
SolverOptions: [1x1 SimBiology.ODESolverOptions]
    SolverType: 'ode15s'
    StopTime: 10
MaximumNumberOfLogs: 50
MaximumWallClock: Inf
    TimeUnits: 'second'
    Type: 'configset'
```

When you simulate modelObj, the simulation stops when 50 logs are created or when the simulation time reaches 10 seconds, whichever comes first.

## See Also

Configset object, MaximumWallClock, OutputTimes, StopTime

**Purpose** Maximum elapsed wall clock time to stop simulation

**Description** MaximumWallClock is a property of a Configset object. This property sets the maximum elapsed wall clock time (seconds) criteria to stop a simulation.

A simulation stops when it meets any of the criteria specified by StopTime, MaximumNumberOfLogs, or MaximumWallClock. However, if you specify the OutputTimes property of the SolverOptions property of the Configset object, then StopTime and MaximumNumberOfLogs are ignored. Instead, the last value in OutputTimes is used as the StopTime criteria, and the length of OutputTimes is used as the MaximumNumberOfLogs criteria.

|                        |             |                                  |
|------------------------|-------------|----------------------------------|
| <b>Characteristics</b> | Applies to  | Object: Configset                |
|                        | Data type   | double                           |
|                        | Data values | Positive scalar. Default is Inf. |
|                        | Access      | Read/write                       |

## **Examples**      **Set Maximum Wall Clock Criteria to Stop Simulation**

Set the maximum wall clock time (in seconds) that triggers a simulation to stop.

Create a model object named cell and save it in a variable named modelObj.

```
modelObj = sbiomodel('cell');
```

Retrieve the configuration set from modelObj and save it in a variable named configsetObj.

```
configsetObj = getconfigset(modelObj);
```

## MaximumWallClock property

---

Configure the simulation stop criteria by setting the `MaximumWallClock` property to 20 seconds. Leave the `StopTime` and `MaximumNumberOfLogs` properties at their default values of 10 seconds and `Inf`, respectively.

```
set(configsetObj, 'MaximumWallClock', 20)
```

View the properties of `configsetObj`.

```
get(configsetObj)
```

```
Active: 1
CompileOptions: [1x1 SimBiology.CompileOptions]
    Name: 'default'
    Notes: ''
RuntimeOptions: [1x1 SimBiology.RuntimeOptions]
SensitivityAnalysisOptions: [1x1 SimBiology.SensitivityAnalysisOptions]
SolverOptions: [1x1 SimBiology.ODESolverOptions]
    SolverType: 'ode15s'
    StopTime: 10
MaximumNumberOfLogs: Inf
MaximumWallClock: 20
    TimeUnits: 'second'
    Type: 'configset'
```

When you simulate `modelObj`, the simulation stops when the simulation time reaches 10 seconds or the wall clock time reaches 20 seconds, whichever comes first.

### See Also

`Configset` object, `MaximumNumberOfLogs`, `OutputTimes`, `StopTime`



**Purpose** Specify nonlinear solver maximum iterations in implicit tau

**Description** The `MaxIterations` property specifies the maximum number of iterations for the nonlinear solver in `impltau`. It is a property of the `SolverOptions` object. `SolverOptions` is a property of the `configset` object.

The implicit tau solver in SimBiology software internally uses a nonlinear solver to solve a set of algebraic nonlinear equations at every simulation step. Starting with an initial guess at the solution, the nonlinear solver iteratively tries to find the solution to the algebraic equations. The closer the initial guess is to the solution, the fewer the iterations the nonlinear solver will take before it finds a solution. `MaxIterations` specifies the maximum number of iterations the nonlinear solver should take before it issues a “failed to converge” error. If you get this error during simulation, try increasing `MaxIterations`. The default value of `MaxIterations` is 15.

## Characteristics

|             |                                    |
|-------------|------------------------------------|
| Applies to  | Object: <code>SolverOptions</code> |
| Data type   | <code>int</code>                   |
| Data values | >0. Default is 15.                 |
| Access      | Read/write                         |

## Examples

This example shows how to change `MaxIterations` settings.

- 1 Retrieve the `configset` object from the `modelObj`, and change the `SolverType` to `impltau`.

```
modelObj = sbiomodel('cell');  
configsetObj = getconfigset(modelObj);  
set(configsetObj, 'SolverType', 'impltau');
```

- 2 Change the `MaxIterations` to 25.

```
set(configsetObj.SolverOptions, 'MaxIterations', 25);
```

## MaxIterations property

---

```
get(configsetObj.SolverOptions, 'MaxIterations')
```

```
ans =
```

```
25
```

### **See Also**

ErrorTolerance, LogDecimation, RandomState

**Purpose** Specify upper bound on ODE solver step size

**Description** MaxStep is a property of the SolverOptions property, which is a property of a configset object. This property specifies the bounds on the size of the time steps. MaxStep is available only for ODE solvers (ode15s, ode23t, ode45, and sundials).

If the differential equation has periodic coefficients or solutions, it might be a good idea to set MaxStep to some fraction (such as 1/4) of the period. This guarantees that the solver does not enlarge the time step too much and step over a period of interest. For more information on MaxStep, see odeset in the MATLAB documentation.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: SolverOptions  |
| Data type   | Positive scalar or empty   |
| Data values | Default value is [] (empty), which is equivalent to setting MaxStep to infinity. |
| Access      | Read/write   |

**See Also** SimBiology property RelativeTolerance  
MATLAB function odeset

# ModelName property

---

**Purpose** Name of model simulated

**Description** The ModelName property shows the name of the model for which the SimData object contains the simulation data.

## Characteristics

|             |                              |
|-------------|------------------------------|
| Applies to  | Object: SimData              |
| Data type   | string                       |
| Data values | Default value is '' (empty). |
| Access      | Read-only                    |

**See Also** Data, DataInfo

**Purpose** Contain all model objects

**Description** The `Models` property shows the models in the SimBiology root. It is a read-only array of model objects.

SimBiology has a hierarchical organization. A model object has the SimBiology root as its `Parent`. Parameter objects can have a model object or kinetic law object as `Parent`. You can display all the component objects with `modelObj.Models` or `get (modelObj, 'Models')`.

## Characteristics

|             |                                      |
|-------------|--------------------------------------|
| Applies to  | Objects: root                        |
| Data type   | Array of model objects               |
| Data values | Model object. Default is [] (empty). |
| Access      | Read-only                            |

**See Also** `sbiomodel`

# Multiplier property

---

**Purpose** Relationship between defined unit and base unit

**Description** The `Multiplier` is the numerical value that defines the relationship between the unit `Name` and the base unit as a product of the `Multiplier` and the base unit. For example, in `Celsius = (5/9)*( Fahrenheit-32)`; `Multiplier` is 5/9 and `Offset` is -32. For `1 mole = 6.0221e23*molecule`, the `Multiplier` is 6.0221e23.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: Unit                             |
| Data type   | double                                   |
| Data values | Nonzero real number. Default value is 1. |
| Access      | Read/write                               |

## Examples

This example shows how to create a user-defined unit, add it to the user-defined library, and query the library.

**1** Create a user-defined unit called `usermole`, whose composition is `molecule` and `Multiplier` property is 6.0221e23.

```
unitObj = sbiounit('usermole', 'molecule', 6.0221e23);
```

**2** Add the unit to the user-defined library.

```
sbioaddtolibrary(unitObj);
```

**3** Query the `Multiplier` property.

```
get(unitObj, 'Multiplier')
```

```
ans =
```

```
1/molarity*second
```

## See Also

`Composition`, `get`, `Offset`, `sbiounit`, `set`

## Purpose

Specify name of object

## Description

The Name property identifies a SimBiology object. Compartments, species, parameters, and model objects can be referenced by other objects using the Name property, therefore Name must be unique for these objects. However, species names need only be unique within each compartment. Parameter names must be unique within a model (if at the model level), or within each kinetic law (if at the kinetic law level). This means that you can have nonunique species names if the species are in different compartments, and nonunique parameter names if the parameters are in different kinetic laws or at different levels. Note that having nonunique parameter names can cause the model to have shadowed parameters and that may not be best modeling practice. For more information on levels of parameters, see “Scope of Parameter Objects”.

Use the function `sbioselect` to find an object with the same Name property value.

In addition, note the following constraints and reserved characters for the Name property in objects:

- Model and parameter names cannot be empty, the word `time`, all whitespace, or contain the characters `[` or `]`.
- Compartment and species names cannot be empty, the word `null`, the word `time` or contain the characters `->`, `<->`, `[` or `]`.
  - However, compartment and species names can contain the words `null` and `time` within the name, such as `nulldrug` or `nullreceptor`.
- Reaction, event, and rule names cannot be the word `time` or contain the characters `[` or `]`.
- If you have a parameter, a species, or compartment name that is not a valid MATLAB variable name, when you write an event function, an event trigger, a reaction, reaction rate equation, or a rule you must enclose that name in brackets. For example, enclose `[DNA polymerase+]` in brackets. In addition, if you have the same species

# Name property

---

in multiple compartments you must qualify the species with the compartment name, for example, `nucleus.[DNA polymerase+]`, `[nuclear complex].[DNA polymerase+]`.

For more information on valid MATLAB variable names, see `matlab.lang.makeValidName`, `matlab.lang.makeUniqueStrings`, and `isvarname`.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Objects: abstract kinetic law, configuration set, compartment, event, kinetic law, model, parameter, reaction, RepeatDose, rule, ScheduleDose, species, unit, or variant |
| Data type   | char string  |
| Data values | Any char string except reserved words and characters   |
| Access      | Read/write   |

## Examples

- 1 Create a model object named `my_model`.

```
modelObj = sbiomodel ('my_model');
```

- 2 Add a reaction object to the model object. Note the use of brackets because the names are not valid MATLAB variable names.

```
reactionObj = addreaction(modelObj, '[Aspartic acid] -> [beta-Aspartyl-P04]')
```

MATLAB returns:

```
SimBiology Reaction Array
```

```
Index:    Reaction:
     1     [Aspartic acid] -> [beta-Aspartyl-P04]
```

- 3 Set the reaction Name and verify.



```
set (reactionObj, 'Name', 'Aspartate kinase reaction');  
get (reactionObj, 'Name')
```

MATLAB returns:

```
ans =
```

```
Aspartate kinase reaction
```

### See Also

addcompartment, addkineticlaw, addparameter, addreaction,  
addrule, addspecies, RepeatDose object, sbiomodel, sbiunit,  
sbiunitprefix, ScheduleDose object

# Normalization property

---

**Purpose** Specify normalization type for sensitivity analysis

**Description** Normalization is a property of the `SensitivityAnalysisOptions` object. `SensitivityAnalysisOptions` is a property of the configuration set object. Use `Normalization` to specify the normalization for the computed sensitivities.

The following values let you specify the type of normalization. The examples show you how sensitivities of a species  $x$  with respect to a parameter  $k$  are calculated for each normalization type:

- 'None' specifies no normalization.

$$\frac{dx(t)}{dk}$$

- 'Half' specifies normalization relative to the numerator (species quantity) only.

$$\left( \frac{1}{x(t)} \right) \left( \frac{dx(t)}{dk} \right)$$

- 'Full' specifies that the data should be made dimensionless.

$$\left( \frac{k}{x(t)} \right) \left( \frac{dx(t)}{dk} \right)$$

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Object: <code>SensitivityAnalysisOptions</code> |
| Data type   | enum  |
| Data values | 'None', 'Half', 'Full'. Default is 'None'.      |
| Access      | Read/write                                      |

**See Also** `Inputs`, `Outputs`, `SensitivityAnalysis`, `SensitivityAnalysisOptions`

**Purpose** HTML text describing SimBiology object

**Description** Use the `Notes` property of an object to store comments about the object. You can include HTML tagging in the notes to render formatted text in the SimBiology desktop.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Objects: compartment, kinetic law, model, parameter, reaction, RepeatDose, rule, ScheduleDose, species, unit, or unit prefix |
| Data type   | char string  |
| Data values | Any char string  |
| Access      | Read/write   |

## Examples

**1** Create a model object.

```
modelObj = sbiomodel ('my_model');
```

**2** Write notes for the model object.

```
set (modelObj, 'notes', '09/01/05 experimental data')
```

**3** Verify the assignment.

```
get (modelObj, 'notes')
```

MATLAB returns:

```
ans =
```

```
09/01/05 experimental data
```

## See Also

`addkineticlaw`, `addparameter`, `addreaction`, `addrule`, `addspecies`, `RepeatDose` object, `sbiomodel`, `sbiounit`, `sbiounitprefix`, `ScheduleDose` object

# Observed property

---

**Purpose** Measured response object name

**Description** Observed is a property of the PKModelMap object. It specifies the name(s) of one or more objects that represent the measured response (the response variable). Specify the names of species, parameters, or compartments that are scoped to a model.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: PKModelMap   |
| Data type   | char string or cell array of strings   |
| Data values | Name of a species, parameter, compartment, or empty. Default is an empty cell array. |
| Access      | Read/write   |

**See Also** “Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters” in the SimBiology User’s Guide, Dosed, Estimated, PKModelMap object

**Purpose** Unit composition modifier

## Description

---

**Note** The Offset property is currently not supported.

---

The Offset is the numerical value by which the unit composition is modified from the base unit. For example, `Celsius = (5/9)*( Fahrenheit-32)`; Multiplier is 5/9 and Offset is -32.

## Characteristics

|             |                            |
|-------------|----------------------------|
| Applies to  | Object: Unit               |
| Data type   | double                     |
| Data values | Real number. Default is 0. |
| Access      | Read/write                 |

## Examples

This example shows how to create a user-defined unit, add it to the user-defined library, and query the library.

- 1 Create a user-defined unit called `celsius2`, whose composition refers to `fahrenheit`, `Multiplier` property is 9/5, and `Offset` property is 32.

```
unitObj = sbiounit('celsius2','fahrenheit',9/5,32);
```

- 2 Add the unit to the user-defined library.

```
sbioaddtolibrary(unitObj);
```

- 3 Query the Offset property.

```
get(unitObj, 'Offset')
```

```
ans =
```

# Offset property

---

32

## **See Also**

Composition, get, Multiplier, sbioaddtolibrary, sbioshowunits, sbiunit, set

**Purpose** Specify species and parameter outputs for sensitivity analysis

**Description** Outputs is a property of the SensitivityAnalysisOptions object. SensitivityAnalysisOptions is a property of the configuration set object.

Use Outputs to specify the species and parameters for which you want to compute sensitivities.

The SimBiology software calculates sensitivities with respect to the values of the parameters and the initial amounts of the species specified in the Inputs property. When you simulate a model with SensitivityAnalysis enabled in the active configuration set object, sensitivity analysis returns the computed sensitivities of the species and parameters specified in Outputs. For a description of the output, see the SensitivityAnalysisOptions property description.

## Characteristics

|            |   |
|------------|---|
| Applies to | Object: SensitivityAnalysisOptions              |
| Data type  | Species or parameter object or array of objects |

---

**Note** If a species or parameter object is determined by a repeated assignment rule, then you cannot use it as an Outputs property.

---

|             |  |
|-------------|--|
| Data values | Species or parameter object, or an array of objects. Default is [ ] (empty array). |
| Access      | Read/write   |

**Examples** This example shows how to set Outputs for sensitivity analysis.

- 1 Import the radio decay model from the SimBiology demos.

## Outputs property

---

```
modelObj = sbmlimport('radiodecay');
```

- 2 Retrieve the configuration set object from modelObj.

```
configsetObj = getConfigset(modelObj);
```

- 3 Add a species to the Outputs property and display it. Use the sbioselect function to retrieve the species object from the model.

```
SimBiology Species Array
```

```
Index:  Compartment:  Name:  InitialAmount:  InitialAmountUnits:
      1      unnamed      z      0      molecule
```

### See Also

Inputs, sbioselect, SensitivityAnalysis, SensitivityAnalysisOptions



**Purpose** Specify times to log deterministic simulation output

**Description** OutputTimes is a property of the SolverOptions property, which is a property of a Configset object. This property specifies the times during a deterministic (ODE) simulation that data is recorded. Time units are specified by the TimeUnits property of the Configset object. OutputTimes is available only for ODE solvers (ode15s, ode23t, ode45, and sundials).

If the criteria set in the MaximumWallClock property causes a simulation to stop before all time values in OutputTimes are reached, then no data is recorded for the latter time values.

The OutputTimes property can also control when a simulation stops:

- The last value in OutputTimes overrides the StopTime property as criteria for stopping a simulation.
- The length of OutputTimes overrides the MaximumNumberOfLogs property as criteria for stopping a simulation.

|                        |             |  |
|------------------------|-------------|--|
| <b>Characteristics</b> | Applies to  | Object: SolverOptions  |
|                        | Data type   | double   |
|                        | Data values | Vector of nonnegative, monotonically increasing values, or [], an empty vector. Default is [], which results in data being logged every time the simulation solver takes a step. |
|                        | Access      | Read/write   |

## **Examples** Specify Times to Log Deterministic Simulation Output

Specify the times during a deterministic (ODE) simulation that data is recorded.

Create a model object named cell and save it in a variable named modelObj.

## OutputTimes property

---

```
modelObj = sbiomodel('cell');
```

Retrieve the configuration set from modelObj and save it in a variable named configsetObj.

```
configsetObj = getConfigset(modelObj);
```

Specify to log output every second for the first 10 seconds of the simulation. Do this by setting the OutputTimes property of the SolverOptions property of ConfigsetObj.

```
set(configsetObj.SolverOptions, 'OutputTimes', [1:10])  
get(configsetObj.SolverOptions, 'OutputTimes')
```

```
ans =
```

```
1 2 3 4 5 6 7 8 9 10
```

When you simulate modelObj, output is logged every second for the first 10 seconds of the simulation. Also, the simulation stops after the 10th log.

### See Also

MaximumNumberOfLogs, MaximumWallClock, SolverOptions, StopTime, TimeUnits

**Purpose** Owing compartment

**Description** Owner shows you the SimBiology compartment object that owns the compartment object. In the compartment object, the Owner property shows you whether the compartment resides within another compartment. The Compartments property indicates whether other compartments reside within the compartment. You can add a compartment object using the method addcompartment.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: compartment                        |
| Data type   | char string                                |
| Data values | Name of compartment object. Default is []. |
| Access      | Read-only                                  |

## Examples

**1** Create a model object named modelObj.

```
modelObj = sbiomodel('cell');
```

**2** Add two compartments to the model object.

```
compartmentObj1 = addcompartment(modelObj, 'nucleus');  
compartmentObj2 = addcompartment(modelObj, 'mitochondrion');
```

**3** Add a compartment to one of the compartment objects.

```
compartmentObj3 = addcompartment(compartmentObj2, 'matrix');
```

**4** Display the Owner property in the compartment objects.

```
get(compartmentObj3, 'Owner')
```

The result shows you the owning compartment and its components:

```
SimBiology Compartment - mitochondrion
```

## Owner property

---

```
Compartment Components:  
Capacity:          1  
CapacityUnits:  
Compartments:     1  
ConstantCapacity: true  
Owner:  
Species:          0
```

**5** Change the owning compartment.

```
set(compartmentObj3, 'Owner', compartmentObj1)
```

### **See Also**

Compartments, Parent

# ParameterNames (CovariateModel) property

---

**Purpose** Names of parameters in CovariateModel object

**Description** The ParameterNames property is a cell array of strings specifying the names of the parameters in the Expression property of a CovariateModel object.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: CovariateModel                             |
| Data type   | Cell array of strings                              |
| Data values | Names of the parameters in the Expression property |
| Access      | Read only  |

**See Also** CovariateModel | Expression

# Parameters property

---

**Purpose** Array of parameter objects

**Description** The Parameters property indicates the parameters in a Model or KineticLaw object. Read-only array of Parameter objects.

The scope of a parameter object is hierarchical and is defined by the parameter's parent. If a parameter is defined with a kinetic law object as its parent, then only the kinetic law object can use the parameter. If a parameter object is defined with a model object as its parent, then components such as rules, events, and kinetic laws (reaction rate equations) can use the parameter.

You can add a parameter to a model object, or kinetic law object with the method `addparameter` and delete it with the method `delete`.

You can view parameter object properties with the `get` command and configure properties with the `set` command.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Objects: <code>model</code> , <code>kineticlaw</code> |
| Data type   | Array of parameter objects                            |
| Data values | Parameter objects. Default value is [] (empty).       |
| Access      | Read-only   |

## Examples

**1** Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');  
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

**2** Define a kinetic law for the reaction object.

```
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');
```

**3** Add a parameter and assign it to the kinetic law object (`kineticlawObj`);

```
parameterObj1 = addparameter (kineticlawObj, 'K1');
```

```
get (kineticlawObj, 'Parameters')
```

```
SimBiology Parameter Array
```

| Index: | Name: | Value: | ValueUnits: |
|--------|-------|--------|-------------|
| 1      | K1    | 1      |             |

- 4** Add a parameter and assign it to the model object (modelObj);

```
parameterObj1 = addparameter(modelObj, 'K2');  
get(modelObj, 'Parameters')  
SimBiology Parameter Array
```

| Index: | Name: | Value: | ValueUnits: |
|--------|-------|--------|-------------|
| 1      | K2    | 1      |             |

## See Also

addparameter, delete, get, set

# ParameterVariableNames property

---

**Purpose** Cell array of reaction rate parameters

**Description** The `ParameterVariableNames` property shows the parameters used by the kinetic law object to determine the `ReactionRate` equation in the reaction object. Use `setParameter` to assign `ParameterVariableNames`. When you assign species to `ParameterVariableNames`, SimBiology software maps these parameter names to `ParameterVariables` in the kinetic law object.

If the reaction is using a kinetic law, the `ReactionRate` property of a reaction object shows the result of a mapping from a “Kinetic Law Definition” on page 3-71. The `ReactionRate` is determined by the kinetic law object `Expression` property by mapping `ParameterVariableNames` to `ParameterVariables` and `SpeciesVariableNames` to `SpeciesVariables`.

## Characteristics

|             |                                 |
|-------------|---------------------------------|
| Applies to  | Object: <code>kineticlaw</code> |
| Data type   | Cell array of strings           |
| Data values | Cell array of parameters        |
| Access      | Read/write                      |

**Examples** Create a model, add a reaction, and assign the `SpeciesVariableNames` for the reaction rate equation.

- 1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');  
reactionObj = addreaction(modelObj, 'a -> c + d');
```

- 2 Create a kinetic law object for the reaction object of type 'Henri-Michaelis-Menten'.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

```
reactionObj.KineticLaw property is configured to kineticlawObj.
```



## ParameterVariableNames property

---

- 3** The 'Henri-Michaelis-Menten' kinetic law has two parameter variables ( $V_m$  and  $K_m$ ) that should to be set. To set these variables:

```
setparameter(kineticlawObj, 'Vm', 'Va');  
setparameter(kineticlawObj, 'Km', 'Ka');
```

- 4** Verify that the parameter variables are correct.

```
get (kineticlawObj, 'ParameterVariableNames')
```

MATLAB returns:

```
ans =  
  
    'Va'    'Ka'
```

### See Also

Expression(AbstractKineticLaw, KineticLaw),  
ParameterVariables, ReactionRate, setparameter,  
SpeciesVariables, SpeciesVariableNames

# ParameterVariables property

---

**Purpose** Parameters in kinetic law definition

**Description** The `ParameterVariables` property shows the parameter variables that are used in the `Expression` property of the abstract kinetic law object. Use this property to specify the parameters in the `ReactionRate` equation. Use the method `set` to assign `ParameterVariables` to a kinetic law definition. For more information, see “Kinetic Law Definition” on page 3-71.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Objects: abstract kinetic law, kineticlaw                       |
| Data type   | Cell array of strings   |
| Data values | Specified by kinetic law definition                             |
| Access      | Read/write in kinetic law definition. Read-only in kinetic law. |

## Examples

Create a model, add a reaction, and assign the `SpeciesVariableNames` for the reaction rate equation.

- 1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');  
reactionObj = addreaction(modelObj, 'a -> c + d');
```

- 2 Create a kinetic law object for the reaction object of the type 'Henri-Michaelis-Menten'.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

reactionObj.KineticLaw property is configured to kineticlawObj.

- 3 The 'Henri-Michaelis-Menten' kinetic law has two parameter variables. To set these variables:

```
get(kineticlawObj, 'ParameterVariables')
```

MATLAB returns:

```
ans =
```

```
    'Vm' 'Km'
```

### See Also

Expression(AbstractKineticLaw, KineticLaw),  
ParameterVariableNames, ReactionRate, set, setparameter,  
SpeciesVariables, SpeciesVariableNames

# Parent property

---

**Purpose** Indicate parent object

**Description** The Parent property indicates the parent object for a SimBiology object (read-only). The Parent property indicates accessibility of the object. The object is accessible to the Parent object and other objects within the Parent object. The value of Parent depends on the type of object and how it was created. All models always have the SimBiology root as the Parent.

## More Information

The following table shows you the different objects and the possible Parent value.

| Object               | Parent   |
|----------------------|--|
| abstract kinetic law | <ul style="list-style-type: none"><li>• [] (empty) until added to library</li><li>• root object upon addition to library</li></ul> |
| compartment          | model object   |
| event                | model object or [] (empty)   |
| kinetic law          | reaction object  |
| model                | root object  |
| parameter            | model object, kinetic law object, or [] (empty)  |
| reaction             | model object or [] (empty)   |
| RepeatDose           | model object or [] (empty)   |
| rule                 | model object or [] (empty)   |
| ScheduleDose         | model object or [] (empty)   |
| species              | compartment  |

| Object                 | Parent  |
|------------------------|---|
| variant                | model object or [ ] (empty)   |
| unit and unit prefixes | <ul style="list-style-type: none"><li>• [ ] (empty) until added to library</li><li>• root object upon addition to library</li></ul> |

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Objects: abstract kinetic law, compartment, event, kinetic law, model, parameter, reaction, RepeatDose, rule, ScheduleDose, species, variant, unit, or unit prefix |
| Data type   | Object   |
| Data values | SimBiology component object or [ ] (empty)   |
| Access      | Read-only  |

## See Also

addkineticlaw, addparameter, addreaction, RepeatDose object, sbiomodel, ScheduleDose object

# PKCompartments property

---

**Purpose** Hold compartments in PK model

**Description** PKCompartments is a property of the PKModelDesign object. It is used to specify the compartments in the PKModelDesign object. Each compartment is a PKCompartment object added using the addCompartment method.

## Characteristics

Applies to Objects: PKModelDesign

Data type object

Data values PKCompartment object

Access Read-only

**See Also** “Create Pharmacokinetic Models” in the SimBiology User’s Guide, addCompartment, PKCompartment object, PKModelDesign object

**Purpose** Array of reaction products

**Description** The `Products` property contains an array of `SimBiology.Species` objects.

`Products` is a 1-by-n species object array that indicates the species that are changed by the reaction. If the `Reaction` property is modified to use a different species, the `Products` property is updated accordingly.

You can add product species to the reaction with `addproduct` function. You can remove product species from the reaction with `rmproduct`. You can also update reaction products by setting the `Reaction` property with the function `set`.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Object: reaction                        |
| Data type   | Array of objects                        |
| Data values | Species objects. Default is [] (empty). |
| Access      | Read-only                               |

## Examples

**1** Create a model object.

```
modelObj = sbiomodel ('my_model');
```

**2** Add reaction objects.

```
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

**3** Verify the assignment.

```
productsObj = get(reactionObj, 'Products')
```

MATLAB returns:

```
SimBiology Species Array
```

```
Index:  Compartment:  Name:  InitialAmount:  InitialAmountUnits:
```

## Products property

---

|   |         |   |   |
|---|---------|---|---|
| 1 | unnamed | c | 0 |
| 2 | unnamed | d | 0 |

**See Also**      `addkineticlaw, addproduct, addspecies, rmproduct`



# RandomEffectNames (CovariateModel) property

---

**Purpose** Names of random effects in CovariateModel object

**Description** The RandomEffectNames property is a cell array of strings specifying the names of the random effects in the Expression property of a CovariateModel object. Names of random effects are denoted with the prefix eta.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: CovariateModel   |
| Data type   | Cell array of strings  |
| Data values | Names of the random effects in the Expression property. These names are denoted with the prefix eta. |
| Access      | Read only  |

**See Also** CovariateModel | Expression

# RandomState property

---

**Purpose** Set random number generator

**Description** The `RandomState` property sets the random number generator for the stochastic solvers. It is a property of the `SolverOptions` object. `SolverOptions` is a property of the `configset` object.

`SimBiology` software uses a pseudorandom number generator. The sequence of numbers generated is determined by the state of the generator, which can be specified by the integer `RandomState`. If `RandomState` is set to integer  $J$ , the random number generator is initialized to its  $J^{\text{th}}$  state. The random number generator can generate all the floating-point numbers in the closed interval  $[2^{(-53)}, 1-2^{(-53)}]$ . Theoretically, it can generate over  $2^{1492}$  values before repeating itself. But for a given state, the sequence of numbers generated will be the same. To change the sequence, change `RandomState`. `SimBiology` software resets the state at startup. The default value of `RandomState` is `[]`.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Objects: <code>SolverOptions</code> for SSA, <code>expltau</code> , <code>impltau</code> |
| Data type   | <code>int</code>   |
| Data values | Default is <code>[]</code> (empty).  |
| Access      | Read/write   |

## Examples

This example shows how to change `RandomState` settings.

- 1 Retrieve the `configset` object from the `modelObj` and change the `SolverType` to `expltau`.

```
modelObj = sbiomodel('cell');  
configsetObj = getconfigset(modelObj);  
set(configsetObj, 'SolverType', 'expltau')
```

- 2 Change the `Randomstate` to 5.

```
set(configsetObj.SolverOptions, 'RandomState', 5);  
get(configsetObj.SolverOptions, 'RandomState')
```

```
ans =
```

```
5
```

### See Also

ErrorTolerance, LogDecimation, MaxIterations

# Rate property

---

**Purpose** Rate of dose

**Description** Rate is a property of a RepeatDose or ScheduleDose object.  
This property defines how fast a dose is given.

---

**Note** If you set the Rate property of a dose, you must also specify the Amount property of the dose, and set the DurationParameterName property to ' '. This is because the duration is calculated from the amount and rate.

---

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Objects: RepeatDose, ScheduleDose  |
| Data type   | double (RepeatDose) or double array (ScheduleDose)                       |
| Data values | Nonnegative real number. Default is 0 (RepeatDose) or [] (ScheduleDose). |
| Access      | Read/write   |

**See Also** RepeatDose object, ScheduleDose object

**Purpose** Units for dose rate

**Description** RateUnits is a property of a PKData, RepeatDose or ScheduleDose object.

- In RepeatDose or ScheduleDose objects, this property defines units for the Rate property.
- In PKData object, this property defines units for the RateLabel property.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: RepeatDose, ScheduleDose, PKData   |
| Data type   | string   |
| Data values | Units from library with dimensions of amount divided by time. You cannot use units of concentration divided by time. Default = " |
| Access      | Read/write   |

## RateUnits property

---

---

**Note** SimBiology uses unit strings including empty units in association with DimensionalAnalysis and UnitConversion features.

- When DimensionalAnalysis and UnitConversion are both false, unit strings are not used. However, SimBiology still performs a minimum level of dimensional analysis to decide whether a reaction rate is in dimensions of amount/time or concentration/time.
- When DimensionalAnalysis is true and UnitConversion is false, units (if not empty) must have consistent dimensions so that SimBiology can perform dimensional analysis. However, the units are not converted.
- When UnitConversion is set to true (which requires DimensionalAnalysis to be true), SimBiology performs a dimensional analysis and converts everything to consistent units. Hence, you must specify consistent units, and no unit strings can be empty. If you have a dimensionless parameter, you must still set its unit string to dimensionless.

---

### See Also

PKData object, ScheduleDose object, RepeatDose object, Rate, RateLabel

**Purpose** Rate of infusion column in data set

**Description** RateLabel is a property of the PKData object. It specifies the column in DataSet that contains the rate of infusion. This applies only when dosing type is infusion. The data set must contain the rate and not an infusion time. The values must be positive and the column cannot contain Inf or -Inf. 0 specifies an infinite rate (equivalent to a bolus dose), and NaN specifies no rate.

## Characteristics

|             |                      |
|-------------|----------------------|
| Applies to  | Objects: PKData      |
| Data type   | char string          |
| Data values | Column header string |
| Access      | Read/write           |

**See Also** “Specify and Classify the Data to Fit” in the SimBiology User’s Guide, PKData object, DosingType

# Reactants property

---

**Purpose** Array of reaction reactants

**Description** The `Reactants` property is a 1-by-n species object array of reactants in the reaction. If the `Reaction` property is modified to use a different reactant, the `Reactants` property will be updated accordingly.

You can add reactant species to the reaction with the `addreactant` method.

You can remove reactant species from the reaction with the `rmreactant` method. You can also update reactants by setting the `Reaction` property with the function `set`.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: reaction                           |
| Data type   | Species object or array of species objects |
| Data values | Species objects. Default is [] (empty).    |
| Access      | Read-only                                  |

## Examples

**1** Create a model object.

```
modelObj = sbiomodel ('my_model');
```

**2** Add reaction objects.

```
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

**3** View the reactants for `reactionObj`.

```
get(reactionObj, 'Reactants')
```

MATLAB returns:

SimBiology Species Array

```
Index:  Compartment:  Name:  InitialAmount:  InitialAmountUnits:
      1      unnamed      a      0
```



2          unnamed          b          0

### See Also

addreactant, addreaction, addspecies, rmreactant

# Reaction property

---

**Purpose** Reaction object reaction

**Description** Property to indicate the reaction represented in the reaction object. Indicates the chemical reaction that can change the amount of one or more species, for example, 'A + B > C'. This property is different from the model object property called Reactions.

See `addreaction` for more information on how the Reaction property is set.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Object: reaction                              |
| Data type   | char string                                   |
| Data values | Valid reaction string. Default is '' (empty). |
| Access      | Read/write                                    |

## Examples

**1** Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');  
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

**2** Verify that the reaction property records the input.

```
get (reactionObj, 'Reaction')
```

MATLAB returns:

```
ans =
```

```
a + b -> c + d
```

**See Also** `addreaction`

**Purpose** Reaction rate equation in reaction object

**Description** The ReactionRate property defines the reaction rate equation. You can define a ReactionRate with or without the KineticLaw property. KineticLaw defines the type of reaction rate. The addkineticlaw function configures the ReactionRate based on the KineticLaw and the species and parameters specified in the kinetic law object properties SpeciesVariableNames and ParameterVariableNames.

The reaction takes place in the reverse direction if the Reversible property is true. This is reflected in ReactionRate. The ReactionRate includes the forward and reverse rate if reversible.

You can specify ReactionRate without KineticLaw. Use the set function to specify the reaction rate equation. SimBiology software adds species variables while creating reactionObj using the addreaction method. You must add the parameter variables (to the modelObj in this case). See the example below.

After you specify the ReactionRate without KineticLaw and you later configure the reactionObj to use KineticLaw, the ReactionRate is unset until you specify SpeciesVariableNames and ParameterVariableNames.

For information on dimensional analysis for reaction rates, see “How Reaction Rates Are Evaluated” .

---

**Note** If you set the ReactionRate property to an expression that is not continuous and differentiable, see “Using Events to Address Discontinuities in Rule and Reaction Rate Expressions” before simulating your model.

---

## Characteristics

|            |                  |
|------------|------------------|
| Applies to | Object: reaction |
| Data type  | char string      |

# ReactionRate property

---

|             |  |
|-------------|--|
| Data values | Reaction rate string. Default is '' (empty). |
| Access      | Read/write                                   |

## Examples

### Add a Reaction Defined by Michaelis-Menten Kinetic Law

Create a model, add a reaction, and assign the expression for the reaction rate equation.

- 1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');  
reactionObj = addreaction(modelObj, 'a -> c + d');
```

- 2 Create a kinetic law object for the reaction object of the type 'Henri-Michaelis-Menten'.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

reactionObj.KineticLaw property is configured to kineticlawObj.

- 3 The 'Henri-Michaelis-Menten' kinetic law has two parameter variables (Vm and Km) and one species variable (S) that you should set. To set these variables, first create the parameter variables as parameter objects (parameterObj1, parameterObj2) with names Vm\_d and Km\_d and assign them to kineticlawObj.

```
parameterObj1 = addparameter(kineticlawObj, 'Vm_d');  
parameterObj2 = addparameter(kineticlawObj, 'Km_d');
```

- 4 Set the variable names for the kinetic law object.

```
set(kineticlawObj, 'ParameterVariableNames', {'Vm_d' 'Km_d'});  
set(kineticlawObj, 'SpeciesVariableNames', {'a'});
```

- 5 Verify that the reaction rate is expressed correctly in the reaction object ReactionRate property.

```
get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =
```

```
Vm_d*a/(Km_d + a)
```

## Add a Reaction without a Kinetic Law

Create a model, add a reaction, and specify ReactionRate without a kinetic law.

- 1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');  
reactionObj = addreaction(modelObj, 'a + b -> c + d');
```

- 2 Specify ReactionRate and verify the assignment.

```
set(reactionObj, 'ReactionRate', 'k*a');  
get(reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =
```

```
k*a
```

- 3 You cannot simulate the model until you add the parameter k to the modelObj.

```
parameterObj = addparameter(modelObj, 'k');
```

SimBiology adds the parameter to the modelObj with default Value = 1.0 for the parameter.

## Define a Custom Hill Kinetic Law that Works with Dimensional Analysis

This example shows how to define a custom reaction rate for the Hill kinetics that is compatible with DimensionalAnalysis feature of SimBiology.

## ReactionRate property

---

This example is useful especially if you are using the built-in Hill kinetic law, but have the kinetic reaction with a non-integer exponent and cannot verify the model because dimensional analysis failed. The

built-in Hill kinetic law has the following expression:  $\frac{V_m * S^n}{K_p + S^n}$ . Suppose

$K_p = K_h^n$ , then you can rewrite the equation as follows:  $\frac{V_m}{\left(\frac{K_h}{S}\right)^n + 1}$ . The redefined Hill kinetic equation is compatible with Dimensional Analysis and allows you to have a non-integer exponent.

Create a SimBiology model.

```
m1 = sbiomodel('m1');
```

Add a compartment, two species, and a reaction.

```
c1 = addcompartment(m1, 'cell');  
s1 = addspecies(m1, 'a');  
s2 = addspecies(m1, 'b');  
r1 = addreaction(m1, 'a -> b');
```

Add a predefined a Hill kinetic law for the reaction.

```
k1 = addkineticlaw(r1, 'Hill-Kinetics');
```

Display the rate expression of the built-in kinetic law.

```
k1.Expression
```

```
ans =
```

```
Vm*S^n/(Kp + S^n)
```

Define parameters, values, and units.

```
p1 = addparameter(k1, 'Vm', 1.0);
```

```
p2 = addparameter(k1, 'n', 1.5);
p3 = addparameter(k1, 'Kp', 2.828);

set(k1, 'ParameterVariableNames', {'Vm','n','Kp'});
set(k1, 'SpeciesVariableNames', {'a'});
set(s1, 'InitialAmount', 2.0);

set(s1, 'InitialAmountUnits', 'mole/liter');
set(s2, 'InitialAmountUnits', 'mole/liter');
set(c1, 'CapacityUnits', 'liter');
set(p1, 'ValueUnits', 'mole/liter/second');
set(p2, 'ValueUnits', 'dimensionless');
set(p3, 'ValueUnits', 'mole/liter');
```

Verify the model.

```
verify(m1)
```

```
Error using SimBiology.Model/verify
--> Error reported from Dimensional Analysis:
Dimensional analysis failed for reaction 'a -> b'.
When using the power function, both the base and exponent must be dimensional
integer constant (for example 2 in 'x^2').
```

You are seeing the error message because SimBiology only allows exponentiation of any dimensionless quantity to any dimensionless power.

Redefine the reaction rate so that it is compatible with dimensional analysis and allows a non-integer exponent.

```
r1.ReactionRate = 'Vm / ( (Kh/a)^n + 1 )';
k1.KineticLaw = 'Unknown';
```

Define the value and units for Kh parameter.

```
p4 = addparameter(k1, 'Kh', 2.0);
set(p4, 'ValueUnits', 'mole/liter');
```

## ReactionRate property

---

Verify the model.

```
verify(m1)
```

You no longer see the error message.

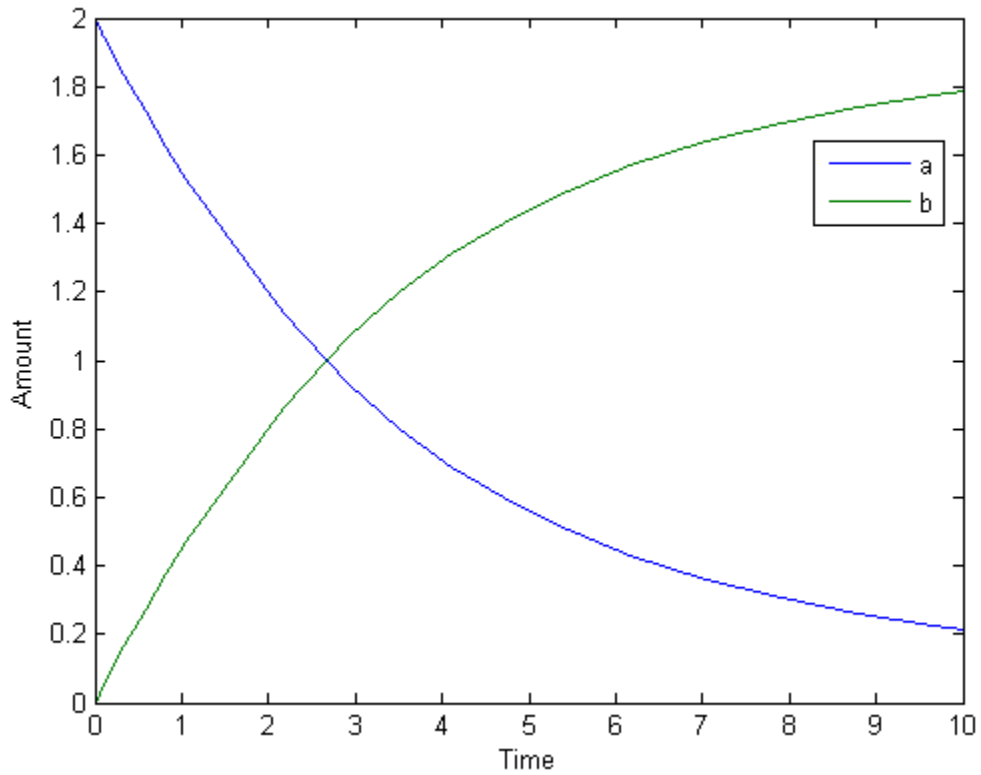
Simulate the model.

```
[t,x,names] = sbiosimulate(m1);
```

Plot the results.

```
plot(t,x);  
xlabel('Time');  
ylabel('Amount');  
legend(names);
```





**See Also** `addparameter`, `addreaction`, `Reversible`

# Reactions property

---

**Purpose** Array of reaction objects

**Description** Property to indicate the reactions in a Model object. Read-only array of reaction objects.

A reaction object defines a chemical reaction that occurs between species. The species for the reaction are defined in the Model object property `Species`.

You can add a reaction to a model object with the method `addreaction`, and you can remove a reaction from the model object with the method `delete`.

## Characteristics

|             |                           |
|-------------|---------------------------|
| Applies to  | Object: model             |
| Data type   | Array of reaction objects |
| Data values | Reaction object           |
| Access      | Read-only                 |

## Examples

**1** Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');  
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

**2** Verify that the reactions property records the input.

```
get (modelObj, 'Reactions')
```

MATLAB returns:

SimBiology Reaction Array

```
Index:   Reaction:  
    1     a + b -> c + d
```

**See Also** `addreaction`, `delete`

**Purpose** Allowable error tolerance relative to state value during a simulation

**Description** `RelativeTolerance` is a property of the `SolverOptions` object, which is a property of a `Configset` object. It is available for the ode solvers (`ode15s`, `ode23t`, `ode45`, and `sundials`).

The `RelativeTolerance` property specifies the allowable error tolerance relative to the state vector at each simulation step. The state vector contains values for all the state variables, for example, amounts for all the species.

If you set the `RelativeTolerance` at `1e-2`, you are specifying that an error of 1% relative to each state value is acceptable at each simulation step.

**Algorithm** At each simulation step, the solver estimates the local error  $e_i$  in the  $i$ th state vector  $y$ . Simulation converges at that time step if  $e_i$  satisfies the following equation:

$$|e_i| \max(\text{RelativeTolerance} * |y_i|, \text{AbsoluteTolerance})$$

Thus at higher state values, convergence is determined by `RelativeTolerance`. As the state values approach zero, convergence is controlled by `AbsoluteTolerance`. The choice of values for `RelativeTolerance` and `AbsoluteTolerance` will vary depending on the problem. The default values should work for first trials of the simulation; however if you want to optimize the solution, consider that there is a trade-off between speed and accuracy. If the simulation takes too long, you can increase the values of `RelativeTolerance` and `AbsoluteTolerance` at the cost of some accuracy. If the results appear to be inaccurate, you can decrease the tolerance values but this will slow down the solver. If the magnitude of the state values is high, you can try to decrease the relative tolerance to get more accurate results.

## Characteristics

|            |                                    |
|------------|------------------------------------|
| Applies to | Object: <code>SolverOptions</code> |
| Data type  | <code>double</code>                |

# RelativeTolerance property

---

|             |  |
|-------------|--|
| Data values | Positive scalar that is <1. Default is 1e-3. |
| Access      | Read/write                                   |

## Examples

This example shows how to change AbsoluteTolerance.

**1** Retrieve the configset object from the modelObj.

```
modelObj = sbiomodel('cell');  
configsetObj = getconfigset(modelObj)
```

**2** Change the AbsoluteTolerance to 1e-8.

```
set(configsetObj.SolverOptions, 'RelativeTolerance', 1.0e-6);  
get(configsetObj.SolverOptions, 'RelativeTolerance')
```

```
ans =
```

```
1.0000e-006
```

## See Also

AbsoluteTolerance

**Purpose** Dose repetitions

**Description** RepeatCount is a property of a RepeatDose object. This property defines the number of doses after the initial dose in a repeat dose series.

---

**Note** When the Interval property is 0, RepeatDose ignores the RepeatCount property, that is, it treats it as though it is set to 0.

---

## Characteristics

|             |                                   |
|-------------|-----------------------------------|
| Applies to  | Object: RepeatDose                |
| Data type   | double                            |
| Data values | Nonnegative integer. Default is 0 |
| Access      | Read/Write                        |

**See Also** ScheduleDose object and RepeatDose object

# Reversible property

---

**Purpose** Specify whether reaction is reversible or irreversible

**Description** The `Reversible` property defines whether a reaction is reversible or irreversible. The rate of the reaction is defined by the `ReactionRate` property. For a reversible reaction, the reaction rate equation is the sum of the rate of the forward and reverse reactions. The type of reaction rate is defined by the `KineticLaw` property. If a reaction is changed from reversible to irreversible or vice versa after `KineticLaw` is assigned, the new `ReactionRate` is determined only if `Type` is `MassAction`. All other `Types` result in unchanged `ReactionRate`. For `MassAction`, the first parameter specified is assumed to be the rate of the forward reaction.

## Characteristics

|             |                                      |
|-------------|--------------------------------------|
| Applies to  | Object: reaction                     |
| Data type   | boolean                              |
| Data values | true, false. Default value is false. |
| Access      | Read/write                           |

## Examples

Create a model, add a reaction, and assign the expression for the reaction rate equation.

- 1 Create model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');  
reactionObj = addreaction(modelObj, 'a -> c + d');
```

- 2 Set the `Reversible` property for the `reactionObj` to true and verify this setting.

```
set (reactionObj, 'Reversible', true)  
get (reactionObj, 'Reversible')
```

MATLAB returns:

```
ans =
```

1

MATLAB returns 1 for true and 0 for false.

In the next steps the example illustrates how the reaction rate equation is assigned for reversible reactions.

- 3 Create a kinetic law object for the reaction object of the type 'MassAction'.

```
kineticlawObj = addkineticlaw(reactionObj, 'MassAction');
```

reactionObj KineticLaw property is configured to kineticlawObj.

- 4 The 'MassAction' kinetic law for reversible reactions has two parameter variables ('Forward Rate Parameter' and 'Reverse Rate Parameter') that you should set. The species variables for MassAction are automatically determined. To set the parameter variables, first create the parameter variables as parameter objects (parameterObj1, parameterObj2) named Kf and Kr and assign the object to kineticlawObj.

```
parameterObj1 = addparameter(kineticlawObj, 'Kf');  
parameterObj2 = addparameter(kineticlawObj, 'Kr');
```

- 5 Set the variable names for the kinetic law object.

```
set(kineticlawObj, 'ParameterVariableNames', {'Kf' 'Kr'});
```

- 6 Verify that the reaction rate is expressed correctly in the reaction object ReactionRate property.

```
get (reactionObj, 'ReactionRate')
```

MATLAB returns:

```
ans =
```

```
Kf*a*b - Kr*c*d
```

## Reversible property

---

### **See Also**

addparameter, addreactant, addreaction, ParameterVariableNames,  
ReactionRate



**Purpose** Specify species and parameter interactions

**Description** The Rule property contains a rule that defines how certain species and parameters should interact with one another. For example, a rule could state that the total number of species A and species B must be some value. Rule is a MATLAB expression that defines the change in the species object quantity or a parameter object Value when the rule is evaluated.

You can add a rule to a model object with the `addrule` method and remove the rule with the `delete` method. For more information on rules, see `addrule` and `RuleType`.

---

**Note** If you set the Rule property for an algebraic rule, rate rule, or repeated assignment rule, and the rule expression is not continuous and differentiable, see “Using Events to Address Discontinuities in Rule and Reaction Rate Expressions” before simulating your model.

---

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: rule   |
| Data type   | char string  |
| Data values | char string defined as species or parameter objects. Default is empty. |
| Access      | Read/write   |

## Examples

**1** Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');  
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

**2** Add a rule.

```
ruleObj = addrule(modelObj, '10-a+b')
```

## Rule property

---

MATLAB returns:

SimBiology Rule Array

| Index: | RuleType: | Rule:  |
|--------|-----------|--------|
| 1      | algebraic | 10-a+b |

### See Also

addrule, delete, “Definitions and Evaluations of Rules”

## Purpose

Specify type of rule for rule object

## Description

The `RuleType` property indicates the type of rule defined by the rule object. A `Rule` object defines how certain species, parameters, and compartments should interact with one another. For example, a rule could state that the total number of species A and species B must be some value. `Rule` is a MATLAB expression that defines the change in the species object quantity or a parameter object `Value` when the rule is evaluated.

You can add a rule to a model object with the `addrule` method and remove the rule with the `delete` method. For more information on rules, see `addrule`.

The types of rules in SimBiology are as follows:

- `initialAssignment` — Lets you specify the initial value of a parameter, species, or compartment capacity, as a function of other model component values in the model.
- `repeatedAssignment` — Lets you specify a value that holds at all times during simulation, and is a function of other model component values in the model.
- `algebraic` — Lets you specify mathematical constraints on one or more parameters, species, or compartments that must hold during a simulation.
- `rate` — Lets you specify the time derivative of a parameter value, species amount, or compartment capacity.

### Constraints on Varying Species Using a Rate Rule

If the model has a species defined in concentration, being varied by a rate rule, and it is in a compartment with varying volume, you can only use `rate` or `initialAssignment` rules to vary the compartment volume.

Conversely, if you are varying a compartment's volume using a `repeatedAssignment` or `algebraic` rules, then you cannot vary a species (defined in concentration) within that compartment, with a rate rule.

# RuleType property

---

The reason for these constraints is that, if a species is defined in concentration and it is in a compartment with varying volume, the time derivative of that species is a function of the compartment's rate of change. For compartments varied by rate rules, the solver has that information.

Note that if you specify the species in amounts there are no constraints.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: rule   |
| Data type   | char string  |
| Data values | 'initialAssignment',<br>'repeatedAssignment' 'algebraic', 'rate'.<br>Default value is 'initialAssignment'. |
| Access      | Read/write   |

## Examples

- 1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');  
reactionObj = addreaction (modelObj, 'a -> b');
```

- 2 Add a rule that specifies the quantity of a species c. In the rule expression, k is the rate constant for a -> b.

```
ruleObj = addrule(modelObj, 'c = k*(a+b)')
```

- 3 Change the RuleType from the default ('algebraic') to 'rate' and verify it using the get command.

```
set(ruleObj, 'RuleType', 'rate');  
get(ruleObj)
```

MATLAB returns all the properties for the rule object.

```
Active: 1  
Annotation: ''  
Name: ''
```

```
Notes: ''
Parent: [1x1 SimBiology.Model]
Rule: 'c = k*(a+b) '
RuleType: 'rate'
Tag: ''
Type: 'rule'
UserData: []
```

### See Also

“Definitions and Evaluations of Rules” in the *SimBiology User’s Guide*,  
addrule, delete

# Rules property

---

**Purpose** Array of rules in model object

**Description** The Rules property shows the rules in a Model object. Read-only array of SimBiology.Rule objects.

A *rule* is a mathematical expression that modifies a species amount or a parameter value. A rule defines how certain species and parameters should interact with one another. For example, a rule could state that the total number of species A and species B must be some value.

You can add a rule to a model object with the `addrule` method and remove the rule with the `delete` method. For more information on rules, see `addrule` and `RuleType`.

## Characteristics

|             |                       |
|-------------|-----------------------|
| Applies to  | Object: model         |
| Data type   | Array of rule objects |
| Data values | Rule object           |
| Access      | Read-only             |

## Examples

**1** Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');  
reactionObj = addreaction (modelObj, 'a + b -> c + d');
```

**2** Add a rule.

```
ruleobj = addrule(modelObj, '10-a+b')
```

MATLAB returns:

SimBiology Rule Array

| Index: | RuleType: | Rule:  |
|--------|-----------|--------|
| 1      | algebraic | 10-a+b |

## See Also

`addrule`, `delete`, “Definitions and Evaluations of Rules”

**Purpose** Information about simulation

**Description** The RunInfo property contains information describing the simulation run that yielded the data in the SimData object.

The following information is stored:

- **Configset** — A struct form of the configuration set used during simulation. This would typically be the model's active configset.
- **Variant** — A struct form of the variant(s) used during simulation.
- **SimulationDate** — The date/time of simulation.
- **SimulationType** — Either 'single run' or 'ensemble run', depending on whether the data object was created using the function sbiosimulate or the function sbioensemblerrun.

## Characteristics

Applies to

Object: SimData

Data type

struct

Data values

Default values are as follows:

```
ConfigSet: []  
SimulationDate: ''  
SimulationType: ''  
Variant: []
```

In practice, the ConfigSet, SimulationDate, and SimulationType fields are rarely empty, since they are populated after simulation.

Access

Read-only

**See Also** StopTime, StopTimeType

# RuntimeOptions property

---

**Purpose** Options for logged species

**Description** The RuntimeOptions property holds options for species that will be logged during the simulation run. The run-time options object can be accessed through this property.

The LogDecimation property of the configuration set object defines how often data is logged.

**Property Summary**

|             |  |
|-------------|--|
| StatesToLog | Specify species, compartment, or parameter data recorded |
| Type        | Display SimBiology object type                           |

## Characteristics

|             |                   |
|-------------|-------------------|
| Applies to  | Object: configset |
| Data type   | Object            |
| Data values | Run-time options  |
| Access      | Read-only         |

## Examples

**1** Create a model object, and retrieve its configuration set.

```
modelObj = sbiomodel('cell');  
configsetObj = getconfigset(modelObj);
```

**2** Retrieve the RuntimeOptions object from the configset object.

```
runtimeObj = get(configsetObj, 'RunTimeOptions')  
Runtime Settings:
```

```
StatesToLog: all
```

**See Also** get, set



## Purpose

Enable or disable sensitivity analysis

## Description

`SensitivityAnalysis` is a property of the `SolverOptions` property, which is a property of a `configset` object. This property lets you compute the time-dependent sensitivities of all the species states defined by the `StatesToLog` property with respect to the `Inputs` that you specify in the `SensitivityAnalysisOptions` property of the configuration set object.

`SensitivityAnalysis` is available only for the ODE solvers (`ode15s`, `ode23t`, `ode45`, and `sundials`)

---

**Note** Models containing the following active components do not support sensitivity analysis:

- Nonconstant compartments
- Algebraic rules
- Events

---

For more information on setting up sensitivity analysis, see `SensitivityAnalysisOptions` . For a description of sensitivity analysis calculations, see “Sensitivity Calculation”.

## Characteristics

|             |                                      |
|-------------|--------------------------------------|
| Applies to  | Object: <code>SolverOptions</code>   |
| Data type   | logical                              |
| Data values | 1, 0, true, false. Default is false. |
| Access      | Read/write                           |

# SensitivityAnalysis property

---

## Examples

This example shows how to enable `SensitivityAnalysis`.

- 1 Retrieve the `configset` object from the `modelObj`.

```
modelObj = sbiomodel('cell');  
configsetObj = getconfigset(modelObj);
```

- 2 Enable `SensitivityAnalysis`.

```
set(configsetObj.SolverOptions, 'SensitivityAnalysis', true);  
get(configsetObj.SolverOptions, 'SensitivityAnalysis')
```

```
ans =
```

```
on
```

## See Also

`SensitivityAnalysisOptions`, `SolverOptions`, `SolverType`,  
`StatesToLog`

# SensitivityAnalysisOptions property

---

**Purpose** Specify sensitivity analysis options

**Description** The SensitivityAnalysisOptions property is an object that holds the sensitivity analysis options in the configuration set object. Sensitivity analysis is supported only for deterministic (ODE) simulations.

---

**Note** The SensitivityAnalysisOptions property controls the settings related to sensitivity analysis. To enable or disable sensitivity analysis, use the SensitivityAnalysis property.

---

Properties of SensitivityAnalysisOptions are summarized in “Property Summary” on page 3-172.

When sensitivity analysis is enabled, the following command

```
[t,x,names] = sbiosimulate(modelObj)
```

returns [t,x,names], where

- **t** is an  $n$ -by-1 vector, where  $n$  is the number of steps taken by the ode solver and **t** defines the time steps of the solver.
- **x** is an  $n$ -by- $m$  matrix, where  $n$  is the number of steps taken by the ode solver and  $m$  is:

Number of species and parameters specified in StatesToLog +  
(Number of sensitivity outputs \* Number of sensitivity input factors)

A SimBiology state includes species and nonconstant parameters.

- **names** is the list of states logged and the list of sensitivities of the species specified in StatesToLog with respect to the input factors.

For an example of the output, see “Examples” on page 3-172.

You can add a number of configuration set objects with different SensitivityAnalysisOptions to the model object with the

# SensitivityAnalysisOptions property

---

addconfigset method. Only one configuration set object in the model object can have the Active property set to true at any given time.

## Property Summary

|               |  |
|---------------|--|
| Inputs        | Specify species and parameter input factors for sensitivity analysis |
| Normalization | Specify normalization type for sensitivity analysis                  |
| Outputs       | Specify species and parameter outputs for sensitivity analysis       |

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: configuration set  |
| Data type   | Object   |
| Data values | SensitivityAnalysisOptions properties as summarized in “Property Summary” on page 3-172. |
| Access      | Read-only  |

## Examples

This example shows how to set SensitivityAnalysisOptions.

- 1 Import the radio decay model from SimBiology demos.

```
modelObj = sbmlimport('radiodecay');
```

- 2 Retrieve the configuration settings and the sensitivity analysis options from modelObj.

```
configsetObj = getconfigset(modelObj);  
sensitivityObj = get(configsetObj, 'SensitivityAnalysisOptions');
```

# SensitivityAnalysisOptions property

---

**3** Add a species and a parameter to the Inputs property. Use the `sbioselect` function to retrieve the species and parameter objects from the model.

**4** Add a species to the Outputs property and display.

SimBiology Species Array

| Index: | Compartment: | Name: | InitialAmount: | InitialAmount: |
|--------|--------------|-------|----------------|----------------|
| 1      | unnamed      | z     | 0              | molecule       |

**5** Enable `SensitivityAnalysis`.

```
set(configsetObj.SolverOptions, 'SensitivityAnalysis', true);  
get(configsetObj.SolverOptions, 'SensitivityAnalysis')
```

```
ans =
```

```
1
```

**6** Simulate and return the results to three output variables. See “Description” on page 3-171 for more information.

```
[t,x,names] = sbiosimulate(modelObj);
```

**7** Display the names.

```
names
```

```
names =
```

```
'x'  
'z'  
'd[z]/d[z]_0'  
'd[z]/d[Reaction1.c]'
```

Display state values `x`.

```
x
```

## SensitivityAnalysisOptions property

---

The display follows the column order shown in `names` for the values in `x`. The rows correspond to `t`.

**See Also** `addconfigset`, `getconfigset`, `SensitivityAnalysis`

**Purpose** Specify model solver options

**Description** The SolverOptions property is an object that holds the model solver options in the configset object. Changing the property SolverType changes the options specified in the SolverOptions object.

Properties of SolverOptions are summarized in “Property Summary” on page 3-175.

## Property Summary

|                           |  |
|---------------------------|--|
| AbsoluteTolerance         | Absolute error tolerance applied to state value during simulation        |
| AbsoluteToleranceScaling  | Control scaling of absolute error tolerance during simulation            |
| AbsoluteToleranceStepSize | Initial guess for time step size for scaling of absolute error tolerance |
| ErrorTolerance            | Specify explicit or implicit tau error tolerance                         |
| LogDecimation             | Specify frequency to log stochastic simulation output                    |
| MaxIterations             | Specify nonlinear solver maximum iterations in implicit tau              |
| MaxStep                   | Specify upper bound on ODE solver step size                              |
| OutputTimes               | Specify times to log deterministic simulation output                     |
| RandomState               | Set random number generator  |
| RelativeTolerance         | Allowable error tolerance relative to state value during a simulation    |

# SolverOptions property

---

|                     |  |
|---------------------|--|
| SensitivityAnalysis | Enable or disable sensitivity analysis |
| Type                | Display SimBiology object type         |

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Object: configset   |
| Data type   | Object  |
| Data values | Solver options depending on SolverType. Default is SolverOptions for default SolverType (ode15s). |
| Access      | Read-only   |

## Examples

This example shows the changes in SolverOptions for various SolverType settings.

- 1 Retrieve the configset object from the modelObj.

```
modelObj = sbiomodel('cell');  
configsetObj = getconfigset(modelObj);
```

- 2 Configure the SolverType to ode45.

```
set(configsetObj, 'SolverType', 'ode45');  
get(configsetObj, 'SolverOptions')
```

```
Solver Settings: (ode)
```

```
AbsoluteTolerance: 1.000000e-006  
RelativeTolerance: 1.000000e-003
```

- 3 Configure the SolverType to ssa.

```
set(configsetObj, 'SolverType', 'ssa');  
get(configsetObj, 'SolverOptions')
```



Solver Settings: (ssa)

```
LogDecimation:      1
RandomState:        []
```

#### 4 Configure the SolverType to impltau.

```
set(configsetObj, 'SolverType', 'impltau');
get(configsetObj, 'SolverOptions')
```

Solver Settings: (impltau)

```
ErrorTolerance:     3.000000e-002
LogDecimation:      1
AbsoluteTolerance:  1.000000e-002
RelativeTolerance:  1.000000e-002
MaxIterations:      15
RandomState:        []
```

#### 5 Configure the SolverType to expltau.

```
set(configsetObj, 'SolverType', 'expltau');
get(configsetObj, 'SolverOptions')
```

Solver Settings: (expltau)

```
ErrorTolerance:     3.000000e-002
LogDecimation:      1
RandomState:        []
```

### See Also

`addconfigset`, `getconfigset`

# SolverType property

---

**Purpose** Select solver type for simulation

**Description** The SolverType property lets you specify the solver to use for a simulation. For a discussion about solver types, see “Choosing a Simulation Solver”.

Changing the solver type changes the options (properties) specified in the SolverOptions property of the configset object. If you change any SolverOptions, these changes are persistent when you switch SolverType. For example, if you set the ErrorTolerance for the expltau solver and then change to impltau when you switch back to expltau, the ErrorTolerance will have the value you assigned.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: Configset  |
| Data type   | enum   |
| Data values | 'ode15s', 'ode23t', 'ode45', 'sundials', 'ssa', 'expltau', 'impltau'. Default is 'ode15s'. |

---

**Note** If your model contains events, you cannot specify 'expltau' or 'impltau' for the SolverType property.

---

---

**Note** If your model contains doses, you cannot specify 'ssa', 'expltau', or 'impltau' for the SolverType property.

---

|        |            |
|--------|------------|
| Access | Read/write |
|--------|------------|

## Examples

1 Retrieve the configset object from the modelObj.

```
modelObj = sbiomodel('cell');
```

```
configsetObj = getconfigset(modelObj)
```

```
Configuration Settings - default (active)
```

```
SolverType:           ode15s  
StopTime:             10.000000
```

```
SolverOptions:  
AbsoluteTolerance:    1.000000e-006  
RelativeTolerance:    1.000000e-003  
SensitivityAnalysis:  false
```

```
RuntimeOptions:  
StatesToLog:          all
```

```
CompileOptions:  
UnitConversion:       false  
DimensionalAnalysis:  true
```

```
SensitivityAnalysisOptions:  
Inputs:               0  
Outputs:              0
```

## 2 Configure the SolverType to ode45.

```
set(configsetObj, 'SolverType', 'ode45')  
configsetObj
```

```
Configuration Settings - default (active)
```

```
SolverType:           ode45  
StopTime:             10.000000
```

```
SolverOptions:  
AbsoluteTolerance:    1.000000e-006  
RelativeTolerance:    1.000000e-003  
SensitivityAnalysis:  false
```

## SolverType property

---

```
RuntimeOptions:
  StatesToLog:          all

CompileOptions:
  UnitConversion:      false
  DimensionalAnalysis: true

SensitivityAnalysisOptions:
  Inputs:              0
  Outputs:             0
```

**See Also**      `getConfigset`, `set`

**Purpose** Array of species in compartment object

**Description** The `Species` property is a property of the compartment object and indicates all the species in a compartment object. `Species` is a read-only array of SimBiology species objects.

In the model object, `Species` contains a flat list of all the species that exist within all the compartments in the model. You should always access a species through its compartment rather than the model object. Use the format `compartmentName.speciesName`, for example, `nucleus.DNA`. Another example of the syntax is `modelObj.Compartments(2).Species(1)`. The `Species` property in the model object might not be available in a future version of the software.

Species are entities that take part in reactions. A species object is added to the `Species` property when a reaction is added to the model object with the method `addreaction`. A species object can also be added to the `Species` property with the method `addspecies`.

If you remove a reaction with the method `delete`, and a species is no longer being used by any of the remaining reactions, the species object is *not* removed from the `Species` property. You have to use the `delete` method to remove species.

There are reserved characters that cannot be used in species object names. See `Name` for more information.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Object: compartment                     |
| Data type   | Array of species objects                |
| Data values | Species object. Default is [ ] (empty). |
| Access      | Read-only                               |

**See Also** `addcompartment`, `addreaction`, `addspecies`, `delete`

# SpeciesVariableNames property

---

**Purpose** Cell array of species in reaction rate equation

**Description** The SpeciesVariableNames property shows the species used by the kinetic law object to determine the ReactionRate equation in the reaction object. Use setspecies to assign SpeciesVariableNames. When you assign species to SpeciesVariableNames, SimBiology software maps these species names to SpeciesVariables in the kinetic law object.

The ReactionRate property of a reaction object shows the result of a mapping from kinetic law definition. The ReactionRate is determined by the kinetic law object Expression property by mapping ParameterVariableNames to ParameterVariables and SpeciesVariableNames to SpeciesVariables.

## Characteristics

|             |                             |
|-------------|-----------------------------|
| Applies to  | Object: kinetic law         |
| Data type   | Cell array of strings       |
| Data values | Cell array of species names |
| Access      | Read/write                  |

## Examples

Create a model, add a reaction, and assign the SpeciesVariableNames for the reaction rate equation.

- 1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');  
reactionObj = addreaction(modelObj, 'a -> c + d');
```

- 2 Create a kinetic law object for the reaction object, of the type 'Henri-Michaelis-Menten'

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

The reactionObj KineticLaw property is configured to kineticlawObj.

## SpeciesVariableNames property

---

- 3** The 'Henri-Michaelis-Menten' kinetic law has one species variable (S) that you should set. To set this variable:

```
setspecies(kineticlawObj, 'S', 'a');
```

- 4** Verify that the species variable is correct.

```
get (kineticlawObj, 'SpeciesVariableNames')
```

MATLAB returns:

```
ans =
```

```
'a'
```

### See Also

Expression(AbstractKineticLaw, KineticLaw),  
ParameterVariables, ParameterVariableNames, ReactionRate,  
setParameter, SpeciesVariables

# SpeciesVariables property

---

**Purpose** Species in abstract kinetic law

**Description** This property shows species variables that are used in the Expression property of the kinetic law object to determine the ReactionRate equation in the reaction object. Use the MATLAB function `set` to assign `SpeciesVariables` to an abstract kinetic law. For more information, see abstract kinetic law.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Objects: abstract kinetic law, kineticlaw                     |
| Data type   | Cell array of strings   |
| Data values | Defined by abstract kinetic law                               |
| Access      | Read/write in abstract kinetic law. Read-only in kinetic law. |

## Examples

Create a model, add a reaction, and assign the `SpeciesVariableNames` for the reaction rate equation.

- 1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');  
reactionObj = addreaction(modelObj, 'a -> c + d');
```

- 2 Create a kinetic law object for the reaction object, of the type 'Henri-Michaelis-Menten'.

```
kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');
```

reactionObj.KineticLaw property is configured to kineticlawObj.

- 3 View the species variable for 'Henri-Michaelis-Menten' kinetic law.

```
get(kineticlawObj, 'SpeciesVariables')
```

MATLAB returns:



```
ans =  
  
'S'
```

### See Also

Expression(AbstractKineticLaw, KineticLaw),  
ParameterVariables, ParameterVariableNames, ReactionRate, set,  
setparameter, SpeciesVariableNames

# StartTime property

---

**Purpose** Start time for initial dose time

**Description** StartTime is a property of a RepeatDose object. For a series of repeated doses, the StartTime property defines the amount of time that elapses before the first (initial) dose is given.

**Characteristics**

|             |   |
|-------------|---|
| Applies to  | Objects: RepeatDose                         |
| Data type   | double                                      |
| Data values | Nonnegative real number. Default value is 0 |
| Access      | Read-write                                  |

**See Also** RepeatDose object

**Purpose** Specify species, compartment, or parameter data recorded

**Description** The StatesToLog property specifies the species, compartment, or parameter data to log during a simulation. This is the data returned in `x` during execution of `[t,x] = sbiosimulate(modelObj)`. By default, all species, nonconstant compartments, and nonconstant parameters are logged.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Object: RuntimeOptions  |
| Data type   | Object or vector of objects   |
| Data values | Species objects, compartment objects, or parameter objects. Default is <code>all</code> , which means all species objects, all compartment objects whose <code>ConstantCapacity = false</code> , and all parameter objects whose <code>ConstantValue = false</code> . |
| Access      | Read/write  |

## Examples

This example shows how to assign species to StatesToLog.

- 1 Create a model object by importing the file `oscillator.xml`.

```
modelObj = sbmlimport('oscillator');
```

- 2 Retrieve the first and second species in `modelObj`.

```
speciesObj1 = modelObj.Species(1);  
speciesObj2 = modelObj.Species(2);
```

- 3 Retrieve the `configsetObj` of `modelObj`.

```
configsetObj = getconfigset(modelObj);
```

- 4 Set the StatesToLog to record three species: two using the retrieved species objects and one using indexing and view the species in StatesToLog.

## StatesToLog property

---

```
set (configsetObj.RuntimeOptions, 'StatesToLog', ...  
    [speciesObj1, speciesObj2, modelObj.Species(3)]);  
get(configsetObj.RuntimeOptions, 'StatesToLog')
```

- 5 Set the StatesToLog property back to the default setting of all.

```
set (configsetObj.RuntimeOptions, 'StatesToLog', 'all');
```

## Purpose

Species coefficients in reaction

## Description

The **Stoichiometry** property specifies the species coefficients in a reaction. Enter an array of **doubles** indicating the stoichiometry of reactants (negative value) and products (positive value). Example: [-1 -1 2].

The **double** specified cannot be 0. The reactants of the reaction are defined with a negative number. The products of the reaction are defined with a positive number. For example, the reaction  $3\text{H} + \text{A} \rightarrow 2\text{C} + \text{F}$  has the **Stoichiometry** value of [-3 -1 2 1].

When this property is configured, the **Reaction** property updates accordingly. In the above example, if the **Stoichiometry** value was set to [-2 -1 2 3], the reaction is updated to  $2\text{H} + \text{A} \rightarrow 2\text{C} + 3\text{F}$ .

The length of the **Stoichiometry** array is the sum of the **Reactants** array and the **Products** array. To remove a product or reactant from a reaction, use the **rmproduct** or **rmreactant** function. Add a product or reactant and set stoichiometry with methods **addproduct** and **addreactant**.

ODE solvers support **double** stoichiometry values such as 0.5. Stochastic solvers and dimensional analysis currently support only integers in **Stoichiometry**, therefore you must balance the reaction equation and specify integer values for these two cases.

$\text{A} \rightarrow \text{null}$  has a stoichiometry value of [- 1].  $\text{null} \rightarrow \text{B}$  has a stoichiometry value of [ 1].

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Object: reaction  |
| Data type   | Double array  |
| Data values | 1-by-n double, where n is length (products) + length (reactants). Default is [ ] (empty). |
| Access      | Read/write  |

# Stoichiometry property

---

## Examples

- 1 Create a reaction object.

```
modelObj = sbiomodel('cell');  
reactionObj = addreaction(modelObj, '2 a + 3 b -> d + 2 c');
```

- 2 Verify the Reaction and Stoichiometry properties for reactionObj.

```
get(reactionObj, 'Stoichiometry')
```

MATLAB returns:

```
ans =  
  
-2    -3     1     2
```

- 3 Set stoichiometry to [-1 -2 2 2].

```
set (reactionObj, 'Stoichiometry', [-1 -2 2 2]);  
get (reactionObj, 'Stoichiometry')
```

MATLAB returns:

```
ans =  
  
-1    -2     2     2
```

- 4 Note with get that the Reaction property updates automatically.

```
get (reactionObj, 'Reaction')
```

MATLAB returns:

```
ans =  
  
a + 2 b -> 2 d + 2 c
```

## See Also

addproduct, addreactant, addreaction, Reaction, rmproduct, rmreactant

**Purpose** Simulation time criteria to stop simulation

**Description** StopTime is a property of a Configset object. This property sets the maximum simulation time criteria to stop a simulation. Time units are specified by the TimeUnits property of the Configset object.

A simulation stops when it meets any of the criteria specified by StopTime, MaximumNumberOfLogs, or MaximumWallClock. However, if you specify the OutputTimes property of the SolverOptions property of the Configset object, then StopTime and MaximumNumberOfLogs are ignored. Instead, the last value in OutputTimes is used as the StopTime criteria, and the length of OutputTimes is used as the MaximumNumberOfLogs criteria.

|                        |             |                                    |
|------------------------|-------------|------------------------------------|
| <b>Characteristics</b> | Applies to  | Object: Configset                  |
|                        | Data type   | double                             |
|                        | Data values | Nonnegative scalar. Default is 10. |
|                        | Access      | Read/write                         |

## Examples **Set Simulation Time Criteria to Stop Simulation**

- 1 Create a model object named cell and save it in a variable named modelObj. Retrieve the configuration set from modelObj and save it in a variable named configsetObj.

```
modelObj = sbiomodel('cell');  
configsetObj = getconfigset(modelObj);
```

- 2 Configure the simulation stop criteria by setting the StopTime property to 20 seconds. Leave the MaximumNumberOfLogs and MaximumWallClock properties at their default values of Inf.

```
set(configsetObj, 'StopTime', 20)  
get(configsetObj)
```

## StopTime property

---

```
Active: 1
CompileOptions: [1x1 SimBiology.CompileOptions]
    Name: 'default'
    Notes: ''
RuntimeOptions: [1x1 SimBiology.RuntimeOptions]
SensitivityAnalysisOptions: [1x1 SimBiology.SensitivityAnalysisOptions]
SolverOptions: [1x1 SimBiology.ODESolverOptions]
    SolverType: 'ode15s'
    StopTime: 20
MaximumNumberOfLogs: Inf
MaximumWallClock: Inf
TimeUnits: 'second'
Type: 'configset'
```

When you simulate `modelObj`, the simulation stops when the simulation time reaches 20 seconds.

### See Also

`Configset` object, `MaximumNumberOfLogs`, `MaximumWallClock`, `OutputTimes`, `TimeUnits`



**Purpose** Specify type of stop time for simulation

---

**Note** StopTimeType will be removed in a future release. Use StopTime,MaximumNumberOfLogs, MaximumWallClock and OutputTimes instead.

---

**Description** StopTimeType is a property of a Configset object. The StopTimeType property sets the type of stop time for a simulation. The stop time is specified in the StopTime property of the configset object. Valid types are approxWallTime, numberOfLogs, and simulationTime. The default is simulationTime.

- `simulationTime` — Specify the stop time for the simulation. The solver determines and sets the time steps and the simulation stops when it reaches the specified StopTime.
- `approxWallTime` — Specify the approximate stop time according to the clock. For example, 10s of `approxWallTime` is approximately 10s of real time.
- `numberOfLogs` — Specify the total number of simulation steps to be recorded during the simulation. For example if you want to log three simulation steps, the `numberOfLogs` is 3. The simulation will stop after the specified `numberOfLogs`.

You can change the StopTimeType setting with the set function.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Object: Configset   |
| Data type   | enum  |
| Data values | <code>approxWallTime</code> , <code>numberOfLogs</code> , and <code>simulationTime</code> |
| Access      | Read/write  |

## StopTimeType property

---

### **See Also**

set, StatesToLog, StopTime, TimeUnits

**Purpose** Specify label for SimBiology object

**Description** The Tag property specifies a label associated with a SimBiology object. Use this property to group objects and then use `sbiobject` to retrieve. For example, use the Tag property in reaction objects to group synthesis or degradation reactions. You can then retrieve all synthesis reactions using `sbiobject`. Similarly, for species objects you can enter and store classification information, for example, membrane protein, transcription factor, enzyme classifications, or whether a species is an independent variable. You can also enter the full form of the name of the species.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Objects: abstract kinetic law, kinetic law, model, parameter, reaction, RepeatDose, rule, ScheduleDose, species |
| Data type   | char string   |
| Data values | Any char string   |
| Access      | Read/write  |

## Examples

**1** Create a model object.

```
modelObj = sbiomodel ('my_model');
```

**2** Add a reaction object and set the Tag property to 'Synthesis Reaction'.

```
reactionObj = addreaction (modelObj, 'a + b -> c + d');  
set (reactionObj, 'Tag', 'Synthesis Reaction')
```

**3** Verify the Tag assignment.

```
get (reactionObj, 'Tag');
```

MATLAB returns:

```
ans =
```

## Tag property

---

'Synthesis Reaction'

### **See Also**

addkineticlaw, addparameter, addreaction, addrule, addspecies,  
RepeatDose object, sbioabstractkineticlaw, sbiomodel, sbioroot,  
ScheduleDose object

**Purpose** Species receiving dose

**Description** TargetName is a property of a RepeatDose or ScheduleDose object. This property defines the SimBiology species receiving the dose. The dose amount increases the species amount at each time interval defined by a repeat dose or at each time point defined by a schedule dose.

The value of TargetName is the name of a species. If the model has more than one species with the same name, TargetName is defined as *compartmentName.speciesName*, where *compartmentName* is the name of the compartment containing the species.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Objects: RepeatDose, ScheduleDose          |
| Data type   | string                                     |
| Data values | Species name. Default value is "" (empty). |
| Access      | Read/Write                                 |

**See Also** ScheduleDose object and RepeatDose object

# Trigger property

---

**Purpose** Event trigger

**Description** Trigger is a property of an Event object

A Trigger is a condition that must become true for an event to execute. You can use a combination of relational and logical operators to build a trigger expression. Trigger can be a string, an expression, or a function handle that when evaluated returns a value of `true` or `false`. A Trigger can access species, parameters, and compartments.

A trigger can contain the keyword `time` and relational operators to trigger an event that occurs at a specific time during the simulation. For example, `time >= x`. In this example trigger, note that:

- The units associated with the keyword `time` are the units for the `TimeUnits` property for the `Configset` object associated with the simulation.
- If `x` is an expression containing compartments, species, or parameters, then any units associated with the expression must have the same dimensions as the keyword `time`.
- If `x` is a raw number, then its dimensions (and units, if unit conversion is on) are assumed to be the same as the keyword `time`.

For more information about how the SimBiology software handles events, see “How Events Are Evaluated”. For examples of event functions, see “Specifying Event Triggers”.

|                        |                   |  |
|------------------------|-------------------|--|
| <b>Characteristics</b> | Applies to        | Object: event  |
|                        | SimBiology type   | String, function handle  |
|                        | SimBiology values | Specify a MATLAB expression as a string. Default is '' (empty string). |
|                        | Access            | Read/write   |

## Examples

- 1 Create a model object, and then add an event object.

```
modelObj = sbmlimport('oscillator');  
eventObj = addevent(modelObj, 'time>= 5', 'OpC = 200');
```

- 2 Set the Trigger property of the event object.

```
set(eventObj, 'Trigger', '(time >=5) && (speciesA<1000)');
```

- 3 Get the Trigger property.

```
get(eventObj, 'Trigger')
```

## See Also

Event object, EventFcns

# Time property

---

**Purpose** Simulation time steps or schedule dose times

**Description** Time is a property of a SimData or ScheduleDose object.

## **SimData Object**

For a simulation, the Time property records the time steps.

## **ScheduleDose Object**

For a series of scheduled doses, the Time property defines the times to give a dose.

A ScheduleDose object defines a series of doses. Each dose can have a different amount, as defined by an amount array in the Amount property, and given at specified times, as defined by a time array in the Time property. A rate array in the Rate property defines how fast each dose is given. At each time point in the time array, a dose is given with the corresponding amount and rate.

## **Characteristics**

|             |  |
|-------------|--|
| Applies to  | Objects: SimData, ScheduleDose   |
| Data type   | double (SimData), double array (ScheduleDose)  |
| Data values | Vector of doubles (SimData)Array of nonnegative real numbers. Default value is [] (ScheduleDose) |
| Access      | Read-only  |

**See Also** ScheduleDose object, SimData object, StopTime, StopTimeType



**Purpose** Show time units for dosing and simulation

**Description** The TimeUnits property specifies time units for these properties:

- StopTime property of a Configset object
- OutputTimes and AbsoluteToleranceStepSize properties of the SolverOptions property of a Configset object
- StartTime and Interval properties of a RepeatDose object
- Time property of a ScheduleDose object
- Time property of a SimData object

---

**Note** If you change the value of the TimeUnits property, make sure:

- You update the values of the Time, StartTime, Interval, StopTime, and OutputTimes properties accordingly.
  - You update raw numbers used in any event triggers that use the keyword `time` accordingly. For more information, see [Trigger](#).
  - The units, if any, associated with expressions used in any event triggers that use the keyword `time`, are consistent with the updated TimeUnits property. For more information, see [Trigger](#).
- 

**Characteristics**

|            |   |
|------------|---|
| Applies to | Objects: Configset, RepeatDose, ScheduleDose, SimData |
| Data type  | string  |

## TimeUnits property

---

|             |   |
|-------------|---|
| Data values | <p>Empty string or a string specifying any unit defined in the Units Library.</p> <p>Default value is:</p> <ul style="list-style-type: none"><li>• <code>second</code> — properties of a <code>Configset</code> object or <code>SimData</code> object for a <code>model</code> object created using <code>sbiomodel</code></li><li>• <code>hour</code> — properties of a <code>Configset</code> object or <code>SimData</code> object for a <code>model</code> object created from a <code>PKModelDesign</code> object</li><li>• <code>''</code> (empty string) — properties of <code>RepeatDose</code> and <code>ScheduleDose</code> objects</li></ul> |
| Access      | <p>Read/write for properties of <code>Configset</code>, <code>RepeatDose</code>, and <code>ScheduleDose</code> objects</p> <p>Read only for properties of <code>SimData</code> objects</p>  |

### See Also

`Configset` object, `RepeatDose` object, `ScheduleDose` object, `SimData` object, `Interval`, `OutputTimes`, `StartTime`, `StopTime`, `Time`

**Purpose** Display SimBiology object type

**Description** The Type property indicates a SimBiology object type. When you create a SimBiology object, the value of Type is automatically defined.

For example, when a Species object is created, the value of the Type property is automatically defined as 'species'.

**Characteristics**

|             |  |
|-------------|--|
| Applies to  | Objects: abstract kinetic law, compartment, configuration set, CompileOptions, event, kinetic law, model, parameter, reaction, RepeatDose, root, rule, ScheduleDose, species, RuntimeOptions, SolverOptions, unit, unitprefix, and variant |
| Data type   | char string  |
| Data values | abstract_kinetic_law, compartment, configset, compileoptions, event, kineticlaw, parameter, reaction, repeatdose, root, rule, runtimeoptions, sbiomodel, scheduledose, species, solveroptions, unit, unitprefix, and variant               |
| Access      | Read-only  |

**See Also** RepeatDose object, sbiomodel, sbioroot, ScheduleDose object, setactiveconfigset

# UnitConversion property

---

**Purpose** Perform unit conversion

**Description** The `UnitConversion` property specifies whether to perform unit conversion for the model before simulation. It is a property of the `CompileOptions` object. `CompileOptions` holds the model's compile time options and is the object property of the `configset` object.

When `UnitConversion` is set to `true`, the SimBiology software converts the matching physical quantities to one consistent unit system in order to resolve them. This conversion is in preparation for correct simulation, but species amounts are returned in the user-specified units.

For example, consider a reaction  $a + b \rightarrow c$ . Using mass action kinetics the reaction rate is defined as  $a \cdot b \cdot k$  where  $k$  is the rate constant of the reaction. If you specify that initial amounts of  $a$  and  $b$  are 0.01M and 0.005M respectively, then units of  $k$  are  $1 / (M \cdot \text{second})$ . If you specify  $k$  with another equivalent unit definition, for example,  $1 / ((\text{molecules/liter}) \cdot \text{second})$ , `UnitConversion` occurs after `DimensionalAnalysis`.

Unit conversion requires dimensional analysis. If `DimensionalAnalysis` is off, and you turn `UnitConversion` on, then `DimensionalAnalysis` is turned on automatically. If `UnitConversion` is on and you turn off `DimensionalAnalysis`, then `UnitConversion` is turned off automatically.

If `UnitConversion` fails, then you see an error when you simulate (`sbiosimulate`).

If `UnitConversion` is set to `false`, the simulation uses the given object values.

Unit conversion involving temperature supports `Celsius` as the temperature unit. Avoid using mixed temperature units as you might get an error.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: CompileOptions (in configset object) |
| Data type   | boolean                                      |
| Data values | true or false. Default value is false.       |
| Access      | Read/write                                   |

---

**Note** SimBiology uses unit strings including empty units in association with DimensionalAnalysis and UnitConversion features.

- When DimensionalAnalysis and UnitConversion are both false, unit strings are not used. However, SimBiology still performs a minimum level of dimensional analysis to decide whether a reaction rate is in dimensions of amount/time or concentration/time.
- When DimensionalAnalysis is true and UnitConversion is false, units (if not empty) must have consistent dimensions so that SimBiology can perform dimensional analysis. However, the units are not converted.
- When UnitConversion is set to true (which requires DimensionalAnalysis to be true), SimBiology performs a dimensional analysis and converts everything to consistent units. Hence, you must specify consistent units, and no unit strings can be empty. If you have a dimensionless parameter, you must still set its unit string to dimensionless.

---

## Examples

This example shows how to retrieve and set unitconversion from the default true to false in the default configuration set in a model object.

1 Import a model.

```
modelObj = sbmlimport('oscillator')
```

# UnitConversion property

---

SimBiology Model - Oscillator

```
Model Components:
  Models:          0
  Parameters:      0
  Reactions:       42
  Rules:           0
  Species:         23
```

- 2 Retrieve the configset object of the model object.

```
configsetObj = getConfigset(modelObj)
```

Configuration Settings - default (active)

```
SolverType:      ode15s
StopTime:        10.000000
```

```
SolverOptions:
  AbsoluteTolerance:  1.000000e-006
  RelativeTolerance:  1.000000e-003
```

```
RuntimeOptions:
  StatesToLog:        all
```

```
CompileOptions:
  UnitConversion:     false
  DimensionalAnalysis: true
```

- 3 Retrieve the CompileOptions object.

```
optionsObj = get(configsetObj, 'CompileOptions')
```

Compile Settings:

```
UnitConversion:    false
DimensionalAnalysis: true
```

- 4 Assign a value of false to UnitConversion.

```
set(optionsObj, 'UnitConversion', true)
```

**See Also**

get, getconfigset, sbiosimulate, set

# UserData property

---

**Purpose** Specify data to associate with object

**Description** Property to specify data that you want to associate with a SimBiology object. The object does not use this data directly, but you can access it using the function `get` or dot notation.

## Characteristics

|             |   |
|-------------|---|
| Applies to  | Objects: abstract kinetic law, configuration set, compartment, data, event, kinetic law, model, parameter, reaction, RepeatDose, rule, ScheduleDose, species, or unit |
| Data type   | Any   |
| Data values | Any. Default is empty.  |
| Access      | Read/write  |

**See Also** RepeatDose object, sbioabstractkineticlaw, sbiomodel, sbioroot, sbiounit, sbiounitprefix, ScheduleDose object



## Purpose

Library of user-defined components

## Description

UserDefinedLibrary is a SimBiology root object property containing all user-defined components of unit, unit prefixes, and kinetic laws that you define. You can add, modify, or delete components in the user-defined library. The UserDefinedLibrary property is an object that contains the following properties:

- **Units** — Contains any user-defined units. You can specify units for compartment capacity, species amounts and parameter values, to do dimensional analysis and unit conversion during simulation. You can display the user-defined units either by using the command `sbiowhos -userdefined -unit`, or by accessing the root object.
- **UnitPrefixes** — Contains any user-defined unit prefixes. You can specify unit prefixes in combination with a valid unit for compartment capacity, species amounts and parameter values, to do dimensional analysis and unit conversion during simulation. You can display the user-defined unit prefixes either by using the command `sbiowhos -userdefined -unitprefix`, or by accessing the root object.
- **KineticLaws** — Contains any user-defined kinetic laws. Use the command `sbiowhos -userdefined -kineticlaw` to see the list of user-defined kinetic laws. You can use user-defined kinetic laws when you use the command `addkineticlaw` to create a kinetic law object for a reaction object. Refer to the kinetic law by name when you create the kinetic law object, for example, `kineticlawObj = addkineticlaw(reactionObj, 'Henri-Michaelis-Menten');`

See “Kinetic Law Definition” on page 3-71 for a definition and more information.

## Characteristics

|            |              |
|------------|--------------|
| Applies to | Object: root |
| Data type  | object       |

# UserDefinedLibrary property

---

|             |   |
|-------------|---|
| Data values | Unit, unit prefix, and abstract kinetic law objects |
| Access      | Read-only   |

Characteristics for UserDefinedLibrary properties:

- **Units**

|             |                             |
|-------------|-----------------------------|
| Applies to  | UserDefinedLibrary property |
| Data type   | Unit objects                |
| Data values | Units                       |
| Access      | Read/write                  |

- **UnitPrefixes**

|             |                             |
|-------------|-----------------------------|
| Applies to  | UserDefinedLibrary property |
| Data type   | Unit prefix objects         |
| Data values | Unit prefixes               |
| Access      | Read/write                  |

- **KineticLaws**

|             |                             |
|-------------|-----------------------------|
| Applies to  | UserDefinedLibrary property |
| Data type   | Abstract kinetic law object |
| Data values | Kinetic laws                |
| Access      | Read/write                  |

## Examples

### Example 1

This example uses the command `sbiowhos` to show the current list of user-defined components.

```
sbiowhos -userdefined -kineticlaw
sbiowhos -userdefined -unit
sbiowhos -userdefined -unitprefix
```

### Example 2

This example shows the current list of user-defined components by accessing the root object.

```
rootObj = sbioroot;
get(rootObj.UserDefinedLibrary, 'KineticLaws')
get(rootObj.UserDefinedLibrary, 'Units')
get(rootObj.UserDefinedLibrary, 'UnitPrefixes')
```

## See Also

`BuiltInLibrary`, `sbioaddtolibrary`, `sbioremovefromlibrary`, `sbioroot`, `sbiounit`, `sbiounitprefix`

# Value property

---

**Purpose** Assign value to parameter object

**Description** The Value property is the value of the parameter object. The parameter object defines an assignment that can be used by the model object and/or the kinetic law object. Create parameters and assign Value using the method `addparameter`.

## Characteristics

|             |                                   |
|-------------|-----------------------------------|
| Applies to  | Object: parameter                 |
| Data type   | double                            |
| Data values | Any double. Default value is 1.0. |
| Access      | Read/write                        |

## Examples

Assign a parameter with a value to the model object.

- 1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel ('my_model');
```

- 2 Add a parameter to the model object (modelObj) with Value 0.5.

```
parameterObj1 = addparameter (modelObj, 'K1', 0.5)
```

MATLAB returns:

SimBiology Parameter Array

| Index: | Name: | Value: | ValueUnits: |
|--------|-------|--------|-------------|
| 1      | K1    | 0.5    |             |

## See Also

`addparameter`

**Purpose** Parameter value units

**Description** The ValueUnits property indicates the unit definition of the parameter object Value property. ValueUnits can be one of the built-in units. To get a list of the built-in units, use the sbioshowunits function. If ValueUnits changes from one unit definition to another, the Value does not automatically convert to the new units. The sbioconvertunits function does this conversion.

You can add a parameter object to a model object or a kinetic law object.

## Characteristics

|             |  |
|-------------|--|
| Applies to  | Object: parameter  |
| Data type   | char string  |
| Data values | Unit from units library. Default is '' (empty string). Note that the default value of an empty string means unspecified. Unspecified units are permitted during dimensional analysis, but not during unit conversion. (Use the string 'dimensionless' to specify dimensionless units.) |
| Access      | Read/write   |

# ValueUnits property

---

---

**Note** SimBiology uses unit strings including empty units in association with DimensionalAnalysis and UnitConversion features.

- When DimensionalAnalysis and UnitConversion are both false, unit strings are not used. However, SimBiology still performs a minimum level of dimensional analysis to decide whether a reaction rate is in dimensions of amount/time or concentration/time.
- When DimensionalAnalysis is true and UnitConversion is false, units (if not empty) must have consistent dimensions so that SimBiology can perform dimensional analysis. However, the units are not converted.
- When UnitConversion is set to true (which requires DimensionalAnalysis to be true), SimBiology performs a dimensional analysis and converts everything to consistent units. Hence, you must specify consistent units, and no unit strings can be empty. If you have a dimensionless parameter, you must still set its unit string to dimensionless.

## Examples

Assign a parameter with a value to the model object.

- 1 Create a model object, and then add a reaction object.

```
modelObj = sbiomodel('my_model');
```

- 2 Add a parameter with Value 0.5, and assign it to the model object (modelObj).

```
parameterObj1 = addparameter(modelObj, 'K1', 0.5, 'ValueUnits', '1/second')
```

MATLAB returns:

```
SimBiology Parameter Array
```

| Index: | Name: | Value: | ValueUnits: |
|--------|-------|--------|-------------|
| 1      | K1    | 0.5    | 1/second    |

### See Also

`addparameter`, `sbioconvertunits`, `sbioshowunits`

# ZeroOrderDurationParameter property

---

**Purpose** Zero-order dose absorption duration

**Description** ZeroOrderDurationParameter is a property of the PKModelMap object. It specifies the name(s) of parameter object(s) that represent the duration of absorption when the DosingType property is ZeroOrder.

Specify the name(s) of parameter object(s) that are:

- Scoped to a model
- Constant, that is, their ConstantValue property is true

When dosing multiple compartments, a one-to-one relationship must exist between the number and order of elements in the ZeroOrderDurationParameter property and the DosingType property. For a dose that is not dosed with zero-order kinetics, use '' (an empty string). For an example, see “Dosing Multiple Compartments in a Model”.

## Characteristics

|            |                                      |
|------------|--------------------------------------|
| Applies to | Object: PKModelMap                   |
| Data type  | char string or cell array of strings |

---

**Tip** If you are not using any zero-order doses, you can set this property to a cell array of empty strings, or simply an empty cell array.

---



# ZeroOrderDurationParameter property

---

|             |   |
|-------------|---|
| Data values | Name of a parameter object or empty. Default is an empty cell array.<br>The parameter object(s) must be: <ul style="list-style-type: none"><li>• Scoped to a model</li><li>• Constant, that is, have a <code>ConstantValue</code> property set to <code>true</code></li></ul> |
| Access      | Read/write  |

## See Also

“Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters” in the SimBiology User’s Guide, `DosingType`, `PKModelMap` object

# ZeroOrderDurationParameter

---

## A

AbsoluteTolerance property  
reference 3-2

AbsoluteToleranceScaling property  
reference 3-4

AbsoluteToleranceStepSize property  
reference 3-5

abstract kinetic law object  
reference 2-8

accelerate method  
reference 2-2 2-164

Active property  
reference 3-6

addcompartment method  
reference 2-10

addCompartment method  
reference 2-15

addconfigset method  
reference 2-19

addcontent method  
reference 2-23

adddose method  
reference 2-25

addevent method  
reference 2-27

addkineticlaw method  
reference 2-30

addparameter method  
reference 2-39

addproduct method  
reference 2-44

addreactant method  
reference 2-47

addreaction method  
reference 2-50

addrule method  
reference 2-57

addspecies method  
reference 2-62

addvariant method

reference 2-68

Amount property  
reference 3-8

AmountUnits property  
reference 3-9

## B

BoundaryCondition property  
reference 3-13

boxplot method  
reference 2-70 to 2-71

BuiltInLibrary property  
reference 3-19

## C

Capacity property  
reference 3-22

CapacityUnits property  
reference 3-23

commit method  
reference 2-72

compartment object  
reference 2-74

Compartments property  
reference 3-26

CompileOptions property  
reference 3-28

Composition property 3-30

configset object  
reference 2-78

Conserved Moieties  
function for 1-27

ConstantAmount property  
reference 3-32

ConstantCapacity property  
reference 3-34

ConstantValue property  
reference 3-35

- construct method
  - reference 2-81
- constructDefaultFixedEffectValues method
  - reference 2-83
- Content property
  - reference 3-37
- copyobj method
  - reference 2-84
- CovariateLabels property
  - reference 3-39
- covariateModel method
  - reference 2-90
- CovariateModel object
  - reference 2-86
- createSimFunction method
  - reference 2-95
- D**
- Data property
  - reference 3-41
- DataCount property
  - reference 3-42
- DataInfo property
  - reference 3-43
- DataNames property
  - reference 3-45
- DataSet property
  - reference 3-46
- DefaultSpeciesDimension property
  - reference 3-47
- delete method
  - reference 2-102
- DependentVarLabel property
  - reference 3-49
- DependentVarUnits property
  - reference 3-50
- DimensionalAnalysis property
  - reference 3-51
- display method
  - reference 2-104
- Dosed property
  - reference 3-55
- DoseLabel property
  - reference 3-56
- DoseUnits property
  - reference 3-57
- DosingType property
  - reference 3-58
- DurationParameterName property
  - reference 3-59
- E**
- EliminationType property
  - reference 3-60
- Ensemble Runs
  - function for 1-40 1-42 1-46
- ErrorTolerance property
  - reference 3-61
- Estimated property
  - reference 3-63
- EstimatedInfo object
  - reference 2-105
- event object
  - reference 2-110
- Exponent property
  - reference 3-67
- Expression property
  - reference 3-71
- F**
- fitted method
  - reference 2-115 2-119
- functions
  - sbioabstractkineticlaw 1-5
  - sbioaddtolibrary 1-24
  - sbioconsmoiety 1-27
  - sbioconvertunits 1-32

sbiocopylibrary 1-34  
 sbiodesktop 1-36  
 sbiodose 1-38  
 sbioensembleplot 1-40  
 sbioensemblerrun 1-42  
 sbioensemblestats 1-46  
 sbiogetmodel 1-82  
 sbiolasterror 1-84  
 sbiolastwarning 1-88  
 sbiooadproject 1-89  
 sbiomodel 1-90  
 sbionlinfit 1-95  
 sbionlmefit 1-103  
 sbionlmefitsa 1-111  
 sbioparamestim 1-129  
 sbioplot 1-153  
 sbioremovefromlibrary 1-156  
 sbioreset 1-158  
 sbioroot 1-161  
 sbiosaveproject 1-181  
 sbioselect 1-183  
 sbioshowunitprefixes 1-196  
 sbioshowunits 1-198  
 sbiosubplot 1-215  
 sbiotrellis 1-217  
 sbiounit 1-224  
 sbiounitcalculator 1-228  
 sbiounitprefix 1-229  
 sbiovariant 1-232  
 sbiowhos 1-235  
 sbmlexport 1-237  
 setactiveconfigset 2-276  
 setparameter 2-278  
 setspecies 2-280  
 simbiology 1-241

**G**

get method  
   reference 2-121

getadjacencymatrix method  
   reference 2-123  
 getconfigset method  
   reference 2-125  
 getdata method  
   reference 2-129  
 getdose method  
   reference 2-136  
 getparameters method  
   reference 2-145  
 getsensmatrix method  
   reference 2-147  
 getspecies method  
   reference 2-150  
 getstoichmatrix method  
   reference 2-152  
 getTable method  
   reference 2-154  
 getvariant method  
   reference 2-157  
 GroupedData object  
   reference 2-159  
 GroupID property  
   reference 3-79  
 GroupLabel property  
   reference 3-80  
 GroupNames property  
   reference 3-81

**H**

HasLag property  
   reference 3-82  
 HasResponseVariable property  
   reference 3-83

**I**

IndependentVarLabel property  
   reference 3-84

IndependentVarUnits property  
reference 3-85

InitialAmount property  
reference 3-86

InitialAmountUnits property  
reference 3-87

Inputs property  
reference 3-90

Interval property  
reference 3-92

## K

kinetic law definition 2-8

kinetic law object  
reference 2-170

KineticLaw property  
reference 3-93

KineticLawName property  
reference 3-95

## L

LagParameter property  
reference 3-97

LagParameterName property  
reference 3-98

LeastSquaresResults object  
reference 2-178

LogDecimation property  
reference 3-99

## M

MaximumNumberOfLogs property  
reference 3-101

MaximumWallClock property  
reference 3-103

MaxIterations property  
reference 3-105

MaxStep property

reference 3-107

methods

- accelerate 2-2 2-164
- addcompartment 2-10
- addCompartment 2-15
- addconfigset 2-19
- addcontent 2-23
- adddose 2-25
- addevent 2-27
- addkineticlaw 2-30
- addparameter 2-39
- addproduct 2-44
- addreactant 2-47
- addreaction 2-50
- addrule 2-57
- addspecies 2-62
- addvariant 2-68
- boxplot 2-70 to 2-71
- commit 2-72
- construct 2-81
- constructDefaultFixedEffectValues 2-83
- copyobj 2-84
- covariateModel 2-90
- createSimFunction 2-95
- delete 2-102
- fitted 2-115 2-119
- get 2-121
- getadjacencymatrix 2-123
- getconfigset 2-125
- getdata 2-129
- getdose 2-136
- getparameters 2-145
- getsensmatrix 2-147
- getspecies 2-150
- getstoichmatrix 2-152
- getTable 2-154
- getvariant 2-157
- plot 2-200 to 2-201
- plotActualVersusPredicted 2-202 to 2-203
- plotResidualDistribution 2-206 to 2-207

plotResiduals 2-205  
 predict 2-208 2-214  
 random 2-218 2-224  
 removeconfigset 2-231  
 removedose 2-233  
 removevariant 2-235  
 rename 2-237  
 reorder 2-239  
 resample 2-244  
 reset 2-247  
 rmcontent 2-249  
 rmproduct 2-252  
 rmreactant 2-254  
 select 2-264  
 selectbyname 2-268  
 set 2-274  
 setTable 2-282  
 summary 2-260  
 verify 2-325  
 Methods  
   display 2-104  
 model object  
   reference 2-181 2-241 2-261 2-316 2-322  
 ModelName property  
   reference 3-108  
 Models property  
   reference 3-109  
 Moiety Conservation  
   function for 1-27  
 Multiplier property  
   reference 3-110  
  
**N**  
 Name property  
   reference 3-111  
 NLINResults object  
   reference 2-185  
 NLMERResults object  
   reference 2-186

Normalization property  
   reference 3-114  
 Notes property  
   reference 3-115

## O

object  
   abstract kinetic law 2-8  
   compartment 2-74  
   configset 2-78  
   CovariateModel 2-86  
   EstimatedInfo 2-105  
   event 2-110  
   GroupedData 2-159  
   kinetic law 2-170  
   LeastSquaresResults 2-178  
   model 2-181 2-241 2-261 2-316 2-322  
   NLINResults 2-185  
   NLMERResults 2-186  
   OptimResults 2-188  
   parameter 2-189  
   PKCompartment 2-191  
   PKData 2-193  
   PKModelDesign 2-196  
   PKModelMap 2-198  
   reaction 2-228  
   root 2-256  
   rule 2-258  
   SimData 2-312  
   simfunction 2-285  
   unit 2-318 2-320  
 Observed property  
   reference 3-116  
 Offset property  
   reference 3-117  
 OptimResults object  
   reference 2-188  
 Outputs property  
   reference 3-119

OutputTimes property

reference 3-121

Owner property

reference 3-123

## P

Parameter Estimation

function for 1-129

parameter object

reference 2-189

Parameters property

reference 3-126

ParameterVariableNames property

reference 3-128

ParameterVariables property

reference 3-130

Parent property

reference 3-132

PKCompartment object

reference 2-191

PKCompartments property

reference 3-134

PKData object

reference 2-193

PKModelDesign object

reference 2-196

PKModelMap object

reference 2-198

plot method

reference 2-200 to 2-201

plotActualVersusPredicted method

reference 2-202 to 2-203

plotResidualDistribution method

reference 2-206 to 2-207

plotResiduals method

reference 2-205

predict method

reference 2-208 2-214

Products property

reference 3-135

properties

AbsoluteTolerance 3-2

AbsoluteToleranceScaling 3-4

AbsoluteToleranceStepSize 3-5

Active 3-6

Amount 3-8 to 3-9

BoundaryCondition 3-13

BuiltInLibrary 3-19

Capacity 3-22

CapacityUnits 3-23

Compartments 3-26

CompileOptions 3-28

Composition 3-30

ConstantAmount 3-32

ConstantCapacity 3-34

ConstantValue 3-35

Content 3-37

CovariateLabels 3-39

Data 3-41

DataCount 3-42

DataInfo 3-43

DataNames 3-45

DataSet 3-46

DefaultSpeciesDimension 3-47

DependentVarLabel 3-49

DependentVarUnits 3-50

DimensionalAnalysis 3-51

Dosed 3-55

DoseLabel 3-56

DoseUnits 3-57

DosingType 3-58

DurationParameterName 3-59

EliminationType 3-60

ErrorTolerance 3-61

Estimated 3-63

Exponent 3-67

Expression 3-71

GroupID 3-79

GroupLabel 3-80



- GroupNames 3-81
  - HasLag 3-82
  - HasResponseVariable 3-83
  - IndependentVarLabel 3-84
  - IndependentVarUnits 3-85
  - InitialAmount 3-86
  - InitialAmountUnits 3-87
  - Inputs 3-90
  - Interval 3-92
  - KineticLaw 3-93
  - KineticLawName 3-95
  - LagParameter 3-97
  - LagParameterName 3-98
  - LogDecimation 3-99
  - MaximumNumberOfLogs 3-101
  - MaximumWallClock 3-103
  - MaxIterations 3-105
  - MaxStep 3-107
  - ModelName 3-108
  - Models 3-109
  - Multiplier 3-110
  - Name 3-111
  - Normalization 3-114
  - Notes 3-115
  - Observed 3-116
  - Offset 3-117
  - Outputs 3-119
  - OutputTimes 3-121
  - Owner 3-123
  - Parameters 3-126
  - ParameterVariableNames 3-128
  - ParameterVariables 3-130
  - Parent 3-132
  - PKCompartments 3-134
  - Products 3-135
  - RandomState 3-138
  - Rate 3-140
  - RateLabel 3-143
  - RateUnits 3-141
  - Reaction 3-146
  - ReactionRate 3-147
  - Reactions 3-154
  - RelativeTolerance 3-155
  - RepeatCount 3-157
  - Reversible 3-158
  - Rule 3-161
  - Rules 3-166
  - RuleType 3-163
  - RunInfo 3-167
  - RuntimeOptions 3-168
  - SensitivityAnalysis 3-169
  - SensitivityAnalysisOptions 3-171
  - SolverOptions 3-175
  - SolverType 3-178
  - Species 3-181
  - SpeciesVariableNames 3-182
  - SpeciesVariables 3-184
  - StartTime 3-186
  - StatesToLog 3-187
  - Stoichiometry 3-189
  - StopTime 3-191
  - Tag 3-195
  - TargetName 3-197
  - Time 3-200
  - TimeUnits 3-201
  - Type 3-203
  - UnitConversion 3-204
  - UserData 3-208
  - UserDefinedLibrary 3-209
  - Value 3-212
  - ValueUnits 3-213
  - ZeroOrderDurationParameter 3-216
- Properties
- Reactants 3-144
- R**
- random method
    - reference 2-218 2-224
  - RandomState property

- reference 3-138
  - Rate property
    - reference 3-140
  - RateLabel property
    - reference 3-143
  - RateUnits property
    - reference 3-141
  - Reactants property
    - reference 3-144
  - reaction object
    - reference 2-228
  - Reaction property
    - reference 3-146
  - ReactionRate property
    - reference 3-147
  - Reactions property
    - reference 3-154
  - RelativeTolerance property
    - reference 3-155
  - removeconfigset method
    - reference 2-231
  - removedose method
    - reference 2-233
  - removevariant method
    - reference 2-235
  - rename method
    - reference 2-237
  - reorder method
    - reference 2-239
  - RepeatCount property
    - reference 3-157
  - resample method
    - reference 2-244
  - reset method
    - reference 2-247
  - Reversible property
    - reference 3-158
  - rmcontent method
    - reference 2-249
  - rmproduct method
    - reference 2-252
  - rmreactant method
    - reference 2-254
  - root object
    - reference 2-256
  - rule object
    - reference 2-258
  - Rule property
    - reference 3-161
  - Rules property
    - reference 3-166
  - RuleType property
    - reference 3-163
  - RunInfo property
    - reference 3-167
  - RuntimeOptions property
    - reference 3-168
- S**
- sbioabstractkineticlaw function
    - reference 1-5
  - sbioaddtolibrary function
    - reference 1-24
  - sbioconsmoiety function
    - reference 1-27
  - sbioconvertunits function
    - reference 1-32
  - sbiocopylibrary function
    - reference 1-34
  - sbiodesktop function
    - reference 1-36
  - sbiodose function
    - reference 1-38
  - sbioensembleplot function
    - reference 1-40
  - sbioenssemblerun function
    - reference 1-42
  - sbioensemblestats function
    - reference 1-46

- sbiogetmodel function
  - reference 1-82
- sbiolasterror function
  - reference 1-84
- sbiolastwarning function
  - reference 1-88
- sbioloadproject function
  - reference 1-89
- sbiomodel function
  - reference 1-90
- sbiolmefit function
  - reference 1-103
- sbiolmefitsa function
  - reference 1-111
- sbioparamestim function
  - reference 1-129
- sbioplot function
  - reference 1-153
- sbioremovefromlibrary function
  - reference 1-156
- sbioreset function
  - reference 1-158
- sbioroot function
  - reference 1-161
- sbiosaveproject function
  - reference 1-181
- sbioselect function
  - reference 1-183
- sbioshowunitprefixes function
  - reference 1-196
- sbioshowunits function
  - reference 1-198
- sbiosubplot function
  - reference 1-215
- sbiotrellis function
  - reference 1-217
- sbiounit function
  - reference 1-224
- sbiounitcalculator function
  - reference 1-228
- sbiounitprefix function
  - reference 1-229
- sbiovariant function
  - reference 1-232
- sbiowhos function
  - reference 1-235
- sbsmlexport function
  - reference 1-237
- select method
  - reference 2-264
- selectbyname method
  - reference 2-268
- Sensitivity Analysis
  - properties for 3-90 3-114 3-119 3-169 3-171
- SensitivityAnalysis property
  - reference 3-169
- SensitivityAnalysisOptions property
  - reference 3-171
- set method
  - reference 2-274
- setactiveconfigset function
  - reference 2-276
- setparameter function
  - reference 2-278
- setspecies function
  - reference 2-280
- setTable method
  - reference 2-282
- simbiology function
  - reference 1-241
- SimData object
  - reference 2-312
- simfunction object
  - reference 2-285
- SolverOptions property
  - reference 3-175
- SolverType property
  - reference 3-178
- Species property
  - reference 3-181

SpeciesVariableNames property  
reference 3-182

SpeciesVariables property  
reference 3-184

StartTime property  
reference 3-186

StatesToLog property  
reference 3-187

Stoichiometry property  
reference 3-189

StopTime property  
reference 3-191

summary method  
reference 2-260

**T**

Tag property  
reference 3-195

TargetName property  
reference 3-197

Time property  
reference 3-200

TimeUnits property  
reference 3-201

Type property

reference 3-203

**U**

unit object  
reference 2-318 2-320

UnitConversion property  
reference 3-204

UserData property  
reference 3-208

UserDefinedLibrary property  
reference 3-209

**V**

Value property  
reference 3-212

ValueUnits property  
reference 3-213

verify method  
reference 2-325

**Z**

ZeroOrderDurationParameter property  
reference 3-216