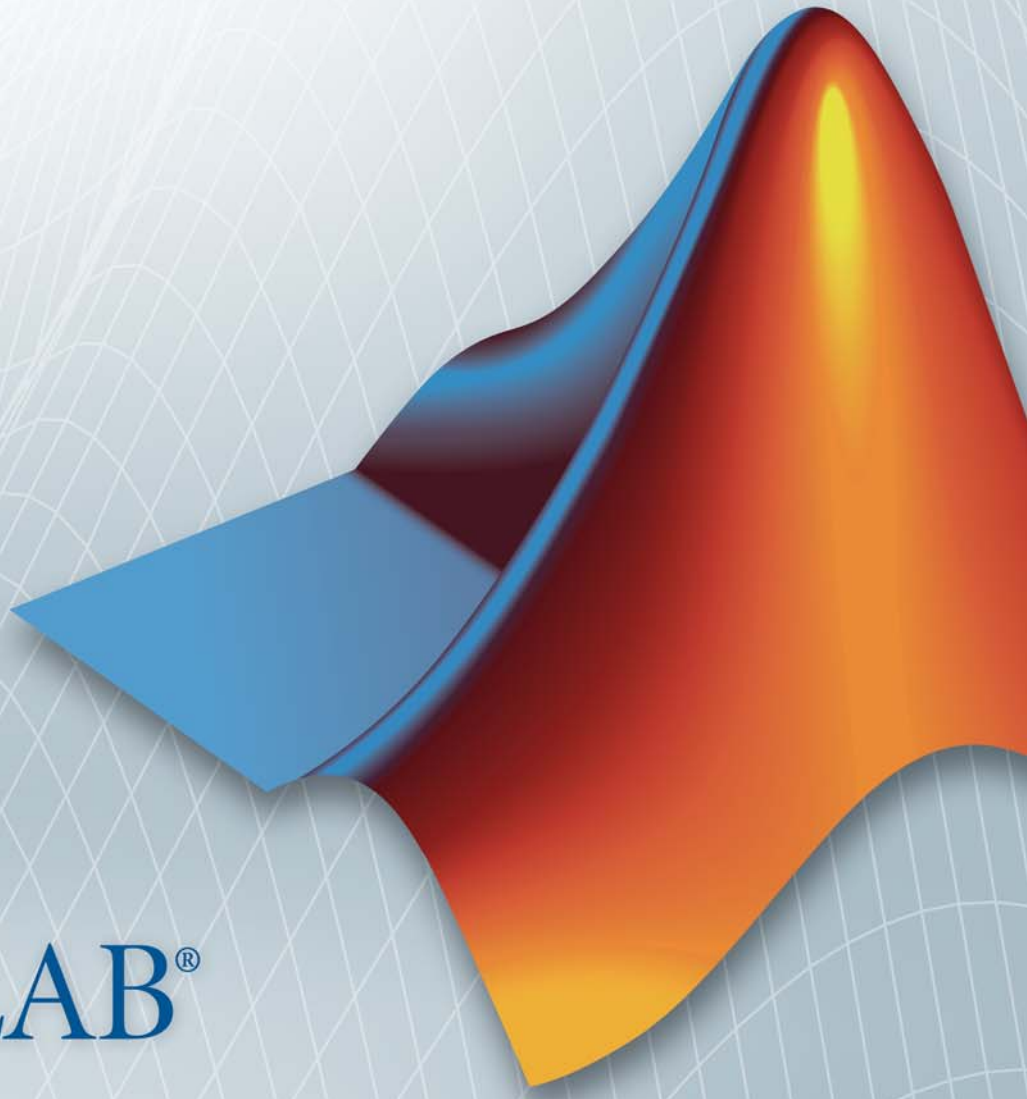
SimBiology® 3
User's Guide



MATLAB®

MathWorks®

How to Contact MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support

`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`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® User's Guide

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `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` 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)

Contents

1 Modeling

2 Structural Analysis

3 Simulation and Analysis

4

A Creating Reaction Rates

B Creating Rate Rules

Models Used in Examples

C

Index

1

Modeling

What is a Model?

In this section...

Model Definition

A SimBiology® *model* is composed of a set of expressions (reactions, differential equations, discrete events), which together describe the dynamics of a biological system. You write expressions in terms of quantities (compartments, species, parameters), which are also enumerated in the model.

Expressions

There are three distinct types of expressions in SimBiology:

- Reactions
- Rules
- Events

Reactions

A *reaction* describes a process such as a transformation, transport, or binding/unbinding process between reactants and products.

Example reactions include:

```
Creatine + ATP <-> ADP + phosphocreatine
cytoplasm.speciesA -> nucleus.speciesA
```

Rules

A *rule* is a class of mathematical expressions that include differential equations, initial assignments, repeated assignments, and algebraic constraints.

For example, you can use a rule to:

- Specify values for model components that are required for comparison with experimental data. For example, specify the active fraction of total protein.
- Assign values to model components based on the values of other components in the model. For example, define a parameter's value as being proportional to a species or another parameter.
- Define mass balance equations.
- For species, use rate rules as an alternative to the differential rate expression generated from reactions.

Events

An *event* describes an instantaneous change in the value of a quantity (compartment, species, parameter). The discrete transition occurs when a user-specified condition becomes true. The condition can be a specific time or a specific time-independent condition.

For example, you can use an event to:

- Activate or deactivate a specific species (activator or inhibitor species)
- Change a parameter value based on external signals
- Change reaction rates in response to addition or removal of a species
- Replicate an experimental condition, such as the addition or removal of an activating agent (such as a drug) to or from a sample

Quantities

SimBiology uses three types of quantities in models:

- Compartments
- Species

- Parameters

Compartments

A *compartment* defines a physically bounded region that contains species. A compartment is characterized by a capacity expressed as volume, area, or length. A compartment can also contain other compartments, which adds hierarchy to a model. For example, a compartment named `cytoplasm` might contain a compartment named `nucleus`, thereby partitioning species based on their location.

Species

A *species* characterizes the state of the biological system by representing the amount (or concentration) present in the system for that entity. Examples of species are `DNA`, `ATP`, and `creatine`. Species' amounts (or concentrations) vary during a simulation as a result of their participation in reactions, differential equations, and events. Therefore, species represent the dynamical state of a biological system.

Parameters

A *parameter* is a quantity that is referred to by expressions. It typically remains constant during a simulation. For example, parameters are used as rate constants in reactions.

You can configure a parameter to vary during a simulation. This is useful, for example, to model the change in a reaction rate given the concentration of a catalyst or a change in temperature.

Model Hierarchy

Quantities are organized as illustrated in the following diagram.

Note the following conditions imposed on model hierarchy:

- Models must contain at least one compartment.
- A compartment can contain one or more compartments.
- Species are always contained within a compartment.

Model Modifiers

In this section...

In addition to expressions and quantities, which are model components, SimBiology provides the following constructs (or objects) that you use to modify or perturb a model from its base configuration.

Variants

A *variants* is a collection of quantities (compartments, species, and/or parameters) that you can use to alter a model's initial or base configuration, which is easier than individually modifying each quantity separately. For example, assuming that a different set of parameter values characterizes differences between wild type and mutant strains, you can use a variant to group parameter values indicative of these strains. You apply variants to a model to evaluate the model behavior under "variant" conditions. Note that the model's original configuration is only temporarily altered, for example during a simulation.

For example, you can use a variant to compare:

- Two different species, such as human versus mouse
- Wild type versus mutant strains
- Different experimental conditions

Doses

A *dose* is used to increment the amount (or concentration) of a species exogenously. For example, you can use a dose to model the instantaneous supply of a drug regimen during the simulation of a model.

How Variants and Doses Modify a Model

The following graphic illustrates how variants and doses modify a model:

Representing a Model and Model Modifiers in SimBiology

In this section...

Constructing a Simple Model

The following code shows how to construct a simple model consisting of one compartment, two species, a parameter, and a reaction:

```
% Create a model named example
model = sbiomodel('example');
% Add a compartment named cell to model
compartment = addcompartment(model, 'cell');
% Add two species, A and B, to the cell compartment
species_1 = addspecies(compartment, 'A');
species_2 = addspecies(compartment, 'B');
% Add a parameter, K1, to model with a value of 3
parameter = addparameter(model, 'K1', 3);
% Add the reaction A -> B to the model
```

```
reaction = addreaction(model, 'A -> B', 'ReactionRate', 'K1');
```

SimBiology Objects

In SimBiology, models and their components are implemented as objects. For example, in the previous code, `model` is a model object composed of a compartment object, `compartment`, which in turn is composed of species, parameter and reaction objects. These objects have properties and methods associated with them, which you use to access and configure them. Use the `get` method to list the property values of an object. Use the `set` method to change the property values of an object.

SimBiology objects are handle objects, which has implications for how they behave during copy operations. In particular, handle objects do not behave as arrays of doubles do in MATLAB®. To learn how handle objects affect copy operations, see Copying Objects in the MATLAB Programming Fundamentals documentation.

Model Object

A `model object` represents a model and is composed of quantities and expressions. Quantities represent the state variables in the system while expressions depict the relationships between quantities and therefore describe the dynamics of the model.

For information about...	See...
Creating a model	`sbiomodel`
Methods and properties of a model	`model object`
Removing models from MATLAB Workspace	`clear`
Deleting models	`sbioreset`

Objects Representing Quantities

The following objects represent quantities in a model:

- Compartment
- Species

- Parameter

Scoping of Compartments, Species, and Parameters

Scoping refers to which object another object is contained in. Scoping affects compartments, species, and parameters.

- **Compartment Object** — A compartment is scoped to (or contained in) a model or another compartment.
- **Species Object** — Although a model can contain multiple compartments, each species is scoped to (or contained in) only one compartment.
- **Parameter Object** — A parameter is scoped to (or contained in) a model or a kinetic law.

Naming of Compartments and Species

Note the following when naming objects within a model:

- Compartment names must be unique within a model.
-

Compartment Object

A `compartment object` represents a compartment, which is a physically isolated region. It lets you associate pools of species to that physically isolated region. It has a capacity associated with it.

All models must contain at least one compartment. A compartment is scoped to a model or another compartment. A compartment contains one or more species. Each compartment within a model must have a unique name.

You can add a compartment explicitly (using the `addcompartment` method) or add a reaction (using the `addreaction` method) to create a compartment.

For information about...	See...
Creating and adding a compartment to a model	`addcompartment`, `addreaction`
Methods and properties of a compartment	`compartment object`

Species Object

A `species object` represents a species, which is the amount of a chemical or entity that participates in reactions. A species is always scoped to a compartment. Each species must

When adding species to a model with multiple compartments, you must specify qualified names, using *`compartmentName.speciesName`*. For example, `nucleus.DNA` denotes the species `DNA` in the compartment `nucleus`.

For information about...	See...
Creating and adding a species to a model	`addspecies`
Methods and properties of a species	`species object`

How Species Amounts Change During Simulations

The amount of a species can remain constant or vary during the simulation of a model. Use the following properties of a `species object` to specify how the amount of a species changes during a simulation:

- `ConstantAmount` property — When set to `true`, the species amount does not change during a simulation. The species can be part of a reaction or rule, but the reaction or rule cannot change its amount. When set to `false`, the species amount is determined by a reaction or a rule, but not both.
- `BoundaryCondition` property — When set to `true`, the species amount is either constant or determined by a rule, but not determined by a chemical reaction. In other words, the simulation does not create a differential rate term from the reactions for this species, even if it is in a reaction, but it can have a differential rate term created from a rule.

Keeping a Species Amount Unchanged

Set `ConstantAmount` to `true` and `BoundaryCondition` to `false` for a constant species, whose amount is not changed by a reaction or rule. In this case, the species acts like a parameter. It cannot be in a reaction, and it cannot be varied by a rule.

ConstantAmount	BoundaryCondition	Reaction	Rule	Changed By
True	False	No	No	Never

Example — Species **E** is not part of the reaction, but it is part of the reaction rate equation. **E** is constant and could be replaced with the constant `Vm = k2*E`.

```
      reaction: S -> P
reaction rate: kcat*E*S/(Km + S)
```

Changing a Species Amount with a Reaction or Rule

Set `ConstantAmount` to `false` and `BoundaryCondition` to `false` for a species whose amount is changed by a reaction or rule, but not both.

ConstantAmount	BoundaryCondition	Reaction	Rule	Changed By
False	False	Yes	No	Reaction
False	False	No	Yes	Rule

Example 1 — Species **A** is part of a reaction, and it is in the reaction rate equation. The species amount or concentration is determined by the reaction. This is the most common category of a species. A differential rate equation for the species is created from the reactions.

```
      reaction: A -> B
reaction rate: k*A
```

Example 2 — Species **E** is not part of the reaction, but it is in the reaction rate equation. **E** varies with another reaction or rule.

```
      reaction: S -> P
reaction rate: kcat*E*S/(Km + S)
```

Example 3 — Species **G** is not part of a reaction, and it is not in a rate equation. **G** varies with an algebraic rule or rate rule.

```
rate rule: dG/dt = k
```

Changing a Species Amount with a Rule When Species is Part of a Reaction

Set `ConstantAmount` to `false` and `BoundaryCondition` to `true` for a species whose amount is changed by a rule, but the species is also part of a reaction, and a differential rate term from the reaction is not created. The amount of the species changes with the rule, and a differential rate term is created from the rule.

ConstantAmount	BoundaryCondition	Reaction	Rule	Changed By
False	True	Yes	Yes	Rule

Example 1 — Species **A** is not changed by the rate equation, but changes according to a rate rule. However, **A** could be in the rate equation that changes other species in the reaction.

```
      reaction: A -> B
reaction rate: k1 or k1*A
    rate rule: dA/dt = k2*A (solution is  A = k2*t)
               (enter in SimBiology as A = k2*A)
```

Example 2 — Species **A** is not in the rate equation, but changes according to an algebraic rule.

```
       reaction: A -> B + C
 reaction rate: k or k*A
algebraic rule: A = 2*C
                (enter in SimBiology as 2*C - A)
```

Keeping a Species Amount Unchanged When Species is Part of a Reaction that Adds or Removes Mass

Set `ConstantAmount` to `false` and `BoundaryCondition` to `true` for a constant species that is part of a reaction, but a differential rate term is not created from the reaction. The differential rate term is created from a rule.

ConstantAmount	BoundaryCondition	Reaction	Rule	Changed By
True	True	Yes	No	Never

During simulation, a differential rate equation is not created for the species. `dSpecies/dt` does not exist.

Example 1 — **A** is a *`infinite source`* and its amount does not change. B increases with a zero order rate (`k` and `k*A` are both constants). A source refers to a species where mass is added to the system.

```
      reaction: A -> B
reaction rate: k or k*A
```

Example 2 — B decreases with a first-order rate, but **A** is an *`infinite sink`* and its amount does not change. A *`sink`* refers to a species where mass is subtracted from the system.

```
      reaction: B -> A
reaction rate: k*B
```

Example 3 — The **null** species is a reserved species name that can act as a source or a sink.

```
      reaction: null -> B
reaction rate: k

      reaction: B -> null
reaction rate: k*B
```

Example 4 — **ATP** and **ADP** are in the reaction and have constant values, but they are not in the reaction rate equation.

```
      reaction: S + ATP -> P + ADP
reaction rate: Vm*S/(Km + S)
```

Parameter Object

A `parameter object` represents a parameter, which is a value that typically remains constant during a simulation. For example, you use parameters to define reaction rate constants. In some circumstances it is useful to allow

parameter values to vary. In these cases you can specify a parameter as nonconstant.

For information about...	See...
Creating and adding a parameter to a model	`addparameter`
Methods and properties of a parameter	`parameter object`

Scope of Parameter Objects

When you create a parameter, you scope it to either a model or a reaction.

Parameters Scoped to a Model. Parameters scoped to a model can be used (or referenced) by any expression (reaction, rule, or event) in the model.

Parameters Scoped to a Reaction. Parameters scoped to a reaction can be used (or referenced) by only the reaction rate expression.

Objects Representing Expressions

The following objects represent expressions in a model:

- Reaction object
- KineticLaw object
- Rule object
- Event object

When Reactions, Rules, and Events Specify Parameters

Reactions, rules and events can specify one or more parameters. A parameter is scoped a model or a kinetic law. Note the following when using a reaction, rule, or event to specify a parameter:

- When a *reaction* specifies a parameter, the parameter can be scoped to the model or the kinetic law that is part of that reaction. If more than one reaction specifies the same parameter, the parameter must be scoped to the

model. If two parameters have the same name, one at the model level and the other at the kinetic law level, the software uses the parameter at the kinetic law level for the reaction rate that specifies the parameter.

- When a *rule* specifies a parameter, the parameter must be scoped to the model.
- When an *event* specifies a parameter, the parameter must be scoped to the model.

For more information, see "Scope of Parameter Objects" on page 1-14.

Reaction Object

A `reaction object` represents a reaction, which is a mathematical expression and other information that describe a transformation, transport, or binding process that changes one or more species. Typically, the change is to the amount of a species.

A `reaction object` includes:

- `Reaction` property — Mathematical expression that describes the reaction
- `ReactionRate` property — Mathematical expression that defines the rate at which the reactants combine to form products. You can provide this information explicitly or use the `KineticLaw` property to populate this information.
- `KineticLaw` property — Object that specifies a rate law that defines the type of reaction rate. Examples include Henri-Michaelis-Menten and Mass Action. The object also specifies `species objects`, or `parameter objects`. This property is optional. It serves as a template for a reaction rate and provides a convenient way of applying a specific rate law to multiple reactions. If you use this property, it automatically populates the `ReactionRate` property.

A reaction is scoped to a model.

For information about...	See...
Creating and adding a reaction to a model	`addreaction`
Methods and properties of a reaction	`reaction object`
Creating and adding a kinetic law to a reaction	`addkineticlaw`
Methods and properties of a kinetic law	`KineticLaw object`

Writing Reaction Expressions

Use any valid MATLAB code to create the mathematical expression for a reaction (`Reaction` property of a `reaction object`). The reaction can specify species.

Following are rules for writing reaction expressions:

- Use spaces before and after species names and stoichiometric values.
- Stoichiometry values must be positive.
- If a stoichiometry value is not specified, it is assumed to be `1`.
- In a model with a single compartment, specify species using *`speciesName`*. In a model with multiple compartments, specify species using qualified names: *`compartmentName`*`.`*`speciesName`*. For example, `nucleus.DNA` denotes the species `DNA` in the compartment `nucleus`.
- Enclose names with non-alphanumeric characters (including spaces) in brackets.

Examples of reaction expressions include:

```
Creatine + ATP <-> ADP + phosphocreatine
glucose + 2 ADP + 2 Pi -> 2 lactic acid + 2 ATP + 2 H2O
cytoplasm.A -> nucleus.A
[compartment 1].[species A] -> [compartment 2].[species A]
```

Writing Reaction Rate Expressions Explicitly

Use any valid MATLAB code to create the mathematical expression for a reaction rate (`ReactionRate` property of a `reaction object`). The reaction rate can specify compartments, species, or parameters.

For example, if you have the following reaction expression:

```
Creatine + ATP <-> ADP + phosphocreatine
```

and the reaction follows Mass Action kinetics, then the reaction rate expression would be:

```
K*Creatine*ATP - Krev*ADP*phosphocreatine
```

Tip If your reaction rate expression is not continuous and differentiable, see "Using Events to Address Discontinuities in Rule and Reaction Rate Expressions" on page 1-34 before simulating your model.

Creating Reaction Rate Expressions Using Kinetic Law Objects

A `KineticLaw object` is scoped to a reaction and specifies:

- A rate law that defines the type of reaction rate. Examples include Henri-Michaelis-Menten and Mass Action.
- species and parameters

A `KineticLaw object` serves as a template for a reaction rate and provides a convenient way of applying a specific rate law to multiple reactions. You can use this object to create a reaction rate, which populates the `ReactionRate` property of the `reaction object`.

For example, if you create a `KineticLaw object` that specifies Henri-Michaelis-Menten for the `KineticLawName`, species `S`, and parameters `Vm` and `Km`, the reaction rate law is:

$$V_{\mathrm{m}} * S / (K_m + S)$$

Then if you create a `reaction object` that specifies the previous `KineticLaw object` and species the following reaction expression:

```
A -> B
```

with `Vm` = `Va` and `Km` = `Ka` and `S` = `A`, then the reaction rate equation is:

Va*A/(Ka + A)

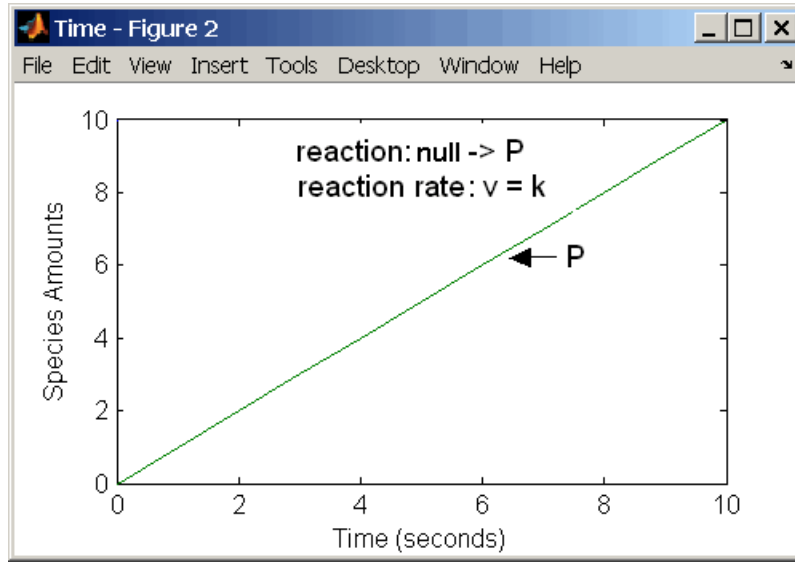
Examples of Creating Reaction Rates

Example of Creating a Zero-Order Reaction. With a zero-order reaction, the reaction rate does not depend on the concentration of reactants. Examples of zero-order reactions are synthesis from a `null` species, and modeling a source species that is added to the system at a specified rate.

```
      reaction: null -> P
reaction rate: k mole/second
      species: P =  0 mole
   parameters: k =  1 mole/second
```

Note When specifying a `null` species, the reaction rate must be defined in units of amount per unit time not concentration per unit time.

Entering the reaction above into the software and simulating produces the following result:



Zero-Order Mass Action Kinetics

Note If the amount of a reactant with zero-order kinetics reaches zero before the end of a simulation, then the amount of reactant can go below zero regardless of the solver or tolerances you set.

Examples of Creating Other Reactions. For examples of creating other reaction rates, see the following sections in Appendix A, "Creating Reaction Rates".

- "Defining Reaction Rates with Mass Action Kinetics" on page A-2
- "Defining Reaction Rates with Enzyme Kinetics" on page A-8

How Reaction Rates Are Evaluated

Reaction Rate Dimensions. When calculating species fluxes, SimBiology must determine whether you specified reaction rates in dimensions of amount/time or concentration/time. When all compartments in a model have a capacity of one unit, amount and concentration are numerically equivalent.

For all other models, the numerical results of the simulation depend on which interpretation SimBiology selects. SimBiology determines whether a reaction rate is in dimensions of amount/time or concentration/time via dimensional analysis of `ReactionRate` expressions. This minimum level of dimensional analysis always occurs, even when `DimensionalAnalysis` and `UnitConversion` are off.

The `DefaultSpeciesDimension` property defines the dimensions of species appearing in a reaction rate. SimBiology infers the dimensions of parameters appearing in a reaction rate from their `ValueUnits` property. If any parameters appearing in a reaction rate expression do not have units, SimBiology interprets the reaction rate in dimensions of amount/time. Therefore, the only way to specify that a reaction rate has dimensions of concentration/time is to assign appropriate units to all parameters.

Reactions Spanning Multiple Compartments . Specify reactions that span compartments using the syntax *`compartment1Name.species1Name –> compartment2Name.species2Name`*. The reaction rate dimensions must resolve to amount/time when:

- Species span multiple compartments.
- The reaction is reversible mass action and the products are in multiple compartments.

Examples. Consider a reaction `a + b  > c`. Using mass action kinetics, the reaction rate is `k*a*b`, where `k` is the rate constant of the reaction. If you specify that initial amounts of `a` and `b` are `0.01 molarity` and `0.005 molarity` respectively, then the reaction rate is in concentration/time (and units of `molarity/second`) if the units of `k` are `1/(molarity*second)`. If you specify `k` with another equivalent unit definition, for example, `1/((moles/liter)*second)`, SimBiology checks whether the physical quantities match. If the physical quantities do not match, you see an error and the model is not simulated.

If, in the previous example, you specify that initial amounts of `a` and `b` are `0.01` and `0.005` respectively, without specifying units, SimBiology checks whether `DefaultSpeciesDimension` is `substance` or `concentration`. If `DefaultSpeciesDimension` is `concentration`, and you set units on the rate constant such that the reaction rate dimensions resolve to concentration/time,

SimBiology scales the species amounts for compartment capacity, and returns the species values in concentration.

If you specify initial amounts of `a` and `b` as `0.01 molarity` and `0.005 mole` respectively, include the volume scaling for `b` in the reaction rate expression. Include volume scaling in the rate constant, and set the units of the rate constant accordingly (`1/(mole*second)` for concentration/time, or `1/(molarity*second)` for amount/time).

Rule Object

A `rule object` represents a rule, which is a mathematical expression that modifies one of the following:

- Compartment capacity
- Species amount
- Parameter value

A rule is scoped to a model. A rule has a `RuleType` property that specifies one of the following four types of rules:

- Algebraic
- Initial Assignment
- Repeated Assignment
- Rate

For information about...	See...
Creating and adding a rule to a model	`addrule`
Methods and properties of a rule	`rule object`

Writing Rule Expressions

Use any valid MATLAB code to create the mathematical expression for a rule. The rule can specify compartments, species, or model-scoped parameters.

Tip

- If you plan to specify a parameter in a rule, set the scope of the parameter to the model.
- If you use an algebraic or rate rule to vary the value of a parameter during the simulation, make sure the `ConstantValue` property of the parameter is set to `false`.
- If your algebraic, repeated assignment, or rate rule expression is not continuous and differentiable, see "Using Events to Address Discontinuities in Rule and Reaction Rate Expressions" on page 1-34 before simulating your model.

Algebraic Rules

An algebraic rule lets you specify mathematical constraints on one or more compartments, species, or parameters that must hold during a simulation. It is evaluated continuously during a simulation.

An algebraic rule takes the form `0 = Expression` and the rule is specified as the `Expression`.

For example, you could write a mass conservation expression such as `species_total = species1 + species2` where `species_total` is the independent variable. In SimBiology, write the rule as `species1 + species2 - species_total`.

An algebraic rule is a convenient way to define mathematical relationships between states. A model can consist of a combination of differential and algebraic relationships.

An algebraic rule is a constraint that is enforced by the solver during simulation. You can use algebraic rules to specify the dynamics for parameters, species, and compartments that are not driven by one or more reactions. The accuracy of the solution depends on the tolerance specified in the `Configset object`, which defines the simulation settings.

An algebraic rule is defined by the equation:

```
f(t,x)= 0
```

Where `t` is simulation time. The variable `x` is species amount, parameter value, or compartment capacity.

An example of an algebraic rule is:

```
x*log(x) - 3
```

Considerations When Imposing Constraints. Consider the mathematical constraint `y = m*x - c`. In the software this rule is written as `m*x - c - y`. If you want to use this rule to determine the value of `y`, then `m`, `x`, and `c` must be variables or constants whose values are known or determined by other equations. In general, the degree of freedom available must match the number of constraints. Therefore, you must ensure that the equation is not overconstrained or underconstrained. In this example, if the equation is underconstrained, it is unclear which variable is being determined by the expression.

Initial Assignment Rules

An initial assignment rule lets you specify the initial value of a compartment capacity, species amount, or parameter value as a function of other component values in the model. It is evaluated once at the beginning of a simulation.

An initial assignment rule is expressed as `Variable = Expression`.

For example, you could write an initial assignment rule to set the initial amount of `species1` to be proportional to `species2`:

```
species1 = k*species2
```

Repeated Assignment Rules

A repeated assignment rule lets you specify a value that holds at all times during simulation, and is a function of other component values in the model. It is evaluated at every time-step during a simulation. These time steps are determined by the solver during the simulation process.

A repeated assignment rule is expressed as `Variable = Expression`.

For example, if you want the capacity of a compartment (`cytoplasm`) to change in response to a change in the concentration of a species (`x`), write a repeated assignment rule to set the capacity of `cytoplasm` to be proportional to `x`.

```
cytoplasm = k*x
```

Where `k` is a specified constant parameter.

Repeated Assignment Versus Algebraic Rules. Repeated assignment rules are mathematically equivalent to algebraic rules, but result in exact solutions, compared to algebraic rules whose accuracy depends on the tolerance specified in the `Configset object`, which defines the simulation settings.

Tip

- If you can solve for the variable, use a repeated assignment rule instead of an algebraic rule.
- In repeated assignment rules, the constrained variable is explicitly defined as the left-hand side, whereas in algebraic rules it is inferred from the degrees of freedom in the system of equations. See also "Considerations When Imposing Constraints" on page 1-23.

Rate Rules

A rate rule lets you specify the time derivative of a compartment capacity, species amount, or parameter value. It is evaluated continuously during a simulation.

A rate rule is determined by `dVariable/dt = Expression`, which is expressed in the software as `Variable = Expression`. For example, to define the rate of change in the quantity of `species3` (`dspecies3/dt`), write the rule in the software as:

```
species3 = k * (species1 + species2)
```

One example of a rate rule is when `species1` is at the boundary of the system, but the rate of input of `species1` to the system can be determined by a rate rule.

A rate rule is defined by the equation:

```
dx/dt = f(t,W)
```

The variable `x` can be a species amount, parameter value, or compartment capacity. The function `f(W)` is an expression that can include other species and parameters. Enter a rate rule using the form

```
x = f(t,W)
```

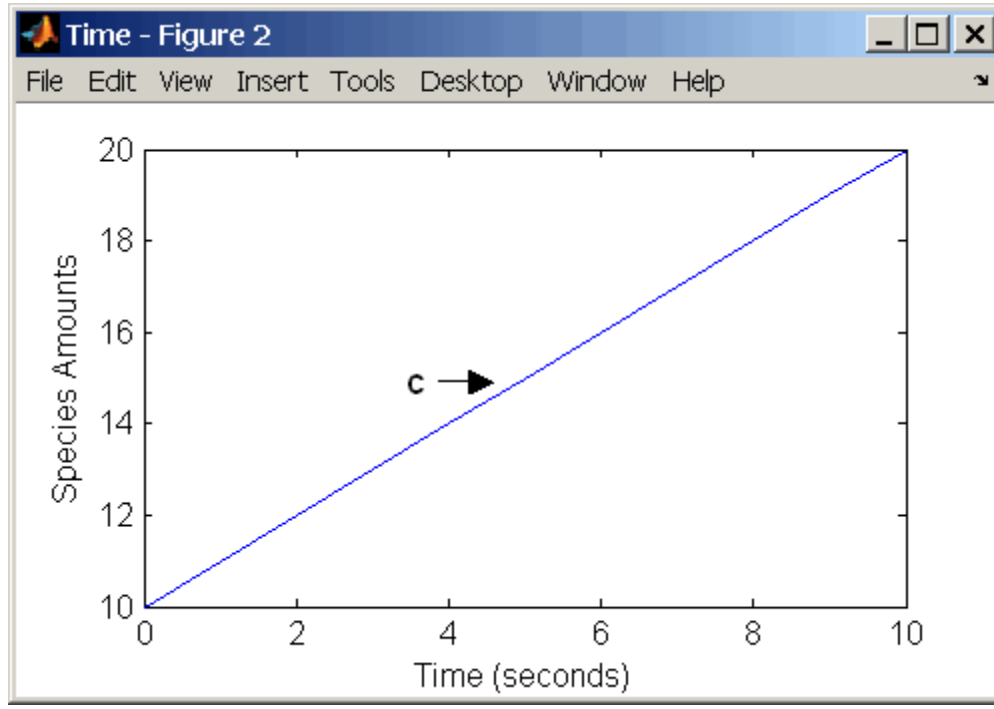
Examples of Creating Rate Rules

Example of Creating a Rate Rule When the Rate of Change Is Constant. You can increase or decrease the amount or concentration of a species by a constant value using a zero-order rate rule. For example, suppose species `c` increases by a constant rate `k`.

```
      reaction: none
rate equation: none
    rate rule: dc/dt = k
     species : c = 10 mole(initial amount)
   parameters: k = 1 mole/second
```

The analytical solution is $c = kt + c_o$, where c_o is the initial amount or concentration of the species `c`.

Enter the rule described above as `c = k`. Set the `RuleType` property to `rate`, enter the values for `c` and `k`, and then simulate.



Alternatively, you could model a constant increase in a species using Mass Action reaction `null -> C`.

Examples of Creating Other Rate Rules. For examples of creating other rate rules, see the following sections in Appendix B, "Creating Rate Rules":

- "Using Rate Rules When the Rate of Change Is Exponential" on page B-4
- "Using Rate Rules When the Rate of Change Is Determined by Another Species " on page B-6
- "Using Rate Rules To Express Differential Rate Equations as Rules" on page B-8

Event Object

An `event object` represents an event, which is a discrete transition in value of a quantity or expression in a model. This discrete transition occurs when a

user-specified condition becomes true. The condition can be a specific time or a time-independent condition.

An event is scoped to a model.

For information about...	See...
Creating and adding an event to a model	`addevent`
Methods and properties of a event	`event object`

For an example of creating and using an event in a model, see "Example — Creating A Model That Changes a Species Amount Using an Event" on page 1-39.

Event Triggers

An event has a `Trigger` property that specifies a condition that must be true to trigger the event to execute.

Typical event triggers are:

- A specific time during simulation — Specify that the event must change the amounts or values of species or parameters. For example, at time = 5 s, increase the amount of an inhibitor species above the threshold to inhibit a given reaction.
- In response to state or changes in the system — Change amounts/values of certain species/parameters in response to events that are not tied to any specific time. For example, when species `A` reaches an amount of `30` molecules, double the value of reaction rate constant `k`; or when temperature reaches `42 C`, inhibit a particular reaction by setting its reaction rate to zero.

Event Functions

An event has an `EventFcns` property that specifies what occurs when the event is triggered. Event functions can range from simple to complex. For example, an event function might:

- Change the values of compartments, species, or parameters.
- Double the value of a reaction rate constant.

Specifying Event Triggers

The `Trigger` property of an event specifies a condition that must become true for an event to execute. Typically, the condition uses a combination of relational and logical operators to build a trigger expression.

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`. For more information see the `Trigger` property reference page.

MATLAB uses specific operator precedence to evaluate trigger expressions. Precedence levels determine the order in which MATLAB evaluates an expression. Within each precedence level, operators have equal precedence and are evaluated from left to right. To find more information on how relational and logical operators are evaluated see "Operators" in the MATLAB Programming Fundamentals documentation.

Some examples of triggers are:

Trigger	Explanation
`'(time >=5) && (speciesA<1000)'`	Execute the event when the following condition becomes true:Time is greater than or equal to `5`, and `speciesA` is less than 1000. **Tip** Using a `&&` (instead of `&`) tells the software to evaluate the first part of the expression for whether the statement is true or false, and skip evaluating the second statement if this statement is false.

Trigger	Explanation
`'(time >=5) || (speciesA<1000)'`	Execute the event when the following condition becomes true: Time is greater than or equal to `5`, or if `speciesA` is less than 1000.
`'(s1 >=10.0) || (time>= 250) && (s2<5.0E17)'`	Execute the event when the following condition becomes true: Species, `s1` is greater than or equal to `10.0` or, time is greater than or equal to `250` and species `s2` is less than `5.0E17`. Because of operator precedence the expression is treated as if it were `'(s1 >=10.0) || ((time>= 250) && (s2<5.0E17))'` Thus, it is always a good idea to use parenthesis to explicitly specify the intended precedence of the statements.
`'((s1 >=10.0) || (time>= 250)) && (s2<5.0E17)'`	Execute the event when the time the following condition becomes true: Species, `s1` is greater than or equal to `10` or time is greater than or equal to `250`, and species `s2` is less than `5.0E17`.
`'((s1 >=5000.0) && (time>= 250)) || (s2<5.0E17)'`	Execute the event when the time the following condition becomes true: Species, `s1` is greater than or equal to `5000` and time is greater than or equal to `250`, or species `s2` is less than `5.0E17`.

Specifying Event Functions

The `EventFcns` property of an event specifies what occurs when the event is triggered. You can use an event function to change the value of a compartment, species, or parameter, or you can specify complex tasks by calling a user-defined function or script.

An event function is either a single valid MATLAB expression (without '`;`' in the expression) or a cell-array of single valid MATLAB expressions.

Some examples of event functions include:

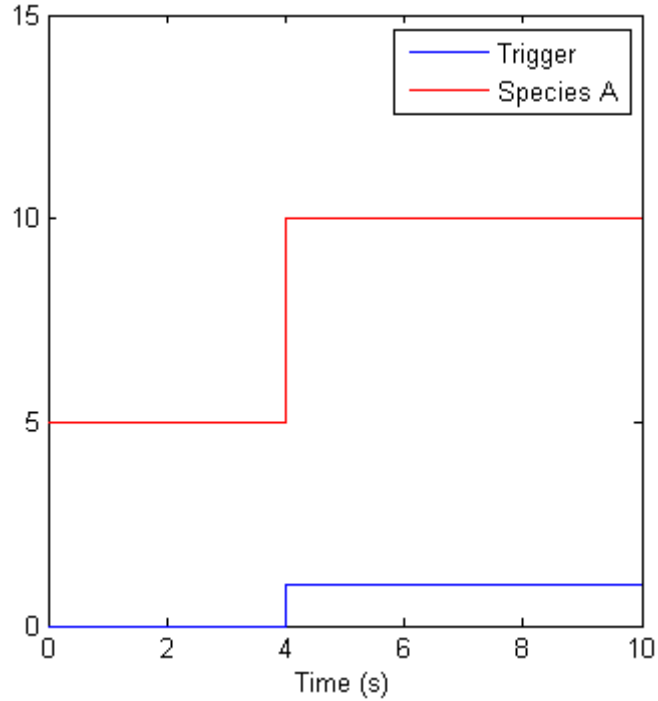
EventFcn	Explanation
`'speciesA = speciesB'`	When the event is executed set the amount of `speciesA` equal to that of `speciesB`.
`'k = k/2'`	When the event is executed halve the value of the rate constant `k`.
`{'speciesA = speciesB', 'k = k/2'}`	When the event is executed set the amount of `speciesA` equal to that of `speciesB`, and halve the value of the rate constant `k`.
`'kC = my_func(A, B, kC)'`	When the event is executed call the user-defined function `my_func()`. This function takes 3 arguments: The first two arguments are the current amounts of two species (`A` and `B`) during simulation and the third argument is the current value of a parameter, `kC`. The function returns the modified value of `kC` as its output.

Simulation Solvers for Models Containing Events

To simulate models containing events, use the deterministic `sundials` solver or the stochastic `ssa` solver. Other solvers do not support events. For more information, see "SUNDIALS Solvers" on page 3-6 and "Stochastic Solvers" on page 3-6.

How Events Are Evaluated

Consider the example of a simple event where you specify that at `4s`, you want to assign a value of `10` to species `A`.



At `time = 4 s` the trigger becomes true and the event executes. In the figure above assuming that `0` is false and `1` is true, when the trigger becomes true, the amount of species `A` is set to 010. In theory, with a perfect solver, the event would be executed exactly at `time = 4.00 s`. In practice there is a very minute delay (for example you might notice that the event is executed at `time = 4.00001 s`). Thus, you must specify that the trigger can become true at or after `4s`, which is `time >= 4 s`.

Trigger	EventFcn
`time >= 4`	`A = 10`

The point at which the trigger becomes true is called a *rising edge*. SimBiology events execute the `EventFcn` *only* at rising edges.

The `Trigger` is evaluated at every time step to check whether the condition specified in the trigger transitions from false to true. The solver detects and tracks *falling edges*, which is when the trigger becomes false, so if another

rising edge is encountered, the event is executed again. If a trigger is already true before a simulation starts, then the event does not execute the at the start of the simulation. The event is not executed until the solver encounters a rising edge. Very rarely, the solver might miss a rising edge; one example of this is when a rising edge follows very quickly after a falling edge, and the step size results in the solver skipping over the transition point.

If the trigger becomes true exactly at the stop time of the simulation, the event may or may not execute. If you want the event to execute, increase the stop time.

Note Since the rising edge is instantaneous and changes the system state, there are two values for the state at the same time. The simulation data thus contains the state before the event and after the event, but both points are at the same time value. This leads to multiple values of the system state at a single instant in time.

Evaluation of Simultaneous Events

When two or more trigger conditions simultaneously become true, the solver executes the events sequentially in the order in which they are listed in the model. You can reorder events using the `reorder` method. For example, consider a case where:

Event Number	Trigger	EventFcn
1	`SpeciesA >= 4`	`SpeciesB = 10`
2	`SpeciesC >= 15`	`SpeciesB = 25`

The solver tries to find the rising edge for these events within a certain level of tolerance. If this results in the two events occurring simultaneously, then the value of `SpeciesB` after the time step in which these two events occur, will be `25`. If you reorder the events to reverse the event order, then the value of `SpeciesB` after the time step in which these two events occur, will be `10`.

Consider an example in which you include event functions that change model components in a dependent fashion. For example, the event function in Event 2 below, stipulates that `SpeciesB` takes the value of `SpeciesC`.

Event Number	Trigger	EventFcn
1	`SpeciesA >= 4`	`SpeciesC = 10`
2	`time >= 15`	`SpeciesB = SpeciesC`

Event 1 and Event 2 may or may not occur simultaneously.

- If Event 1 and Event 2 do not occur simultaneously, when Event 2 is triggered, `SpeciesB` is assigned the value that `SpeciesC` has at the time of the event trigger.
- If Event 1 and Event 2 occur simultaneously, the solver executes Event 1 first, then executes Event 2. In the above example, if `SpeciesC = 15` when the events are triggered, after the events are executed, `SpeciesC = 10` and `SpeciesB = 10`.

Evaluation of Multiple Event Functions

Consider an event function in which you specify that the value of a model component (`SpeciesB`) is dependent on the value of model component (`SpeciesA`), but `SpeciesA` also is changed by the event function.

Trigger	EventFcn
`time >= 4`	`{'SpeciesA = 10, SpeciesB = SpeciesA'}`

The solver stores the value of `SpeciesA` at the rising edge and before any event functions are executed and uses this stored value to assign `SpeciesB` its value. So in the above example if `SpeciesA = 15` at the time the event is triggered, after the event is executed, `SpeciesA = 10` and `SpeciesB = 15`.

When One Event Triggers Another Event

In the example below, Event 1 includes an expression in the event function that causes Event 2 to be triggered, (assuming that `SpeciesA` has amount less than `5` when Event 1 is executed).

Event Number	Trigger	EventFcn
1	`time >= 5`	`{'SpeciesA = 10, SpeciesB = 5'}`
2	`SpeciesA >= 5`	`SpeciesC = SpeciesB`

When Event 1 is triggered, the solver evaluates and executes Event 1 with the result that `SpeciesA = 10`, and `SpeciesB = 5`. Now, the trigger for Event 2 becomes true (assuming that `SpeciesA` is below `5`) and the solver executes the event function for Event 2. Thus, `SpeciesC = 5` at the end of this event execution.

You can thus have event cascades of arbitrary length, for example, Event 1 triggers Event 2, which in turn triggers Event 3, and so on.

Cyclical Events

In some situations, a series of events can trigger a cascade that becomes cyclical. Once you trigger a cyclical set of events, the only way to stop the simulation is by pressing **Ctrl+C**. You lose any data acquired in the current simulation. An example of cyclical events is shown below. This example assumes that `Species B <= 4` at the start of the cycle.

Event Number	Trigger	EventFcn
1	`SpeciesA > 10`	`{SpeciesB = 5, SpeciesC = 1'}`
2	`SpeciesB > 4`	`{SpeciesC = 10, SpeciesA = 1'}`
3	`SpeciesC > 9`	`{SpeciesA = 15, SpeciesB = 1'}`

Using Events to Address Discontinuities in Rule and Reaction Rate Expressions

The solvers provided with the SimBiology software will give inaccurate results when the following expressions are not continuous and differentiable:

- Repeated assignment rule
- Algebraic rule

- Rate rule
- Reaction rate

Either ensure that the previous expressions are continuous and differentiable or use events to reset the solver at the discontinuity, as described in Deterministic Simulation of a Model Containing a Discontinuity.

Objects Representing Model Modifiers

Variant Object

A `variant object` represents a variant, which is an alternate value for a compartment, species, or parameter in a model. You can apply this alternate value during a simulation, which lets you evaluate model behavior with a different value, without having to search and replace the value, or create an additional model with the new value.

You can use a variant to store an alternate value for any of the following:

- Compartment `Capacity` property
- Species `InitialAmount` property
- Parameter `Value` property

The alternate value applies temporarily, only during a simulation, and does not alter the model's values permanently. If you determine that the values in a variant accurately define your model, you can permanently replace the values in your model with the values stored in the variant object by using the `commit` method.

Creating and Using Variants

1. Create a `variant object` and add it to a model using the `addvariant` method.
2. (Optional) Set the `Active` property of the `variant object` to true if you always want the variant to be applied before simulating the model.

3 Enter the model and variant object as input arguments to `sbiosimulate`. This applies the variant only for the current simulation and supersedes any active variant objects on the model.

   or

   If you followed step 2, simply call `sbiosimulate` on the model object to apply the variant.

For an example of creating and using a variant in a model, see "Example — Creating A Model That Changes a Parameter Value Using a Variant" on page 1-44.

For information about...	See...
Creating and adding a variant to a model	`addvariant`
Creating a stand-alone variant	`sbiovariant`
Methods and properties of a variant	`Variant object`
Appending contents to variants	`addcontent`
Replacing model values permanently with values from a variant	`commit`

Applying Multiple Variants in a Model

When you use multiple variants during a simulation, and 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.

If you specify variants as arguments to `sbiosimulate`, this applies the variants for the current simulation in the order that they are specified, and supersedes any active variant objects on the model.

Similarly, in the variant contents (`Content` property), if there are duplicate specifications for a property's value, the last occurrence for the property in the contents is used during simulation.

Dose Object

A `dose object` represents one or more increases to the amount of a species during a model simulation. It lets you increase the amount of a species during a simulation, either at specific times or time intervals. The increase in the amount of a species occurs only during a simulation, and does not alter the species' value permanently.

For example, you can use a dose to represent the addition of a drug to a model every two hours.

There are two types of `dose objects`:

- `ScheduleDose object` — Applies a dose to a single species at a pre-defined list of times.
- `RepeatDose object` — Repeatedly applies a dose to a single species at regularly spaced time intervals.

Repeat doses and schedule doses support three kinds of increases to species amounts:

- Instantaneous increase (or step change) in the amount of a species
- Increase at a fixed rate over a period of time calculated from the dose amount
- Increase at a fixed rate calculated from the dose amount and dose duration

By using repeat doses and schedule doses with SimBiology models, you can easily model common dosing strategies, such as bolus, infusion, zero-order absorption, and first-order absorption.

Creating and Using Dose Objects Associated With a Model

1. Create a dose and add it to the model using the `adddose` method.
2. Configure the properties of the dose. For example, set the `TargetName` property to the name of the species in the model that will receive the dose. Set the `Active` property to true to use it during a simulation.
3. Enter the model as an input argument to `sbiosimulate` to apply the dose.

SimBiology will simulate the model using all active doses associated with the model.

For information about...	See...
Creating and adding a dose to a model	`adddose`
Creating a stand-alone dose	`sbiodose`
Methods and properties of a dose	`ScheduleDose object` and `RepeatDose object`

Example — Creating A Model That Changes a Species Amount Using an Event

In this section...

Overview

About the Example Model

This example uses the model described in Model of the Yeast Heterotrimeric G Protein Cycle.

This table shows the reactions used to model the G protein cycle and the corresponding rate parameters (rate constants) for each reaction. For reversible reactions, the forward rate parameter is listed first.

No.	Name	Reaction[1]	Rate Parameters
1	Receptor-ligand interaction	`L + R <-> RL`	`kRL`, `kRLm`
2	Heterotrimeric G protein formation	`Gd + Gbg -> G`	`kG1`
3	G protein activation	`RL + G -> Ga + Gbg + RL`	`kGa`
4	Receptor synthesis and degradation	`R <-> null`	`kRdo`, `kRs`
5	Receptor-ligand degradation	`RL -> null`	`kRD1`

No.	Name	Reaction[1]	Rate Parameters
6	G protein inactivation	`Ga -> Gd`	`kGd`

[1] Legend of species: L = ligand (alpha factor), R = alpha-factor receptor, Gd = inactive G-alpha-GDP, Gbg = free levels of G-beta:G-gamma complex, G = inactive Gbg:Gd complex, Ga = active G-alpha-GTP

About the Example

This example shows how to add an event to a model to trigger a time-based change. The event modifies the amount of ligand (`L`), thus modeling a delay in the addition of α-factor to the cell culture.

For information on events and how they are evaluated, see "Event Object" on page 1-26.

Opening the Example Model

Load the `gprotein` example project, which includes the variable `m1`, a model object:

```
sbioloadproject gprotein
```

The `m1` model object appears in the MATLAB Workspace.

Adding an Event to the Example Model

1. Set the `InitialAmount` of species `L` (ligand) to be `0.0` when the simulation starts:

   ```
   speciesObj = sbioselect(m1, 'Type', 'species', 'Name', 'L');
   set(speciesObj, 'InitialAmount', 0);
   ```

2. Add an event to the `m1` model object. Configure the event to set the amount of species `L` (ligand) and to trigger when the simulation time equals `100`:

   ```
   evt = addevent(m1, 'time >= 100', 'L = 6.022E17');
   ```

Simulating the Model

1 Configure the simulation settings (`configset object`) for the `m1` model object to log all states during the simulation:

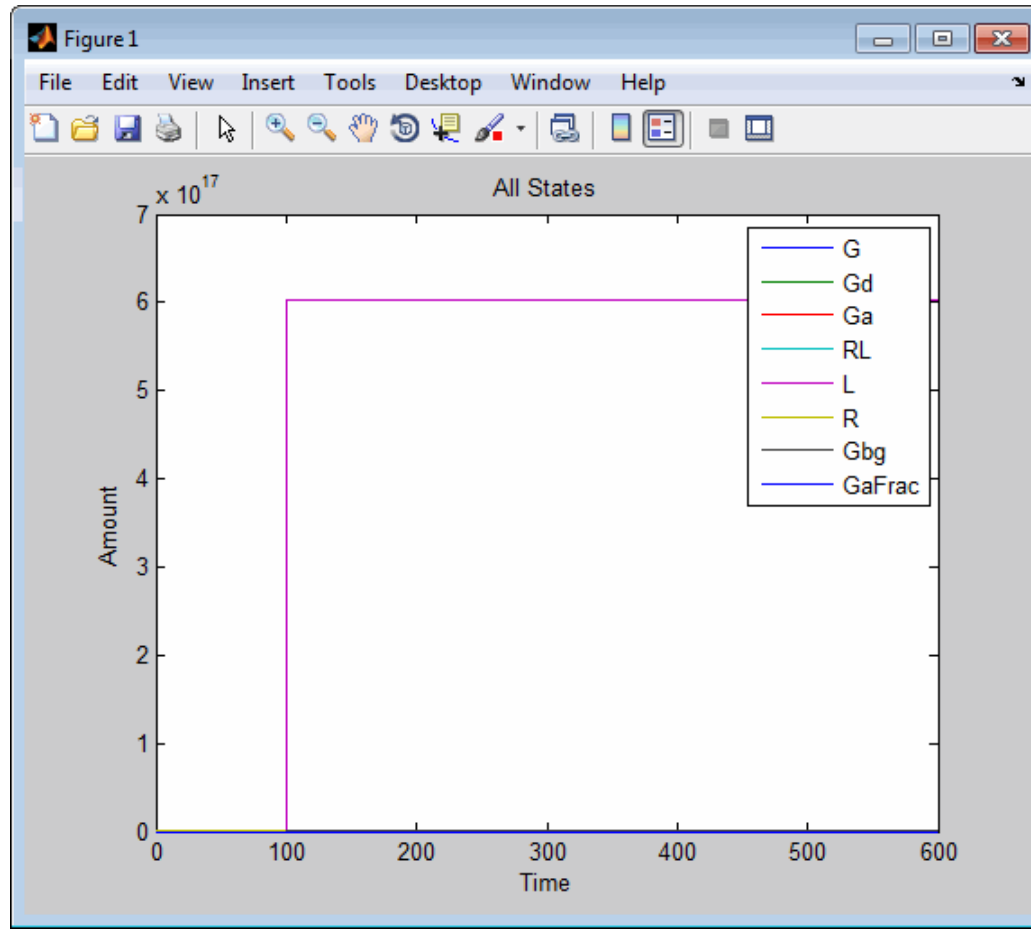
```
cs = getconfigset(m1);
set(cs.RuntimeOptions, 'StatesToLog', 'all');
```

2 Simulate the model:

```
[t,x,names] = sbiosimulate(m1);
```

3 Plot the results:

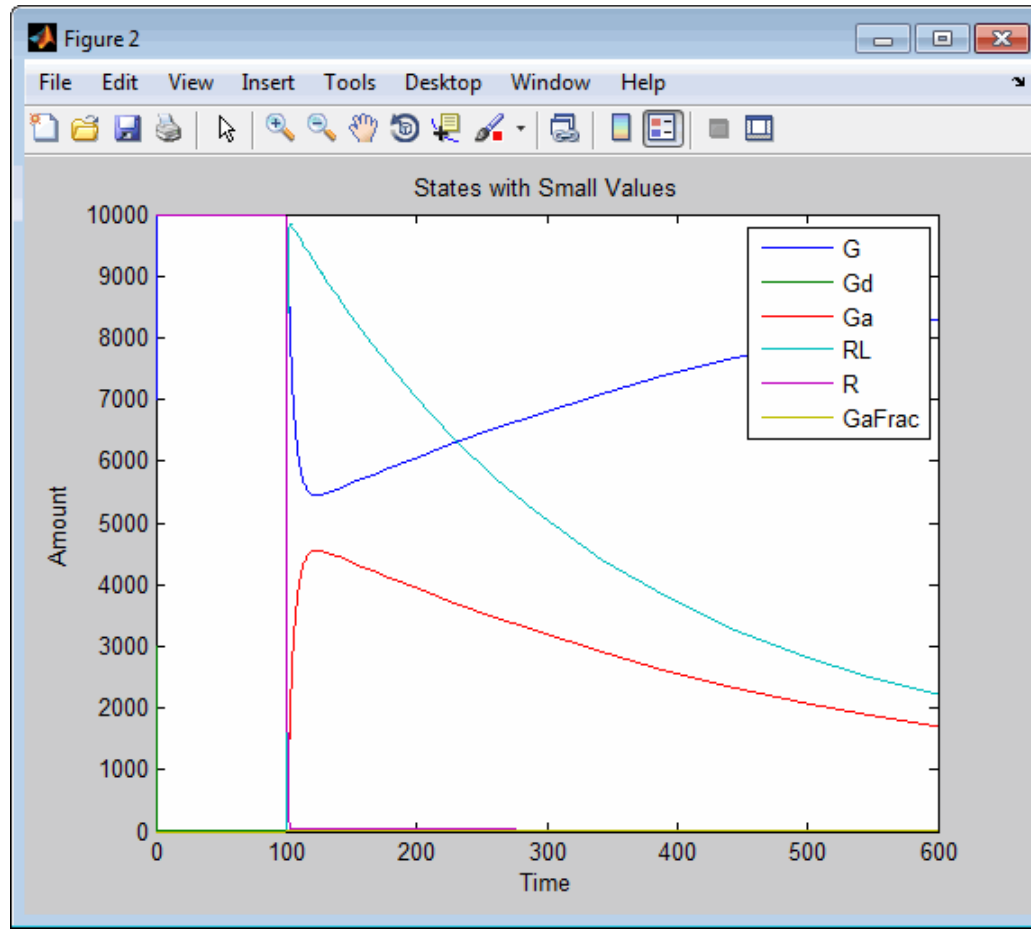
```
plot(t,x)
legend(names)
xlabel('Time'); ylabel('Amount');title('All States')
```



Notice that the species `L` amount increases when the event triggers at simulation time `100`. The plot does not show the other species in the model due to the wide range in species amounts.

4 To see plots of the species with smaller amounts, plot all species except the 5th (species `L`) and 7th (species `Gbg`).

```
figure; plot(t, x(:, [1:4 6 8]))
legend(names{[1:4 6 8]})
xlabel('Time'); ylabel('Amount');title('States with Small Values')
```



Notice the increase in the activation of G protein (species `Ga`, shown in red) after ligand (species `L`) is added at simulation time `100`.

Example — Creating A Model That Changes a Parameter Value Using a Variant

In this section...

Overview

About the Example Model

This example uses the model described in Model of the Yeast Heterotrimeric G Protein Cycle.

This table shows the reactions used to model the G protein cycle and the corresponding rate parameters (rate constants) for each reaction. For reversible reactions, the forward rate parameter is listed first.

No.	Name	Reaction[1]	Rate Parameters
1	Receptor-ligand interaction	`L + R <-> RL`	`kRL`, `kRLm`
2	Heterotrimeric G protein formation	`Gd + Gbg -> G`	`kG1`
3	G protein activation	`RL + G -> Ga + Gbg + RL`	`kGa`
4	Receptor synthesis and degradation	`R <-> null`	`kRdo`, `kRs`
5	Receptor-ligand degradation	`RL -> null`	`kRD1`

No.	Name	Reaction[1]	Rate Parameters
6	G protein inactivation	`Ga -> Gd`	`kGd`

[1] Legend of species: L = ligand (alpha factor), R = alpha-factor receptor, Gd = inactive G-alpha-GDP, Gbg = free levels of G-beta:G-gamma complex, G = inactive Gbg:Gd complex, Ga = active G-alpha-GTP

About the Example

This example shows how to apply a variant to a model that contains a parameter value for a G protein cycle in a wild-type strain. The variant represents a parameter value for a G protein cycle in a mutant strain. Thus, when you simulate the model without applying the variant, you see results for the wild type strain, and when you simulate the model with the variant, you see results for the mutant strain.

About the Variant Created in This Example

The value of the parameter `kGd` is `0.11` for the wild-type strain and `0.004` for the mutant strain. To represent the mutant strain, store an alternate value of `0.004` for the `kGd` parameter in a variant. Then apply this variant when simulating the model.

For information on variants, see “Variant Object” on page 1-35.

Opening the Example Model

Load the `gprotein` example project, which includes the variable `m1`, a model object:

```
sbioloadproject gprotein
```

The `m1` model object appears in the MATLAB Workspace.

Applying an Alternate Parameter Value Using a Variant

1 Add a variant to the `m1` model object:

```
v1 = addvariant(m1, 'mutant_strain');
```

2 Add content to the variant object `v1`. Specifically, add a parameter `kGd` with a value of `0.004`:

```
addcontent(v1, {'parameter', 'kGd', 'Value', 0.004});
```

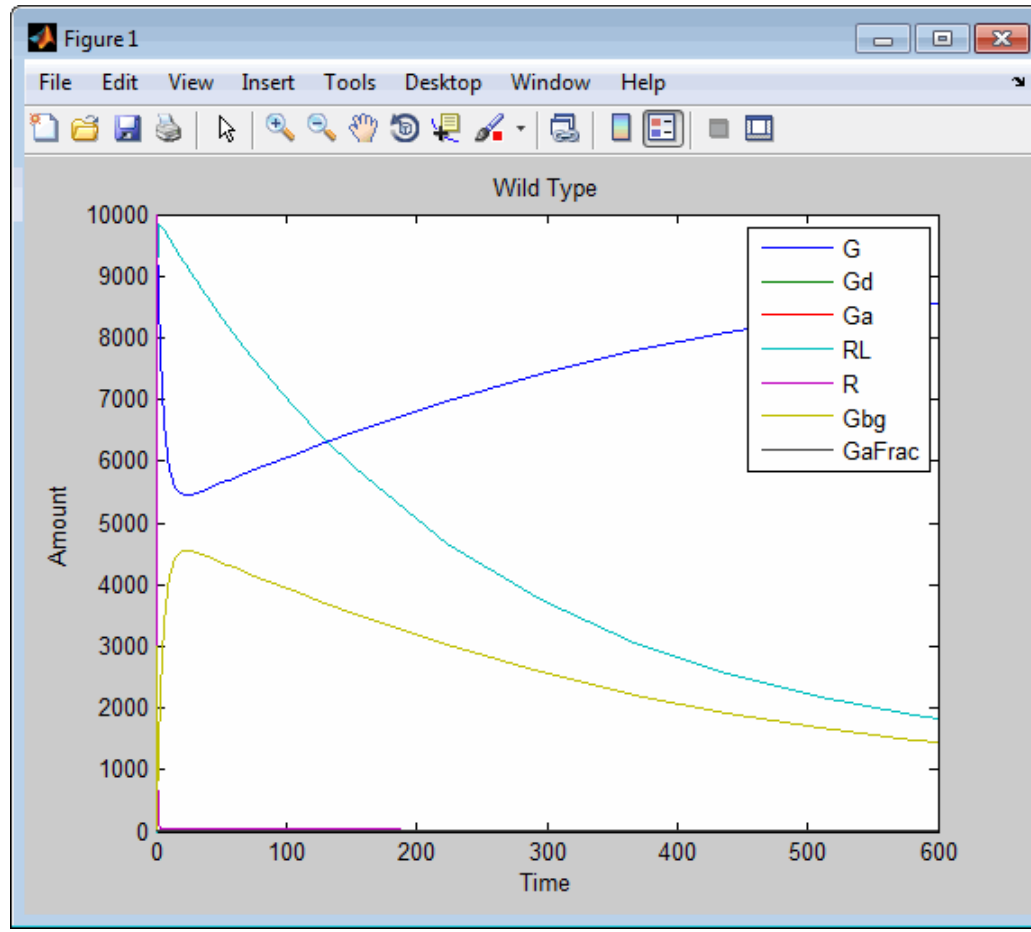
Simulating the Model With and Without the Variant

1 Simulate the model of the wild-type strain, that is by not applying the variant:

```
[t,x,names] = sbiosimulate(m1);
```

2 Plot the results of the wild-type strain (no variant):

```
plot(t,x)
legend(names)
xlabel('Time'); ylabel('Amount'); title('Wild Type')
```



3 Simulate the model of the mutant strain by applying the variant:

```
[t,x,names] = sbiosimulate(m1, v1);
```

4 Plot the results of the mutant strain (variant):

```
figure; plot(t,x)
legend(names)
xlabel('Time'); ylabel('Amount'); title('Mutant')
```

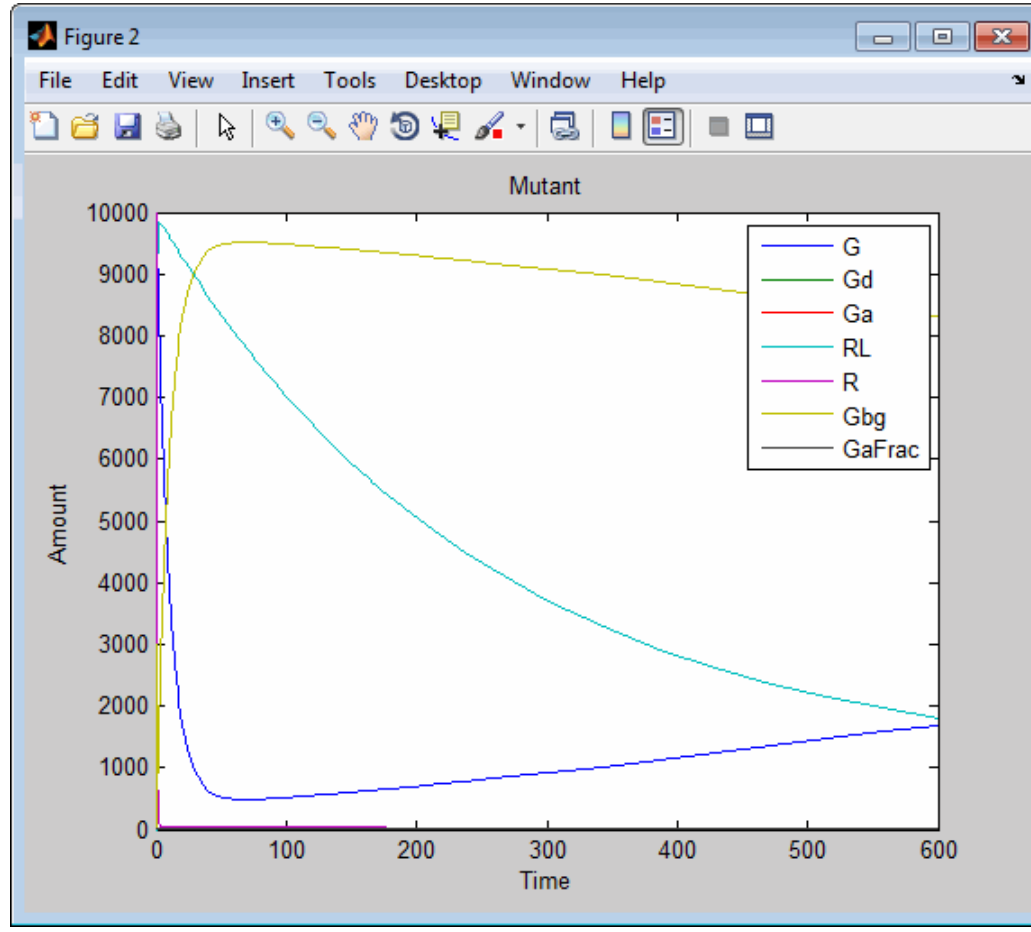


Figure 2
File Edit View Insert Tools Desktop Window Help
Mutant
Amount
Time
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
0
100
200
300
400
500
600
G
Gd
Ga
RL
R
Gbg
GaFrac

Example — Creating A Model That Uses a User-Defined Function in an Expression

In this section...

Overview

Prerequisites for the Example

This example assumes you have a working knowledge of:

- MATLAB desktop
- Creating and saving MATLAB programs

About the Example Model

This example uses the model described in Model of the Yeast Heterotrimeric G Protein Cycle.

This table shows the reactions used to model the G protein cycle and the corresponding rate parameters (rate constants) for each reaction. For reversible reactions, the forward rate parameter is listed first.

No.	Name	Reaction[1]	Rate Parameters
1	Receptor-ligand interaction	`L + R <-> RL`	`kRL`, `kRLm`
2	Heterotrimeric G protein formation	`Gd + Gbg -> G`	`kG1`
3	G protein activation	`RL + G -> Ga + Gbg + RL`	`kGa`
4	Receptor synthesis and degradation	`R <-> null`	`kRdo`, `kRs`
5	Receptor-ligand degradation	`RL -> null`	`kRD1`
6	G protein inactivation	`Ga -> Gd`	`kGd`

[1] Legend of species: L = ligand (alpha factor), R = alpha-factor receptor, Gd = inactive G-alpha-GDP, Gbg = free levels of G-beta:G-gamma complex, G = inactive Gbg:Gd complex, Ga = active G-alpha-GTP

Assumptions of the Example

This example assumes that:

- An inhibitor (`Inhib` species) slows the inactivation of the active G protein (reaction 6 above, `Ga > Gd`).
- The variation in the amount of inhibitor (`Inhib` species) is defined in a custom function, `inhibvalex`.
- The inhibitor (`Inhib` species) affects the reaction by changing the amount of rate parameter `kGd`.

About the Example

This example shows how to create and call a custom function in a SimBiology expression. Specifically, it shows how to use a custom function in a rule expression.

About Using Custom Functions in SimBiology Expressions

You can use custom functions in:

- Reaction rate expressions (`ReactionRate` property)
- Rule expressions (`Rule` property)
- Event expressions (`EventFcns` property or `Trigger` property)

The requirements for using custom functions in SimBiology expressions are:

- Create a custom function. For more information, see `function`.
- Change the current folder to the folder containing your custom MATLAB file. Do this by using the `cd` command or by using the Current Folder field in the MATLAB desktop toolbar. Alternatively, add the folder containing your file to the search path. Do this by using the `addpath` command or see "Adding Folders to the Search Path".
- Call the custom function in a SimBiology reaction, rule, or event expression.

Tip If your rule or reaction rate expression is not continuous and differentiable, see "Using Events to Address Discontinuities in Rule and Reaction Rate Expressions" on page 1-34 before simulating your model.

Creating a Custom Function

The following procedure creates a custom function, `inhibvalex`, which lets you specify how the inhibitor amount changes over time. The inputs are time, the initial amount of inhibitor, and a parameter that governs the amount of inhibitor. The output of the function is the amount of inhibitor.

1. In the MATLAB desktop, select **File > New > Script**, to open the MATLAB Editor.
2. Copy and paste the following function declaration:

```
% inhibvalex.m
function Cp = inhibvalex(t, Cpo, kel)

% This function takes the input arguments t, Cpo, and kel
% and returns the value of the inhibitor Cp.
% You can later specify the input arguments in a
% SimBiology rule expression.
```

```
% For example in the rule expression, specify:
% t as time (a keyword recognized as simulation time),
% Cpo as a parameter that represents the initial amount of inhibitor,
% and kel as a parameter that governs the amount of inhibitor.

if  t < 400
    Cp = Cpo*exp(-kel*(t));
else
    Cp = Cpo*exp(-kel*(t-400));
end
```

3 Save the file (name the file `inhibvalex.m`) in a directory that is on the MATLAB search path, or to a directory that you can access.

4 If the location of the file is not on the MATLAB search path, change the working directory to the file location.

Opening the Example Model

Load the `gprotein` example project, which includes the variable `m1`, a model object:

```
sbioloadproject gprotein
```

The `m1` model object appears in the MATLAB Workspace.

Adding the Custom Function to the Example Model

The following procedure creates a rule expression that calls the custom function, `inhibvalex`, and specifies the three input values to this function.

1 Add a repeated assignment rule to the model that specifies the three input values to the custom function, `inhibvalex`:

```
rule1 = addrule(m1, 'Inhib = inhibvalex(time, Cpo, Kel)',...
                'repeatedAssignment');
```

The `time` input is a SimBiology keyword recognized as simulation time

2 Create the two parameters used by the `rule1` rule and assign values to them:

```
p1 = addparameter(m1, 'Cpo', 250);
p2 = addparameter(m1, 'Kel', 0.01);
```

3 Create the species used by the `rule1` rule:

```
s1 = addspecies(m1.Compartments, 'Inhib');
```

Defining a Rule to Affect Parameter Value

The value of rate parameter `kGd` is affected by the amount of inhibitor present in the system. Add a rule to the model to describe this action, but first change the `ConstantValue` property of the parameter `kGd` so that it can be varied by a rule.

1 Change the `ConstantValue` property of the `kGd` parameter to `false`.

```
p3 = sbioselect(m1, 'Type', 'parameter', 'Name', 'kGd');
set(p3, 'ConstantValue', false)
```

2 Add a repeated assignment rule to the model to define how the `kGd` parameter is affected by the `Inhib` species.

```
rule2 = addrule(m1, 'kGd = 1/Inhib', 'repeatedAssignment');
```

Adding an Event to Reset the Solver at a Discontinuity

The custom function, `inhibvalex`, introduces a discontinuity in the model when time = 400. To ensure accurate simulation results, add an event to the model to reset the solver at the time of the discontinuity. Set the event to trigger at the time of the discontinuity (time = 400). The event does not need to modify the model, so create an event function that multiplies a species value by 1:

```
addevent(m1, 'time>=400', 'G=1*G');
```

Simulating the Modified Model

1 Configure the simulation settings (`configset object`) for the `m1` model object to log all states during the simulation:

```
cs = getconfigset(m1);
set(cs.RuntimeOptions, 'StatesToLog', 'all')
```

2 Simulate the model:

```
simDataObj = sbiosimulate(m1);
```

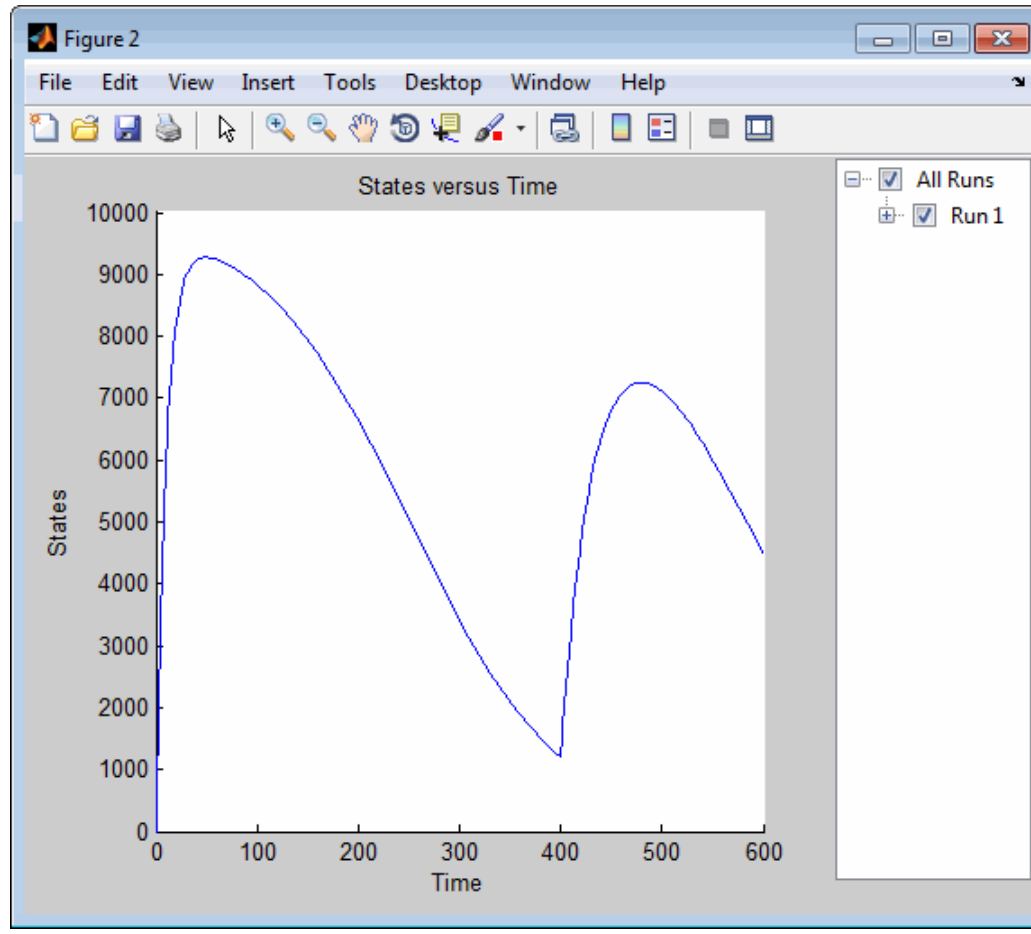
3 Plot the results:

```
sbioplot(simDataObj)
```

The plot does not show the species of interest due to the wide range in species amounts/concentrations.

4 Plot only the species of interest. `Ga`:

```
GaSimDataObj = selectbyname(simDataObj, 'Ga');
sbioplot(GaSimDataObj)
```



Notice the change in the profile of species `Ga` at time = `400` seconds (simulation time). This is the time when the inhibitor amount is changed to reflect the re-addition of inhibitor to the model.

5 Plot only the inhibitor (`Inhib` species):

```
InhibSimDataObj = selectbyname(simDataObj, 'Inhib');
sbioplot(InhibSimDataObj)
```

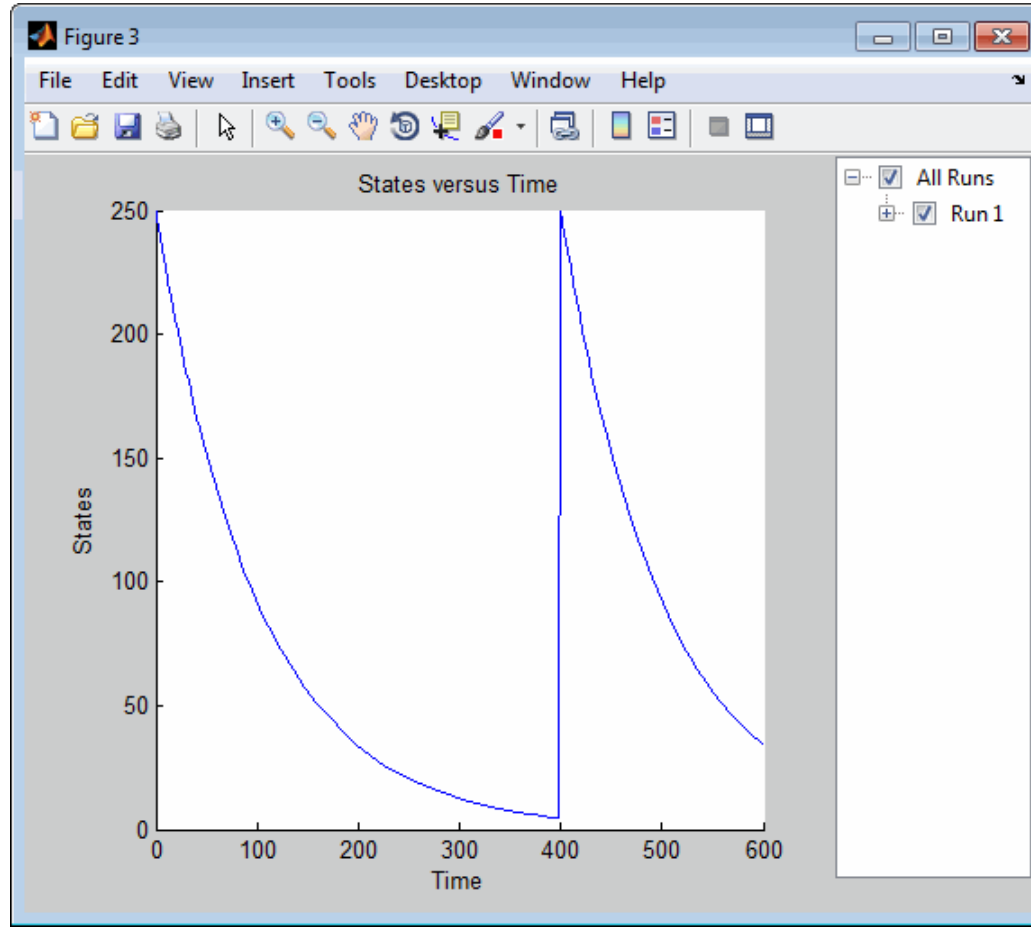


Figure 3
File Edit View Insert Tools Desktop Window Help
States versus Time
All Runs
Run 1
250
200
150
100
50
0
States
0
100
200
300
400
500
600
Time

See Also

To learn about...	Refer to...
Functions	`function` in the *MATLAB Function Reference*.
Changing the working directory to the directory containing the function file	`cd` in the *MATLAB Function Reference*.
Adding the directory containing function files to the MATLAB search path	`addpath` in the *MATLAB Function Reference* or "Adding Folders to the Search Path" in the *MATLAB Desktop Tools and Development Environment*.

2

Structural Analysis

Overview of Structural Analysis

Structural analyses let you verify and investigate the structure of your model and its quantities and expressions before actually simulating the model. These static inspections help you to:

- Confirm the model is ready for simulation.
- Better understand the relationships between quantities and expressions in the model.

Verifying a Model

In this section...

What is Model Verification?

SimBiology has functionality that helps you find and fix warnings that you might need to be aware of, and errors that would prevent you from simulating and analyzing your model.

Model verification checks many aspects of the model including:

- Model structure
- Validity of mathematical expressions
- Dimensional analysis
- Unit conversion issues

When to Verify a Model

You can check your model for warnings and errors at any time when constructing or working with your model. For example:

- Verify your model during construction to ensure that the model is complete.
- Verify the model after changing simulation settings, dimensional analysis settings, or unit conversion settings.

Analyses such as simulation, scanning, and parameter fitting automatically verify a model.

Tip Repeatedly running a task using a different variant or setting a different value for the `InitialAmount` property of a species, the `Capacity` property of a compartment, or the `Value` property of a parameter, generates warnings only the first time you simulate a model. Use the verification functionality described in this section to display warnings again.

Verifying That a Model Has No Warnings or Errors

Use the `verify` method to see a list of warnings and errors in your model.

Use the `sbiolastwarning` and `sbiolasterror` functions to return the last warning and last error encountered during verification.

See Also

For an example of verifying a model, see "Example — Verifying a Model" on page 2-5.

Example — Verifying a Model

1 Create a model with a reaction that references `K1`, an undefined parameter:

```
% Create a model named example
model = sbiomodel('example');
% Add a compartment named cell to model
compartment = addcompartment(model, 'cell');
% Add two species, A and B, to the cell compartment
species_1 = addspecies(compartment, 'A');
species_2 = addspecies(compartment, 'B');
% Add the reaction A -> B to the model
reaction = addreaction(model, 'A -> B', 'ReactionRate', 'K1');
```

2 Verify the model to check for warnings and errors:

```
verify(model)
```

```
??? --> Error reported from Expression Validation:
The name 'K1' in reaction 'A -> B' does not refer to any in-scope species,
parameters, or compartments.
```

3 Address the error by defining the parameter `K1`:

```
% Add a parameter, K1, to the model with a value of 3
parameter = addparameter(model, 'K1', 3);
```

4 Verify the model again:

```
verify(model)
```

Determining Conserved Moieties

In this section...

Introduction to Moiety Conservation

Conserved moieties represent quantities that are conserved in a system, regardless of the individual reaction rates.

Consider this simple network:

```
reaction 1: A -> B
reaction 2: B -> C
reaction 3: C -> A
```

Regardless of the rates of reactions 1, 2, and 3, the quantity `A + B + C` is conserved throughout the dynamic evolution of the system. This conservation is termed structural because it depends only on the structure of the network, rather than on details such as the kinetics of the reactions involved. In the context of systems biology, such a conserved quantity is sometimes referred to as a conserved moiety. A typical, real-world example of a conserved moiety is adenine in its various forms ATP, ADP, AMP, etc. Finding and analyzing conserved moieties can yield insights into the structure and function of a biological network. In addition, for the quantitative modeler, conserved moieties represent dependencies that can be removed to reduce a system's dimensionality, or number of dynamic variables. In the previous simple network, in principle, it is only necessary to calculate the time courses for `A` and `B`. After this is done, `C` is fixed by the conservation relation.

Algorithms for Conserved Cycle Calculations

The `sbioconsmoiety` function analyzes conservation relationships in a model by calculating a complete set of linear conservation relations for the species in the model object.

`sbioconsmoiety` lets you specify one of three algorithms based on the nature of the model and the required results:

- `'qr'` — `sbioconsmoiety` uses an algorithm based on `QR` factorization. From a numerical standpoint, this is the most efficient and reliable approach.
- `'rreduce'` — `sbioconsmoiety` uses an algorithm based on row reduction, which yields better numbers for smaller models. This is the default.
- `'semipos'` — `sbioconsmoiety` returns conservation relations in which all the coefficients are greater than or equal to zero, 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 can return a different number of relations. Mathematically speaking, this algorithm returns a generating set of vectors for the space of semipositive conservation relations.

In some situations, you may be interested in the dimensional reduction of your model via conservation relations. Recall the simple model, presented in "Introduction to Moiety Conservation" on page 2-6, that contained the conserved cycle `A + B + C`. Given `A` and `B`, `C` is determined by the conservation relation; the system can be thought of as having only two dynamic variables rather than three. The `'link'` algorithm specification caters to this situation. In this case, `sbioconsmoiety` partitions the species in the model into independent and dependent sets and calculates the dependence of the dependent species on the independent species.

Consider a general system with an `n`-by-`m` stoichiometry matrix `N` of rank `k`, and suppose that the rows of `N` are permuted (which is equivalent to permuting the species ordering) so that the first `k` rows are linearly independent. The last `n – k` rows are then necessarily dependent on the first `k` rows.

The matrix `N` can be split into the following independent and dependent parts,

$$\mathrm{N} = \begin{pmatrix} N_R \\ N_D \end{pmatrix}$$

where R in the independent submatrix N_R denotes 'reduced'; the (n – k)-by-k link matrix L0 is defined so that N_D = L0*N_R. In other words, the link matrix gives the dependent rows N_D of the stoichiometry matrix, in terms of the independent rows N_R. Because each row in the stoichiometry matrix corresponds to a species in the model, each row of the link matrix encodes how one dependent species is determined by the k independent species.

See Also

For examples of determining conserved moieties, see:

- "Example — Determining Conserved Moieties" on page 2-9
- The demo Finding Conserved Quantities in a Pathway Model

Example — Determining Conserved Moieties

1 Load the Goldbeter Mitotic Oscillator project, which includes the variable `m1`, a model object:

```
sbioloadproject Goldbeter_Mitotic_Oscillator_with_reactions
```

The `m1` model object appears in the MATLAB Workspace.

2 Display the species information:

```
m1.Compartments.Species

SimBiology Species Array

Index:  Compartment:  Name:  InitialAmount:  InitialAmountUnits:
 1       unnamed       C        0.01
 2       unnamed       M        0.01
 3       unnamed       Mplus    0.99
 4       unnamed       Mt       1
 5       unnamed       X        0.01
 6       unnamed       Xplus    0.99
 7       unnamed       Xt       1
 8       unnamed       V1       0
 9       unnamed       V3       0
 10      unnamed       AA       0
```

3 Display the reaction information:

```
m1.Reactions

SimBiology Reaction Array

   Index:    Reaction:
   1         AA -> C
   2         C -> AA
   3         C + X -> AA + X
   4         Mplus + C -> M + C
   5         M -> Mplus
   6         Xplus + M -> X + M
   7         X -> Xplus
```

4 Use the simplest form of the `sbioconsmoiety` function and display the results. The default call to `sbioconsmoiety`, in which no algorithm is specified, uses an algorithm based on row reduction.

```
[g sp] = sbioconsmoiety(m1)

g =

     0     1     1     0     0     0
     0     0     0     1     1     0
     0     0     0     0     0     1


sp =

    'C'
    'M'
    'Mplus'
    'X'
    'Xplus'
    'AA'
```

The columns in `g` are labeled by the species `sp`. Thus the first row describes the conserved relationship, `M + Mplus`. Notice that the third row indicates that the species `AA` is conserved, which is because `AA` is constant (`ConstantAmount = 1`).

5 Call `sbioconsmoiety` again, this time specifying the semipositive algorithm to explore conservation relations in the model. Also specify to return the conserved moieties in a cell array of strings, instead of a matrix.

```
cons_rel = sbioconsmoiety(m1,'semipos','p')

cons_rel =

    'AA'
    'X + Xplus'
    'M + Mplus'
```

6 Use the `'link'` option to study the dependent and independent species.

```
[SI,SD,LO,NR,ND] = sbioconsmoiety(m1, 'link');
```

7 Show the list of independent species:

```
SI

SI =

    'C'
    'M'
    'X'
```

8 Show the list of dependent species:

```
SD

SD =

    'Mplus'
    'Xplus'
    'AA'
```

9 Show the link matrix relating `SD` and `SI` by converting the `LO` output from a sparse matrix to a full matrix:

```
LO_full = full(LO)

LO_full =

         0   -1.0000         0
         0         0   -1.0000
         0         0         0
```

10 Show the independent stoichiometry matrix, N_R by converting the `NR` output from a sparse matrix to a full matrix:

```
NR_full = full(NR)

NR_full =

     1    -1    -1     0     0     0     0
     0     0     0     1    -1     0     0
```

```
     0     0     0     0     0     1    -1
```

11 Show the dependent stoichiometry matrix, N_D by converting the `ND` output from a sparse matrix to a full matrix:

```
ND_full = full(ND)

ND_full =

     0     0     0    -1     1     0     0
     0     0     0     0     0    -1     1
     0     0     0     0     0     0     0
```

Determining the Adjacency Matrix for a Model

In this section...

What Is an Adjacency Matrix?

An *adjacency matrix* lets you easily determine:

- The reactants and products in a specific reaction in a model
- The reactions that a specific species is part of, and whether the species is a reactant or product in that reaction

An adjacency matrix is an *N*-by-*N* matrix, where *N* equals the total number of species and reactions in a model. Each row corresponds to a species or reaction, and each column corresponds to a species or reaction.

The matrix indicates which species and reactions are involved as reactants and products:

- Reactants are represented in the matrix with a `1` at the appropriate location (row of species, column of reaction). Reactants appear above the diagonal.
- Products are represented in the matrix with a `1` at the appropriate location (row of reaction, column of species). Products appear below the diagonal.
- All other locations in the matrix contain a `0`.

For example, if a `model object` contains one reaction equal to `A + B -> C` and the `Name` property of the reaction is `R1`, the adjacency matrix is:

```
      A     B     C    R1
A     0     0     0     1
B     0     0     0     1
C     0     0     0     0
R1    0     0     1     0
```

Example — Retrieving an Adjacency Matrix for a Model

Retrieve an adjacency matrix for a model by passing the `model object` as an input argument to the `getadjacencymatrix` method.

1. Read in `m1`, a model object, using `sbmlimport`:

```
m1 = sbmlimport('lotka.xml');
```

2. Get the adjacency matrix for `m1`:

```
[M, Headings] = getadjacencymatrix(m1)

M =

   (5,1)        1
   (5,2)        1
   (6,3)        1
   (7,4)        1
   (1,5)        1
   (2,5)        1
   (2,6)        1
   (3,6)        1
   (3,7)        1


Headings =

    'x'
    'y1'
    'y2'
    'z'
    'Reaction1'
    'Reaction2'
    'Reaction3'
```

3. Convert the adjacency matrix from a sparse matrix to a full matrix to more easily see the relationships between species and reactions:

```
M_full = full(M)
```

```
M_full =

     0     0     0     0     1     0     0
     0     0     0     0     1     1     0
     0     0     0     0     0     1     1
     0     0     0     0     0     0     0
     1     1     0     0     0     0     0
     0     0     1     0     0     0     0
     0     0     0     1     0     0     0
```

Determining the Stoichiometry Matrix for a Model

In this section...

What Is a Stoichiometry Matrix?

A *stoichiometry matrix* lets you easily determine:

- The reactants and products in a specific reaction in a model, including the stoichiometric value of the reactants and products
- The reactions that a specific species is part of, and whether the species is a reactant or product in that reaction

A stoichiometry matrix is an *M*-by-*N* matrix, where *M* equals the total number of reactions in a model, and *N* equals the total number of species in a model. Each row corresponds to a reaction, and each column corresponds to a species.

The matrix indicates which species and reactions are involved as reactants and products:

- Reactants are represented in the matrix with their stoichiometric value at the appropriate location (row of reaction, column of species). Reactants appear as negative values.
- Products are represented in the matrix with their stoichiometric value at the appropriate location (row of reaction, column of species). Products appear as positive values.
- All other locations in the matrix contain a `0`.

For example, consider a `model object` containing two reactions. One reaction (named `R1`) is equal to `2 A + B -> 3 C`, and the other reaction (named `R2`) is equal to `B + 3 D -> 4 A`. The stoichiometry matrix is:

```
      A    B   C    D
R1   -2   -1   3    0
R2    4   -1   0   -3
```

Example — Retrieving a Stoichiometry Matrix for a Model

Retrieve a stoichiometry matrix for a model by passing the `model object` as an input argument to the `getstoichmatrix` method.

1 Read in `m1`, a model object, using `sbmlimport`:

```
m1 = sbmlimport('lotka.xml');
```

2 Get the stoichiometry matrix for `m1`:

```
[M,objSpecies,objReactions] = getstoichmatrix(m1)

M =

   (2,1)        1
   (2,2)       -1
   (3,2)        1
   (3,3)       -1
   (4,3)        1


objSpecies =

    'x'
    'y1'
    'y2'
    'z'


objReactions =

    'Reaction1'
    'Reaction2'
    'Reaction3'
```

3 Convert the stoichiometry matrix from a sparse matrix to a full matrix to more easily see the relationships between species and reactions:

```
M_full = full(M)

M_full =

     0     0     0
     1    -1     0
     0     1    -1
     0     0     1
```

3

Simulation and Analysis

Overview of Simulation and Analysis

In this section...

After creating models in SimBiology, you can simulate and analyze them.

Typical Workflow

To simulate a model, SimBiology:

1. Converts the model expressions and quantities to a system of differential equations.
2. Uses deterministic or stochastic solvers to numerically solve these equations.
3. Determines the changes in species amounts and parameter values over time.

SimBiology also lets you analyze models. These analyses include a basic simulation of the model as well as additional evaluations such as:

- Calculate sensitivities
- Estimate parameters

See Also

There are demos that show you how to use the command line to simulate and analyze:

- To access demos included with the software, see SimBiology Demos.
- To access demos on the Web, see SimBiology Demos and Webinars.

Simulating Models

In this section...

Simulating a Model Using sbiosimulate

Simulate a model by providing the `model object` as an input argument to the `sbiosimulate` function.

When you simulate a model, you can return results (time points, state data, and state names) in two forms:

- Three separate arrays
- One `SimData object`

A `SimData object` also includes metadata such as the types and names for the logged states, the configuration set used during simulation, and the date of the simulation. It is a convenient way of keeping time data, state data, and associated metadata together. A `SimData object` has associated properties and methods, which you can use to access and manipulate the data.

For more information on simulating a model, see "Example — Simulating a Model and Viewing Results" on page 3-12.

Plotting Simulation Results

If you return time and state data from a simulation in three output arguments, you can use these arguments as inputs to the `plot` function to view your results. For more information, see `sbiosimulate`.

If you return time and state data from a simulation in a `SimData object`, you can use the `SimData object` as an input to the `sbioplot` function to view your results.

For more information on plotting simulation results, see "Example — Simulating a Model and Viewing Results" on page 3-12.

Interpreting Simulation Results

After running a simulation, you may see negative amounts or concentrations for species in the results plot or data array. These negative values can be either:

- Slightly negative due to numerical noise introduced by the simulation process. In this case, you can interpret these values as `0`.
- Significantly negative due to the dynamics in your model not being physical, that is, the dynamics in the system are driving a particular species to be negative. In this case, examine your reaction rate expressions to ensure they implement correct dynamics.

Configuring Stop Time and Other Simulation Settings

A model has a configuration set (`configset object`) associated with it to control the simulation. You can edit the properties of a `configset object` to control all aspects of the simulation, including:

- Simulation stop time
- Simulation time units
- Simulation solver
- Solver error tolerances
- Maximum time step size (ODE solvers only)
- Data to record

- Sensitivity analysis options
- Whether to perform dimensional analysis and unit conversion during simulation

To view the `configset object`, provide the `model object` as an input argument to the `getconfigset` method.

To edit the properties of a `configset object`, use the `set` method.

For more information on viewing and editing the stop time and other simulation settings, see "Example — Simulating a Model and Viewing Results" on page 3-12.

Choosing a Simulation Solver

To simulate a model, the SimBiology software converts a model to a system of differential equations. It then uses a solver function to compute solutions for these equations at different time intervals, giving the model's states and outputs over a span of time.

Available solvers are:

- **ODE Solvers** — These include Nonstiff Deterministic Solvers and Stiff Deterministic Solvers. The solver functions implement numerical integration methods for solving initial value problems for ordinary differential equations (ODEs). Beginning at the initial time with initial conditions, they step through the time interval, computing a solution at each time step. If the solution for a time step satisfies the solver's error tolerance criteria, it is a successful step. Otherwise, it is a failed attempt; the solver shrinks the step size and tries again. For more information, see "ODE Solvers" in the MATLAB Mathematics documentation.
- **SUNDIALS Solvers** — Use with models containing events. At a fundamental level the core algorithms for the SUNDIALS solvers are similar to those for some of the solvers in the MATLAB ODE suite and work as described above in ODE Solvers. For more information, see "SUNDIALS Solvers" on page 3-6.
- **Stochastic Solvers** — Use with models containing a small number of molecules. Stochastic solvers include stochastic simulation algorithm,

explicit tau-leaping algorithm, and implicit tau-leaping algorithm. For more information, see "Stochastic Solvers" on page 3-6.

SUNDIALS Solvers

SUNDIALS (Suite of Nonlinear and Differential/Algebraic Equation Solvers) are part of a freely available third-party package developed at Lawrence Livermore National Laboratory. All other ODE solvers used for simulation of SimBiology models, such as `ode45` and `ode15s`, are part of the MATLAB ODE suite.

When you specify `sundials` for the solver, the software chooses one of two SUNDIALS solvers, CVODE or IDA, as appropriate for your model:

- **CVODE** is a solver for systems of ODEs, both nonstiff and stiff. This is used when a model has no algebraic rules.
- **IDA** is a differential-algebraic equation (DAE) solver, used when one or more algebraic rules are present.

If your model has events and you want to simulate with a deterministic solver, you must select `sundials`. The other ODE solvers do not support events.

For more information on the SUNDIALS solvers, see `http://www.llnl.gov/casc/sundials/description/description.html`.

Stochastic Solvers

When to Use Stochastic Solvers

The stochastic simulation algorithms provide a practical method for simulating reactions that are stochastic in nature. Models with a small number of molecules can realistically be simulated stochastically, that is, allowing the results to contain an element of probability, unlike a deterministic solution.

To use a stochastic solver to simulate a model, ensure all reactions in the model have their `KineticLaw` property set to `MassAction`.

Tip When simulating a model using a stochastic solver, you can increase the `LogDecimation` property of the `configset object` to record fewer data points and decrease run time.

During a stochastic simulation of a model, the software ignores any rate, assignment, or algebraic rules if present in the model. Depending on the model, stochastic simulations may take more computation time than deterministic simulations.

Stochastic Simulation Algorithm (SSA)

The Chemical Master Equation (CME) describes the dynamics of a chemical system in terms of the time evolution of probability distributions. Directly solving for this distribution is impractical for most realistic problems. The stochastic simulation algorithm (SSA) instead efficiently generates individual simulations that are consistent with the CME, by simulating each reaction using its propensity function. Thus, analyzing multiple stochastic simulations to determine the probability distribution is more efficient than directly solving the CME.

Advantage

- This algorithm is exact.

Disadvantages

- Because this algorithm evaluates one reaction at a time, it may be too slow for models with a large number of reactions.

- If the number of molecules of any reactants is huge, it may take a long time to complete the simulation.

Explicit Tau-Leaping Algorithm

Because the stochastic simulation algorithm may be too slow for many practical problems, this algorithm was designed to speed up the simulation at the cost of some accuracy. The algorithm treats each reaction as being independent of the others. It automatically chooses a time interval such that the relative change in the propensity function for each reaction is less than your error tolerance. After selecting the time interval, the algorithm computes the number of times each reaction occurs during the time interval and makes the appropriate changes to the concentration of various chemical species involved.

Advantages

- This algorithm can be orders of magnitude faster than the SSA.
- You can use this algorithm for large problems (if the problem is not numerically stiff).

Disadvantages

- This algorithm sacrifices some accuracy for speed.
- This algorithm is not good for stiff models.
- You need to specify the error tolerance so that the resulting time steps are of the order of the fastest time scale.

Implicit Tau-Leaping Algorithm

Like the explicit tau-leaping algorithm, the implicit tau-leaping algorithm is also an approximate method of simulation designed to speed up the simulation at the cost of some accuracy. It can handle numerically stiff problems better than the explicit tau-leaping algorithm. For deterministic systems, a problem is said to be numerically stiff if there are “fast” and “slow” time scales present in the system. For such problems, the explicit tau-leaping method performs well only if it continues to take small time steps that are of the order of the fastest time scale. The implicit tau-leaping method can potentially take much larger steps and still be stable. The algorithm treats each reaction as being

independent of others. It automatically selects a time interval such that the relative change in the propensity function for each reaction is less than the user specified error tolerance. After selecting a time interval, the algorithm computes the number of times each reaction occurs during the time interval and makes the appropriate changes to the concentration of various chemical species involved.

Advantages

- This algorithm can be much faster than the SSA. It is also usually faster than the explicit tau-leaping algorithm.
- You can use this algorithm for large problems and also for numerically stiff problems.
- The total number of steps taken is usually less than the explicit-tau-leaping algorithm.

Disadvantages

- This algorithm sacrifices some accuracy for speed.
- There is a higher computational burden for each step as compared to the explicit tau-leaping algorithm. This leads to a larger CPU time per step.
- This method often dampens perturbations of the slow manifold leading to a reduced state variance about the mean.

Ensemble Runs of Stochastic Simulations

Because stochastic simulations rely on an element of probability, sequential runs produce different results. Therefore, multiple stochastic runs are needed to determine the probability distribution of the simulation results.

Ensemble runs perform multiple simulations of a model using a stochastic solver. They let you gather data from multiple stochastic runs of the model so you can compare and analyze fluctuations in the behavior of a model over repeated stochastic simulations.

Running Ensemble Simulations. The following functions let you perform and analyze ensemble runs at the command line:

- `sbioensemblerun` — Perform a stochastic ensemble run of the MATLAB model object.
- `sbioensembleplot` — Show a 2-D distribution plot or a 3-D shaded plot of the time varying distribution of one or more specified species.
- `sbioensemblestats` — Get mean and variance as a function of time for all the species in the model used to generate ensemble data by running `sbioensemblerun`.

References

[1] Gibson M.A., Bruck J. (2000), "Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels," Journal of Physical Chemistry, 105:1876–1899.

[2] Gillespie D. (1977), "Exact Stochastic Simulation of Coupled Chemical Reactions," The Journal of Physical Chemistry, 81(25): 2340–2361.

[3] Gillespie D. (2000), "The Chemical Langevin Equation," Journal of Chemical Physics, 113(1): 297–306.

[4] Gillespie D. (2001), "Approximate Accelerated Stochastic Simulation of Chemically Reacting Systems," Journal of Chemical Physics,115(4):1716–1733.

[5] Gillespie D., Petzold L. (2004), "Improved Leap-Size Selection for Accelerated Stochastic Simulation," Journal of Chemical Physics, 119:8229–8234

[6] Rathinam M., Petzold L., Cao Y., Gillespie D. (2003), "Stiffness in Stochastic Chemically Reacting Systems: The Implicit Tau-Leaping Method," Journal of Chemical Physics, 119(24):12784–12794.

[7] Moler, C. (2003), "Stiff Differential Equations Stiffness is a subtle, difficult, and important concept in the numerical solution of ordinary differential equations," MATLAB News & Notes.

See Also

For examples of simulating models, see:

- “Example — Simulating a Model and Viewing Results” on page 3-12
- Analysis of Stochastic Ensemble Data in SimBiology demo

Example — Simulating a Model and Viewing Results

In this section...

Overview

About the Example Model

This example uses the model described in "Model of the Yeast Heterotrimeric G Protein Cycle" on page C-19 to illustrate model simulation.

This table shows the reactions used to model the G protein cycle and the corresponding rate parameters (rate constants) for each mass action reaction. For reversible reactions, the forward rate parameter is listed first.

No.	Name	Reaction[1]	Rate Parameters
1	Receptor-ligand interaction	`L + R <-> RL`	`kRL`, `kRLm`
2	Heterotrimeric G protein formation	`Gd + Gbg -> G`	`kG1`
3	G protein activation	`RL + G -> Ga + Gbg + RL`	`kGa`
4	Receptor synthesis and degradation	`R <-> null`	`kRdo`, `kRs`
5	Receptor-ligand degradation	`RL -> null`	`kRD1`

No.	Name	Reaction[1]	Rate Parameters
6	G protein inactivation	`Ga -> Gd`	`kGd`

[1] Legend of species: L = ligand (alpha factor), R = alpha-factor receptor, Gd = inactive G-alpha-GDP, Gbg = free levels of G-beta:G-gamma complex, G = inactive Gbg:Gd complex, Ga = active G-alpha-GTP

About the Example

This example shows how to configure simulation settings and simulate a model, saving the results in a `SimData object`. It then illustrates how to plot all species in the `SimData object`, as well as plot only the species or parameter of interest, such as:

- `GaFrac` — The fraction of total `Ga` that is active
- `Ga`, `G`, and `Gd` — The species that contain G-alpha units

The example also illustrates how to extract data from the `SimData object` for analysis.

Loading the Example Model

Load the `gprotein.sbproj` project, which includes the variable `m1`, a model object:

```
sbioloadproject gprotein
```

The `m1` model object appears in the MATLAB Workspace.

Configuring Simulation Settings

Set the simulation solver to `ode15s` and set a stop time of `500` by editing the `SolverType` and `StopTime` properties of the `configset object` associated with the `m1` model:

```
csObj = getconfigset(m1);
set(csObj, 'SolverType', 'ode15s', 'StopTime', 500)
```

Simulating the Model

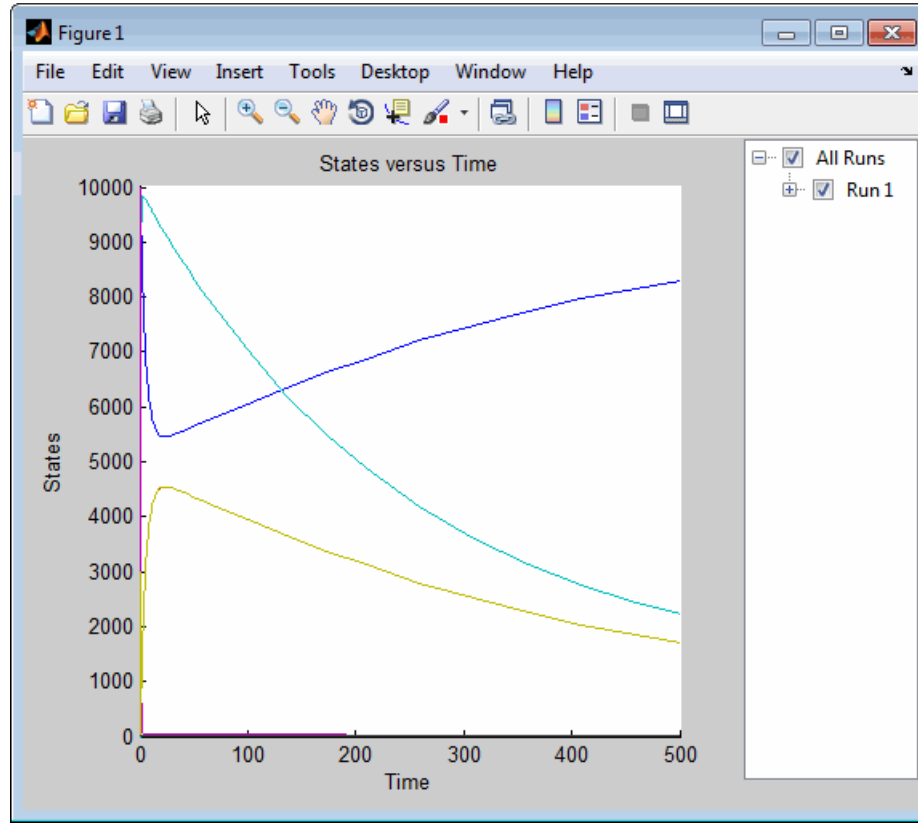
Simulate the model, saving the results in a `SimData object`:

```
simDataObj = sbiosimulate(m1);
```

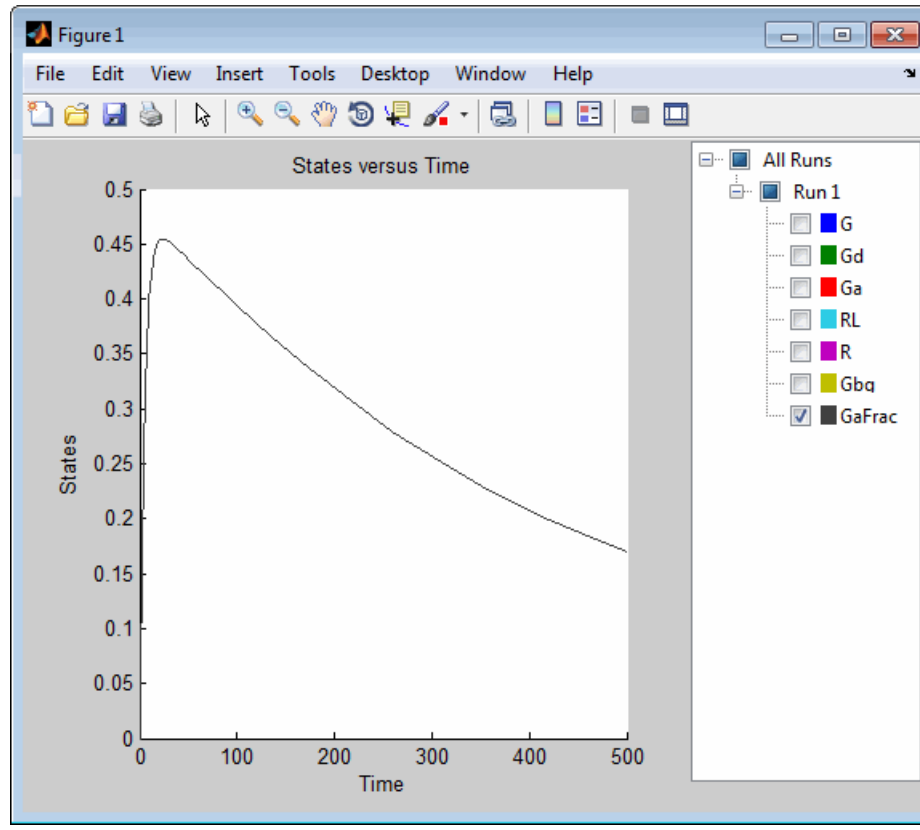
Plotting Simulation Results

1 Plot all the species in the model:

```
sbioplot(simDataObj)
```

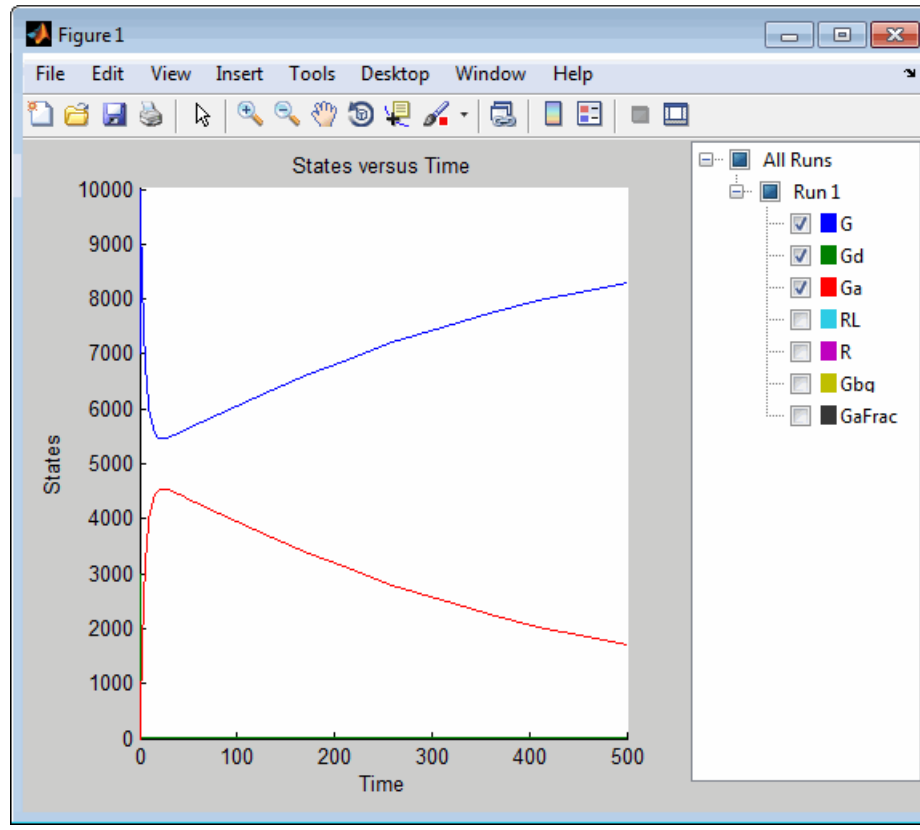


2 Show only the plot of the `GaFrac` parameters by expanding **Run 1**, and then clearing all check boxes other than `GaFrac`:



Notice the scale of the *y*-axis changes as you remove species from the plot.

3 Show only the plots of the species containing G-alpha units by selecting the `G`, `Gd`, and `Ga` check boxes, and clearing the `GaFrac` check box:



4 Alternately, create the previous plots by using the `selectbyname` method to create `SimData objects` containing data for only the species of interest, and then plotting these smaller objects:

```
GafracObj = selectbyname(simDataObj, 'GaFrac');
GaAllObj = selectbyname(simDataObj, {'Ga', 'G', 'Gd'});
sbioplot(GafracObj)
sbioplot(GaAllObj)
```

Extracting Data for Analysis

1 Use the `selectbyname` method to extract arrays containing the time points, state data, and state names for a subset of the data, namely the `Ga`, `G`, and `Gd` species that contain G-alpha units:

```
[times, data, names] = selectbyname(simDataObj, {'Ga', 'G', 'Gd'});
```

2 The previous plot shows that at the start of the simulation, the total G-alpha units (`G + Gd + Ga`) = `10000`. Check the conservation of mass of the G-alpha units by summing their values to see if they equal `10000`:

```
GaTotal = sum(data, 2);
```

3 View the sum of the values for the G-alpha units for the last 10 time points in the simulation:

```
GaTotal(end-9:end)

ans =

  1.0e+004 *

    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
```

Calculating Sensitivities

In this section...

About Calculating Sensitivities

Calculating sensitivities lets you determine which species in a model is most sensitive to a specific condition (for example, a drug). Calculating sensitivities calculates the time-dependent sensitivities of all the species states with respect to species initial conditions and parameter values in the model.

Thus, if a model has a species `x`, and two parameters `y` and `z`, the time-dependent sensitivities of `x` with respect to each parameter value are the time-dependent derivatives

$$\frac{\partial x}{\partial y}, \frac{\partial x}{\partial z}$$

where, the numerator is the sensitivity output and the denominators are the sensitivity inputs to sensitivity analysis.

For more information on the calculations performed, see "References" on page 3-21.

Model Requirements for Calculating Sensitivities

Sensitivity analysis is supported only by the ordinary differential equation (ODE) solvers. The software calculates local sensitivities by combining the original ODE system for a model with the auxiliary differential equations for the sensitivities. The additional equations are derivatives of the original

equations with respect to parameters. This method is sometimes called "forward sensitivity analysis" or "direct sensitivity analysis". This larger system of ODEs is solved simultaneously by the solver.

SimBiology sensitivity analysis uses "complex-step approximation" to calculate derivatives of reaction rates. This technique yields accurate results for the vast majority of typical reaction kinetics, which involve only simple mathematical operations and functions. When a reaction rate involves a nonanalytic function, this technique can lead to inaccurate results. In this case, either sensitivity analysis is disabled, or sensitivity analysis warns you that the computed sensitivities may be inaccurate. An example of such a nonanalytic function is the MATLAB function `abs`. If sensitivity analysis gives questionable results on a model whose reaction rates contain unusual functions, you may be running into limitations of the complex-step method. Contact MathWorks Technical Support for additional information.

Note Models containing the following active components do not support sensitivity analysis:

- Algebraic rules
- Repeated assignment rules
- Rate rules
- Events
- Doses

Setting SolverOptions Properties

Perform sensitivity analysis at the command line by setting the following properties of the `SolverOptions` property of your `configset` object, before using the `sbiosimulate` function:

- `SensitivityAnalysis` — Set to `true` to calculate the time-dependent sensitivities of all the species states defined by the `SpeciesOutputs` property with respect to the initial conditions of the species specified

in SpeciesInputFactors and the values of the parameters specified in ParameterInputFactors.

- SensitivityAnalysisOptions — An object that holds the sensitivity analysis options in the configuration set object. Properties of SensitivityAnalysisOptions are:
  - SpeciesOutputs — Specify the species for which you want to compute the sensitivities. This is the numerator as described in "About Calculating Sensitivities" on page 3-18.
  - SpeciesInputFactors — Specify the species with respect to which you want to compute the sensitivities. Sensitivities are calculated with respect to the InitialAmount property of the specified species. This is the denominator, described in "About Calculating Sensitivities" on page 3-18.
  - ParameterInputFactors — Specify the parameters with respect to which you want to compute the sensitivities of the species outputs in your model. Sensitivities are calculated with respect to the values of the specified parameters. This is the denominator, described in "About Calculating Sensitivities" on page 3-18.
  - Normalization — Specify the normalization for the calculated sensitivities:
    - 'None' — No normalization
    - 'Half' — Normalization relative to the numerator (species output) only
    - 'Full' — Full dedimensionalization

    For more information about normalization, see Normalization in the *SimBiology Reference*.

Calculating Sensitivities of a Model and Viewing Results

After setting SolverOptions properties, calculate the sensitivities of a model by providing the model object as an input argument to the sbiosimulate function.

The sbiosimulate function returns a SimData object containing the following simulation data:

- Time points, state data, state names, and sensitivity data
- Metadata such as the types and names for the logged states, the configuration set used during simulation, and the date of the simulation

A `SimData object` is a convenient way of keeping time data, state data, sensitivity data, and associated metadata together. A `SimData object` has properties and methods associated with it, which you can use to access and manipulate the data.

See Also

For examples of calculating sensitivities, see:

- "Example — Calculating Sensitivities" on page 3-22
- Parameter Scanning, Parameter Estimation, and Sensitivity Analysis in the Yeast Heterotrimeric G Protein Cycle demo

References

Ingalls, B.P, and Sauro, H.M. (2003). Sensitivity analysis of stoichiometric networks: an extension of metabolic control analysis to non-steady state trajectories. J Theor Biol. *222(1)*, 23–36.

Martins, J.R.R.A., Sturdza, P., and Alanso, J.J. (Jan. 2001). The connection between the complex-step derivative approximation and algorithmic differentiation. AIAA Paper *2001–0921*.

Martins, J.R.R.A., Kroo, I.M., and Alanso, J.J. (Jan. 2000). An automated method for sensitivity analysis using complex variables. AIAA Paper *2000–0689*.

Example — Calculating Sensitivities

In this section...

Overview

About the Example Model

This example uses the model described in "Model of the Yeast Heterotrimeric G Protein Cycle" on page C-19 to illustrate SimBiology sensitivity analysis options.

This table lists the reactions used to model the G protein cycle and the corresponding rate parameters (rate constants) for each mass action reaction. For reversible reactions, the forward rate parameter is listed first.

No.	Name	Reaction[1]	Rate Parameters
1	Receptor-ligand interaction	`L + R <-> RL`	`kRL`, `kRLm`
2	Heterotrimeric G protein formation	`Gd + Gbg -> G`	`kG1`
3	G protein activation	`RL + G -> Ga + Gbg + RL`	`kGa`
4	Receptor synthesis and degradation	`R <-> null`	`kRdo`, `kRs`
5	Receptor-ligand degradation	`RL -> null`	`kRD1`

No.	Name	Reaction[1]	Rate Parameters
6	G protein inactivation	`Ga -> Gd`	`kGd`

[1] Legend of species: L = ligand (alpha factor), R = alpha-factor receptor, Gd = inactive G-alpha-GDP, Gbg = free levels of G-beta:G-gamma complex, G = inactive Gbg:Gd complex, Ga = active G-alpha-GTP

About the Example

Assume that you are calculating the sensitivity of species `Ga` with respect to every parameter in the model. Thus, you want to calculate the time-dependent derivatives

$$\frac{\partial(Ga)}{\partial(kRLm)}, \frac{\partial(Ga)}{\partial(kRL)}, \frac{\partial(Ga)}{\partial(kG1)}, \frac{\partial(Ga)}{\partial(kGa)} \ldots$$

Loading and Configuring the Model for Sensitivity Analysis

1 The `gprotein_norules.sbproj` 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
```

The `m1` model object appears in the MATLAB Workspace.

2 The options for sensitivity analysis are in the configuration set object. Get the configuration set object from the model:

```
csObj = getconfigset(m1);
```

3 Use the `sbioselect` function, which lets you query by type, to retrieve the `Ga` species from the model:

```
Ga = sbioselect(m1,'Type','species','Where','Name','==','Ga');
```

4 Set the SpeciesOutputs property of the SensitivityAnalysisOptions object to the Ga species:

```
set(csObj.SensitivityAnalysisOptions,'SpeciesOutputs', Ga);
```

5 Use the sbioselect function, which lets you query by type, to retrieve all the parameters from the model and store the vector in a variable, pif:

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

6 Set the ParameterInputFactors property of the SensitivityAnalysisOptions object to the pif variable containing the parameters:

```
set(csObj.SensitivityAnalysisOptions,'ParameterInputFactors', pif);
```

7 Enable sensitivity analysis in the configuration set object (csObj) by setting the SensitivityAnalysis option to true:

```
set(csObj.SolverOptions, 'SensitivityAnalysis', true);
```

8 Set the Normalization property of the SensitivityAnalysisOptions object to perform 'Full' normalization:

```
set(csObj.SensitivityAnalysisOptions,'Normalization', 'Full');
```

Performing Sensitivity Analysis

Simulate the model and return the data to a SimData object:

```
simDataObj = sbiosimulate(m1);
```

Extracting and Plotting Sensitivity Data

You can extract sensitivity results using the getsensmatrix method of a SimData object. In this example, R is the sensitivity of the species Ga with respect to eight parameters. This example shows how to compare the variation of sensitivity of Ga with respect to various parameters, and find the parameters that affect Ga the most.

1 Extract sensitivity data in output variables T (time), R (sensitivity data for species Ga), snames (names of the states specified for sensitivity analysis), and ifacs (names of the input factors used for sensitivity analysis):

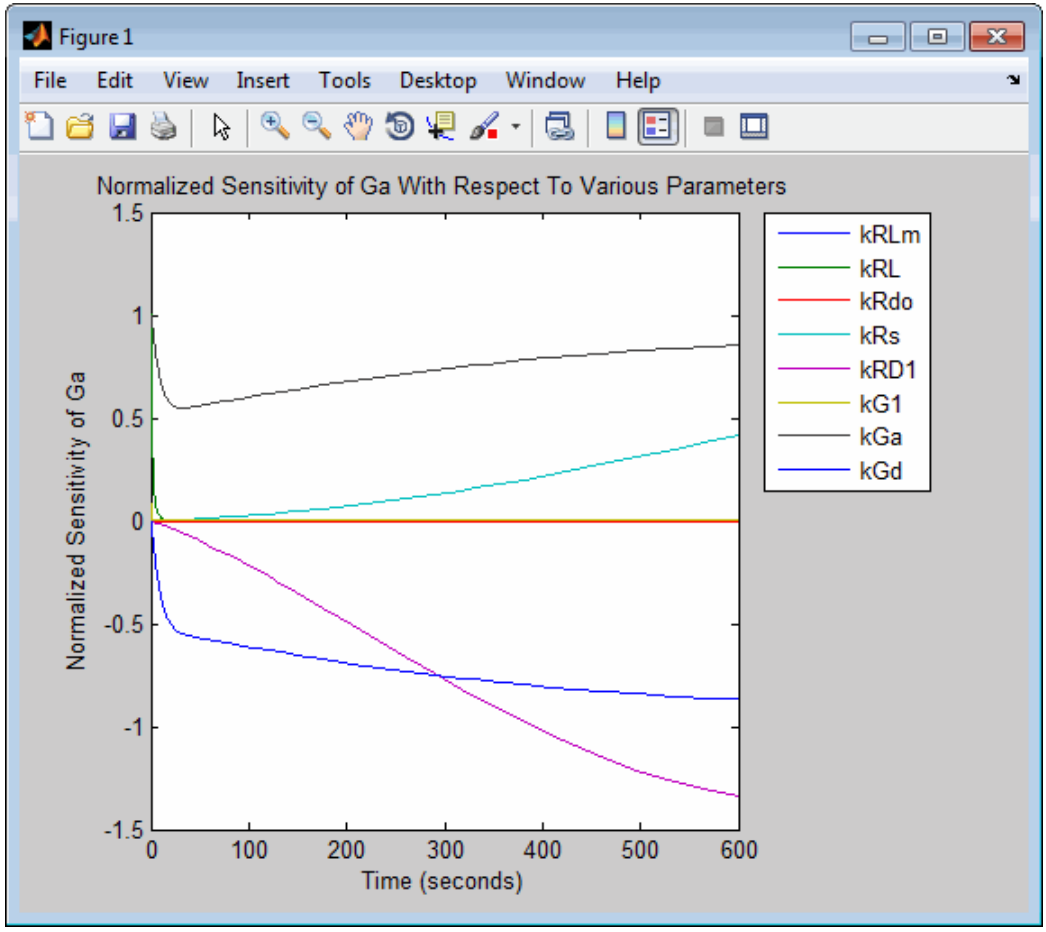
```
[T, R, snames, ifacs] = getsensmatrix(simDataObj);
```

2 Because `R` is a 3-D array with dimensions corresponding to times, output factors, and input factors, reshape `R` into columns of input factors to facilitate visualization and plotting:

```
R2 = squeeze(R);
```

3 After extracting the data and reshaping the matrix, plot the data:

```
% Open a new figure
figure;
% Plot time (T) against the reshaped data R2
plot(T,R2);
title('Normalized Sensitivity of Ga With Respect To Various Parameters');
xlabel('Time (seconds)');
ylabel('Normalized Sensitivity of Ga');
% Use the ifacs variable containing the
% names of the input factors for the legend
% Specify legend location and appearance
leg = legend(ifacs, 'Location', 'NorthEastOutside');
set(leg, 'Interpreter', 'none');
```



From the previous plot you can see that `Ga` is most sensitive to parameters `kGd`, `kRs`, `kRD1`, and `kGa`. This suggests that the amounts of active G protein in the cell depends on the rate of:

- Receptor synthesis
- Degradation of the receptor-ligand complex
- G protein activation
- G protein inactivation

Estimating Parameters

In this section...

About Parameter Estimation

Parameter estimation lets you estimate the values of unknown parameters in a model by fitting the model simulation results to experimental data. This technique is especially useful for parameters that you do not measure directly. This technique is appropriate when you have a fairly complete data set for one individual.

Estimating Parameters of a Model

You estimate one or more parameters in your model using the `sbioparamestim` function.

If you do not have Optimization Toolbox™ installed, then `sbioparamestim` uses the MATLAB function `fminsearch` as the default method for the parameter estimation.

If you have Optimization Toolbox and, optionally, Global Optimization Toolbox installed, then `sbioparamestim` uses the `lsqnonlin` function as the default method for the parameter estimation. However, you can specify other optimization functions from these toolboxes as the parameter estimation method.

See Also

For more information on parameter estimation, see the `sbioparamestim` reference page.

For examples of estimating parameters, see:

- "Example — Estimating Parameters Using sbioparamestim" on page 3-29
- Parameter Scanning, Parameter Estimation, and Sensitivity Analysis in the Yeast Heterotrimeric G Protein Cycle demo

About Population Fitting

SimBiology lets you perform individual and population fitting, which is parameter estimation for grouped sets of experimental data. This functionality requires Statistics Toolbox™ Version 7.3 or later. This type of parameter estimation is useful for pharmacokinetic/pharmacodynamic (PKPD) models, in which you typically want to fit your model to a population of data. This technique is appropriate when you have an incomplete data set for many individuals.

You perform individual fitting using the `sbionlinfit` function.

You perform population fitting using the `sbionlmefit` or `sbionlmefitsa` function.

For more information, see the following:

- "Creating Pharmacokinetic Models" on page 4-17
- "Parameter Fitting in Pharmacokinetic Models" on page 4-32
- "Fitting Pharmacokinetic Model Parameters" on page 4-34
- The demo Modeling the Population Pharmacokinetics of Phenobarbital in Neonates

Example — Estimating Parameters Using sbioparamestim

In this section...

Overview

About the Example Model

This example illustrates parameter estimation using time-course data from one experiment, using the `sbioparamestim` function. For information on all available parameter estimation and population fitting techniques, see "Estimating Parameters" on page 3-27.

This example uses the model described in "Model of the Yeast Heterotrimeric G Protein Cycle" on page C-19 to illustrate parameter estimation.

This table lists the reactions used to model the G protein cycle and the corresponding rate parameters (rate constants) for each mass action reaction. For reversible reactions, the forward rate parameter is listed first.

No.	Name	Reaction[1]	Rate Parameters
1	Receptor-ligand interaction	`L + R <-> RL`	`kRL`, `kRLm`
2	Heterotrimeric G protein formation	`Gd + Gbg -> G`	`kG1`

No.	Name	Reaction[1]	Rate Parameters
3	G protein activation	`RL + G -> Ga + Gbg + RL`	`kGa`
4	Receptor synthesis and degradation	`R <-> null`	`kRdo`, `kRs`
5	Receptor-ligand degradation	`RL -> null`	`kRD1`
6	G protein inactivation	`Ga -> Gd`	`kGd`

[1] Legend of species: L = ligand (alpha factor), R = alpha-factor receptor, Gd = inactive G-alpha-GDP, Gbg = free levels of G-beta:G-gamma complex, G = inactive Gbg:Gd complex, Ga = active G-alpha-GTP

About the Example

The study used to build the example model (Yi et al., 2003) reported the estimated value of parameter `kGd` as `0.11` for the wild-type strain. In "Example — Calculating Sensitivities" on page 3-22, the analysis showed that `Ga` is sensitive to parameters `kGd`, `kRs`, `kRD1`, and `kGa`.

This example shows:

- How to estimate the parameter `kGd` and determine its effect on the model
- How to estimate parameters `kGd`, `kRs`, `kRD1`, and `kGa` to obtain a better fit to the experimental data

Loading the Example Model

The `gprotein.sbproj` project contains a model for the wild-type strain (stored in variable `m1`). Load the G protein model for the wild-type strain:

```
sbioloadproject gprotein
```

The `m1` model object appears in the MATLAB Workspace.

Defining Experimental Data

The study used for this example (Yi et al., 2003) reports, in a plot, the experimental data as the fraction of active G protein. Store the time data for

the experimental results in a variable, `tExpt`, and store the values for the fraction of active G protein in a variable, `GaFracExpt`:

```
tExpt  = [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]';
```

Note For this simple example, you stored the experimental data in a variable in the MATLAB Workspace by typing the data values. However, typically, you import larger data sets into a MATLAB variable. For more information about importing data into variables, see "Importing Data" in the MATLAB documentation.

Simulating the G Protein Model

1 Simulate the model and return the results to a `SimData object`:

```
simDataObj = sbiosimulate(m1);
```

2 Retrieve the time and state data for the `GaFrac` parameter:

```
[tOrig, GaFracOrig] = selectbyname(simDataObj,'GaFrac');
```

Calculating R^2 for the G Protein Model

R^2 is the square of the correlation between the response values and the predicted response values. Therefore, R^2 measures how successful the fit is in explaining the variation of the data.

1 Calculate the sum of squares about the mean (SST):

```
sst = norm(GaFracExpt - mean(GaFracExpt))^2;
```

2 Interpolate the data to get time points that match the time points in the experimental data using the `cubic` interpolation method:

```
GaFracResampled = interp1(tOrig, GaFracOrig, tExpt, 'cubic');
```

3 Calculate the sum of squares due to error (SSE):

```
sse = norm(GaFracExpt - GaFracResampled)^2;
```

4 Calculate the R^2 value for the simulation data before parameter estimation:

```
rSquareOrig = 1-sse/sst

rSquareOrig =

    0.8967
```

For more information about the functions used here, see the `norm` and `interp1` reference pages.

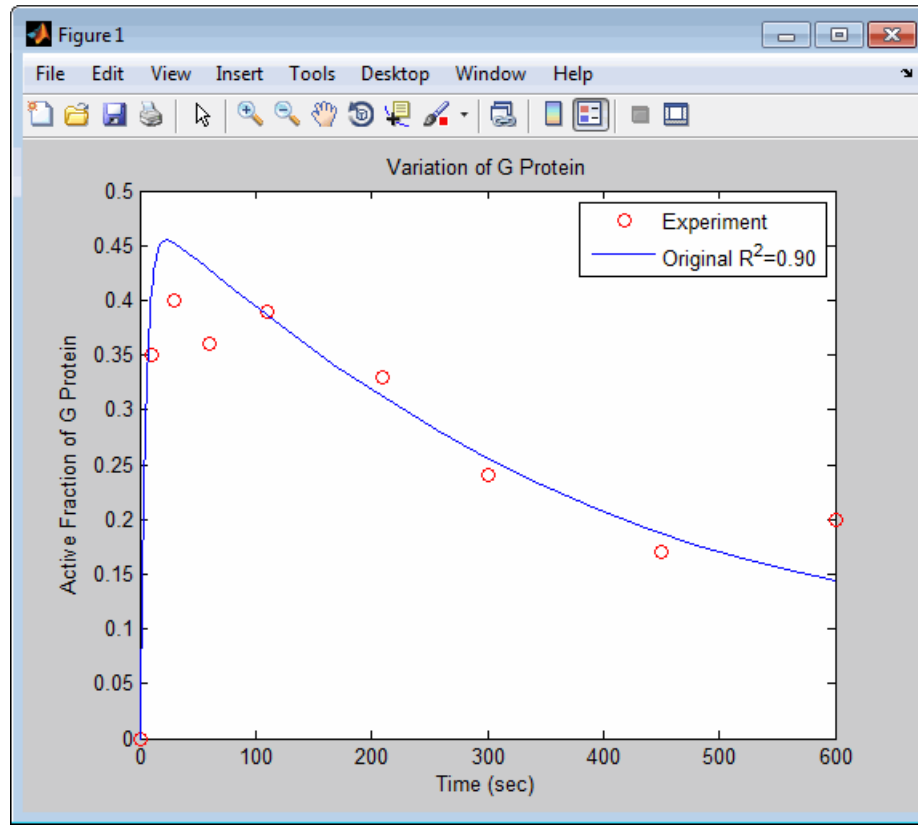
Plotting the Experimental Results and Simulation Data

1 Plot the experimental data for active G protein:

```
plot(tExpt, GaFracExpt, 'ro');
title('Variation of G Protein');
xlabel('Time (sec)');
ylabel('Active Fraction of G Protein');
legend('Experiment');
```

2 Plot the simulation data in the same plot:

```
hold on;
plot(tOrig, GaFracOrig);
legendText = {'Experiment', sprintf('Original R^2=%4.2f',...
               rSquareOrig)};
legend(legendText{:});
```



Leave this figure window open so you can use it to plot and compare results of using the estimated parameters later in this example.

Estimating the kGd Parameter in the G Protein Model

The study used to build the G protein model reported an estimated value of `0.11` for the parameter `kGd` in the wild-type strain (Yi et al., 2003). This example estimates the value of `kGd`.

1 Create a variable for the parameter to estimate. Also create a variable for the parameter corresponding to the experimental data to which you are fitting:

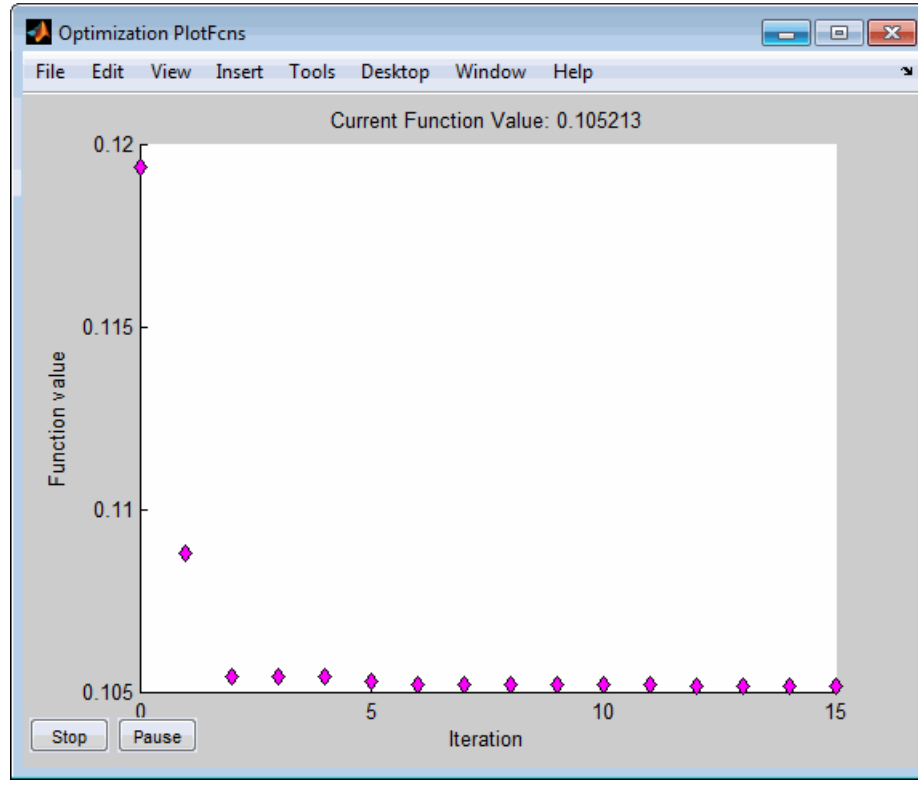
```
paramToEst = sbioselect(m1, 'Name', 'kGd');
GaFrac = sbioselect(m1, 'Name', 'GaFrac');
```

2 Specify plotting of each iteration of the parameter estimation to see how optimization is progressing:

```
opt = optimset('PlotFcns',@optimplotfval,'MaxIter',15);
```

3 Use the current value of the `kGd` parameter in the model as the starting value for optimization:

```
[estValues1, result1] = sbioparamestim(m1, tExpt, GaFracExpt, ...
                            GaFrac, paramToEst, {}, {'fminsearch',opt});
```



Simulating and Plotting Results Using the Estimated Parameter

Use the estimated value of the `kGd` parameter to see how it affects simulation results.

1. Use a variant to store the estimated value of `kGd`:

```
estvarObj = addvariant (m1, 'Optimized kGd');
addcontent(estvarObj, {'parameter', 'kGd', 'Value', estValues1});
```

2. Apply the value stored in the variant, simulate the model, and return the results:

```
simDataObj1 = sbiosimulate(m1, estvarObj );
[t1, GaFrac1] = selectbyname(simDataObj1,'GaFrac');
```

3. Calculate the R^2 value with the new estimate obtained using `'fminsearch'`:

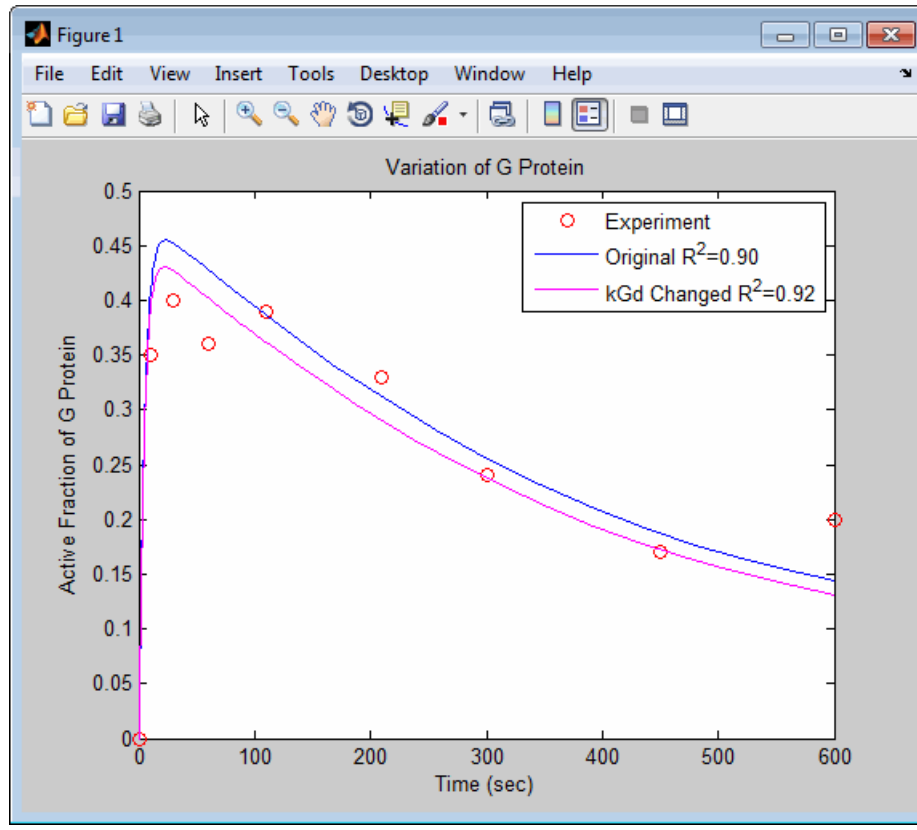
```
GaFrac1Resampled = interp1(t1, GaFrac1, tExpt, 'cubic');
sse1 = norm(GaFracExpt - GaFrac1Resampled)^2;
rSquare1 = 1 - sse1/sst

rSquare1 =

    0.9199
```

4. Plot the data and compare. If you left the previous figure open, because `hold` is `on`, the new plot appears in the existing figure to facilitate the comparison:

```
plot(t1, GaFrac1, 'm-');
legendText{end + 1} = sprintf('kGd Changed R^2=%4.2f', rSquare1);
legend(legendText{:});
```



The figure shows the best fit achieved by changing the parameter `kGd`.

Estimating Other Parameters in the G Protein Model

The example illustrating sensitivity analysis ("Example — Calculating Sensitivities" on page 3-22) showed that `Ga` is sensitive to parameters `kRs`, `kRD1`, `kGa`, and `kGd`. Based on the results from the sensitivity analysis, this tutorial shows you how to estimate these parameters. The sensitivity data is presented in "Extracting and Plotting Sensitivity Data" on page 3-24.

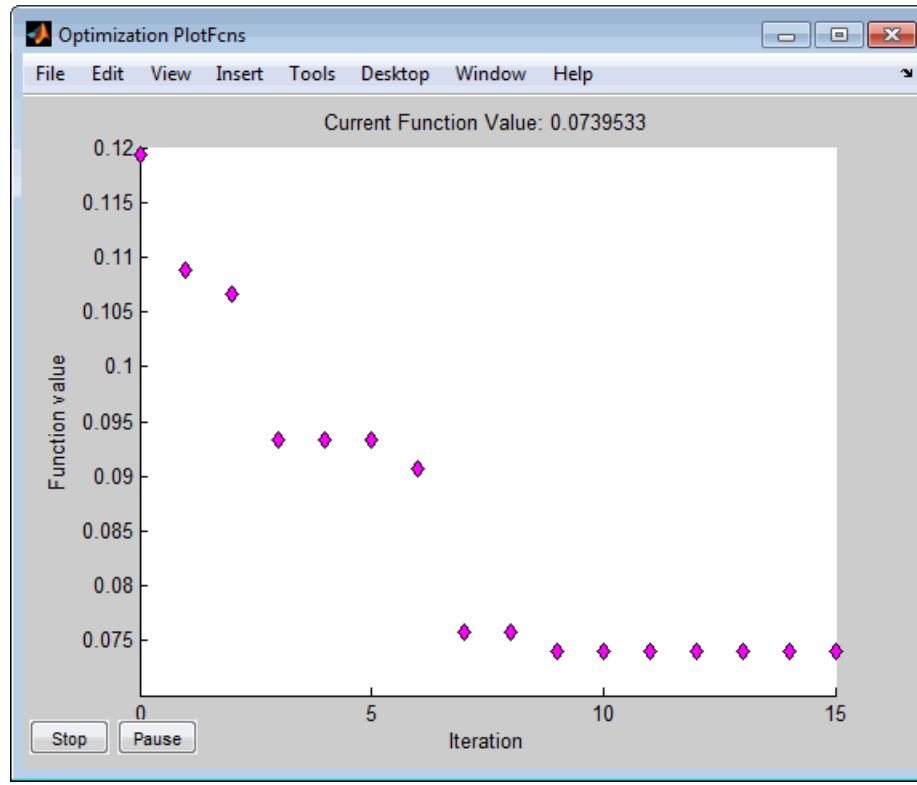
Note Although this example estimates four parameters to fit the data, there is no published experimental data that verifies these values, and this example is only for illustration.

1 Create a variable containing the parameters to estimate:

```
paramsToEst = [sbioselect(m1, 'Name', 'kRs');...
               sbioselect(m1, 'Name', 'kRD1');...
               sbioselect(m1, 'Name', 'kGa');...
               sbioselect(m1, 'Name', 'kGd')];
```

2 Estimate the parameters. Use the current values of parameters in the model as the starting values for optimization. Use the `opt` variable you created previously to specify plotting of each iteration of the parameter estimation to see how optimization is progressing:

```
[estValues2, result2] = sbioparamestim(m1, tExpt, GaFracExpt,...
                        GaFrac, paramsToEst, {}, {'fminsearch',opt});
```



3 Compare original parameter values and the estimated parameter values obtained with `'fminsearch'`:

```
% Original parameter values
paramsToEst

SimBiology Parameter Array

Index:    Name:    Value:    ValueUnits:
1         kRs      4
2         kRD1     0.004
3         kGa      1e-005
4         kGd      0.11

% Estimated parameter values
```

```
num2str(estValues2)

ans =

      4.549
  0.0031018
9.0068e-006
    0.12381
```

4 Calculate the R^2 value using the new estimates obtained with `'fminsearch'`:

```
estvarObj2 = addvariant(m1, 'Optimized kRs, kRD1, kGa, and kGd');
addcontent(estvarObj2, ...
    {{'parameter', 'kRs', 'Value', estValues2(1)}, ...
    {'parameter', 'kRD1', 'Value', estValues2(2)}, ...
    {'parameter', 'kGa', 'Value', estValues2(3)}, ...
    {'parameter', 'kGd', 'Value', estValues2(4)}});
simDataObj2 = sbiosimulate(m1, estvarObj2);
[t2, GaFrac2] = selectbyname(simDataObj2, 'GaFrac');
GaFrac2Resampled = interp1(t2, GaFrac2, tExpt, 'cubic');
sse2 = norm(GaFracExpt - GaFrac2Resampled)^2;
rSquare2 = 1 - sse2/sst

rSquare2 =

    0.9603
```

5 Plot the data and compare. If you left the previous figure open, because `hold` is `on`, the new plot appears in the existing figure to facilitate the comparison:

```
plot(t2, GaFrac2, 'g-');
legendText{end + 1} = sprintf('4 Constants Changed R^2=%4.2f',...
                      rSquare2);
legend(legendText{:});
```

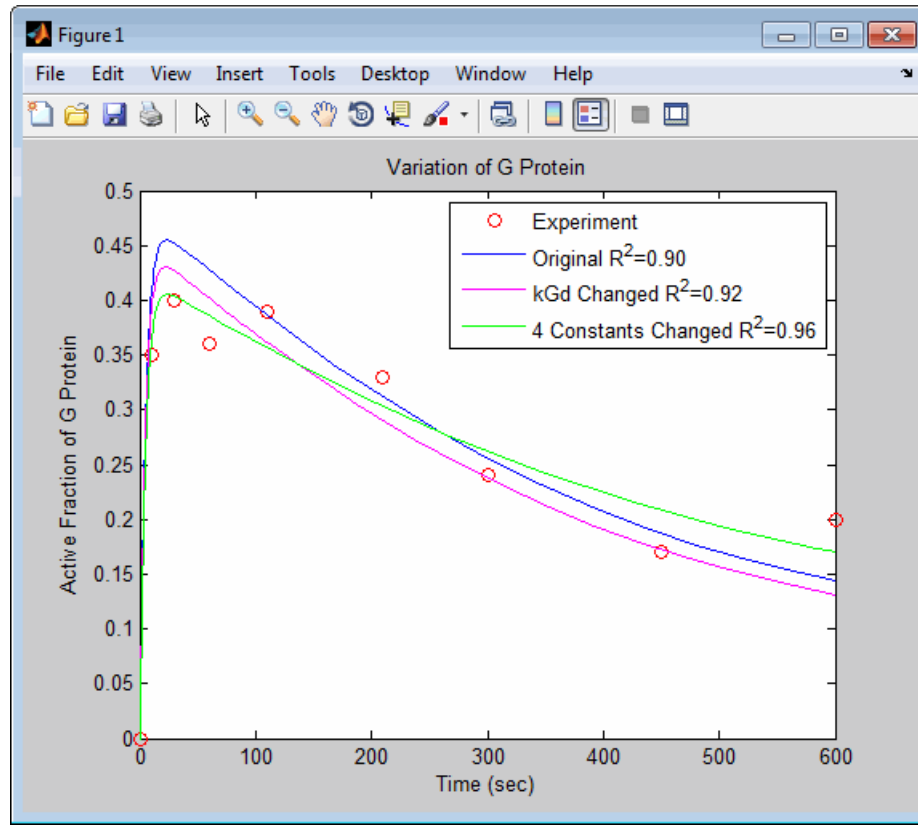


Figure 1
File Edit View Insert Tools Desktop Window Help
Variation of G Protein
Experiment
Original R2=0.90
kGd Changed R2=0.92
4 Constants Changed R2=0.96
Active Fraction of G Protein
Time (sec)
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
0
100
200
300
400
500
600

Accelerating Model Simulations and Analyses

In this section...

What Is Acceleration?

Normally, when simulating or analyzing a model in SimBiology, you express the model in MATLAB code. You can accelerate the simulation or analysis by converting the model to compiled C code, which executes faster. Because this compilation step has a small time overhead, acceleration is not recommended for individual simulations of small models. However, for large models, or for repeated simulations during analysis, acceleration can provide a significant speed increase that outweighs the small time overhead.

What Simulations and Analyses Can Be Accelerated?

You can accelerate the following:

- Simulating models
- Calculating sensitivities

Note For parameter estimations (using `sbioparamestim`) and population fittings (using `sbionlinfit`, `sbionlmefit`, or `sbionlmefitsa`), acceleration is automatically enabled, if the prerequisites for accelerating simulations and analyses are met.

When to Accelerate Simulations and Analyses

The functionality to accelerate simulations performs optimally under the following conditions:

- Running many simulations with different initial conditions
- Running very long simulations (for example, simulations that take longer than a minute to run)

Prerequisites for Accelerating Simulations and Analyses

To prepare your models for accelerated simulations:

1. Install a C compiler (if one is not already installed on your system). For a current list of supported compilers, see Supported and Compatible Compilers at `www.mathworks.com`.

   Tip On 32-bit Windows® platforms, the lcc compiler is automatically installed. However, for better performance of the acceleration functionality, you may want to install a supported compiler other than lcc. You can also use `mex -setup` to choose and configure a different C compiler, as described in "Building MEX-Files" in the MATLAB External Interfaces documentation.

2. Run `mex -setup` once after compiler installation, as described in the `mex` reference page.

3. Ensure that any user-defined functions in your model can be used for code generation from MATLAB, so they can convert to compiled C. For more information, see the Code Generation from MATLAB documentation.

Accelerating a Simulation

Accelerating simulations is a two step process:

1. Use the `sbioaccelerate` function to prepare your model for accelerated simulations. Use the same input arguments that you plan to use with `sbiosimulate`. For example:

```
sbioaccelerate(modelObj, configsetObj, doseObj);
```

This step prepares your model for acceleration and may take a minute or longer to complete for very large models.

Note You need to run `sbioaccelerate` again, before running simulations, if you make any modifications to this model, other than:

- Changes to any variants
- Changes to values for the `InitialAmount` property of species
- Changes to the `Capacity` property of compartments
- Changes to the `Value` property of parameters

---

2. Use the `sbiosimulate` function with the same input arguments that you used with `sbioaccelerate`. For example:

```
simdataObj = sbiosimulate(modelObj, configsetObj, doseObj);
```

Troubleshooting Accelerated Simulations

If you have user-defined functions, do not use persistent variables in these functions. Persistent variables are not compatible with the functionality used for accelerating simulations.

If you specify user-defined functions in SimBiology expressions, you might see the following warning if your code is not compatible with code generation from MATLAB:

```
The SimBiology Expression and any user-defined functions
could not be accelerated. Please check that these expressions
and any user-defined functions are supported for code generation
as described in the Code Generation from MATLAB documentation.
```

where *Expression* is any of the following:

- Reaction rate/rule expression
- Repeated assignment rule expression

- Event trigger expression
- Event function expression

For more information, see the Code Generation from MATLAB documentation.

4

Pharmacokinetic Modeling

Pharmacokinetic Modeling Functionality

In this section...

Overview

SimBiology software extends the MATLAB computing environment for analyzing pharmacokinetic (PK) data using models. The software lets you do the following:

- Create models — Use a model construction wizard. Alternatively, extend any model with pharmacodynamic (PD) model components, or build higher fidelity models. See "Model" on page 4-4 for more information.
- Fit data — Fit nonlinear, mixed-effects models to data, and estimate the fixed and random effects, or fit the data using nonlinear least squares. For more information, see "Analyze Data Using Models" on page 4-4.
- Generate diagnostic plots — For more information, see "Analyze Data Using Models" on page 4-4.

The software lets you work with different model structures, thus letting you try multiple models to see which one produces the best results.

Required and Recommended Software for Pharmacokinetic Modeling

Required Software

MATLAB	Provides a command-line interface and an integrated software environment. For instructions, see the MATLAB installation documentation for your platform. If you have installed MATLAB and want to check which other MathWorks® products are installed, enter `ver` in the MATLAB Command Window.
Statistics Toolbox (Version 7.3 (R2010a) or greater)	Provides fitting tools including functions used to analyze nonlinear mixed effects.

Recommended Software

C Compiler	Required to prepare the model for accelerating simulations. For list of supported compilers, see Supported and Compatible Compilers.
Optimization Toolbox	Optimization Toolbox extends the MATLAB technical computing environment with tools and widely used algorithms for standard and large-scale optimization. These algorithms solve constrained and unconstrained, continuous and discrete problems. If the Optimization Toolbox product is installed, you can specify additional methods for likelihood maximization. If you do not have this product, SimBiology

uses `fminsearch` provided by MATLAB for likelihood maximization.

How This Product Supports Pharmacokinetic Modeling

Import and Work with Data

You can import tabular data into the SimBiology desktop or the MATLAB Workspace. The supported file types are `.xls`, `.csv`, and `.txt`. You can specify that the data is in a NONMEM® formatted file. The import process interprets the columns according to the NONMEM definitions.

From the SimBiology desktop, you can filter the raw data to suppress outliers, visualize data using common plots (such as `plot`, `semilog`, `scatter`, or `stairs`), and perform basic statistical analysis. You also can use functions to process and visualize the data at the command line.

Model

SimBiology provides an extensible modeling environment. You can do any of the following:

- Create a PK model using a model construction wizard to specify the number of compartments, the route of administration, and the type of elimination.
- Extend any model with pharmacodynamic (PD) model components, or build higher fidelity models.
- Build or load your own SimBiology, or SBML model.

For more information on building SimBiology models, see Chapter 1, “Modeling”.

Analyze Data Using Models

Perform both individual and population fits to grouped longitudinal data:

- Individual fit — Fit data using nonlinear least-squares method, specify parameter transformations, estimate parameters, and calculate residuals and the estimated coefficient covariance matrix.
- Population fit — Fit data, specify parameter transformations, and estimate the fixed effects and the random sources of variation on parameters using nonlinear mixed-effects models.

  You can use the following methods to estimate the fixed effects:

  - `LME` — Linear mixed-effects approximation
  - `RELME` — Restricted LME approximation
  - `FO` — First-order estimate
  - `FOCE` — First-order conditional estimate

  For more information about each of these methods, see `nlmefit` in the Statistics Toolbox documentation.
- Population fit using a stochastic algorithm — Fit data, specify parameter transformations, and estimate the fixed effects and the random sources of variation on parameters, using the Stochastic Approximation Expectation-Maximization (SAEM) algorithm. SAEM is more robust with respect to starting values. This functionality relaxes assumption of constant error variance.

  For more information, see `nlmefitsa` in the Statistics Toolbox documentation.

In addition, you can generate diagnostic plots that show:

- The predicted time courses and observations for an individual or the population
- Observed versus predicted values
- Residuals versus time, group, or predictions
- Distribution of the residuals
- A box-plot for random effects or parameter estimates from individual fitting

Using the Command Line Versus the SimBiology Desktop

SimBiology extends MATLAB and lets you access pharmacokinetic modeling functionality at the command line and in the graphical SimBiology desktop.

Use the command line to write and save scripts for batch processing and to automate your workflow.

Use the SimBiology desktop to interactively change and iterate through the model workflow. The SimBiology desktop lets you encapsulate models, data, tasks, task settings, and diagnostic plots into one convenient package, namely a SimBiology project.

Furthermore, if you are using the SimBiology desktop and want to learn about using the command line, the MATLAB code capture feature in the desktop lets you see the commands and export files for further scripting in the MATLAB editor.

Accessing a Pharmacokinetic Modeling Demo

For a demo showing pharmacokinetic modeling functionality at the command line, click the following link to open the demo in MATLAB: Modeling the Population Pharmacokinetics of Phenobarbital in Neonates.

Acknowledgements: Tobramycin Data Set

Acknowledgements for data in the `tobramycin.txt` file in the `/matlab/toolbox/simbio/simbiodemos` folder:

[1] Original Publication: Aarons L, Vozeh S, Wenk M, Weiss P, and Follath F. “Population pharmacokinetics of tobramycin.” *Br J Clin Pharmacol.* 1989 Sep;28(3):305–14.

Data set provided by Dr. Leon Aarons, (`laarons@fs1.pa.man.ac.uk`)

The data in the `tobramycin.txt` file were downloaded from the Web site of the Resource Facility for Population Kinetics http://www.rfpk.washington.edu. Funding source: NIH/NIBIB grant P41-EB01975.

The original data set was modified as follows:

- Header comments were removed.
- The file was converted to a tab-delimited format.
- Missing values in the `HT` column were denoted with `"."` instead of `100000000.000`.

Importing Data — Supported Files and Data Types

In this section...

Supported Files and Data Types

You can import tabular data to the SimBiology desktop or to the MATLAB Workspace. The supported file types are `.xls`, `.csv`, and `.txt`. You can specify that the data is in a NONMEM formatted file. The import process interprets the columns according to the NONMEM definitions. For more information see "Support for Importing NONMEM Formatted Files" on page 4-8.

From the SimBiology desktop, you can filter the raw data to suppress outliers, visualize data using common plots (such as `plot`, `semilog`, `scatter`, or `stairs`), and perform basic statistical analysis. You also can use functions to process and visualize the data at the command line.

Note If your data set contains dosing information that is infusion data, the data set must contain the rate and not an infusion duration.

Unit Conversion

Regardless of whether unit conversion functionality is on or off, dosing in the data file must be expressed in amounts (or as `amount/time` for infusion rate). By default **Unit Conversion** is off, so you must ensure that units for the data are consistent with each other. If you want to turn on unit conversion, see "Unit Conversion for Imported Data and the Model" on page 4-27 .

Support for Importing NONMEM Formatted Files

You can specify that the data is in a NONMEM formatted file. The following table highlights the interpretation of this data in SimBiology software.

Column Header	Interpretation
`ID`	Text or numeric values that identify the record. The import process assumes that contiguous data with the same value contains data from one individual. If the data contains non-contiguous references to the same value, the import process assigns the second ID encountered an indexed valued derived from the group first encountered. For example, if the ID columns contains `[1 1 1 2 2 2 1 1 1]`, the IDs assigned are `1, 2, 1_1`.
`TIME`	Monotonically increasing positive values within each group, indicating time of observation or dose. The data file can specify clock (`2:30`) or decimal values (`6.25`). The import process assigns a value of `0` to the first `TIME` value in the data file. The import process assigns subsequent values relative to the first value. For example the import process interprets [10:05 10:30 11 12:30 21.3] as: [0 0.25 0.95 2.25 14.2]. If the data file also contains a `DATE` column, the import process uses it with the `TIME` column in calculating the relative `TIME` values. The column cannot contain `Inf`.
`DATE`, `DAT1`, `DAT2`, or `DAT3`	Defines the day of the observation or the dose. The column can contain the month as a number (`9`) or a string (`Sep`). Specify date in the following formats: • `DATE` — The column can specify `month/day/year` or `month-day-year`. If you specify two numbers, the import process assumes they are month and day. • `DAT1` — The column can specify `day/month/year` or `day-month-year`. If you specify two numbers, the import process assumes they are day and month.

Column Header	Interpretation
	• `DAT2` — The column can specify `year/month/day` or `year-month-day`. If you specify two numbers, the import process assumes they are month and day. • `DAT3` — The column can specify `year/day/month` or `year-day-month`. If you specify two numbers, the import process assumes they are day and month. Note the following additional assumptions: • If you specify only one number, the import process assumes it is the day • You can omit the year or specify 1, 2, 3, or 4 digits. If you specify two-digit years, it is assumed to be in the 1900s.
`DV`	Numeric value of an observation. Column cannot contain `Inf` or `-Inf`.
`MDV`	Defines whether a row describes an observation: • Row contains 0 — Observation event • Row contains 1 — Not an observation event
`EVID`	Defines the type of event described for the row in the record: • 0 — Observation event; row contains an observed value. • 1 — Dose event; row describes a dose. • 2 — Other event; row describes some other event such as measurement of a covariate. If a column contains values for dose, but `EVID` is not `1`, the import process ignores the value. You see a warning and the value is ignored.

Column Header	Interpretation
	The import process does not support values `3` and `4`. You see a warning and the value is ignored.
`CMT`	Indicates which compartment is used for observation value or for dose received. The interpretation also depends on `EVID`: • Observation event (`EVID = 0`) — `CMT` column indicates which compartment was used for observation value. • Dose Event (`EVID = 1`) — `CMT` column indicates which compartment received the dose.
`AMT`	Positive number indicating dose. `0` or `NaN` specifies no dose administered. The column cannot contain `Inf`.
`RATE`	Positive number indicating rate of infusion. `0` specifies an infinite rate (equivalent to a bolus dose), and `NaN` specifies no rate. The column cannot contain `Inf`.
`II`	Positive number defining the time between doses.
`ADDL`	When the data specifies a number of identical serial doses at specific intervals (defined by `II`), `ADDL` specifies the number of doses in the series excluding the initial dose. If the data specifies `II` but not `ADDL`, then SimBiology assumes that the dosing occurs for the duration of that data record.

Unsupported NONMEM Definitions

The import process does not support (and therefore ignores) the rows containing the following values or definitions:

- `EVID` values 3 and 4
- `SS` column for specifying steady state doses

- `PCMT` column to define whether to compute a prediction for the row
- `CALL` column for calling the ERROR or the PK subroutine
- If rate is specified as being less than zero, it is assumed to be zero

Creating a Data File with SimBiology Definitions

If you are creating a file containing population data that you want to later import into SimBiology, create the data file with the following columns:

- Group column — Specify text or numeric values. The rows in the file that have the same Group column value are for the same individual.
- Time column — Specify monotonically increasing positive values within each group that define the time of the dose, observation and/or covariate measurements.
- Zero or more dosing columns — Create one dosing column for each compartment being dosed. In each column, specify positive values representing doses in amount that are added to a species. Use `0` or `NaN` to specify that no dose was applied at the specified time. This is useful for times when an observation was recorded but no dose was applied.
- Zero, or more rate columns — Specify positive values. Zero specifies an infinite rate and `NaN` specifies that no rate applies. The rate column is associated with a dosing column and defines the rate at which the dose is administered.
- Zero or more observation columns — Specify numeric values or `NaN`s. You can only specify one observation value at a particular time for each group. `NaN` values define that no observation was recorded at the specified time. This is useful for times when a dose was applied but no observation was recorded.
- Zero or more covariate columns — Specify numeric values or `NaN`s. Each value defines the covariate value at the specified time. `NaN` values define that no covariate observation was recorded at the specified time.

Importing Data

In this section...

Importing Data From NONMEM Formatted Files

Use the `sbionmimport` function to import data from NONMEM formatted files. To import the data without NONMEM interpretation of column headers, see "Importing Data Using the dataset Function" on page 4-14.

To prepare the data file for import, remove any comments that are present at the beginning of the file and select one of the following methods to import your data:

- If the data file contains only the column header values shown in "Support for Importing NONMEM Formatted Files" on page 4-8, use the syntax shown in the following example:

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

- If the data file has column header labels different from the table shown in "Support for Importing NONMEM Formatted Files" on page 4-8 and you want to apply NONMEM interpretation of headers:

  1. Create a NONMEM file definition object. This object lets you define what the column headers in the data file mean in SimBiology. In the following example, the column containing response values is `CP`, whereas in NONMEM formatted files the column is labelled `DV`.

     To use the tobramycin data set [1], create a NONMEM file definition object and define the following:

     ```
     def = sbionmfiledef;
     def.DoseLabel = 'DOSE';
     def.GroupLabel = 'ID';
     ```

```
def.TimeLabel = 'TIME';
def.DependentVariableLabel = 'CP';
def.MissingDependentVariableLabel = 'MDV';
def.EventIDLabel = 'EVID';
def.ContinuousCovariateLabels = {'WT', 'HT', 'AGE', 'SEX', 'CLCR'};
```

Your file can contain any name for column headings. See `sbionmfiledef` for the list of properties you can configure in the NONMEM file definition object.

2. Use the `sbionmimport` function to import your data file with the column header definitions as specified in the NONMEM file definition object. For example, browse to *`matlabroot`*`/toolbox/simbio/simbiodemos/` (where *`matlabroot`* is the folder where MATLAB is installed).

```
[ds, PKDataObj] = sbionmimport('tobramycin.txt', def, ...
   'TreatAsEmpty', '.');
```

This example shows you how to obtain the PKData object, `PKDataObj`, while importing, since you will use the PKData object in fitting the model later.

The `sbionmimport` function accepts property-name-value pairs accepted by `dataset`. For example, if the data set does not contain column headers, use `'ReadVarNames', false` to specify that `sbionmimport` should read values from the first row of the file.

For information about creating a model to fit the data, see "Creating PK Models" on page 4-19.

Importing Data Using the dataset Function

Use the `dataset` function to import tabular data with named columns into an array that you can use in fitting and analysis at the command line. Use this function when you want to import the data without NONMEM interpretation of column headers. The `dataset` function lets you specify parameter/value pair arguments in which you can specify options such as the type of delimiter, and whether the first row contains header names. For more information, see `dataset`.

To prepare the data file for import, remove any comments that are present at the beginning of the file.

Examples:

```
% text files
data = dataset('file', 'tobramycin.txt')
% text files with . in place of missing values
data = dataset('file', 'tobramycin.txt', 'TreatAsEmpty', '.')

% For Excel files
data = dataset('xlsfile', 'tobramycin.xls')
```

You can also construct the `dataset` array from variables in the MATLAB Workspace.

```
% Create a 10x2 array
x = rand(10,2);
% Construct a dataset array containing x
data = dataset({x(:, 1), 'Column1'}, {x(:,2), 'Column2'})
```

If you import the data as separate variables containing doubles, you can construct the `dataset` array by concatenating the variables.

```
% Create 2 10x1 vectors
x = rand(10,1);
y = rand(10,1);
% Construct a dataset array containing x and y
data = dataset({x, 'Column1'}, {y, 'Column2'})
```

After you finish analyzing your data, you can export any new variables to a variety of file formats. The “Exporting Data” section of the MATLAB documentation describes how to export data from the MATLAB Workspace.

Other Resources for Importing Data

For detailed information about supported data formats and the functions for importing data into the MATLAB Workspace, see the “Importing Data” section of the MATLAB documentation. You also can import data using the MATLAB Import Wizard (see “Tips for Using the Import Wizard” in the

MATLAB documentation). Use the Import Wizard, to import data as text files (such as `.txt` and `.dat`), MAT-files, and spreadsheet files, (such as `.xls`).

The MATLAB Import Wizard processes the data source. The wizard recognizes data delimiters, as well as row or column headers, to facilitate the process of data selection and importation into the MATLAB Workspace. You can import the data to the SimBiology desktop from the MATLAB Workspace.

Creating Pharmacokinetic Models

In this section...

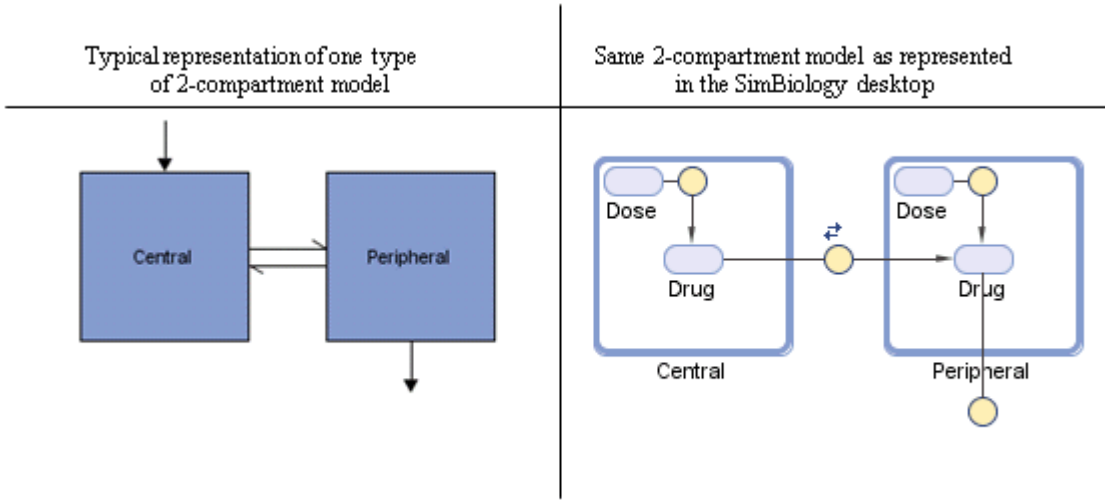
Overview

To start modeling, you can:

- Create a PK model using a model construction wizard that lets you specify the number of compartments, the route of administration, and the type of elimination.
- Extend any model to build higher fidelity models.
- Build or load your own model. Load a SimBiology project or SBML model.

How SimBiology Models Represent Pharmacokinetic Models

The following figure compares a model as typically represented in pharmacokinetics with the same model shown in the SimBiology model diagram. For this comparison, assume that you are modeling administration of a drug using a two-compartment model with any dosing input and linear elimination kinetics. (The model structure remains the same with any dosing type.)



Comparison of views in PK

Note the following:

- SimBiology represents the concentration or amount of a drug in a given compartment or volume by a species *object* contained within the compartment.
- SimBiology represents the exchange or flow of the drug between compartments and the elimination of the drug by reactions.
- SimBiology represents intercompartmental clearance by a parameter (`Q`) which specifies the clearance between the compartments.
- SimBiology drives the dosing schedule with a combination of species (`Drug` and/or `Dose`) and reactions (`Dose -> Drug`), depending on whether the administration into the compartment follows bolus, zero-order, infusion, or first-order dosing kinetics. For more information on the components added and parameters estimated, see "About Dosing Types" on page 4-21.

You can also view this model as a regression function, `y = f(k,u)`, where `y` is the predicted value, given values of an input `u`, and parameter values `k`. In SimBiology the model represents `f`, and the model is used to generate a regression function if `y`, `k`, and `u` are identified in the model.

Creating PK Models

To create a PK model with the specified number of compartments, dosing type, and method of elimination:

1. Create a `PKModelDesign` object. The `PKModelDesign` object lets you specify the number of compartments, route of administration, and method of elimination, which SimBiology uses to construct the model object with the necessary compartments, species, reactions, and rules.

   ```
   pkm = PKModelDesign;
   ```

2. Add a compartment specifying the compartment name, and optionally, the type of dosing, and the method of elimination. Also specify whether the data contains a response variable measured in this compartment and whether the dose(s) have time lags. For example, if using the tobramycin data set [1], specify a compartment named `Central`, with `Bolus` for the `DosingType` property, `linear-clearance` for the `EliminationType` property, and `true` for the `HasResponseVariable` property.

   ```
   pkc1 = addCompartment(pkm, 'Central', 'DosingType', 'Bolus', ...
                         'EliminationType', 'linear-clearance', ...
                         'HasResponseVariable', true);
   ```

   For a description of other `DosingType` and `EliminationType` property values, see “About Dosing Types” on page 4-21 and “About Elimination Types” on page 4-24.

   For a description of the `HasResponseVariable` property, see `HasResponseVariable`. At least one compartment in a model must have a response. Although SimBiology supports multiple responses per compartment, when adding compartments to a `PKModelDesign` object, you are limited to one response per compartment.

Note To add a compartment that has a time lag associated with any dose that targets it, set the `HasLag` property to `true`:

```
pkc_lag = addCompartment(pkm, 'Central', 'DosingType', 'Bolus', ...
                         'EliminationType', 'linear-clearance', ...
                         'HasResponseVariable', true, 'HasLag', true);
```

Or after adding a compartment, set its `HasLag` property to `true`:

```
pkc1.HasLag = true;
```

---

3. Optionally, add a second compartment named `Peripheral`, with no dosing, no elimination, and no time lag. Set the `HasResponseVariable` property to `true`. If you are using the tobramycin data set [1], skip this step and use only one compartment.

   ```
   pkc2 = addCompartment(pkm, 'Peripheral', 'HasResponseVariable', true);
   ```

   The model construction process adds the necessary parameters, including a parameter representing intercompartmental clearance `Q`. You can add more compartments by repeating this step. The addition of each compartment creates a chain of compartments in the order of compartment addition, with a bidirectional flow of the drug between compartments in the model.

   Use the handle to the compartment (`pkc1` or `pkc2`), to change compartment properties.

4. Construct a SimBiology model object.

   ```
   [modelObj, PKModelMapObj] = pkm.construct
   ```

   The `construct` method returns a SimBiology model object (`modelObj`) and a `PKModelMap` object (`PKModelMapObj`) that contains the mapping of the model components to the elements of the regression function. For more information about the `PKModelMap` object, see "Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters" on page 4-29.

Note If you change the `PKModelDesign` object, you must create a new model object using the `construct` method. Changes to the `PKModelDesign` do not automatically propagate to a previously constructed model object.

5 Perform parameter fitting as shown in "Fitting Pharmacokinetic Model Parameters" on page 4-34.

The model object and the `PKModelMap` object are input arguments for the `sbionlmefit`, `sbionlmefitsa` and `sbionlinfit` functions used in parameter fitting.

For information on ...	See ...
Dosing types	"About Dosing Types" on page 4-21
Elimination types	"About Elimination Types" on page 4-24
Parameter fitting	"Fitting Pharmacokinetic Model Parameters" on page 4-34
Simulating the model and a description of configuration sets	• "About Simulation Settings and Specifying Alternate Values for Initial Estimates" on page 4-51 • "Simulating Models" on page 3-3

About Dosing Types

When creating models, SimBiology creates the following model components for each compartment in the model, regardless of the dosing type:

- Two species (`Drug_CompartmentName` and `Dose_CompartmentName`) for each compartment.
- A reaction (`Dose_CompartmentName -> Drug_CompartmentName`) for each compartment, governed by mass action kinetics.
- A parameter (`ka_CompartmentName`) for each compartment, representing the absorption rate of the drug when absorption follows first-order kinetics.

This is the forward rate parameter for the `Dose_CompartmentName -> Drug_CompartmentName` reaction.

- A parameter (`TkO_CompartmentName`) for each compartment, representing the duration of drug absorption when absorption follows zero-order kinetics.
- A parameter (`TLag_CompartmentName`) for each compartment, representing the time lag for any dose that targets that compartment and also that is specified as having a time lag.

For dosing types that have a fixed infusion or absorption duration (`infusion` and `zero-order`), you can use overlapping doses. The doses are additive.

The following table describes the dosing types, the default parameters to estimate, and lists the model components created and used for dosing.

Dosing Type	Description	SimBiology Model Components Used	Default Parameters to Estimate
`''`(empty string)	No dose	The species (`Drug_CompartmentName`) in each compartment	None
SimBiology desktop — `bolus` Command line — `Bolus`	Assumes that the drug amount is increased instantly at the dose time. In the SimBiology model, the initial concentration of the drug is based on dose amount and volume of the compartment containing the drug.	The species (`Drug_CompartmentName`) in each compartment	None

Dosing Type	Description	SimBiology Model Components Used	Default Parameters to Estimate
SimBiology desktop — `infusion` Command line — `Infusion`	Assumes that the infused drug amount increases at a constant known absorption (or infusion) rate over a known duration. The imported data set must contain the rate and not an infusion duration. SimBiology uses this information to change the species concentration at the constant rate over the duration specified in the data set.	The species (`Drug_CompartmentName`) in each compartment	None
SimBiology desktop — `zero-order` Command line — `ZeroOrder`	Assumes that the drug is added at a constant rate over fixed, but unknown duration.	• The species `Drug_CompartmentName` in each compartment • The parameter (`Tk0_CompartmentName`) in each compartment that has zero-order dosing. This parameter represents the duration of drug absorption	`Tk0_CompartmentName` (absorption duration)
SimBiology desktop — `first-order` Command line — `FirstOrder`	Assumes that the rate at which the drug is absorbed is not constant. In the SimBiology model, absorption rate is assumed	• A species (`Dose_CompartmentName`) representing the dose amount before it is absorbed • A species (`Drug_CompartmentName`) for each compartment	`ka_CompartmentName` (absorption rate)

Dosing Type	Description	SimBiology Model Components Used	Default Parameters to Estimate
	to be governed by mass-action kinetics.	• A parameter (`ka_CompartmentName`) representing the absorption rate of the drug • A `MassAction reaction (Dose_CompartmentName > Drug_CompartmentName)` with forward rate parameter (`ka_CompartmentName`)	

If you are using a custom model, or want to simulate a model with the dosing schedule applied, see the following additional sources of information:

For information on ...	See ...
Preparing the model before simulating	"Prerequisites for Using Custom SimBiology Models in Parameter Fitting" on page 4-28

About Elimination Types

Elimination Type	Description	SimBiology Model Components Created	Default Parameters to Estimate
SimBiology desktop — `Linear {Elimination Rate, Volume}` Command line — `'linear'`	Assumes simple mass-action kinetics in the elimination of the drug. In the SimBiology model, elimination is specified by mass-action kinetics with the	• A parameter representing the elimination rate (`ke_CompartmentName`) • A `MassAction reaction (drug > null)` with forward rate parameter (`ke_CompartmentName`)	• Compartment volume (`Capacity` property) • Elimination rate constant (`ke_CompartmentName`) • Inter-compartmental clearance (`Q`) when

Elimination Type	Description	SimBiology Model Components Created	Default Parameters to Estimate
	elimination rate constant specified by the forward rate parameter (`ke`).	specific to the compartement	there is more than one compartment. See "About Intercompartmental Clearance" on page 4-26.
SimBiology desktop — `Linear {Clearance, Volume}` Command line — `'linear-clearance'`	Assumes simple mass-action kinetics in the elimination of the drug. In the SimBiology model, similar to `Linear {Elimination Rate, Volume}`. But, in addition, this option lets you specify the model in terms of clearance (`Cl`) where, `Cl = ke * volume`).	• A parameter representing the clearance (`Cl_CompartmentName`) • A parameter representing the elimination rate constant (`ke_CompartmentName`) • An `InitialAssignment` rule that initializes `ke_CompartmentName` based on the initial values for `Cl_CompartmentName` and compartment volume • A `MassAction` reaction (`drug > null`) with forward rate parameter (`ke_CompartmentName`)	• Compartment volume (`Capacity` property) • Clearance (`Cl_CompartmentName`) • Inter-compartmental clearance (`Q`) when there is more than one compartment. See "About Intercompartmental Clearance" on page 4-26.
SimBiology desktop — `Enzymatic (Michaelis-Menten)`	Assumes that elimination is governed by	• Parameter representing the	• Compartment volume (`Capacity` property)

Elimination Type	Description	SimBiology Model Components Created	Default Parameters to Estimate
Command line — `'enzymatic'`	Michaelis-Menten kinetics.	Michaelis constant, (`Km_CompartmentName`) • A parameter for maximum velocity (`Vm_CompartmentName` • A reaction with Michaelis-Menten kinetics (`drug -> null`), with kinetic law parameters `Vm_CompartmentName` and `Km_CompartmentName`	• Parameter (`Km_CompartmentName`) • Parameter (`Vm_CompartmentName`) • Inter-compartmental clearance (`Q`) when there is more than one compartment. See "About Intercompartmental Clearance" on page 4-26

About Intercompartmental Clearance

The compartments created when you generate a SimBiology model form a chain and each pair of linked compartments are connected by a transport reaction similar to linear elimination. The addition of two compartments, `C1` and `C2`, generates a reversible mass-action reaction `C1.Drug_C1 <-> C2.Drug`. The forward rate parameter is the compartmental clearance, Q_{12}, divided by the volume of `C1`. The reverse rate parameter is Q_{12}, divided by the volume of `C2`.

The process of adding each pair of compartments in the chain C_m and C_n generates the following model components:

- A parameter Q_{mn} representing the compartmental clearance between those two compartments. This parameter is added to the list of parameters to be estimated (`PKModelMapObj.Estimated` property).
- A parameter (`kmn`) representing the rate of transfer of the drug from `Cm` to `Cn`, where $k_{mn} = Q_{mn}/V_m$.
- A parameter (`knm`) representing the rate of `Cn` to `Cm`, where $k_{nm} = Q_{mn}/V_n$.

- A reversible mass-action reaction between the two compartments, `Cm.Drug_Cm <-> Cn.Drug_Cn`, with forward rate parameter `kmn`, and reverse rate parameter `knm`.
- An initial assignment rule that initializes the value of the parameter `kmn`, based on the initial values for `Cm` and `Qmn`.
- An initial assignment rule that initializes the value of the parameter `knm`, based on the initial values for `Cn` and `Qmn`.

Unit Conversion for Imported Data and the Model

Unit conversion converts the matching physical quantities to one consistent unit system in order to resolve them. This conversion is in preparation for correct simulation, but SimBiology returns the physical quantities in the model in units that you specify.

Regardless of whether unit conversion is `on` or `off`, you must express dosing data in amount. By default, **Unit Conversion** is `off`, so you must ensure that units for the data and the model are consistent with one another.

If **Unit Conversion** is `on`, you must specify units. If using the SimBiology desktop, specify units in the **Raw Data** tab, when data is selected in the **Project Explorer**. If using the command line, specify units in the `PKData` object.

Parameters in the model have default units. If unit conversion is `on`, you can change the units as long as the dimensions are consistent. These default units, which you might use to specify the values for the initial guess, are as follows.

Physical Quantity or Model Parameter	Unit
Capacity (compartment volume)	`liter`
First-order elimination rate	`1/second`
`Km` — Michaelis constant	`milligram/liter`
`Vm` — (`Vmax`) Maximum reaction-velocity (Michaelis-Menten kinetics)	`milligram/second`
Clearance	`liter/second`

Physical Quantity or Model Parameter	Unit
`Tk0` (absorption duration)	`second`
`ka` (absorption rate)	`1/second`

Use the configuration settings options to turn unit conversion `on` or `off`. For more information see, "Simulating Models" on page 3-3 in the SimBiology documentation.

See "How Reaction Rates Are Evaluated" on page 1-19 in the SimBiology documentation for more information on dimensional analysis for reaction rates.

Prerequisites for Using Custom SimBiology Models in Parameter Fitting

Overview

If you created a PK model using either the `PKModelDesign` object's `construct` method at the command line or the wizard in the SimBiology desktop, you can skip this section. This section provides information about working with a custom SimBiology model.

When using a custom model, you must provide information about whether dosing is applicable and define which components of the SimBiology model represent the observed response, the dose, and the estimated parameters. Use the `PKModelMap` object to define these settings as shown in "Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters" on page 4-29.

Another point to consider is the solver you use in performing simulations. During fitting the solver type must be `sundials` to support any events in the model. See "About Simulation Settings and Specifying Alternate Values for Initial Estimates" on page 4-51, for more information.

Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters

The `PKModelMap` object holds information about the dosing type and defines which components of the SimBiology model represent the observed response, the dose, and the parameters to be estimated.

If you are using a custom SimBiology model that you did not create using either the `PKModelDesign` object's `construct` method or the wizard, you must create a `PKModelMap` object to define these relationships.

Consider the following regression function, `y = f(k,u)`, where `y` is the measured or observed response, given values of an input `u`, and parameter values `k`. In SimBiology, the model represents `f`, which is used to generate the regression function, if `y`, `k`, and `u` are identified in the model. You must, therefore, use the `PKModelMap` object to define which components of the model represent `y`, `k`, and `u`. If applicable, the `PKModelMap` object also needs information on the type of dosing or input being given to the model.

1. Import an SBML model:

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

2. Create a `PKModelMap` object:

   ```
   PKModelMapObj = PKModelMap;
   ```

3. Use the name of the model component to specify the corresponding property in the `PKModelMap` object.

Model Component Represents	PKModelMap Object Property
Object being driven by an input	`Dosed`
Measured response	`Observed`
Parameters to be estimated	`Estimated`

   For example:

   ```
   set(PKModelMapObj, 'Observed', 'unnamed.y1');
   ```

```
set(PKModelMapObj, 'Estimated', {'Reaction1.c1', 'Reaction2.c2'});
```

Note When specifying species names, qualify the name with the compartment name in the form `compartmentName.speciesName` (for example, `nucleus.DNA`). For names of parameters scoped at the reaction level, use `reactionName.parameterName`. For parameters scoped at the model level, you do not have to qualify the name.

4 Use the `DosingType` property to specify the type of dosing, if applicable. The allowed types are `''`, `'Bolus'`, `'Infusion'`, `'FirstOrder'`, and `'ZeroOrder'`.

For example:

```
set(PKModelMapObj, 'DosingType', 'Bolus');
```

Note When using custom models with `DosingType` set to `zero-order`, you must include a parameter that represents the duration of drug absorption. Set the `ZeroOrderDurationParameter` property of the `PKModelMap` object to the name of the duration parameter. For example, `set(PKModelMapObj, 'ZeroOrderDurationParameter', 'Kdo');`.

The previous example sets the observed response to a species `y1`, contained by a compartment (`unnamed`), and sets the parameters to be estimated to the parameters `c1` and `c2` that are scoped to the reactions, `Reaction1` and `Reaction2`, respectively.

For information on ...	See ...
`PKModelMap` object properties and allowed values	`PKModelMap object Dosed`, `DosingType`, `Estimated`, and `Observed`, `ZeroOrderDurationParameter`
Allowed dosing types	"About Dosing Types" on page 4-21

For information on ...	See ...
Parameter scoping	"When Reactions, Rules, and Events Specify Parameters" on page 1-14
Parameter fitting	"Fitting Pharmacokinetic Model Parameters" on page 4-34

Dosing Multiple Compartments in a Model

1. Use the name of the model component to specify the `Dosed` property in the `PKModelMap` object.

   For example, assume that a model contains two compartments named `Central` and `Peripheral`. Specify the species names in the dosed compartments. For example:

   ```
   set(PKModelMapObj, 'Dosed, {'Central.Drug_Central', ...
      'Peripheral.Drug_Peripheral'});
   ```

2. Use the `DosingType` property to specify the type of dosing if applicable. The allowed types are `''`, `'Bolus'`, `'Infusion'`, `'FirstOrder'`, and `'ZeroOrder'`. When specifying dosing for multiple compartments, the order in the `Dosed` property is the order in which the dosing type is applied.

   For example, if `Central` takes zero-order dosing and `Peripheral` takes a first-order dosing enter:

   ```
   set(PKModelMapObj, 'DosingType', {'ZeroOrder', 'FirstOrder'});
   ```

3. Because the model includes `zero-order` as a `DosingType`, you must include a parameter that represents the duration of drug absorption and is used when simulating the model with dosing information or during fitting. Set the `ZeroOrderDurationParameter` property of the `PKModelMap` object to the name of the duration parameter. For example,

   ```
   set(PKModelMapObj, 'ZeroOrderDurationParameter', {'Kdo', ''})
   ```

   Specify the parameters in the same order as the species in the `Dosed` property.

Parameter Fitting in Pharmacokinetic Models

In this section...

Parameter Fitting Functionality

SimBiology lets you perform individual and population fitting on grouped data. This functionality uses Statistics Toolbox features (Version 7.3 or later).

- Individual fit — Fit data separately for each individual using the nonlinear least squares method, estimate parameters, and calculate residuals and the estimated coefficient covariance matrix.
- Population fit — Estimate the fixed effects and the random sources of variation on parameters, using nonlinear mixed-effects models.

  You can use the following methods to estimate the fixed effects:

  - `LME` — Linear mixed-effects approximation
  - `RELME` — Restricted LME approximation
  - `FO` — First-order estimate
  - `FOCE` — First-order conditional estimate

The following results are returned for population fitting:

- The maximized log-likelihood for the fitted model
- The estimated error variance for the fitted model
- The Akaike information criterion for the fitted model
- The Bayesian information criterion for the fitted model
- The standard errors for the estimates of the fixed effects
- The error degrees of freedom for the model
- The weighted residuals for the fitted model

In addition, you can generate diagnostic plots that show:

- The predicted time courses and observations for an individual or the population
- Observed versus predicted values
- Weighted residuals versus time, group, or predictions
- Distribution of the weighted residuals
- A box-plot for random effects or parameter estimates from individual fitting

Prerequisites for Parameter Fitting

Before you fit parameters, the SimBiology desktop or the MATLAB Workspace must contain the following:

- Data to use in the fitting (See "Importing Data — Supported Files and Data Types" on page 4-8 for more information.)
- A model to fit (See "Creating Pharmacokinetic Models" on page 4-17 for more information.)

If you plan to use the command line, see the following for more information:

"Fitting Pharmacokinetic Model Parameters" on page 4-34

Fitting Pharmacokinetic Model Parameters

In this section...

Fitting Parameters

The following steps show one of the workflows you can use at the command line to fit a PK model and estimate parameters:

1. Import data as shown in "Importing Data" on page 4-13.
2. Specify the structural model by creating a PK model as shown in "Creating PK Models" on page 4-19. Alternatively, if you have a SimBiology model that you want to use in fitting, see "Prerequisites for Using Custom SimBiology Models in Parameter Fitting" on page 4-28.
3. Classify the data set to use in fitting. See "Specifying and Classifying the Data to Fit" on page 4-35.
4. Specify the initial guesses for the parameters to be estimated, as shown in "Setting Initial Estimates" on page 4-37.

5. Perform individual or population fits:
   - For individual fits:
     - (Optional) Set tolerances and specify maximum iterations.
   - For population fits:
     - Specify the statistical model:

       Specify the covariate model and the covariance matrix. See "Specifying a Covariate Model" on page 4-40 and "Specifying the Covariance Pattern of Random Effects" on page 4-41.

       Specify the error model. See "Specifying an Error Model" on page 4-43.
     - (Optional) Set tolerances and specify maximum iterations.
6. Obtain and visualize results.

Specifying and Classifying the Data to Fit

In order to use the imported data in fitting, you must identify required columns in the data set that was previously imported as shown in "Importing Data" on page 4-13.

Use the `PKData` object to specify the data set containing the observed data to use in fitting. The properties of the `PKData` object specify what each column in the data represents.

To create the `PKData` object:

1. Create the `PKData` object for the data set `data`.

   ```
   pkDataObject = PKData(data);
   ```

   `PKData` assigns the data set `data` to the read-only `DataSet` property.

2. Use the column headers in the data set to specify the following properties for the column in the data set.

Column in Data Set Represents	PKData Object Property
Group identification labels	`GroupLabel`
Independent variable (For example, time)	`IndependentVarLabel`
Dependent variable (For example, measured response)	`DependentVarLabel`
Amount of dose given	`DoseLabel`
Rate of infusion (when applicable). Data must contain rate (`amount/time`) and not infusion time.	`RateLabel`
Covariates (For example, age, gender, weight)	`CovariateLabels`

For example, for the tobramycin data set [1]:

```
pkDataObject.GroupLabel          = 'ID';
pkDataObject.IndependentVarLabel = 'Time';
pkDataObject.DependentVarLabel   = 'Response';
pkDataObject.DoseLabel           = 'Dose';
pkDataObject.CovariateLabels     = {'WT','HT','AGE', 'SEX', 'CLCR'};
```

Note For the subset of data belonging to a single group (as defined by the column in your data set that represents group identification labels, which you map to 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 number of observations at different times cause some time points to be weighted more.

Tip If dosing applies to more than one compartment in the model, specify the `DoseLabel` property as follows:

```
pkDataObject.DoseLabel =  {'Dose1', 'Dose2'};
```

`Dose1` and `Dose2` are names of columns containing dose information for compartments. A one-to-one relationship must exist between the number and order of elements in the `DoseLabel` property and the `Dosed` property of the corresponding `PKModelMap` object.

Tip If your model measures multiple responses, specify the `DependentVarLabel` property as follows:

```
pkDataObject.DependentVarLabel =  {'Response1', 'Response2'};
```

`Response1` and `Response2` are names of columns containing response measurements. A one-to-one relationship must exist between the number and order of elements in the `DependentVarLabel` property and the `Observed` property of the corresponding `PKModelMap` object.

When you assign a column containing group identification labels to the `GroupLabel` property, `PKData` sets these read-only properties as follows:

- The `GroupNames` property is set to the unique names found in the group column.
- The `GroupID` property is set to an integer corresponding to the unique names found in the group column.

Setting Initial Estimates

To set the initial estimates (or initial guesses) for the parameters with fixed effects to estimate, first identify the sequence of the parameters in the model by querying the `PKModelMap` object. Next, construct a vector, `beta0`, containing the initial conditions. For information about `PKModelMap` objects, see step 4 in "Creating PK Models" on page 4-19.

1 Query the `Estimated` property of the `PKModelMap` object:

```
PKModelMapObj.Estimated
```

MATLAB returns the sequence of the parameters to be estimated. For example:

```
ans =

    'Central'
    'Cl_Central'
```

2 Set the initial estimates for the parameters. For example:

```
beta0 = [10.0, 1.0];
```

For information on ...	See ...
The parameters added to the model	• "About Dosing Types" on page 4-21 • "About Elimination Types" on page 4-24
Default units for the above parameters	"Unit Conversion for Imported Data and the Model" on page 4-27

Specifying a Nonlinear, Mixed-Effects Model

Suppose data for a nonlinear regression model falls into one of m distinct groups $i = 1, ..., m$. (Specifically, suppose that the groups are not nested.) To specify a general, nonlinear, mixed-effects (NLME) model for this data:

1 Define group-specific model parameters φ_i as linear combinations of fixed effects β and random effects b_i.

2 Define response values y_i as a nonlinear function f of the parameters and group-specific covariate variables X_i.

The model is:

$$\varphi_i = A_i\beta + B_i b_i, where\ b_i \sim N(0,\Psi)$$
$$y_i = f(\varphi_i, X_i) + \varepsilon_i$$
$$Alternatively, \log y_i = \log f(\varphi_i, X_i) + \varepsilon_i$$

This formulation of the nonlinear, mixed-effects model uses the following notation:

φ_i	A vector of group-specific model parameters
β	A vector of fixed effects, modeling population parameters
b_i	A vector of multivariate, normally distributed, group-specific, random effects
A_i	A group-specific design matrix for combining fixed effects
B_i	A group-specific design matrix for combining random effects
X_i	A data matrix of group-specific covariate values
y_i	A data vector of group-specific response values
f	A general, real-valued function of φ_i and X_i
ε_i	• For `sbionlmefit`, you can specify different error models as shown in "Specifying an Error Model" on page 4-43. • For `sbionlmefitsa`, you can specify different error models as shown in "Specifying an Error Model" on page 4-43.
Ψ	A covariance matrix for the random effects
σ^2	The error variance, assumed to be constant across observations

For example, consider a one-compartment model with first-order dosing and linear clearance. The group-specific parameters (φ) in the model are clearance (Cl), compartment volume (V), and absorption rate constant (k_a). From the model:

$$\begin{pmatrix} Cl \\ V \\ ka \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \beta_{Cl} \\ \beta_V \\ \beta_{ka} \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} b_{Cl} \\ b_V \\ b_{ka} \end{pmatrix}$$

In SimBiology, B_i is an identity matrix. That is, `sbionlmefit` does not support the specification of a different random-effects design matrix. You can alter the design matrices, as necessary, to introduce weighting of individual effects.

The Statistics Toolbox function `nlmefit` fits the general, nonlinear, mixed-effects model to data, estimating the fixed and random effects. The function also estimates the covariance matrix Ψ for the random effects. Additional diagnostic outputs allow you to assess trade-offs between the number of model parameters and the goodness of fit. See "Mixed-Effects Models" in the Statistics Toolbox documentation for more information.

Specifying a Covariate Model

If the NLME model assumes a group-dependent covariate such as weight (w), the model becomes:

$$\begin{pmatrix} Cl \\ V \\ ka \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & w_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \beta_{Cl} \\ \beta_V \\ \beta_{ka} \\ \beta w \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} b_{Cl} \\ b_V \\ b_{ka} \end{pmatrix}$$

Thus, the parameter for clearance (Cl) for an individual is $Cl_i = \beta_{Cl} + \beta_{Cl/w} * w_i + b_{Cli}$.

Assuming that `1` is the initial estimate for β_{Cl} and `0` is the initial estimate for $\beta_{Cl/w}$, use the `PKCovariateModel.Expression` property to define the covariate model.

```
covmodel = PKCovariateModel;
covmodel.Expression = ({'Cl = 1*exp(0*weight)'});
```

Use the `PKCovariateModel.getInitialEstimate` method to determine the order of elements:

```
[initialEstimates elementNames] = ...
   covmodel.getInitialEstimate(PKModelMapObj, PKDataObj)];
disp('Element Names:');
disp(elementNames);
```

Your output appears as follows:

```
Element Names:
    'Central'
    'Cl_Central'
    'Central/WEIGHT'
    'Cl_Central/WEIGHT'
```

Update the initial estimates using the values estimated from fitting the base model (model without covariate effects):

```
covmodel.setInitialEstimate(PKModelMapObj, PKDataObj, ...
   [nlmeResults.estimate' 0 0 ]);
disp('Covariate model expression with updated initial estimates:');
disp(covmodel.Expression)
```

Your output appears as follows:

```
Covariate model expression with updated initial estimates:
    'Central = 1.40855891462797 * exp(0*(WEIGHT))'
    'Cl_Central = 0.00613734942978561 * exp(0*(WEIGHT))'
```

Specifying the Covariance Pattern of Random Effects

By default, the function you use to perform population fits (`nlmefit` or `nlmefitsa`) assumes a diagonal covariance matrix (no covariance among the random effects). To specify a different covariance pattern of random effects, use the `'CovPattern'` option. In the previous example, assuming that each of the parameters has random effects and that Cl and V exhibit covariance, the covariance pattern of random effects would be a logical array:

$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

1. Create an options `struct` with the specified covariance pattern:

   ```
   options.CovPattern = [1, 1, 0; 1, 1, 0; 0, 0, 1];
   ```

2. Specify the arguments for `sbionlmefit` or `sbionlmefitsa`:

   ```
   [results, simdataI, simdataP] = sbionlmefit(modelObj,...
        PKModelMapObj, PKDataObj, beta0, options)
   ```

If you are using the tobramycin data set [1], do the following:

1. Create an options `struct` with the specified covariance pattern:

```
options.CovPattern = [1, 0; 0, 1];
```

2. Specify the arguments for `sbionlmefit`:

```
[results, simdataI, simdataP] = sbionlmefit(modelObj,...
     PKModelMapObj, PKDataObj, beta0, options)

results =

    estimate: [2x1 double]
        phiI: [2x97 double]
        phiP: [2x97 double]
        beta: [2x1 double]
         psi: [2x2 double]
       stats: [1x1 struct]
           b: [2x97 double]

results.estimate

ans =

   21.6535
    3.7172

results.beta

ans =

    3.0752
    1.3130
```

For more information, see `nlmefit` or `nlmefitsa` in the Statistics Toolbox documentation.

Fitting the model and estimating the covariance matrix Ψ often leads to further refinements. A relatively small estimate for the variance of a random effect suggests that it can be removed from the model. Similarly, relatively small estimates for covariances among certain random effects suggest that a full covariance matrix is unnecessary. Since random effects are unobserved, Ψ must be estimated indirectly. Specifying a diagonal or block-diagonal covariance pattern for Ψ can improve convergence and efficiency of the fitting algorithm.

Specifying an Error Model

You can specify error models with the `sbionlmefitsa` or `sbionlmefit` function.

To define an error model, use the `ErrorModel` option. Each model defines the error using a standard normal (Gaussian) variable e, the function value f, and one or two parameters, a and b. You can specify the following error models:

- `constant`: `y = f + a*e`
- `proportional`: `y = f + b*f*e`
- `combined`: `y = f + (a+b*f)*e`
- `exponential`: `y = f*exp(a*e)` or `log(y) = log(f) + a*e`

1. Use `ErrorModel` in `nlmefitsa` or `nlmefit`. Create a `struct` with the specified error model. For example:

   ```
   options.ErrorModel = 'proportional';
   ```

2. (Optional) Specify starting values for parameters of the error model using `ErrorParameters`. For example:

   ```
   options.ErrorParameters = 1;
   ```

Tip To specify `a` and `b` for `combined`, enter:

```
options.ErrorParameters = [1 1];
```

---

3 Specify the arguments for `sbionlmefitsa` or `sbionlmefit`, as shown in "Performing Population Fitting Using sbionlmefit or sbionlmefitsa" on page 4-45.

See also `nlmefitsa` or `nlmefit`in the Statistics Toolbox documentation.

Specifying Parameter Transformations

To specify parameter transformations, use the `ParamTransform` option in `sbionlinfit`, `sbionlmefit` and `sbionlmefitsa`. The `ParamTransform` option lets you specify either no transformation, or the `log`, `probit`, or `logit` transformation.

The underlying algorithm in `nlmefit` assumes that parameters follow a normal distribution. This assumption may not hold for biological parameters that are constrained to be positive, such as volume and clearance. You may specify a transformation function for the estimated parameters, so that the transformed parameters follow a normal distribution.

By default, the SimBiology fitting functions choose a log transform for all estimated parameters. Parameters that are constrained between the values `0` and `1`, like absorption fraction, can be transformed by the `probit` or `logit` transformations described below.

The `probit` function is the inverse cumulative distribution function (CDF) associated with the standard normal distribution. To apply the `probit` transform to a variable `x` in MATLAB, use the Statistics Toolbox function `norminv`: `t = norminv(x)`. To untransform a variable `t`, use the function `normcdf`: `x = normcdf(t)`.

The `logit` function is the inverse of the sigmoid function. To apply the `logit` transform to a variable `x` in MATLAB, use the following expression: `t = log(x) - log(1-x)`. To untransform the variable `t`, use `x = 1/(1+exp(-t))`.

1 For the `ParamTransform` option, specify a vector of values equal to the number of parameters to be estimated. The values must be one of the integer codes listed in `nlmefitsa` or `nlmefit` specifying the transformation for the corresponding value of the parameters to be estimated. For example

```
options.ParamTransform = [0 1 2];
```

See `nlmefit` and `nlmefitsa` for more information.

2 Specify the arguments for `sbionlmefit` or `sbionlmefitsa`, as shown in "Performing Population Fitting Using sbionlmefit or sbionlmefitsa" on page 4-45.

For individual fitting, see "Performing Individual Fitting Using sbionlinfit" on page 4-49.

Performing Population Fitting Using sbionlmefit or sbionlmefitsa

The `sbionlmefit` and `sbionlmefitsa` functions let you specify a SimBiology model that you want to use in fitting. These functions use the `nlmefit` and `nlmefitsa` functions from the Statistics Toolbox to fit data with both fixed and random sources of variation using nonlinear mixed-effects and return the estimates. `nlmefit` fits the model by maximizing an approximation to the marginal likelihood with random effects integrated out assuming the following:

- Random effects are multivariate, normally distributed, and independent between groups.
- Observation errors are independent, identically normally distributed, and independent of random effects.

1 (Optional) Set the tolerance and maximum iteration options. Use an options structure that is an argument for `sbionlmefit`:

```
options.Options.TolX = 1.0E-4;
options.Options.TolFun = 1.0E-4;
options.Options.MaxIter = 200;
```

2 Specify the model object, the `PKModelMap` object, the `PKData` object, the `PKCovariateModel` object, a vector containing the initial estimates for the fixed effects, and the options:

```
[results, simdataI, simdataP] = sbionlmefit(modelObj,...
 PKModelMapObj, pkDataObject, PKCovariateModelObject, beta0, options);
```

or, if `options` includes an error model, you must use `sbionlmefitsa`:

```
[results, simdataI, simdataP] = sbionlmefitsa(modelObj,...
 PKModelMapObj, pkDataObject, PKCovariateModelObject, beta0, options);
```

Note If your population fit uses multiple doses, make sure each element in the `Dosed` property of the `PKModelMap object` is unique.

Note In your `PKData object`, 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 number of observations at different times cause some time points to be weighted more.

`sbionlmefit` and `sbionlmefitsa` return the following:

- A `results` structure containing estimated values and other statistics. For more information, see the `sbionlmefit` and `sbionlmefitsa` reference pages.
- *`simdataI`*, a `SimData object` containing the data from simulating the model using the estimated parameter values for individuals, which includes both the fixed and random effects.
- *`simdataP`*, `SimData object` containing the data from simulating the model using the estimated parameter values for the population, which includes only the fixed effects.

3 Plot the data from the data set. For example, in the imported data set used for fitting, `ds`, `ID`, `Time`, and `Response` are the column headers for the columns containing group IDs, time, and the response variable, respectively.

```
p = sbiotrellis(ds, 'ID', 'Time', 'Response')
```

Note If your data set has multiple responses, with column headers `Response1` and `Response2` containing the response variables, you plot the data as follows:

```
Response = {'Response1', 'Response2'}
p = sbiotrellis(ds, 'ID', 'Time', Response)
```

4 Use the `plot` method on the trellis plot object `p`, returned by `sbiotrellis` to overlay data, using default values for the second and third input arguments.

```
p.plot(simdataP, [], '', PKModelMapObj.Observed);
```

For a description of the results, see `sbionlmefit` in the SimBiology documentation.

For more information, see the following topics in the Statistics Toolbox documentation:

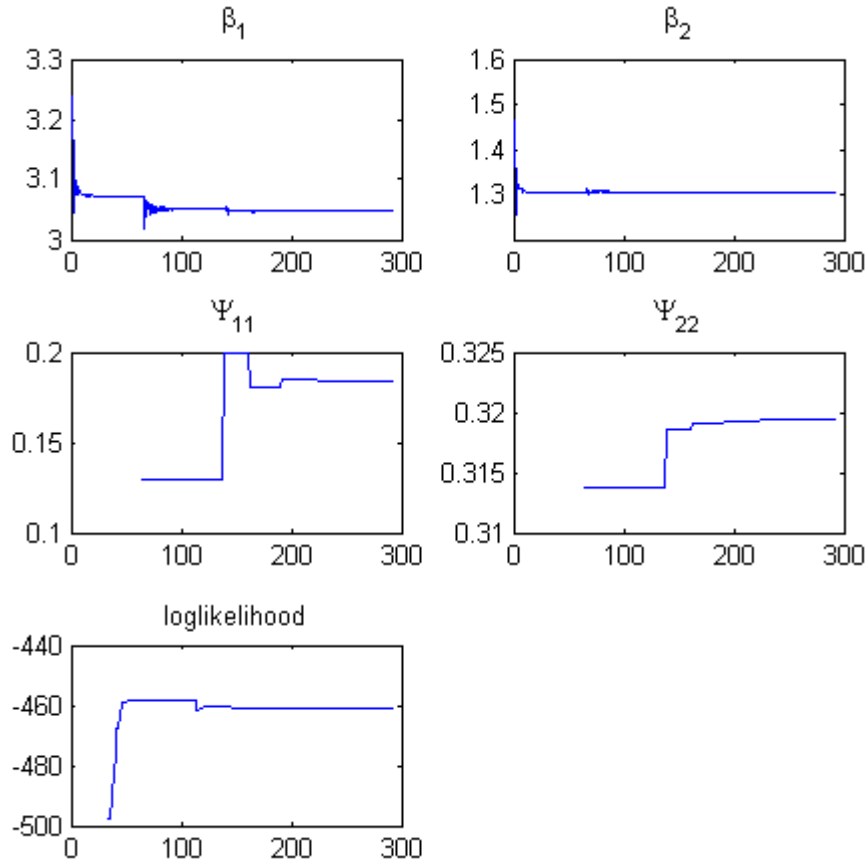
- "Nonlinear Regression Models"
- "Mixed-Effects Models"
- `nlmefit`

Obtaining the Status of Fitting

The `sbiofitstatusplot` function dynamically plots the progress of the fitting task. During the task, the function plots the fixed effects (ß), the estimates for the diagonal elements of the covariance matrix for the random effects (Ψ), and the log-likelihood. This functionality is useful for large and complex models when you expect the time to return the results to be longer than a few minutes. Use the options structure that is an argument for the `sbionlmefit` function:

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

The following figure shows the type of plots obtained.



Tips for interpreting status plots:

- The fitting function tries to maximize the log-likelihood. When the plot begins to display a flat line, this might indicate that maximization is complete. Try setting the maximum iterations to a lower number to reduce the number of iterations you need and improve performance. For information on how to set iteration options, see “Performing Population Fitting Using sbionlmefit or sbionlmefitsa” on page 4-45.

- Plots for the fixed effects (ß) and the estimates for the diagonal elements of the covariance matrix for the random effects (Ψ), should show convergence. If you see oscillations, or jumps without accompanying improvements in the log-likelihood, the model may be over-parameterized. Try the following:
  - Reduce the number of fixed effects.
  - Reduce the number of random effects.
  - Simplify the covariance matrix pattern of random effects.

See also `sbiofitstatusplot` in the SimBiology documentation.

Performing Individual Fitting Using sbionlinfit

The `sbionlinfit` function lets you specify a SimBiology model to fit using the `nlinfit` function (individual fit). The `nlinfit` function fits using nonlinear least squares and returns parameter estimates, residuals, and the estimated coefficient covariance matrix.

1 (Optional) Set the tolerance and maximum iteration options:

```
options.TolX = 1.0E-8;
options.TolFun = 1.0E-8;
options.MaxIter = 100;
```

2 Specify the model object, the `PKModelMap` object, the `PKData` object, a vector containing the initial estimates for the fixed effects, and the options:

```
[results, simdataI] = sbionlinfit(modelobj,...
 PKModelMapObj, PKDataObj, beta0, options);
```

Note If your individual fit uses multiple doses, make sure each element in the `Dosed` property of the `PKModelMap object` is unique.

Note In your `PKData object`, 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 number of observations at different times cause some time points to be weighted more.

`sbionlinfit` returns the following:

- A `results` array of structures, each containing the following for one group:
  - `beta` — Fitted coefficients
  - `R` — Residuals
  - `J` — Jacobian of `modelObject`
  - `COVB` — Estimated covariance matrix for the fitted coefficients
  - `mse` — Estimate of the error of the variance term
- *`simdataI`* contains the data from simulating the model using the estimated parameter values, for individuals.

3 Plot the data from the data set. For example, in the imported data set (`ds`), `ID`, `Time` and `Response` are the column headers for the columns containing group IDs, time, and the response variable respectively.

```
p = sbiotrellis(ds, 'ID', 'Time', 'Response')
```

Note If your data set has multiple responses, with column headers `Response1` and `Response2` containing the response variables, you plot the data as follows:

```
Response = {'Response1', 'Response2'}
p = sbiotrellis(ds, 'ID', 'Time', Response)
```

4 Use the plot method on the trellis plot object `p`, returned by `sbiotrellis` to overlay data, using default values for the second and third input arguments.

```
p.plot(simdataI, [], '', PKModelMapObj.Observed);
```

For more information, see "Nonlinear Regression Models" and `nlinfit` in the Statistics Toolbox documentation.

About Simulation Settings and Specifying Alternate Values for Initial Estimates

Use the `Variant object` to store and apply alternate values for model components during a simulation. For more information about variants and how to add variants see .

Note The fitting functions, `population fit (NLMEFIT)` and `individual fit (NLINFIT)`, use the initial estimate values as the initial parameter estimates when fitting parameters (overriding the values in the variant). Specify initial estimate values, as shown in "Setting Initial Estimates" on page 4-37.

Use the `Configset object` to change settings for simulations. For more information about performing simulations, see `sbiosimulate`. The model object created using the `PKModelDesign` object's `construct` method contains a default configuration set that uses the `sundials` solver.

If you change the solver type in the configuration set during fitting, SimBiology temporarily changes the solver to `sundials` to support events in the model. SimBiology reverses the change after returning the results. If you want to change tolerance options for simulations select a deterministic solver and use the tolerance options provided in the deterministic solver. The fitting functions (`sbionlmefit` or `sbionlinfit`) will use the tolerance options specified in the deterministic solver.

A

Creating Reaction Rates

Defining Reaction Rates with Mass Action Kinetics

Definition of Mass Action Kinetics

Mass action describes the behavior of reactants and products in an elementary chemical reaction. Mass action kinetics describes this behavior as an equation where the velocity or rate of a chemical reaction is directly proportional to the concentration of the reactants.

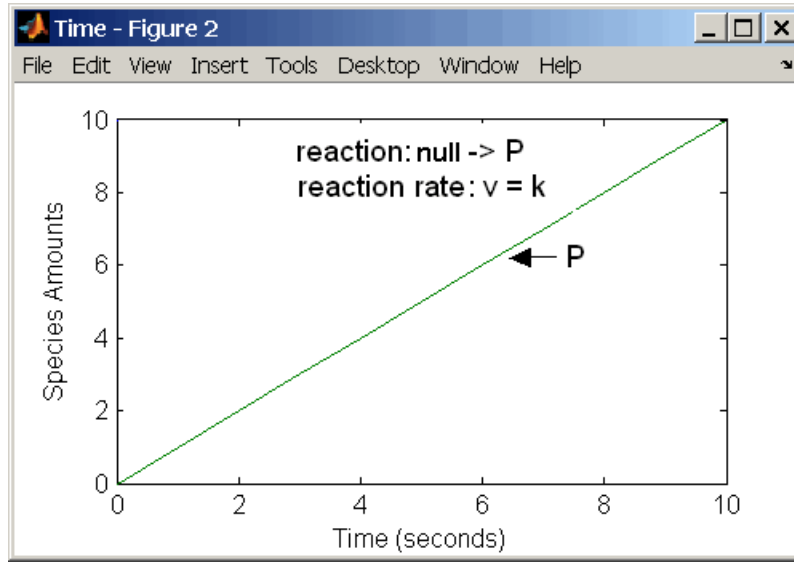
Zero-Order Reactions

With a zero-order reaction, the reaction rate does not depend on the concentration of reactants. Examples of zero-order reactions are synthesis from a `null` species, and modeling a source species that is added to the system at a specified rate.

```
      reaction: null -> P
 reaction rate: k mole/second
       species: P =  0 mole
    parameters: k =  1 mole/second
```

Note When specifying a `null` species, the reaction rate must be defined in units of amount per unit time not concentration per unit time.

Entering the reaction above into the software and simulating produces the following result:



Zero-Order Mass Action Kinetics

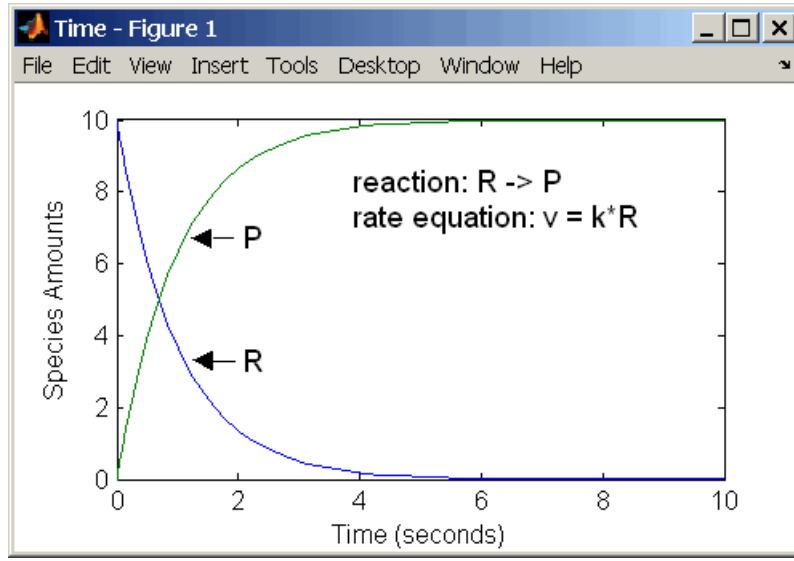
Note If the amount of a reactant with zero-order kinetics reaches zero before the end of a simulation, then the amount of reactant can go below zero regardless of the solver or tolerances you set.

First-Order Reactions

With a first-order reaction, the reaction rate is proportional to the concentration of a single reactant. An example of a first-order reaction is radioactive decay.

```
      reaction: R -> P
 reaction rate: k*R mole/(liter*second)
       species: R = 10 mole/liter
                P =  0 mole/liter
    parameters: k =  1 1/second
```

Entering the reaction above into the software and simulating produces the following results:



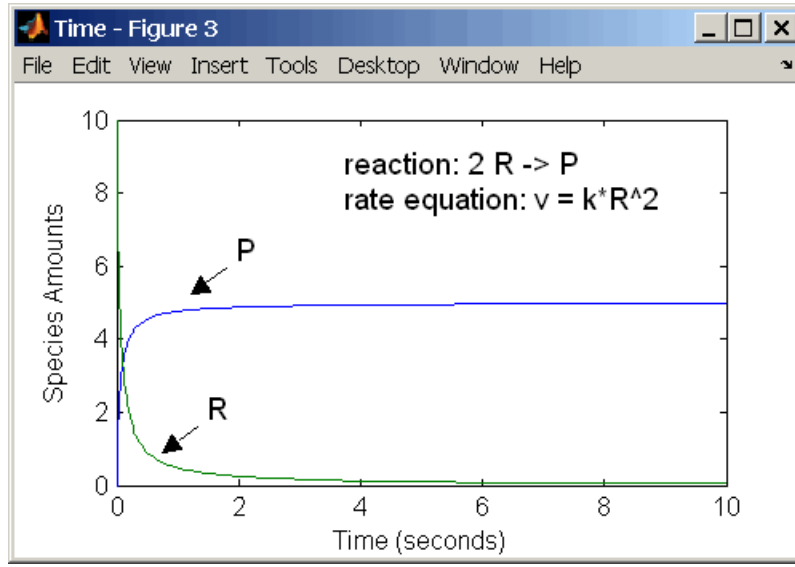
First-Order Mass Action Kinetics

Second-Order Reactions

A second-order reaction has a reaction rate that is proportional to the square or the concentration of a single reactant or proportional to two reactants. Notice the space between the reactant coefficient and the name of the reactant. Without the space, `2R` would be considered the name of a species.

```
      reaction: 2 R -> P
 reaction rate: k*R^2 mole/(liter*second)
       species: R = 10 mole/liter
                P =  0 mole/liter
    parameters: k =  1 liter/(mole*second)
```

Entering the reaction above into the software and simulating produces the following results:

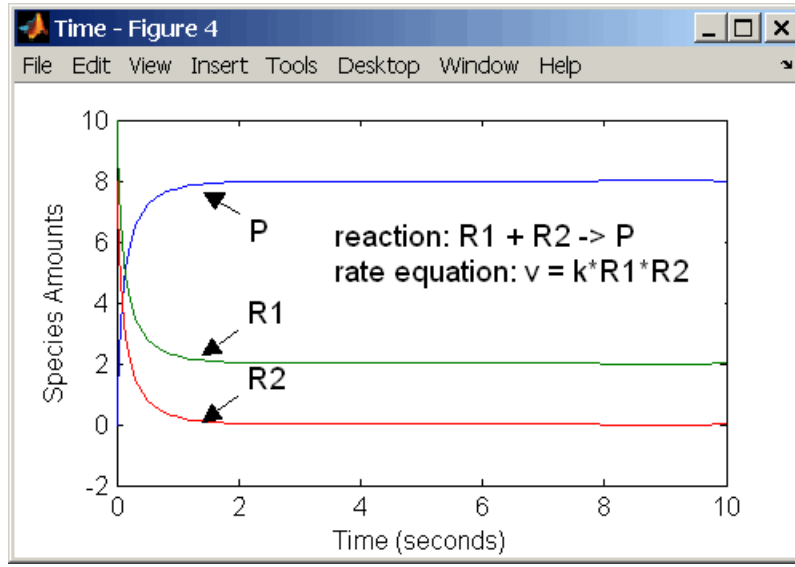


Second-Order Kinetics with Single Reactant

With two reactants, the reaction rate depends on the concentration of two of the reactants.

```
      reaction: R1 + R2 -> P
reaction rate: k*R1*R2 mole/(liter*second)
      species: R1 = 10 mole/liter
               R2 =  8 mole/liter
                P =  0 mole/liter
   parameters:  k =  1 liter/(mole*second)
```

Enter the reaction above into the software and simulating produces the following results. There is a difference in the final values because the initial amount of one of the reactants is lower than the other. After the first reactant is used up, the reaction stops.



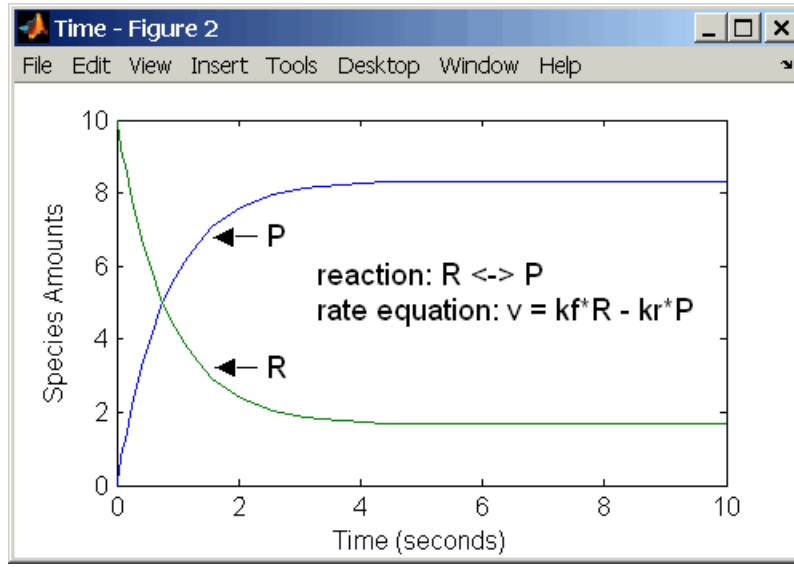
Second-Order Kinetics with Two Reactants

Reversible Mass Action

You can model reversible reactions with two separate reactions or with one reaction. With a single reversible reaction, the reaction rates for the forward and reverse reactions are combined into one expression. Notice the angle brackets before and after the hyphen to represent a reversible reaction.

```
      reaction: R <-> P
 reaction rate: kf*R - kr*P mole/(liter*second)
       species: R = 10   mole/liter
                P =  0   mole/liter
    parameters: kf = 1    1/second
                kr = 0.2  1/second
```

Entering the reaction above into the software and simulating produces the following results. At equilibrium when the rate of the forward reaction equals the reverse reaction, `v = kf*R - kr*P = 0` and `P/R = kf/kr`.


Time - Figure 2
File Edit View Insert Tools Desktop Window Help
Species Amounts
Time (seconds)
P
R
reaction: R <-> P
rate equation: v = kf*R - kr*P
0
2
4
6
8
10

Defining Reaction Rates with Enzyme Kinetics

Simple Model for Single Substrate Catalyzed Reactions

A simple model for enzyme-catalyzed reactions starts a substrate `S` reversibly binding with an enzyme `E`. Some of the substrate in the substrate/enzyme complex is converted to product `P` with the release of the enzyme.

$$S + E \underset{k1r}{\overset{k1}{\rightleftharpoons}} ES \xrightarrow{k2} E + P$$

$$v1 = k1[S][E], \quad v1r = k1r[ES], \quad v2 = k2[ES]$$

This simple model can be defined with

- Differential rate equations. See “Enzyme Reactions with Differential Rate Equations” on page A-8.
- Reactions with mass action kinetics. See “Enzyme Reactions with Mass Action Kinetics” on page A-10.
- Reactions with Henri-Michaelis-Menten kinetics. See “Enzyme Reactions with Irreversible Henri-Michaelis-Menten Kinetics” on page A-11.

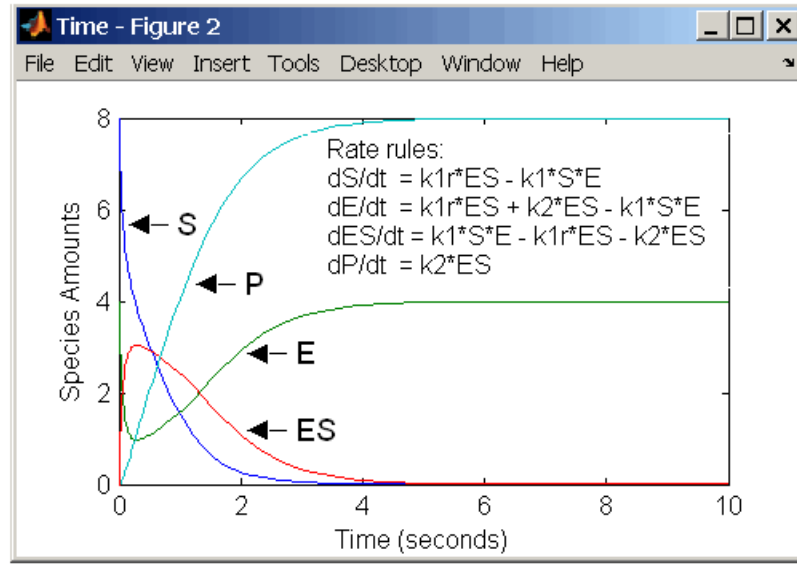
Enzyme Reactions with Differential Rate Equations

The reactions for a single-substrate enzyme reaction mechanism (see “Simple Model for Single Substrate Catalyzed Reactions” on page A-8) can be described with differential rate equations. You can enter the differential rate equations into the software as rate rules.

```
     reactions: none
reaction rate: none
   rate rules: dS/dt  = k1r*ES - k1*S*E
               dE/dt  = k1r*ES + k2*ES - k1*S*E
               dES/dt = k1*S*E - k1r*ES - k2*ES
               dP/dt  = k2*ES
      species: S =  8   mole
               E =  4   mole
              ES =  0   mole
               P =  0   mole
```

```
   parameters:  k1 = 2   1/(mole*second)
               k1r = 1   1/second
                k2 = 1.5 1/second
```

Remember that the rate rule `dS/dt = f(x)` is written in a SimBiology rate rule expression as `S = f(x)`. For more information about rate rules see "Rate Rules" on page 1-24.



Alternatively, you could remove the rate rule for `ES`, add a new species `Etotal` for the total amount of enzyme, and add an algebraic rule `0 = Etotal - E - ES`, where the initial amounts for `Etotal` and `E` are equal.

```
      reactions: none
  reaction rate: none
     rate rules: dS/dt = k1r*ES - k1*S*E
                 dE/dt = k1r*ES + k2*ES - k1*S*E
                 dP/dt = k2*ES
 algebraic rule: 0 = Etotal - E - ES
        species: S =  8   mole
                 E =  4   mole
                ES =  0   mole
```

```
                P =  0   mole
           Etotal =  4   mole
     parameters: k1 = 2   1/(mole*second)
                 k1r = 1   1/second
                  k2 = 1.5 1/second
```

Enzyme Reactions with Mass Action Kinetics

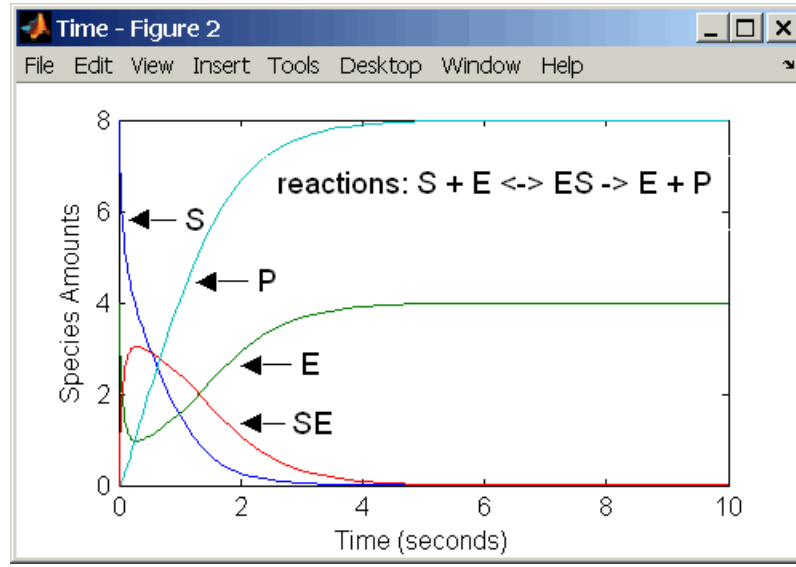
Determining the differential rate equations for the reactions in a model is a time-consuming process. A better way is to enter the reactions for a single substrate enzyme reaction mechanism directly into the software. The following example using models an enzyme catalyzed reaction with mass action kinetics. For a description of the reaction model, see "Simple Model for Single Substrate Catalyzed Reactions" on page A-8.

```
     reaction: S + E -> ES
reaction rate: k1*S*E (binding)

     reaction: ES -> S + E
reaction rate: k1r*ES (unbinding)

     reaction: ES -> E + P
reaction rate: k2*ES (transformation)
      species: S =  8   mole
               E =  4   mole
              ES =  0   mole
               P =  0   mole
   parameters: k1  = 2   1/(mole*second)
               k1r = 1   1/second
               k2  = 1.5 1/second
```

The results for a simulation using reactions are identical to the results from using differential rate equations.



Enzyme Reactions with Irreversible Henri-Michaelis-Menten Kinetics

Representing an enzyme-catalyzed reaction with mass action kinetics requires you to know the rate constants `k1`, `k1r`, and `k2`. However, these rate constants are rarely reported in the literature. It is more common to give the rate constants for Henri-Michaelis-Menten kinetics with the maximum velocity `Vm=k2*E` and the constant `Km = (k1r + k2)/k1`. The reaction rate for a single substrate enzyme reaction using Henri-Michaelis-Menten kinetics is given below. For information about the model, see “Simple Model for Single Substrate Catalyzed Reactions” on page A-8.

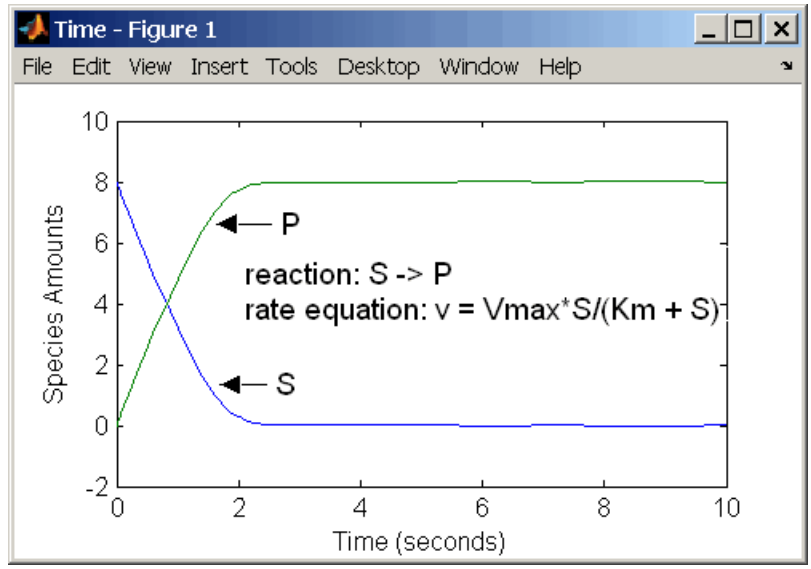
$$v = \frac{Vmax[S]}{Km + [S]}$$

The following example models an enzyme catalyzed reaction using Henri-Michaelis-Menten kinetics with a single reaction and reaction rate equation. Enter the reaction defined below into the software and simulate.

```
      reaction: S -> P
reaction rate: Vmax*S/(Km + S)
```

```
    species:    S =  8    mole
                P =  0    mole
parameters: Vmax =  6    mole/second
              Km =  1.25 mole
```

The results show a plot slightly different from the plot using mass action kinetics. The differences are due to assumptions made when deriving the Michaelis-Menten rate equation.



B

Creating Rate Rules

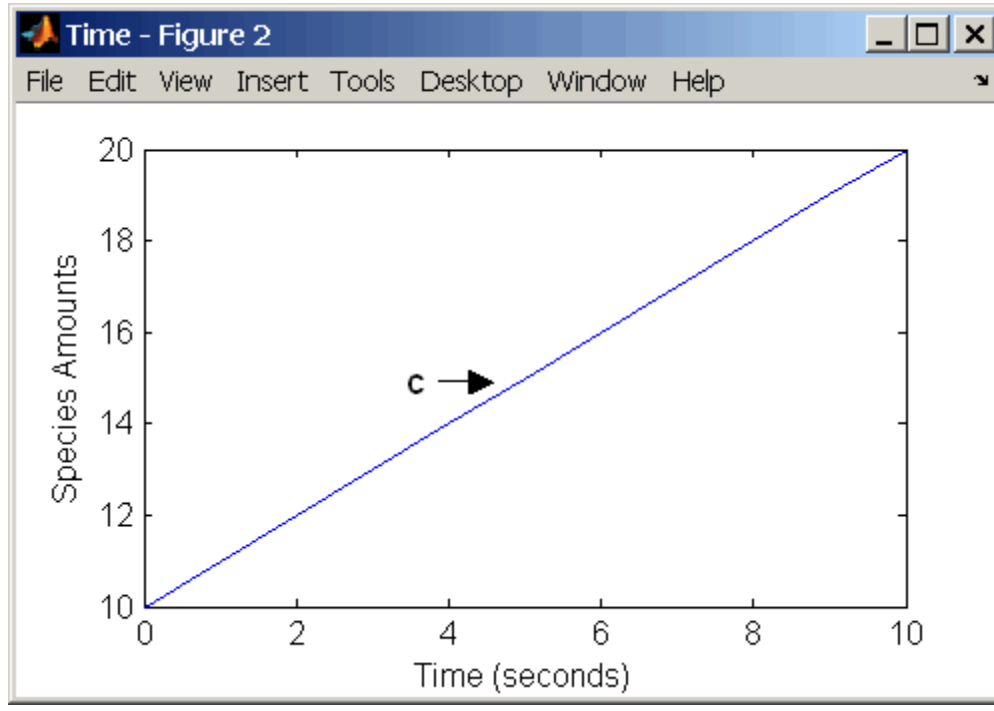
Using Rate Rules When the Rate of Change Is Constant

You can increase or decrease the amount or concentration of a species by a constant value using a zero-order rate rule. For example, suppose species `c` increases by a constant rate `k`.

```
      reaction: none
 rate equation: none
     rate rule: dc/dt = k
      species : c = 10 mole(initial amount)
    parameters: k = 1 mole/second
```

The analytical solution is $c = kt + c_o$, where c_o is the initial amount or concentration of the species `c`.

Enter the rule described above as `c = k`. Set the `RuleType` property to `rate`, enter the values for `c` and `k`, and then simulate.



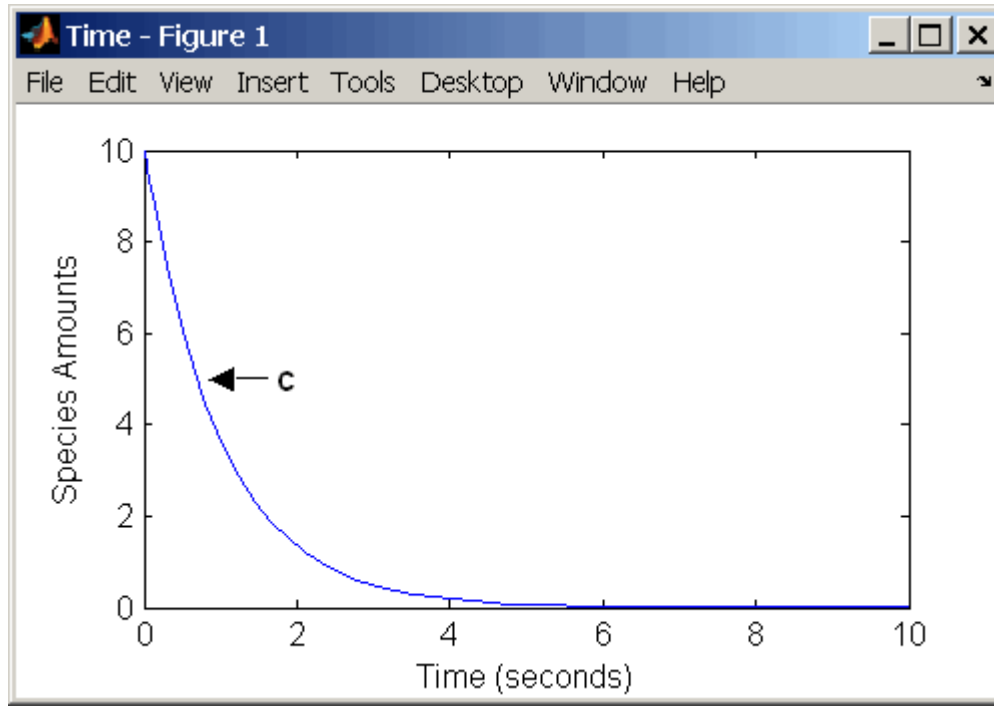
Alternatively, you could model a constant increase in a species using Mass Action reaction `null -> C`.

Using Rate Rules When the Rate of Change Is Exponential

You can change the amount of a species similar to a first-order reaction using a first-order rate rule. For example, suppose the species `c` decays exponentially.

The solution for the rate rule `dc/dt = -k*c` is $c = c_0 e^{-kt}$.

Enter the rate rule described above and the simulate.



Notice that if the amount of a species `c` is determined by a rate rule and `c` is also in a reaction, `c` must have its `BoundaryCondition` property set to `true`. For example, with a reaction `a -> c` and a rate rule `dc/dt = k*c`, set the `BoundaryCondtion` property for `c` so that a differential rate term is not created from the reaction. The amount of `c` is determined solely by a differential rate term from the rate rule.

If the `BoundaryCondtion` property is set to `false`, you will get the following error message:

```
Invalid rule variable 'in a reaction or another rule'.
```

Using Rate Rules When the Rate of Change Is Determined by Another Species

A species from one reaction can determine the rate of another reaction if it is in the second reaction rate equation. Similarly, a species from a reaction can determine the rate of another species if it is in the rate rule that defines that other species.

```
      reaction: a -> b
 rate equation: v = -k1*a
     rate rule: dc/dt = k2*a
       species: a = 10 mole
                b =  0 mole
                c =  5 mole
    parameters: k1 = 1 1/second
                k2 = 1 1/second
```

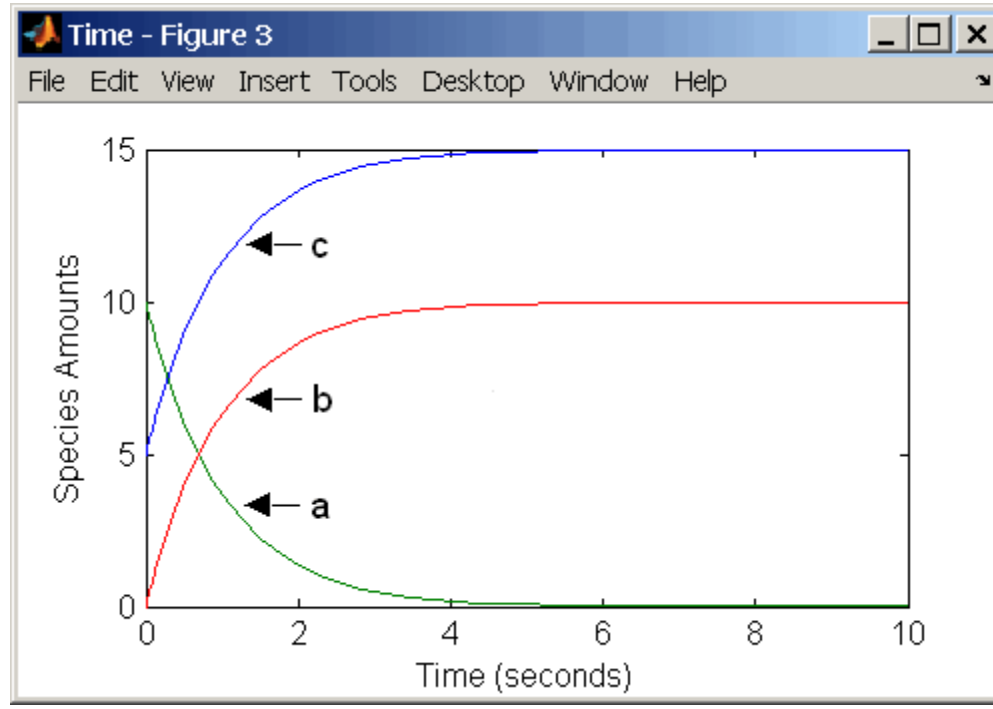
The solution for the species in the reaction are:

$$a=a_o e^{-k1t} \quad \text{and} \quad b=a_o(1-e^{-k1t})$$

With the rate rule `dc/dt = k2*a` dependent on the reaction, `dc/dt =` $k_2(a_o e^{-k1t})$, and the solution is:

$$c = c_o + k_2 a_o / k_1 (1 - e^{-k_1 t})$$

Enter the reaction and rule described above and simulate.


Time - Figure 3
File Edit View Insert Tools Desktop Window Help
Species Amounts
Time (seconds)
c
b
a
0
5
10
15
0
2
4
6
8
10

Using Rate Rules To Express Differential Rate Equations as Rules

Many mathematical models in the literature are described with differential rate equations for the species. You could manually convert the equations to reactions, or you could enter the equations as rate rules. For example, you could enter the following differential rate equation for a species `C`:

$$\frac{dC}{dt} = v_i - v_d X \frac{C}{K_c + C} - k_d C$$

as a rate rule in SimBiology:

```
C = vi - (vd*X*C)/(Kc + C) - kd*C
```

C

Models Used in Examples

Minimal Cascade Model for a Mitotic Oscillator

Albert Goldbeter modified a model with enzyme cascades [Goldbeter and Koshland 1981] to fit cell cycle data from studies with embryonic cells [Goldbeter 1991]. He used this model to demonstrate thresholds with enzyme cascades and periodic behavior caused by negative feedback.

There are two SimBiology model variations using Goldbeter's model. The first model uses the differential rate equations directly from Goldbeter's paper. The second model is built with reactions using Henri-Michaelis-Menten kinetics.

In this section...

Goldbeter Model

About the Goldbeter Model

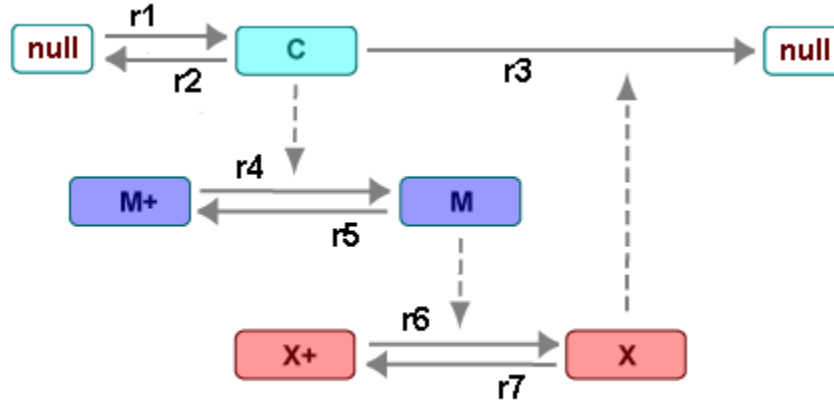
Albert Goldbeter created a simple cell division model from studies with embryonic cells [Goldbeter 1991]. This model demonstrates thresholds with enzyme cascades and periodic behavior caused by negative feedback.

There are six species in Goldbeter's minimal mitotic oscillator model [Goldbeter 1991].

- C — Cyclin. The periodic behavior of cyclin activates and deactivates an enzyme cascade.

- M+, M — Inactive (phosphorylated) and active forms of cdc2 kinase. Kinases catalyze the addition of phosphate groups onto amino acid residues.
- X+, X — Inactive and active (phosphorylated) forms of a cyclin protease. Proteases degrade proteins by breaking peptide bonds.

The reactions are labeled `r1` to `r7` on the following diagram.



This model shows:

- How thresholds with cdc2 kinase activation (M+ -> M) and protease activation (X+ -> X) can occur as the result of covalent modification (for example, phosphorylation or dephosphorylation), but without the need for positive feedback.
- How periodic behavior with cdc2 kinase activation can occur with negative feedback and the time delay associated with activation/deactivation enzyme cascades.

Reaction Descriptions and Model Assumptions

The following list describes each of the reactions in Goldbeter's minimal mitotic oscillator with some of the simplifying assumptions. For a more detailed explanation of the model, see [Goldbeter 1991].

- Cyclin (C) is synthesized at a constant rate (r1) and degraded at a constant rate (r2).
- Cyclin (C) does not complex with cdc2 kinase (M).
- Cyclin (C) activates cdc2 kinase (M+ -> M) by increasing the velocity of the phosphatase that activates the kinase. Inactive cdc2 kinase (M+) is activated by removing inhibiting phosphate groups (r4).
- The amount of deactivating kinase (not modeled) for the cdc2 kinase (M) is constant. Active cdc2 kinase (M) is deactivated by adding inhibiting phosphate group (r5).
- The activation of cyclin protease (X+ -> X) by the active cdc2 kinase (M) is direct without other intervening cascades. Cyclin protease (X) is activated by adding phosphate groups (r6).
- The amount of deactivating phosphatase (not modeled) for the cyclin protease (X) is constant. Active cyclin protease (X) is deactivated by removing the activating phosphate groups (r7).
- The three species of interest are cyclin (C), active dephosphorylated cdc2 kinase (M), and active phosphorylated protease (X). The total amounts of (M + M+) and (X + X+) are constant.

Mathematical Model

Goldbeter's minimal mitotic oscillator model is defined with three differential rate equations and two algebraic equations that define changing parameters in the rate equations.

Differential Rate Equation 1, Cyclin (C). The following differential rate equation is from [Goldbeter 1991] for cyclin (C).

$$\frac{dC}{dt} = v_i - v_d X \frac{C}{K_d + C} - k_d C$$

Differential Rate Equation 2, Kinase (M). The following differential rate equation is for cdc2 kinase (M). Notice that (1 - M) is the amount of inactive (phosphorylated) cdc2 kinase (M+).

$$\frac{dM}{dt} = V_1 \frac{(1-M)}{K_1 + (1-M)} - V_2 \frac{M}{K_2 + M}$$

$$V_1 = \frac{VM_1[C]}{K_c + [C]}$$

Differential Rate Equation 3, Protease (X). Differential rate equations for cyclin protease (`X`). Notice that `(1 - X)` is the amount of inactive (unphosphorylated) cyclin protease (`X+`).

$$\frac{dX}{dt} = V_3 \frac{(1-X)}{K_3 + (1-X)} - V_4 \frac{X}{K_4 + X}$$

$$V_3 = VM_3[M]$$

SimBiology Model with Rate Rules

SimBiology Model with Rules

In the literature, many biological models are defined using differential rate and algebraic equations. With SimBiology software, you can enter the equations directly as SBML rules. The example in this section uses Goldbeter's mitotic oscillator to illustrate this point.

Writing differential rate equations in an unambiguous format that a software program can understand is a fairly simple process.

- Use an asterisk to indicate multiplication. For example, `k[a]` is written `k*a`.
- Remove square brackets that indicate concentration from around species. The units associated with the species will indicate concentration (`moles/liter`) or amount (`moles`, `molecules`).

SimBiology software uses square brackets around species and parameter name to allow names that are not valid MATLAB variable names. For example, you could have a species named `glucose-6-phosphate dehydrogenase` but you need to add brackets around the name in reaction rate and rule equations.

- Use parentheses to clarify the order of evaluation for mathematical operations. For example, do not write a Henri-Michaelis-Menten rate as `Vm*C/Kd + C`, because `Vm*C` is divided by `Kd` before adding `C`, and then `C` is added to the result.

The following equation is the rate rule for "Differential Rate Equation 1, Cyclin (C)" on page C-4:

```
dC/dt = vi - (vd*X*C)/(Kd + C) - kd*C
```

The following equations are the rate and `repeatedAssignment` rules for "Differential Rate Equation 2, Kinase (M)" on page C-4:

```
dM/dt = (V1*Mplus)/(K1 + Mplus) - (V2*M)/(K2 + M)
V1 = (VM1*C)/(Kc + C)
Mplus = Mt - M
```

The following equations are the rate and `repeatedAssignment` rules for "Differential Rate Equation 3, Protease (X)" on page C-5:

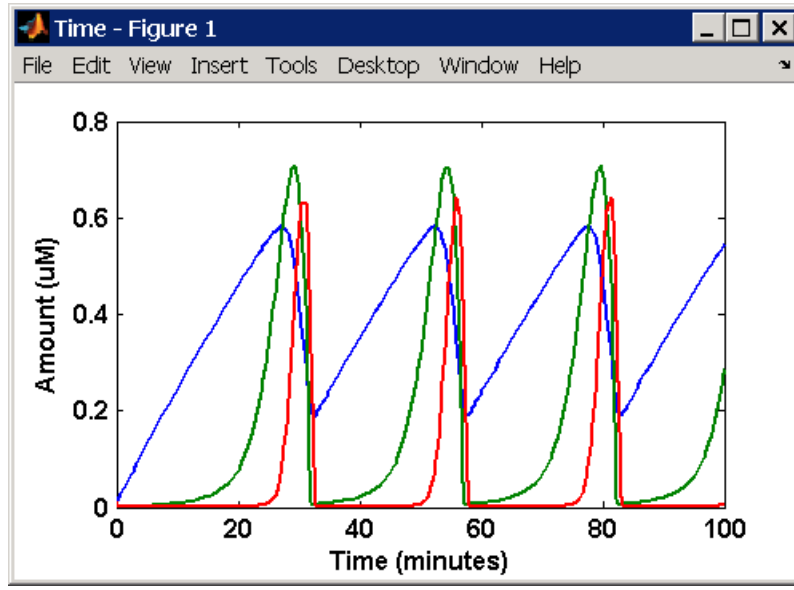
```
dX/dt = (V3*Xplus)/(K3 + Xplus) - (V4*X)/(K4 + X)
V3 = VM3*M
Xplus = Xt - X
```

Rules. The active (`M`) and inactive (`Mplus`) forms of the kinase are assumed to be part of a conserved cycle with the total concentration (Mt) remaining constant during the simulation. You need only one differential rate equation with a mass balance equation to define the amounts of both species. Similarly, the active (`X`) and inactive (`Xplus`) forms of the protease are part of a second conserved cycle.

SimBiology Simulation with Rules

This is a simulation of Goldbeter's minimal mitotic oscillator using differential rate and algebraic equations. Simulate with the `sundials` solver and plot

species `C`, `M`, and `X`. For a description of the model, see "SimBiology Model with Rules" on page C-5.



SimBiology Model with Reactions

Converting Differential Rate Equations to Reactions

In the literature, many models are defined with differential rate equations. With SimBiology software, creating the differential equations from reactions is unnecessary; you can enter the reactions and let the software calculate the equations.

Some models are defined with differential rate equations, and you might need the reactions to be compatible with your model. Two rules you can use to convert differential rate equations to reactions are:

- **For a positive term** — The species described by the equation is placed on the right as a product, and the species in the term are placed on the left as reactants.
- **For a negative term** — The species described by the equation is placed on the left as a product, and the species in the term are also placed on the left as reactants.

  You need to determine the products using additional information, for example, a reaction diagram, a description of the model, or an understanding of a reaction. If a reaction is catalyzed by a kinase, then you can conclude that the product has one or more additional phosphate groups.

A simple first-order reaction has differential rate equation `dR/dt = +kr[P] - kf[R]`. The negative term implies that the reaction is `R -> ?` with an unknown product. The positive term identifies the product and completes the reaction, `R <-> P`.

Reactions R1 to R3 from Equation E1. The differential rate equation 1 is repeated here for comparison with the reactions. See "Differential Rate Equation 1, Cyclin (C)" on page C-4.

$$\frac{dC}{dt} = v_i - v_d X \frac{C}{K_d + C} - k_d C$$

The reaction and reaction rate equations from the differential rate equation E1 are given below:

```
r1      reaction: null -> C
   reaction rate: vi

r2      reaction: C -> null
   reaction rate: kd*C

r3      reaction: C -> null
   reaction rate: (vd*X*C)/(Kd + C)
```

Reactions R4 and R5 from Equation E2. The differential rate equation 2 and algebraic equation 2 are repeated here for comparison with the reactions. See "Differential Rate Equation 2, Kinase (M)" on page C-4.

$$\frac{dM}{dt} = V_1 \frac{(1-M)}{K_1 + (1-M)} - V_2 \frac{M}{K_2 + M}$$

$$V_1 = \frac{VM_1[C]}{K_c + [C]}$$

The reaction and reaction rate equations from the differential rate equation E2 are given below:

```
r4              reaction: Mplus -> M
           reaction rate: V1*Mplus/(K1 + Mplus)
 repeatedAssignment rule: V1 = VM1*C/(Kc + C)

r5      reaction: M -> Mplus
   reaction rate: V2*M/(K2 + M)
```

Reactions R6 and R7 from Equation E3. The differential rate equation for equation 3 and algebraic equation 3 is repeated here for comparison with the reactions.

$$\frac{dX}{dt} = V_3 \frac{(1-X)}{K_3 + (1-X)} - V_4 \frac{X}{K_4 + X}$$

```
V3 = VM3*[M]
```

The reaction and reaction rate equations from the differential rate equation E3 are given below:

```
r6               reaction: Xplus -> X
            reaction rate: V3*Xplus]/(K3 + Xplus)
  repeatedAssignment rule: V3 = VM3*M

r7      reaction: X -> Xplus
   reaction rate: V4*X/(K4 + X)
```

Calculating Initial Values for Reactions

After you converted the differential rate equations to the reactions and reaction rate equations, you can start to fill in initial values for the species (reactants and products) and parameters.

The initial values for parameters and amounts for species are listed with four different units in the same dimension:

- A — Original units in the Goldbeter 1991 paper.
- B — Units of concentration with time converted to second. When converting a to b, use `1 minute = 60 second` for parameters.

$$\frac{\text{X uM}}{\text{minute}} \text{ x } \frac{\text{1e-6 mole/liter}}{\text{1 uM}} \text{ x } \frac{\text{1 minute}}{\text{60 second}} = \frac{\text{Y mole}}{\text{liter*second}}$$

- C — Units of amount as moles. When converting concentration to moles, use a cell volume of `1e-12` liter and assume that volume does not change.

$$\frac{\text{Y mole}}{\text{liter*second}} \text{ x } \frac{\text{1e-12 liter}}{} = \frac{\text{Z mole}}{\text{second}}$$

- D — Units of amount as molecules. When converting amount as moles to molecules, use `6.022e23 molecules = 1 mole`.

$$\frac{\text{Z mole}}{\text{second}} \text{ x } \frac{\text{6.022e23 molecule}}{\text{1 mole}} = \frac{\text{N molecules}}{\text{second}}$$

With dimensional analysis on and unit conversion off, select all of the units for one letter. For example, select all of the As. If dimensional analysis and unit conversion are on, you can mix and match letters and get the same answer.

Reaction 1 Cyclin Synthesis.

R1		Value	Units
reaction	null -> C	----	----
reaction rate	vi	----	A. uM/minute
		----	B. mole/(liter*second)

R1		Value	Units
		----	C. mole/second
		----	D. molecule/second
parameters	vi	0.025	A. uM/minute
		4.167e-10	B. mole/(liter*second)
		4.167e-22	C. mole/second
		205	D. molecule/second
species	C	0.01	A. uM
		1e-8	B. mole/liter
		1.0e-20	C. mole
		6.022e+3	D. molecule

Reaction 2 Cyclin Undifferentiated Degradation.

R2		Value	Units
reaction	`C -> null`	----	----
reaction rate	`kd*C`	----	A. uM/minute
		----	B. mole/(liter*second)
		----	C. mole/second
		----	D. molecule/second
parameters	`kd`	0.010	A. 1/minute
		1.6667e-4	B, C, D. 1/second
species	`C`	0.01	A. uM
		1e-8	B. mole/liter
		1.0e-20	C. mole
		6.022e+3	D. molecule

Reaction 3 Cyclin Protease Degradation.

R3		Value	Units
reaction	`C -> null`	----	----
reaction rate	`(vd*X*C)/(Kd + C)`	----	A. uM/minute
		----	B. mole/(liter*second)
		----	C. mole/second
		----	D. molecule/second
parameter	`vd`	0.25	A. 1/minute
		0.0042	B, C, D. 1/second
parameter	`Kd`	0.02	A. uM
		2.0e-8	B. mole/liter
		2.0e-020	C. mole
		12044	D. molecule
species	`C` (substrate)	0.01	A. uM
		1e-8	B. mole/liter
		1.0e-20	C. mole
		6.022e+3	D. molecule
species	`X` (enzyme)	0.01	A. uM
		1e-8	B. mole/liter
		1.0e-20	C. mole
		6.022e+3	D. molecule

Reaction 4 Cdc2 Kinase Activation.

R4		Value	Units
reaction	`Mplus -> M`	----	----
reaction rate	`(V1*Mplus)/(K1 + Mplus)`	----	A. uM/minute

R4		Value	Units
		----	B. mole/(liter*second)
		----	C. mole/second
		----	D. molecule/second
repeatedAssignment rule	V1 = (VM1*C)/(Kc + C)	----	
parameter	V1 (variable by rule)	0.00	A. uM/minute
			B. mole/(liter*second)
			C. mole/second
			D. molecule/second
parameter	VM1	3.0	A. uM/minute
		5.0e-8	B. mole/(liter*second)
		5.0000e-020	C. mole/second
		30110	D. molecule/second
parameter	Kc	0.5	A. uM
		5.0000e-7	B. mole/liter
		5.0e-19	C. mole
		3.011e+5	D. molecule
parameter	K1	0.005	A. uM
		5e-9	B. mole/liter
		5e-21	C. mole
		3.011e+3	D. molecule
species	Mplus (inactive substrate)	0.99	A. uM
		9.9e-7	B. mole/liter
		9.9e-19	C. mole
		5.962e+5	D. molecule
species	M (active product)	0.01	A. uM

R4		Value	Units
		1e-8	B. mole/liter
		1.0e-20	C. mole
		6.022e+3	D. molecule
species	C	0.01	A. uM
		1e-8	B. mole/liter
		1.0e-20	C. mole
		6.022e+3	D. molecule

Reaction 5 Cdc2 Kinase Deactivation.

R5		Value	Units
reaction	`M -> M_plus`	----	----
reaction rate	`(V2*M)/(K2 + M)`	----	A. uM/minute
		----	B. (mole/liter-second)
		----	C. mole/second
		----	D. molecule/second
parameter	`V2`	1.5	A. uM/minute
		2.5000e-008	B. mole/liter-second
		2.5000e-020	C. mole/second
		15055	D. molecule/second
parameter	`K2`	0.005	A. uM
		5.0000e-009	B. mole/liter
		5.0000e-021	C. mole
		3011	D. molecule
		1.0e-20	C. mole
species	`Mplus` (inactive)	0.99	A. uM
		9.9e-7	B. mole/liter

R5		Value	Units
		9.9e-19	C. mole
		5.962e+5	D. molecule
species	M (active)	0.01	A. uM
		1e-8	B. mole/liter
		1.0e-20	C. mole
		6.022e+3	D. molecule

Reaction 6 Protease Activation.

R6		Value	Units
reaction	Xplus -> X	----	----
reaction rate	(V3*Xplus)/(K3 + Xplus)	----	A. uM/minute
		----	B. mole/(liter*second)
		----	C. mole/second
		----	D. molecule/second
repeatedAssignment rule	V3 = VM3*M	----	
parameter	V3 (variable by rule)		A. uM/minute
			B. mole/liter-second
			C. mole/second
			D. molecule/second
parameter	VM3	1.0	A. 1/minute
		0.0167	B, C, D. 1/second
parameter	K3	0.005	A. uM
		5e-9	B. mole/liter
		5e-21	C. mole
		3.011e+3	D. molecule

R6		Value	Units
species	Xplus (inactive substrate)	0.99	A. uM
		9.9e-7	B. mole/liter
		9.9e-19	C. mole
		5.962e+5	D. molecule
species	X (active product)	0.01	A. uM
		1e-8	B. mole/liter
		1.0e-20	C. mole
		6.022e+3	D. molecule
species	M (enzyme)	0.01	A. uM
		1e-8	B. mole/liter
		1.0e-20	C. mole
		6.022e+3	D. molecule

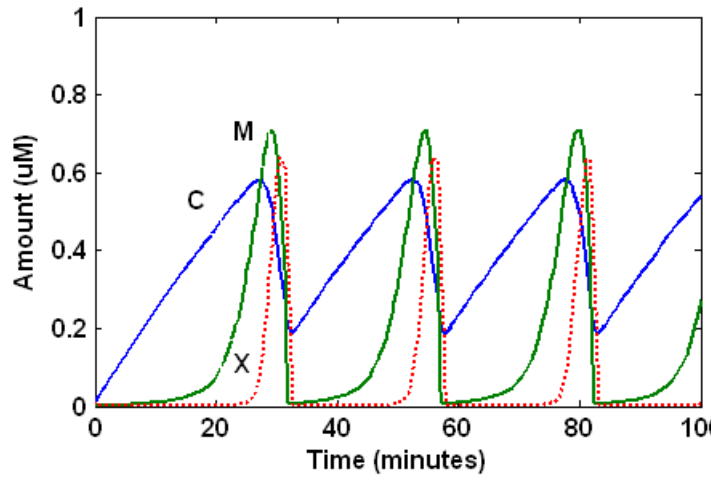
Reaction 7 Protease Deactivation.

R7		Value	Units
reaction	X -> X_plus	----	----
reaction rate	(V4*X)/(K4 + X)	----	A. uM/minute
		----	B. mole/(liter*second)
		----	C. mole/second
		----	D. molecule/second
parameter	V4	0.5	A. uM/minute
		8.3333e-009	B. mole/(liter*second)
		8.3333e-021	C. mole/second
		5.0183e+003	D. molecule/second
parameter	K4	0.005	A. uM
		5e-9	B. mole/liter

R7		Value	Units
		5e-21	C. mole
		3011	D. molecule
species	`Xplus` (inactive)	0.99	A. uM
		9.9e-7	B. mole/liter
		9.9e-19	C. mole
		5.962e+5	D. molecule
species	`X` (active)	0.01	A. uM
		1e-8	B. mole/liter
		1.0e-20	C. mole
		6.022e+3	D. molecule

SimBiology Simulation with Reactions

This is a simulation of Goldbeter's minimal mitotic oscillator with rate and algebraic equations. Simulate with the `sundials` solver and plot species `C`, `M`, and `X`. For a description of the model, see "SimBiology Model with Reactions" on page C-7.



References

[1] Goldbeter A. (1991), "A minimal cascade model for the mitotic oscillator involving cyclin and cdc2 kinase," Proceedings of the National Academy of Sciences USA, 88:9107-9111.

[2] Goldbeter A., Koshland D. (1981), "An amplified sensitivity arising from covalent modification in biological systems," Proceedings of the National Academy of Sciences USA, 78:6840-6844.

[3] Goldbeter A., Koshland D. (1984), "Ultrasensitivity in biochemical systems controlled by covalent modification," The Journal of Biological Chemistry, 259:14441-14447.

[4] Goldbeter A., home page on the Web, `http://www.ulb.ac.be/sciences/utc/GOLDBETER/agoldbet.html`

[5] Murray A.W., Kirschner M.W. (1989), "Cyclin synthesis drives the early embryonic cell cycle," Nature, 339:275-280.

Model of the Yeast Heterotrimeric G Protein Cycle

In this section...

Background on G Protein Cycles

G Proteins

Cells rely on signal transduction systems to communicate with each other and to regulate cellular processes. G proteins are GTP-binding proteins that are involved in the regulation of many cellular processes. There are two known classes of G proteins: the monomeric G proteins (one GTPase), and the heterotrimeric G proteins (three different monomers). The G proteins usually facilitate a step requiring energy. This energy is supplied by the hydrolysis of GTP by a GTPase activating protein (GAP). The exchange of GDP for GTP is catalyzed by a guanine nucleotide releasing protein (GNRP) [Alberts et al. 1994].

$$Gprotein + GTP \underset{GNRP}{\overset{GAP}{\rightleftharpoons}} Gprotein + GDP$$

G protein-coupled receptors (GPCRs) are the targets of many pharmaceutical agents. Some estimates suggest that 40 to 50% of currently marketed drugs target GPCRs and that 40% of current drug discovery focus is on GPCR targets. Some examples include those for reducing stomach acid (ranitidine which targets histamine H2 receptor), migraine (sumatriptan, which targets a serotonin receptor subtype), schizophrenia (olanzapine, which targets serotonin and dopamine receptors), allergies (desloratadine, which targets histamine receptors). One approach in pharmaceutical research is to model

signaling pathways to analyze and predict both downstream effects and effects in related pathways. This tutorial examines model building and analysis of the G protein cycle in the yeast pheromone response pathway using the SimBiology desktop.

G Proteins and Pheromone Response

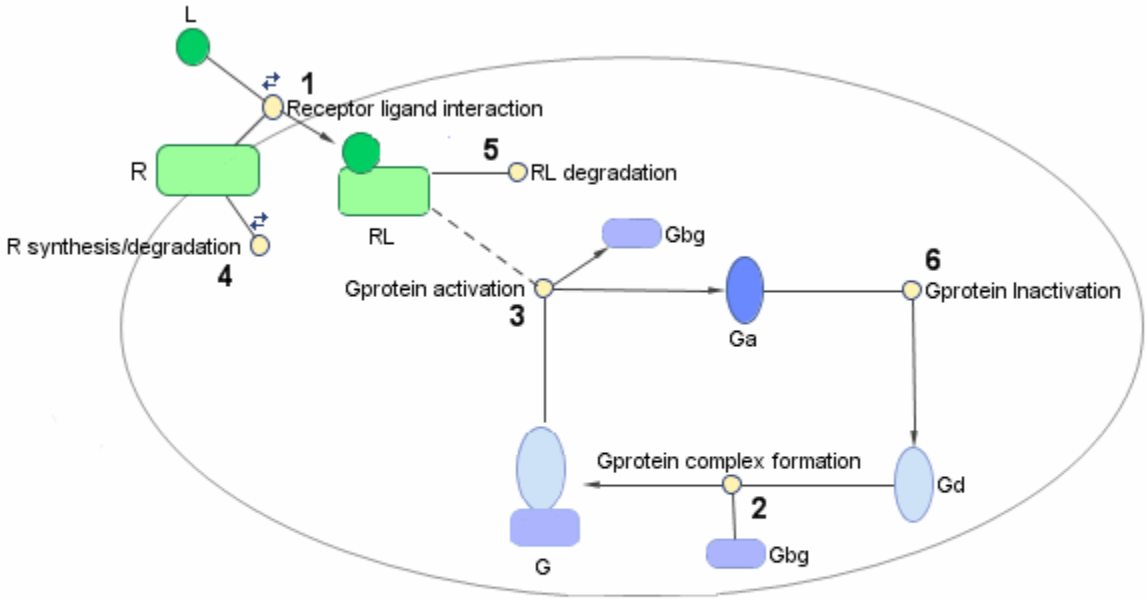
In the yeast *Saccharomyces cerevisiae*, G protein signaling in pheromone response is a well characterized signal transduction pathway. The pheromone secreted by *alpha* cells activates the G protein-coupled α-factor receptor (Ste2p) in *a* cells which results in a variety of cell responses including cell-cycle arrest and synthesis of new proteins. The authors of the study performed a quantitative analysis of this cycle, compared the regulation of G protein activation in wild-type yeast haploid *a* cells with cells containing mutations that confer supersensitivity to α-factor. They analyzed the data in the context of cell-cycle arrest and pheromone-induced transcriptional activation and developed a mathematical model of the G protein cycle that they used to estimate rates of activation and deactivation of active G protein in the cell.

Modeling a G Protein Cycle

Reactions Overview

Systems biologists represent biological pathways and processes as reactions with reaction rates, and treat the components of these pathways as individual species.

The G protein cycle in the yeast pheromone-response pathway can be condensed into a set of biochemical reactions. These reactions are complex formation, transformation, or disassociation reactions that Yi and colleagues [Yi et al. 2003] use to simplify and describe the system. In this example, α-factor, α-factor receptor, and the G protein subunits are all treated as species participating in reactions. The system can be graphically represented as follows.



Graphical Representation of the G protein cycle in yeast pheromone response. The numbers represent reaction numbers referenced in the text. L = Ligand (alpha factor), R = alpha-factor receptor, Gbg = free levels of G-beta:G-gamma complex, Ga = active G-alpha-GTP, Gd = inactive G-alpha-GDP, G = inactive Gbg:Gd complex.

The following table shows you the reactions used to model the G protein cycle and the corresponding rate constants (rate parameters) for each reaction. For reversible reactions, the forward rate parameter is listed first.

No.	Name	Reaction	Rate Parameters
1	Receptor-ligand interaction	`L + R <-> RL`	`kRL, kRLm`
2	Heterotrimeric G protein formation	`Gd + Gbg -> G`	`kG1`
3	G protein activation	`RL + G -> Ga + Gbg + RL`	`kGa`

No.	Name	Reaction	Rate Parameters
4	Receptor synthesis and degradation	`R <-> null`	`kRdo`, `kRs`
5	Receptor-ligand degradation	`RL -> null`	`kRD1`
6	G protein inactivation	`Ga -> Gd`	`kGd`

Note that in reaction 3 (G protein activation), `RL` appears on both sides of the reaction. This is because `RL` is treated as a modifier or catalyst, and the model assumes that there is no synthesis or consumption of `RL` in this reaction.

The authors use a set of ordinary differential equations (ODEs) to describe the system. In the software, you can represent the biological pathway as a system of biochemical reactions and the software creates the ODEs for you. Alternatively, if you have a set of ODEs that describe your system you can enter these as rate rules. For an example of modeling using rate rules, see "SimBiology Model with Rate Rules" on page C-5.

Assumptions, Experimental Data, and Units in the G Protein Model

The authors have obtained experimental data either through their own measurements or through published literature. As with any other model, the G protein cycle model simplifies the biological process while also trying to reconcile the experimental data. Consider these points:

- Reaction 2 — Binding and formation of the heterotrimeric G protein complex is treated as a single-step reaction.
- Reaction 3 — Activation of G protein is modeled as a single-step. Guanine nucleotide exchange factors (GEFs) are not modeled.
- Reactions 3 and 6 — The parameters for the rate of G protein activation and deactivation (`kGa` and `kGd`) have been estimated based on the dose response curves in the reference paper. The SimBiology model being built in this tutorial directly uses those values.
- Reactions 4 and 5 — Receptor synthesis and degradation are handled purely as two simple reaction steps.

- Reaction 6 — Deactivation of G protein by the regulator of G protein signaling (RGS) protein Sst2p is modeled as a single step. Sst2p is not modeled.

  The reaction is modeled with an estimated reaction rate of `0.11` s^{-1}) in the Sst2p containing wild-type strain. The uncatalyzed reaction rate is estimated to be `0.004` s^{-1} in a strain with a deletion of SST2 (*sst2Δ*, mutant strain).

- Free GDP, GTP, and Pi are not included in the model.

This tutorial shows you how to plot the experimental data over the simulation plot of the active G protein fraction. You can estimate the values of the experimental data of interest for this example from the coordinates of the plots found in Figure 5 of the reference paper [Yi et al. 2003]. The following values were obtained by comparing the coordinates of the standards with those of the unknowns in the figure.

Time	Fraction of Active Gα (Experimental)
0	0.00
10	0.35
30	0.40
60	0.36
110	0.39
210	0.33
300	0.24
450	0.17
600	0.20

Note The SimBiology **Dimensional Analysis** feature is not used in this tutorial. For this tutorial, the values of all species are converted to have the unit `molecule`, and all rate parameters are converted to have either the unit `1/second` or the units `1/(molecule*second)`, depending on whether the reaction is first or second order. You should leave the **InitialAmountUnits** box for species and the **ValueUnits** box for rate parameters empty for the models in this tutorial.

References

[1] Tau-Mu Yi, Hiroaki Kitano, and Melvin I. Simon. A quantitative characterization of the yeast heterotrimeric G protein cycle. PNAS (2003) vol. 100, 10764-10769.

[2] Alberts, B., Bray, D., Lewis, J., Raff, M., Roberts, K., and Watson, J.D. Molecular Biology of the Cell, 3rd edition, Garland Publishing, 1994.

Model of M-Phase Control in *Xenopus* Oocyte Extracts

John Tyson's Computational Cell Biology Lab created a mathematical model for M-phase control in *Xenopus* oocyte (frog egg) extracts [Marlovits et al. 1998]. The M-phase control model shows principles by which you can apply phosphorylation and regulatory loops in your own models. Publications typically list systems of ordinary differential equations (ODEs) that represent a model system. This example shows you how to interpret these ODEs in the form of reaction pathways that are easier to represent and visualize in SimBiology software.

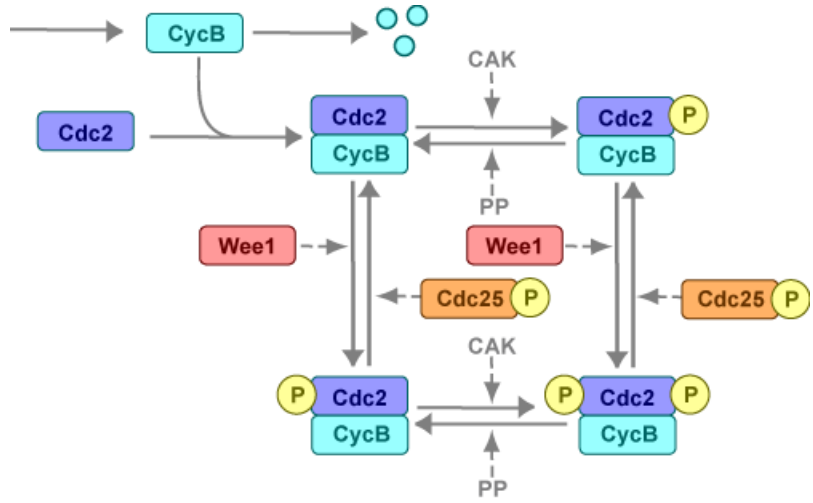
The model is centered around M-phase promoting factor (MPF). There are two positive feedback loops where MPF increases its synthesis and a negative feedback loop where MPF decreases its amount by increasing its degradation.

In this section...

M-Phase Control Model

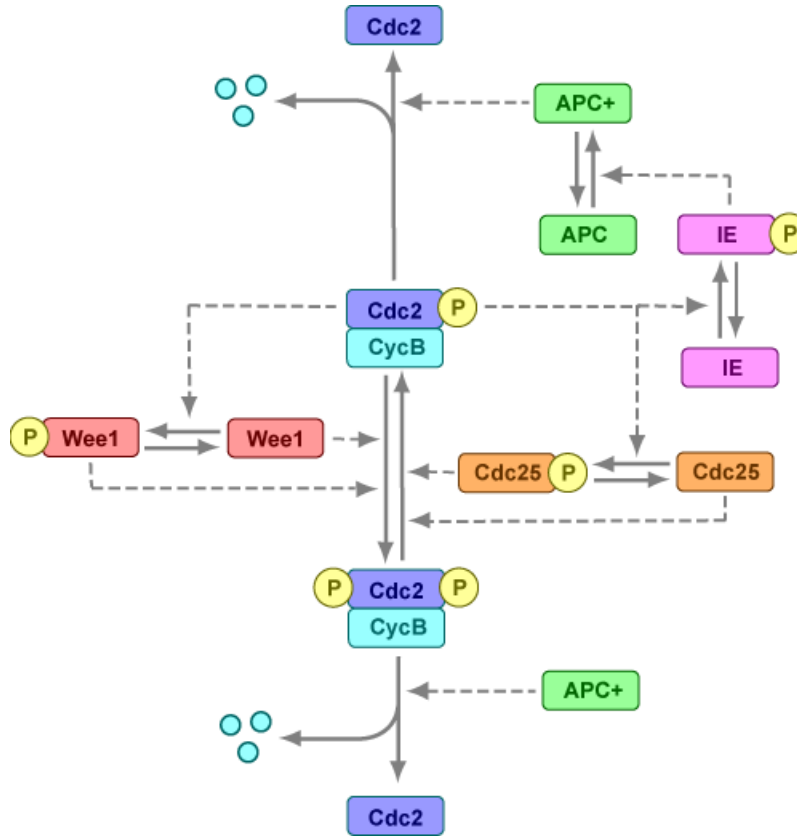
Synthesis Reactions

Cyclin B (CycB) dimerizes with Cdc2 kinase (Cdc2) to form M-phase promoting factor (MPF).



Regulation Reactions with Active MPF

Positive feedback loops with M-phase promoting factor (MPF) activate the Cdc25 phosphatase and deactivate the Wee1 kinase. A negative feedback loop with MPF activates anaphase-promoting complex (APC) that regulates the degradation of the Cyclin B subunit.



M-Phase Control Equations

About the Rate Equations in This Example

Models in systems biology are commonly described in the literature with differential rate equations. However, SimBiology software defines a model using reactions. This section shows you how to convert models published in the literature to a SimBiology format. The equation numbers match the published paper for this model [Marlovits et al. 1998]. Equations that are missing in the sequence involve the Cdk inhibitor (CKI) protein, which is not currently modeled in the SimBiology version.

Converting Differential Equations to Reactions

The rules for writing reaction and reaction rate equations from differential rate equations include not only the equations but also an understanding of the reactions. `dx/dt` refers to the species the differential rate equation is defining. `kinetics` refers to the species in the reaction rate.

- Positive terms: Rate species are placed on right side of the reactions; reaction rate equation species are placed on the left.

$$kinetics \rightarrow \frac{dx}{dt}$$

- Negative terms: Rate species are placed on the left side of the reaction because the species are being used up in some way; reaction rate equation species are also placed on left. You need to deduce the products from additional information about the model.

$$\text{kinetics or } (\frac{dx}{dt}) \rightarrow \text{products?}$$

The following table will help you deduce the products for a reaction. In this example, by convention, phosphate groups on the right side of a species name are activating while phosphate groups on left are inhibiting.

Enzyme	Description	Reaction
`wee1`	Kinase, add inhibiting phosphate group	MPF —> P-MPF
`cdc25`	Phosphatase, remove inhibiting phosphate group	P-MPF —> MPF + P
`kcak`	Kinase, add activating phosphate group	MPF —> MPFp
`kpp`	Phosphatase, remove activating phosphate group	MPF-P —> MPF + P
`MPF`	Kinase, add activating or inhibiting phosphate group	Wee1/Cdc25/IE —> X-P or P-X
`ki`	Add inhibiting Cki	Cki + MPF —> Cki:MPF
`kir`	Remove inhibiting Cki	Cki:MPF —> Cki + MPF

Equation 1, Cyclin B

Differential rate equation for cyclin B [Marlovits et al. 1998].

$$\frac{d[\text{CycB}]}{dt} = +\text{k1 -k2[CycB] -k3[Cdc2][CycB]}$$

Rate rule using SimBiology format for the differential rate equation 1. For a model using this rule, see "SimBiology Model with Rate and Algebraic Rules" on page C-36.

```
Rule 1  [CycB] = k1 - K2*[CycB] - k3*[Cdc2]*[CycB]
```

Reaction and reaction rate equations derived from the differential rate equation. For a model using these reactions, see "SimBiology Model with Reactions and Algebraic Rules" on page C-43.

```
Reaction 1   AA -> CycB           v = k1
Reaction 2   CycB -> AA           v = K2*[CycB]
Reaction 3   Cdc2 + CycB -> MPF   v = k3*[Cdc2]*[CycB]
```

Equation 2, M-Phase Promoting Factor

Differential rate equation for M-phase promoting factor (MPF) [Marlovits 1998]. Note that the parameter name `kcakr` [Marlovits et al. 1998] is changed to `kpp` [Borisuk 1998] in the following reaction equations. MPF is a heterodimer of cdc2 kinase and cyclin B.

$$\frac{d[\mathrm{MPF}]}{dt} = +\mathrm{k3[Cdc2][CycB]} - \mathrm{K2[MPF]} + \mathrm{kpp[MPFp]} - \mathrm{kcak[MPF]} + \mathrm{Kcdc25[pMPF]} - \mathrm{Kwee1[MPF]} + \mathrm{kir[Cki{:}MPF]} - \mathrm{ki[MPF][Cki]}$$

Rate rule using SimBiology format for the differential rate equation 1. For a model using this rule, see "SimBiology Model with Rate and Algebraic Rules" on page C-36.

```
Rule 2 MPF = kpp*MPFp - (Kwee1 + kcak + K2)*MPF +
Kcdc25*pMPF + k3*Cdc2*CycB
```

Reaction and reaction rate equations derived from the differential rate equation. For a model using these reactions, see "SimBiology Model with Reactions and Algebraic Rules" on page C-43. A reaction name in parentheses denotes a reaction repeated in another differential rate equation.

```
(Reaction 3) Cdc2 + CycB -> MPF     v = k3*[Cdc2]*[CycB]
Reaction 4   MPF -> Cdc2 + AA       v = K2*[MPF]
Reaction 5   MPFp -> MPF            v = kpp*[MPFp]
Reaction 6   MPF -> MPFp            v = kcak*[MPF]
Reaction 7   pMPF -> MPF            v = Kcdc25*[pMPF]
Reaction 8   MPF -> pMPF            v = Kwee1*[MPF]
```

Equation 3, Inhibited M-Phase Promoting Factor

Differential rate equation for inhibited M-phase promoting factor (pMPF) [Marlovits 1998].

$$\frac{d[\text{pMPF}]}{dt} = -\text{K2[pMPF]} + \text{kpp[pMPFp]} - \text{kcak[pMPF]} + \text{Kwee1[MPF]} - \text{Kcdc25[pMPF]} + \text{kd[Cki:pMPF]}$$

Rate rule using SimBiology format for the differential rate equation 3. For a model using this rule, see "SimBiology Model with Rate and Algebraic Rules" on page C-36.

```
Rule 3 pMPF = Kwee1*MPF - (Kcdc25 + kcak +
K2)*pMPF + kpp*pMPFp
```

Reaction and reaction rate equations derived from the differential rate equation. For a model using these reactions, see "SimBiology Model with Reactions and Algebraic Rules" on page C-43.

```
Reaction 11  pMPF -> Cdc2 + AA      v = K2*[pMPF]
Reaction 12  pMPFp -> pMPF          v = kpp*[pMPFp]
Reaction 13  pMPF -> pMPFp          v = kcak*[pMPF]
(Reaction 8) MPF -> pMPF            v = Kwee1*[MPF]
(Reaction 7) pMPF -> MPF            v = Kcdc25*[pMPF]
```

Equation 4, Inhibited and Activated M-Phase Promoting Factor

Differential rate equation for inhibited and activated M-phase promoting factor (pMPFp) [Marlovits 1998].

$$\frac{d[\text{pMPFp}]}{dt} = \text{-K2[pMPFp]} + \text{kcak[pMPF]} - \text{kpp[pMPFp]} + \text{Kwee1[MPFp]} - \text{Kcdc25[pMPFp]} + \text{kd[Cki:pMPFp]}$$

Rate rule using SimBiology format for the differential rate equation. For a model using this rule, see "SimBiology Model with Rate and Algebraic Rules" on page C-36.

```
Rule 4 pMPFp = Kwee1*MPFp - (kpp + Kcdc25 + K2)*pMPFp
+ kcak*pMPF
```

Reaction and reaction rate equations derived from the differential rate equation. For a model using these reactions, see "SimBiology Model with Reactions and Algebraic Rules" on page C-43.

```
Reaction 15   pMPFp -> Cdc2 + AA       v = K2*[pMPFp]
(Reaction 13) pMPF -> pMPFp            v = kcak*[pMPF]
(Reaction 12) pMPFp -> pMPF            v = kpp*[pMPFp]
Reaction 16   MPFp -> pMPFp            v = Kwee1*[MPFp]
Reaction 17   pMPFp -> MPFp            v = Kcdc25*[pMPFp]
```

Equation 5, Activated M-Phase Promoting Factor

Differential rate equation for activated M-phase promoting factor (MPFp) [Marlovits 1998].

$$\frac{d[\text{MPFp}]}{dt} = \text{-K2[MPFp]} + \text{kcak[MPF] -kpp[MPFp]} + \text{Kcdc25[pMPFp] -Kwee1[MPFp]} + \text{kir[CKI:MPFp] -ki[CKI][MPFp]}$$

Rate rule using SimBiology format for the differential rate equation 1. For a model using this rule, see "SimBiology Model with Rate and Algebraic Rules" on page C-36.

```
Rule 5  MPFp = kcak*MPF - (kpp + Kwee1 + K2)*MPFp
+ Kcdc25*pMPFp
```

Reaction and reaction rate equations derived from the differential rate equation. For a model using these reactions, see "SimBiology Model with Reactions and Algebraic Rules" on page C-43.

```
Reaction 19    MPFp -> MPF + AA        v = K2*[MPFp]
(Reaction 6)  MPF -> MPFp             v = kcak*[MPF]
(Reaction 5)  MPFp -> MPF             v = kpp*[MPFp]
(Reaction 17) pMPFp -> MPFp           v = Kcdc25*[pMPFp]
(Reaction 16) MPFp -> pMPFp           v = Kwee1*[MPFp]
```

Equation 11, Cell Division Control 25

Differential rate equation for activating and deactivating Cdc25 [Marlovits 1998].

$$\frac{d[\mathrm{Cdc25p}]}{dt} = +\frac{\mathrm{k25}[\mathrm{MPFp}][\mathrm{Cdc25}]}{\mathrm{Km25}+[\mathrm{Cdc25}]} - \frac{\mathrm{k25r}[\mathrm{Cdc25p}]}{\mathrm{Km25r}+[\mathrm{Cdc25p}]}$$

Rate rule in SimBiology format for the differential rate equation 1. For a model using this rule, see "SimBiology Model with Rate and Algebraic Rules" on page C-36. Note that since there isn't a rate rule for `Cdc25`, its amount is written as `(TotalCdc25 - Cdc25p)`.

```
Rule 11 Cdc25p = (k25*MPFp*(TotalCdc25 - Cdc25p))/(Km25 +
(TotalCdc25 - Cdc25p)) - (k25r*PPase*Cdc25p)/(Km25r + Cdc25p)
```

Reaction and reaction rate equations derived from the differential rate equation. For a model using these reactions, see "SimBiology Model with Reactions and Algebraic Rules" on page C-43.

```
Reaction 36 Cdc25 -> Cdc25p,  v = k25*[MPFp]*[Cdc25]/(Km25 + [Cdc25])
Reaction 37 Cdc25p -> Cdc25,  v = k25r*[Cdc25p]/(Km25r + [Cdc25p])
```

Equation 12, Wee1 Activation/Deactivation

Differential rate equation for activating and deactivating Wee1 kinase [Marlovits 1998]. The kinase (MPFp) phosphorylates active Wee1 (Wee1) to its inactive form (Wee1p). The dephosphorylation of inactive Wee1 (Wee1p) is by an unknown phosphatase.

$$\frac{d[Wee1]}{dt} = -\frac{kw[MPFp][Wee1]}{Kmw+[Wee1]} + \frac{kwr[Wee1P]}{Kmwr+[Wee1P]}$$

Rate rule in SimBiology format for the differential rate equation 1. For a model using this rule, see "SimBiology Model with Rate and Algebraic Rules" on page C-36.

```
Rule 12  Wee1p = (kw*MPFp*(TotalWee1 - Wee1p))/(Kmw + (TotalWee1 - Wee1p))
                                    - (kwr*Wee1p)/(Kmwr + Wee1p)
```

Reaction and reaction rate equations derived from the differential rate equation. For a model using these reactions, see "SimBiology Model with Reactions and Algebraic Rules" on page C-43.

```
reaction 38 Wee1 -> Wee1p, v = (kw*[MPFp]*[Wee1])/(Kmw + [Wee1])
reaction 39 Wee1p -> Wee1, v = (kwr*[Wee1p])/(Kmwr + [Wee1p])
```

Equation 13, Intermediate Enzyme Activation/Deactivation

Differential rate equation for activating and deactivating the intermediate enzyme (IE) [Marlovits 1998]. The active kinase (MPFp) phosphorylates the inactive intermediate enzyme (IE) to its active form (IEp).

$$\frac{d[IEp]}{dt} = +\frac{kie[MPFp][IE]}{Kmie + [IE]} - \frac{kier[IEp]}{Kmier + [IEp]}$$

Rate rule in SimBiology format for the differential rate equation 1. For a model using this rule, see "SimBiology Model with Rate and Algebraic Rules" on page C-36.

```
Rule 13 IEp = (kie*MPFp*(TotalIE - IEp))/(Kmie + (TotalIE - IEp))
                - (kier*IEp)/(Kmier + IEp)
```

Reaction and reaction rate equations derived from the differential rate equation. For a model using these reactions, see "SimBiology Model with Reactions and Algebraic Rules" on page C-43.

```
reaction 40 IE -> IEp, v = (kie*[MPFp]*[IE])/(Kmie + [IE])
reaction 41 IEp -> IE, v = (kier*[IEp])/(Kmier + [IEp])
```

Equation 14, APC Activation/Deactivation

Differential rate equation for [Marlovits 1998].

$$\frac{d[APCa]}{dt} = +\frac{kap[IEP][APCi]}{Kmap + [APCi]} - \frac{kapr[APCa]}{Kmapr + [APCa]}$$

Rate rule in SimBiology format for the differential rate equation 1. For a model using this rule, see "SimBiology Model with Rate and Algebraic Rules" on page C-36.

```
Rule 14 APCa = (kap*IEp*(TotalAPC - APCa))/(Kmap + (TotalAPC - APCa))
                - (kapr*APCa)/(Kmapr + APCa)
```

Reaction and reaction rate equations derived from the differential rate equation. For a model using these reactions, see "SimBiology Model with Reactions and Algebraic Rules" on page C-43.

```
Reaction 42 APCi -> APCa, v = (kap*[IEp]*[APCi])/(Kmap + [APCi])
Reaction 43 APCa -> APCi,  v = (kapr*[APCa])/(Kmapr + [APCa])
```

Equation 17, Rate Parameter K2

Algebraic equation to define the rate parameter `K2` [Marlovits 1998]. Inactive APC (APCi) is catalyzed by IE (intermediate enzyme) to active APC (APCa).

$$k2 = V2'[APC] + V2''[APC']$$

Algebraic rule in SimBiology format for the algebraic equation 17. For a model using this rule, see "SimBiology Model with Rate and Algebraic Rules" on page C-36.

```
Algebraic Rule 17  V2i*(TotalAPC - APCa) + V2a*APCa - K2
```

Algebraic rule when simulating with reactions. For a model using this rule with reactions, see "SimBiology Model with Reactions and Algebraic Rules" on page C-43. `V2'` is renamed to `V2i` and `V2` is renamed to `V2a`. APCi (APC) is the inactive form of the enzyme while APCa (APC') is the active form. `K2` is the independent variable.

```
Algebraic Rule 1 (V2i*APCi) + (V2a*APCa) - K2
```

Equation 18, Rate Parameter Kcdc25

Algebraic equation to define the rate parameter `Kcdc25` [Marlovits 1998]. Inactive Cdc25 (`Cdc25`) is phosphorylated by MPF to active Cdc25 (Cdc25p).

$$kcdc25 = V_{25}'[Cdc25] + V_{25}''[Cdc25p]$$

Algebraic rule in SimBiology format for the algebraic equation 18. For a model using this rule, see "SimBiology Model with Rate and Algebraic Rules" on page C-36.

```
Algebraic Rule 18  V25i*(TotalCdc25 - Cdc25p) + V25a*Cdc25p - Kcdc25
```

Algebraic rule when simulating with reactions. `Kcdc25` is the independent variable. For a model using this rule with reactions, see "SimBiology Model with Reactions and Algebraic Rules" on page C-43.

```
Algebraic Rule 2 (V25i*Cdc25) + (V25a*Cdc25p) - Kcdc25
```

Equation 19, Rate Parameter Kwee1

Algebraic equation to define the rate parameter [Marlovits 1998]. Active Wee1 (`Wee1`) is phosphorylated by MPF to inactive Wee1 (`Wee1p`).

$$k_{wee1} = V_{wee1}'[Wee1p] + V_{wee1}''[Wee1]$$

Algebraic rule in SimBiology format for rate parameter equation 19. For a model using this rule, see "SimBiology Model with Rate and Algebraic Rules" on page C-36.

```
Algebraic Rule 19  Vwee1i*Wee1p + Vwee1a*(TotalWee1 - Wee1p) - Kwee1
```

Algebraic rule when simulating with reactions. `Kwee1` is the independent variable. For a model using this rule with reactions, see "SimBiology Model with Reactions and Algebraic Rules" on page C-43.

```
Algebraic Rule 3 (Vwee1i*Wee1p) + (Vwee1a*Wee1) - Kwee1
```

SimBiology Model with Rate and Algebraic Rules

Overview

There is one rate rule for each equation defining a species and one algebraic rule for each variable parameter in the M-phase control model [Marlovits 1998]. For a list and description of the equations, see "M-Phase Control Equations" on page C-27.

A basic model includes rate rules 1 to 5 and 11 to 14 with algebraic rules 17, 18, and 19.

Writing Differential Rate Equations as Rate Rules

Writing differential rate equations in an unambiguous format that a software program can understand is a simple process when you follow the syntax rules for programming languages.

- Use an asterisk to indicate multiplication. For example, `k[A]` is written `k*A` or `k*[A]`. The brackets around the species `A` do not indicate concentration.
- SimBiology uses square brackets around species and parameter name to allow names that are not valid MATLAB variable names. For example, you could have a species named `glucose-6-phosphate dehydrogenase` but you need to add brackets around the name in reaction rate and rule equations.

  ```
  [glucose-6-phosphate dehydrogenase]
  ```

- Use parentheses to clarify the order of evaluation for mathematical operations. For example, do not write Henri-Michaelis-Menten reaction rates as `Vm*C/Kd + C`, because `Vm*C` is divided by `Kd` before adding `C` to the result. Instead, write this reaction rate as `(Vm*C)/(Kd + C)`.

Species

The following table lists species in the model with their initial amounts. There are three variable parameters modeled as species (`K2`, `Kcdc25`, and `KWee1`). You could also model the variable parameters as parameters with the property **ConstantAmount** cleared.

Name	InitialAmount	InitialAmountUnits	ConstantAmount
CycB	0.0		☐
MPF	0.0		☐
pMPF	0.0		☐
pMPFp	0.0		☐
MPFp	0.0		☐
Cdc25p	0.0		☐
Wee1p	0.0		☐
IEp	0.0		☐
APCa	0.0		☐
Kcdc25	0.0		☐
Kwee1	0.0		☐
K2	0.0		☐
Wee1	0.0		☐
TotalCdc25	1.0		☑
TotalWee1	1.0		☑
TotalAPC	1.0		☑
TotalIE	1.0		☐
PPase	1.0		☑
AntiAPC	1.0		☑
Cdc2	100.0		☑

Parameters

The following table lists parameters in the model with their initial values. The property **ConstantValue** is selected for all of the parameters.

Name	Value	ValueUnits	ConstantValue
Vweea	1.0		☑
k1	1.0		☑
Kmwr	1.0		☑
Kmapr	1.0		☑
Km25r	1.0		☑
kcak	0.64		☑
V2a	0.25		☑
V25a	0.17		☑
kier	0.15		☑
kapr	0.13		☑
kap	0.13		☑
kwr	0.1		☑
k25r	0.1		☑
Kmw	0.1		☑
Km25	0.1		☑
kie	0.02		☑
kw	0.02		☑
k25	0.02		☑
V25i	0.017		☑
Vweei	0.01		☑
Kmier	0.01		☑
Kmie	0.01		☑
Kmap	0.01		☑
V2i	0.0050		☑
k3	0.0050		☑
kpp	0.0040		☑

Rate Rule 1, Cyclin B (CycB)

The rate rule is from "Equation 1, Cyclin B" on page C-29.

```
rate rule: CycB = k1 - K2*CycB - k3*Cdc2*CycB
```

```
   species: CycB = 0 nM
            Cdc2 = 100 nM, [x]constant
parameters: k1 = 1 nM/minute
            K2 = 0 1/minute, []constant
            k3 = 0.005 1/(nM*minute)
```

`K2` is a variable rate parameter whose value is defined by an algebraic rule. See "Algebraic Rule 17, Rate Parameter K2" on page C-43. Its value varies from `0.005` to `0.25 1/minute`.

Rate Rule 2, M-Phase Promoting Factor (MPF)

The rate rule is from "Equation 2, M-Phase Promoting Factor" on page C-30.

```
 rate rule: MPF = kpp*MPFp - (Kwee1 + kcak + K2)*MPF + Kcdc25*pMPF
                    + k3*Cdc2*CycB
   species: MPF = 0 nM
            MPFp = 0 nM
            pMPF = 0 nM
parameters: kpp = 0.004 1/minute
            kcak = 0.64 1/minute
            k3 = 0.005 1/(nM*minute)
            K2 = 0 1/minute
            Kcdc25 = 0 1/minute
            Kwee1 = 0 1/minute
```

`K2`, `Kcdc25`, and `Kwee1` are variable rate parameters whose values are defined by algebraic rules. See "Algebraic Rule 17, Rate Parameter K2" on page C-43, "Algebraic Rule 18, Rate Parameter Kcdc25" on page C-43, and "Algebraic Rule 19, Rate Parameter Kwee1" on page C-43.

Rate Rule 3, Inhibited M-Phase Promoting Factor (pMPF)

The rate rule is from "Equation 3, Inhibited M-Phase Promoting Factor" on page C-31.

```
rate rule: pMPF = Kwee1*MPF - (Kcdc25 + kcak + K2)*pMPF + kpp*pMPFp
```

Rate Rule 4, Activated but Inhibited M-Phase Promoting Factor (pMPFp)

The rate rule is from "Equation 4, Inhibited and Activated M-Phase Promoting Factor" on page C-31.

```
rate rule: pMPFp = Kwee1*MPFp - (kpp + Kcdc25 + K2)*pMPFp + kcak*pMPF
```

Rate Rule 5, Activated M-Phase Promoting Factor (MPFp)

The rate rule is from "Equation 5, Activated M-Phase Promoting Factor" on page C-32.

```
rate rule: MPFp = kcak*MPF - (kpp + Kwee1 + K2)*MPFp + Kcdc25*pMPFp
```

Rate Rule 11, Activated Cdc25 (Cdc25p)

The rate rule is from "Equation 11, Cell Division Control 25" on page C-33.

```
rate rule: Cdc25p = (k25*MPFp*(TotalCdc25 - Cdc25p))/(Km25 + (TotalCdc25 - Cdc25p))
                      - (k25r*PPase*Cdc25p)/(Km25r + Cdc25p)
```

Rate Rule 12, Inhibited Wee1 (Wee1p)

The rate rule is from "Equation 12, Wee1 Activation/Deactivation" on page C-33.

```
rate rule: Wee1p = (kw*MPFp*(TotalWee1 - Wee1p))/(Kmw + (TotalWee1 - Wee1p))
                    - (kwr*PPase*Wee1p)/(Kmwr + Wee1p)
```

Rate Rule 13, Activated Intermediate Enzyme (IEp)

The rate rule is from "Equation 13, Intermediate Enzyme Activation/Deactivation" on page C-34.

```
rate rule: IEp = (kie*MPFp*(TotalIE - IEp))/(Kmie + (TotalIE - IEp))
                        - (kier*PPase*IEp)/(Kmier + IEp)
```

Rate Rule 14, Activated APC (APCa)

The rate rule is from "Equation 14, APC Activation/Deactivation" on page C-34.

```
rate rule: APCa = (kap*IEp*(TotalAPC - APCa))/(Kmap + (TotalAPC - APCa))
                    - (kapr*AntiAPC*APCa)/(Kmapr + APCa)
```

Algebraic Rule 17, Rate Parameter K2

`K2` is a variable rate parameter whose value is determined by the amount of active and inactive `APC`. The algebraic rule is from "Equation 17, Rate Parameter K2" on page C-35.

```
algebraic rule: V2i*(TotalAPC - APCa) + V2a*APCa - K2
       species: APCi = 1 nM
                APCa = 0 nM
                TotalAPC = 1 nM [x]constant
    parameters: K2  = 0 or 0.25 1/minute, []constant
                V2i = 0.005 1/(nM*minute)
                V2a = 0.25  1/(nM*minute)
```

Algebraic Rule 18, Rate Parameter Kcdc25

`Kcdc25` is a variable rate parameter whose value is determined by the amount of active and inactive `Cdc25`. The algebraic rule is from "Algebraic Rule 18, Rate Parameter Kcdc25" on page C-43.

```
algebraic rule: V25i*(TotalCdc25 - Cdc25p) + V25a*Cdc25p - Kcdc25
```

Algebraic Rule 19, Rate Parameter Kwee1

`Kwee1` is a variable rate parameter whose value is determined by the amount of active and inactive `Wee1`. The algebraic rule is from "Equation 19, Rate Parameter Kwee1" on page C-36.

```
algebraic rule: Vweei*Wee1p + Vweea*(TotalWee1 - Wee1p) - Kwee1
```

SimBiology Model with Reactions and Algebraic Rules

Overview

There can be one or more reactions for an equation defining a species and one algebraic rule for each variable parameter in the M-phase control model [Marlovits 1998]. For a list and description of the equations, see "M-Phase Control Equations" on page C-27.

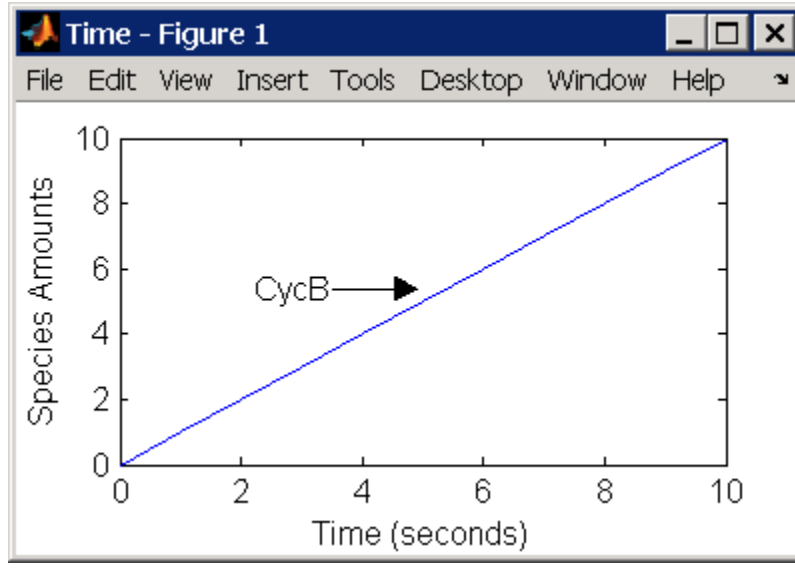
A basic model includes reactions 1 to 8, 11 to 13, 15 to 17, 19, and 36 to 43 with algebraic rules from equations 17, 18, and 19.

Reaction 1, Synthesis of Cyclin B

Cyclin B is synthesized at a constant rate.

```
      reaction: AA -> CycB
 reaction rate: k1 nM/minute
     parameter: k1 = 1 nM/minute
       species: CycB = 0 nM
                AA = 100 nM [x]constant [x]boundary
```

Simulate reaction 1 with the `sundials` solver.



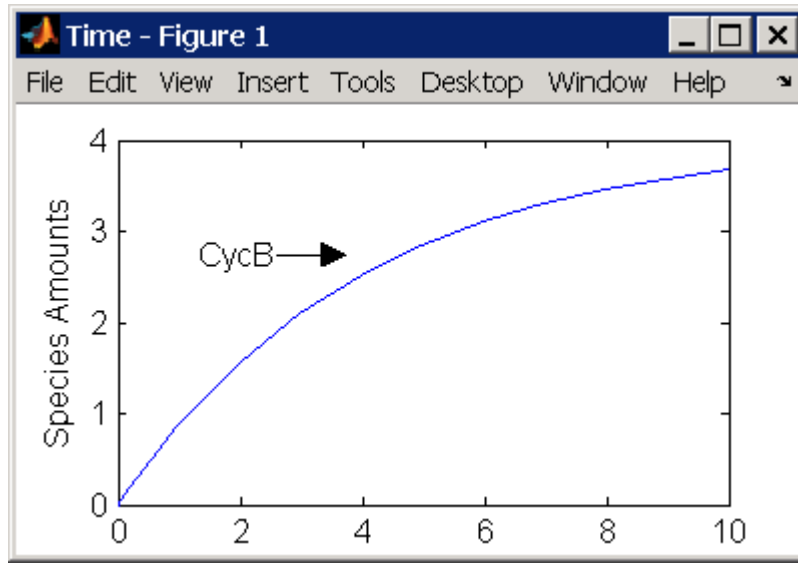
Reaction 2, Degradation of Cyclin B

Cyclin B is degraded at the end of the M-phase.

```
        reaction: CycB -> AA
   reaction rate: K2*CycB nM/minute
      parameters: K2  = 0 1/minute, []constant, variable by rule
                  V2i = 0.005 1/nM*minute
                  V2a = 0.25  1/nM*minute
         species: CycB = 0 nM
                  APCi = 1 nM
                  APCa = 0 nM
                  AA = 100 nM [x]constant [x]boundary
algebraic rule: (V2i*APCi) + (V2a*APCa) - K2
```

Initially, Cyclin B degradation is low. This implies the amount of active APC (APCa) = 0 and inactive APC (APCi) = APCtotal = 1 nM.

Test the algebraic rule by simulating reactions 1 and 2 with `APCi = 0` and `APCa = 1`.

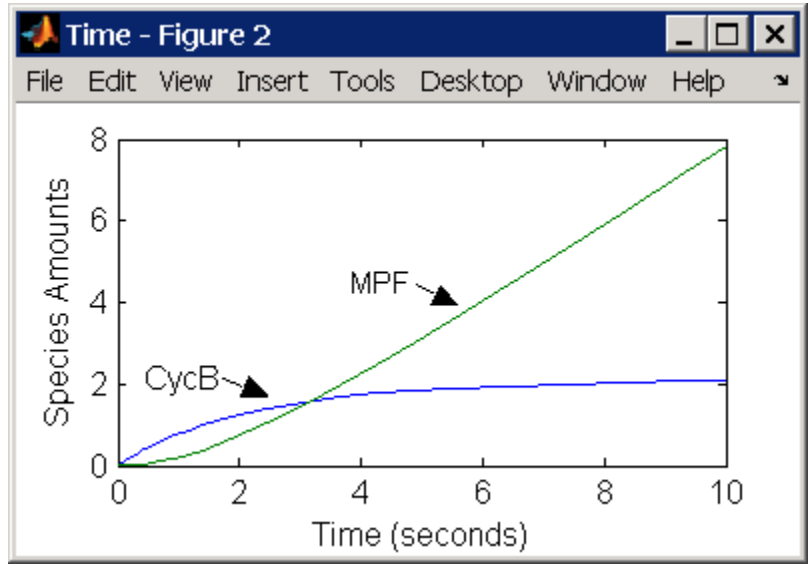


Reaction 3, Dimerization of Cyclin B with Cdc2 Kinase

Cyclin B dimerizes with Cdc2 kinase to form M-phase promoting factor (MPF).

```
      reaction: Cdc2 + CycB -> MPF
 reaction rate: k3*Cdc2*CycB nM/minute
    parameters: k3 =   0.005 1/(nM*minute)
       species: Cdc2 = 100     nM
                CycB =   0     nM
                MPF  =   0     nM
```

Test the model by simulating with `K2 = 0.25`.



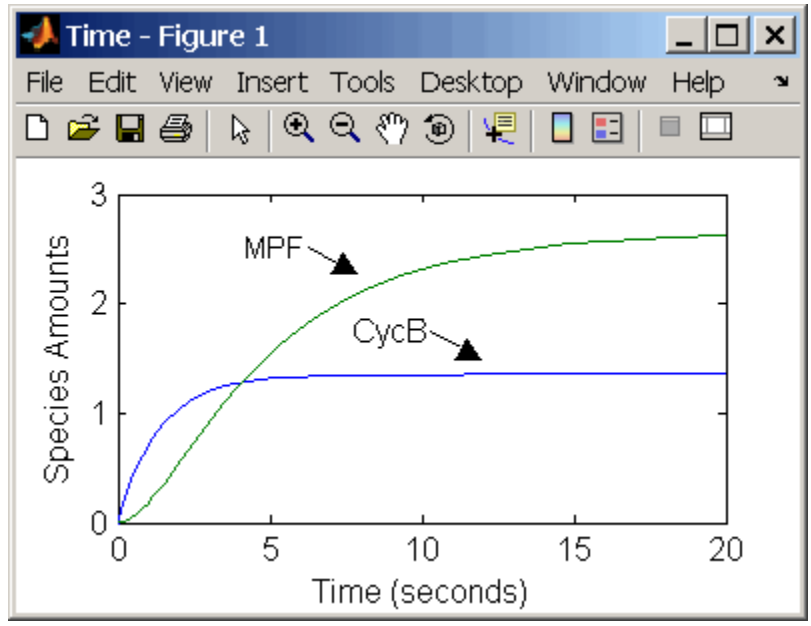
Reaction 4, Degradation of Cyclin B on MPF

Cyclin B is tagged with ubiquitin groups and degrades while bound to Cdc2.

```
      reaction: MPF -> Cdc2 + AA
 reaction rate: K2*[MPF]
    parameters: K2 = 0 or 0.25 1/minute, variable by rule
                v2i = 0.005 1/(nM*minute)
                v2a = 0.25 1/(nM*minute)
       species: MPF = 0 nM
```

```
                APCi = 1 nM
                APCa = 0 nM
                AA = 100 nM [x]constant [x]boundary
algebraic rule: (v2i*APCi) + (v2a*APCa) - K2
```

Test the simulation with `APCa = 1 and APCi = 0`. Because the amount of APCa (active) is high, K2 increases and the degradation starts to balance the synthesis of MPF.



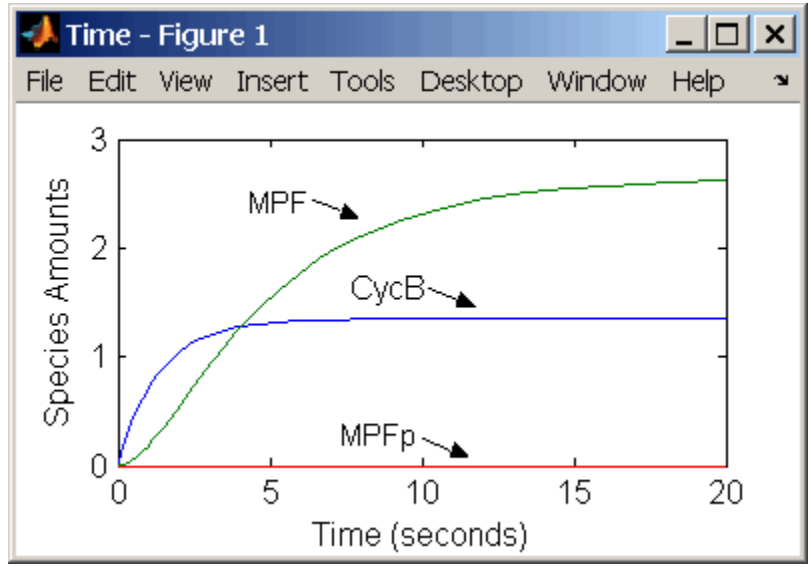
Reaction 5, Deactivation of Active MPF

Active MPF (MPFp) is dephosphorylated on Thr-161 by an unknown phosphatase (PP) to inactive MPF (MPF).

```
      reaction: MPFp -> MPF
reaction rate: kpp*[MPFp]
   parameters: kpp = 0.004 1/minute
      species: MPFp = 0 nM
               MPF = 0 nM
```

`kcakr = 0.004` 1/minute [Marlovits 1998, p. 175], but is renamed to `kpp` [Borisuk 1998].

Test simulation with APCa = 1 and APCi = 0. MPF increases without reaching steady state.



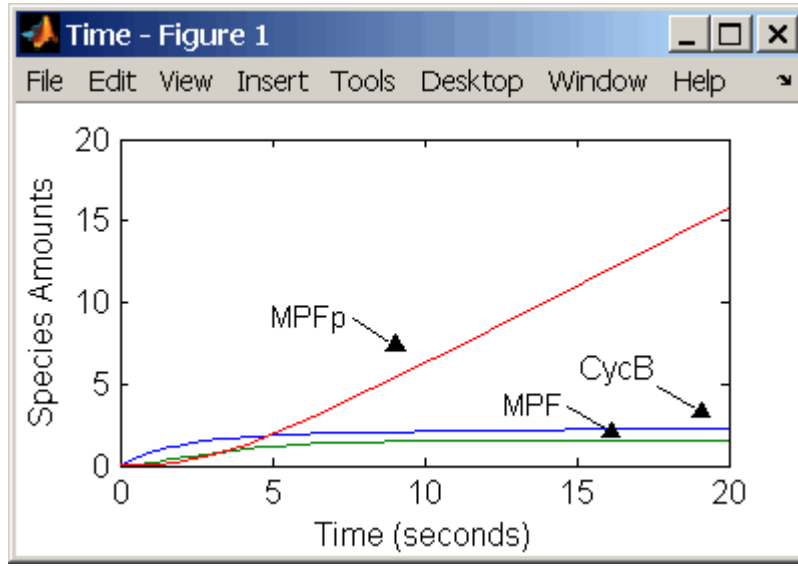
Reaction 6, Activation of MPF

Inactive MPF (MPF) is phosphorylated on Thr-161 by an unknown cyclin activating kinase (CAK).

```
      reaction: MPF -> MPFp
 reaction rate: kcak*[MPF]
    parameters: kcak = 0.64 1/minute
       species: MPF = 0 nM
                MPFp = 0 nM
```

The kinase reaction that phosphorylates MPF to the active form is 160 times faster than the phosphatase reaction that dephosphorylates active MPF.

Simulate the model with reactions 1 to 6. Notice that after adding reaction 6, most of the product goes to active MPF (MPFp).



Reaction 7, Remove Inhibiting Phosphate from Inhibited MPF

Cdc25 phosphatase removes the inhibiting phosphate groups at the threonine 14 and tyrosine 15 residues on Cdc2 kinase.

```
      reaction: pMPF -> MPF
 reaction rate: Kcdc25*[pMPF]
    parameters: Kcdc25 = 0.0 1/minute or 0.017 1/minute, variable by
                                                          algebraic rule
                V25i = 0.017 1/(mM*minute)
                V25a = 0.17 1/mM*minute
       species: pMPF = 0 nM
                MPF = 0 nM
                Cdc25 = 1 nM (inactive)
                Cdc25p = 0 nM (active)
algebraic rule: (V25i*Cdc25) + (V25a*Cdc25p) - Kcdc25
```

Initially, all of the Cdc25 phosphatase is in the inactive form (Cdc25).

Enter the initial value for Kcdc25 as 0.0 and let the first time step calculate the value from the rule, or enter an initial value using the rule.

Initially, set **ConstantAmount** for Cdc25 and Cdc25p to test reactions 1 through 7. Then after you can add the reactions to regulate the Cdc25 phosphatase by clearing the **ConstantAmount** property.

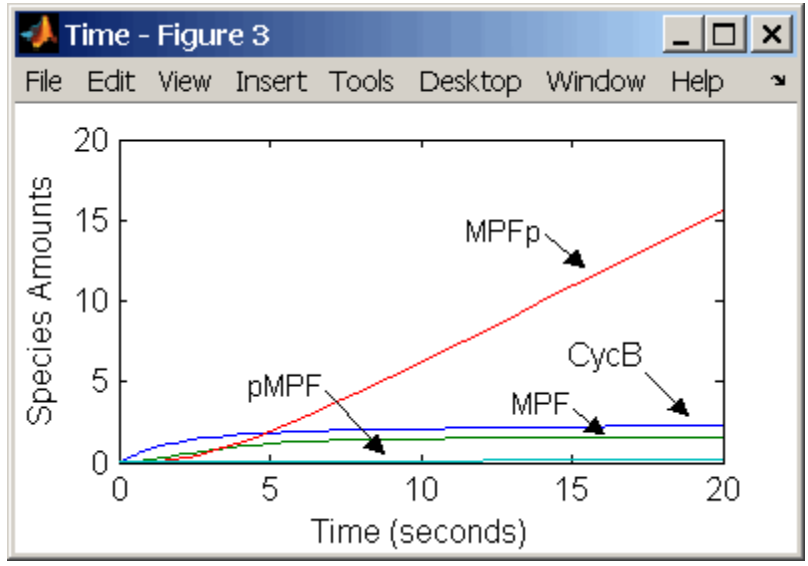
Reaction 8, Inhibition of MPF by Phosphorylation

Addition of inhibiting phosphate groups by Wee1 kinase to inhibit active M-phase promoting factor (MPF). Myt1 kinase is also involved with the phosphorylation, but its contribution is grouped with Wee1.

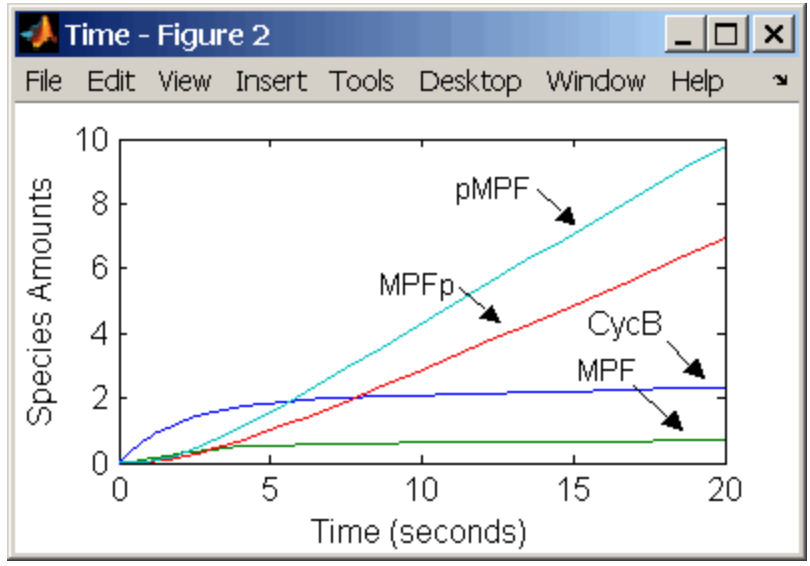
```
      reaction: MPF -> pMPF
 reaction rate: Kwee1*[MPF]
    parameters: Kwee1 = 0.0 1/minute or 0.01 1/minute, variable by
                                                        algebraic rule
                Vwee1i = 0.01 1/(nM*minute)
                Vwee1a = 1.0 1/(nM*minute)
       species: MPF = 0 nM
                pMPF = 0 nM
                Wee1p = 1 nM (inactive)
                Wee1 = 0 nM  (active)
 algebraic rule: (Vwee1i*Wee1p) + (Vwee1a*Wee1) - Kwee1
```

The initial capitalization for the parameter Kwee1 is a convention to indicate that this value changes during the simulation.

Test the simulation for reactions 1 through 8 with `Wee1p` (inactive) = `1` and `Wee1` (active) = `0`.



Test the simulation with `Wee1p` (inactive) = `0` and `Wee1` (active) = `1`.



Reaction 11, Degradation of Cyclin B on Inhibited MPF

Degradation of cyclin B (`CycB`) on inhibited MPF (`pMPF`). Cyclin B is tagged with ubiquitin groups and degrades while bound to Cdc2.

```
       reaction: pMPF -> Cdc2 + AA
  reaction rate: K2*[pMPF] nM/minute
     parameters: K2 = 0 or 0.25 1/minute, variable by rule
                 V2i = 0.005 1/nM*minute
                 V2a = 0.25 1/nM*minute
        species: MPF = 0 nM
                 APCi = 1 nM
                 APCa = 0 nM
                 AA = 100 nM [x]constant [x]boundary
                 Cdc2 = 100 nm
algebraic rule: (V2i*APCi) + (V2a*APCa) - K2
```

Test the simulation with Wee1 active (`Wee1 = 1`) and APC active (`APCi = 1`).

Reaction 12, Deactivation of MPF to Inhibited MPF

Inhibited/active MPF (pMPFp) is dephosphorylated on Thr-161 by an unknown phosphatase (PP) to inhibited MPF (pMPF). Compare reaction 12 with reaction 5.

```
      reaction: pMPFp -> pMPF
reaction rate: kpp*[pMPFp]
   parameters: kpp = 0.004 1/minute
      species: pMPFp = 0 nM
               pMPF  = 0 nM
```

Reaction 13, Activation of Inhibited MPF

Inhibited MPF (pMPF) is phosphorylated on Thr-161 by an unknown cyclin-activating kinase (CAK). Compare reaction 13 with reaction 6.

```
      reaction: pMPF -> pMPFp
reaction rate: kcak*[pMPF] nM/minute
   parameters: kcak = 0.64 1/minute
      species: pMPF = 0 nM
               pMPFp = 0 nM
```

Test the simulation with Wee1p = 1 (inactive)/ Wee1 = 0 and then test with Wee1p = 0 (inactive)/ Wee1 = 1.

Reaction 15, Degradation of Cyclin B on Active but Inhibited MPF

Degradation of cyclin B (CycB) on inhibited MPF (pMPF). Cyclin B is tagged with ubiquitin groups and degrades while bound to cdc2 kinase.

```
       reaction: pMPFp -> Cdc2 + AA
  reaction rate: K2*[pMPFp] nM/minute
     parameters: K2 = 0 or 0.25 1/minute, variable by rule
                 v2i = 0.005 1/nM*minute
                 v2a = 0.25 1/nM*minute
        species: MPF = 0 nM
                 APCi = 1 nM
                 APCa = 0 nM
                 AA = 100 nM [x]constant [x]boundary
                 Cdc2 = 100 nm
algebraic rule: (V2i*APCi) + (V2a*APCa) - K2
```

Reaction 16, Inhibit MPF by Phosphorylation

Addition of inhibiting phosphate groups by Wee1 kinase to inhibit active M-phase promoting factor (MPF). Myt1 kinase is also involved with the phosphorylation, but its contribution is grouped with Wee1.

```
       reaction: MPFp -> pMPFp
  reaction rate: Kwee1*[MPFp] nM/minute
     parameters: Kwee1 = 1/minute []constant, variable by rule
                 Vweei = 0.01 1/nM*minute
                 Vweea = 1 1/nM*minute
        species: MPFp = 0 nM
                 pMPFp = 0 nM
                 Wee1p = 1 nM (inactive)
                 Wee1 = 0 nM (active)
algebraic rule: (Vwee1i*Wee1p) + (Vwee1a*Wee1) - Kwee1
```

Reaction 17, Remove Inhibiting Phosphate from Activated MPF

Remove the inhibiting phosphate group from pMPFp with cdc25 phosphatase.

```
      reaction: pMPFp -> MPFp
 reaction rate: Kcdc25*[pMPFp]
    parameters: Kcdc25 = 0 1/minue, []constant, variable by rule
                V25i = 0.017 1/nM*minute
                V25a = 0.17 1/nM*minute
       species: pMPFp = 0 nM
                MPFp = 0 nM
 algebraic rule: (V25i*Cdc25) + (V25a*Cdc25p) - Kcdc25
```

Reaction 19, Degradation of Cyclin B on Activated MPF

Degradation of cyclin B (`CycB`) on inhibited MPF (`pMPF`). Cyclin B is tagged with ubiquitin groups and degrades while bound to cdc2 kinase.

```
       reaction: MPFp -> MPF + AA
  reaction rate: K2*[MPFp] nM/minute
     parameters: K2 = 0 or 0.25 1/minute, variable by rule
                 V2i = 0.005 1/nM*minute
                 V2a = 0.25 1/nM*minute
        species: MPF = 0 nM
                 MPFp = 0 nM
                 APCi = 1 nM
                 APCa = 0 nM
                 AA = 100 nM [x]constant [x]boundary
                 Cdc2 = 100 nm
 algebraic rule: (V2i*APCi) + (V2a*APCa) - K2
```

Reaction 36, Activation of Cdc25 by Activated MPF

Activation of cdc25 phosphatase by phosphorylation with active M-phase promoting factor (`MPFp`).

```
       reaction: Cdc25 + (MPFp) -> Cdc25p + (MPFp)
 reaction rate: (k25*[MPFp]*[Cdc25])/(Km25 + [Cdc25])
    parameters: k25 = 0.02 1/minute
                Km25 = 0.1 nM
```

```
        species: Cdc25 = 1 nM (inactive)
                 Cdc25p = 0 nM (active)
```

Initially MPF is inhibited (MPF* reacts to pMPF*).

Reaction 37, Deactivation of Cdc25

Deactivation of cdc25 phosphatase by dephosphorylation with an unknown phosphatase.

```
      reaction: Cdc25p -> Cdc25
reaction rate: (k25r*[Cdc25p])/(Km25r + [Cdc25p])
   parameters: k25r = 0.1 nM/minute
               Km25r = 1 nM
      species: Cdc25 = 1 nM (inactive)
               Cdc25p = 0 nM (active)
```

Reaction 38, Deactivation of Wee1 by Active MPF

Deactivation of Wee1 kinase by phosphorylation with active M-phase promoting factor (MPFp).

```
      reaction: Wee1 + (MPFp) -> Wee1p + (MPFp)
reaction rate: (kw*[MPFp]*[Wee1])/(Kmw + [Wee1]) nM/minute
   parameters: kw = 0.02 1/minute
               Kmw = 0.1 nM
      species: Wee1p = 1 nM (inactive)
               Wee1 = 0 nM (active)
```

Initially MPF is inhibited (MPF* reacts to pMPF*).

Reaction 39, Activation of Wee1

Activation of Wee1 kinase by dephosphorylation with an unknown kinase.

```
      reaction: Wee1p -> Wee1
reaction rate: (kwr*[Wee1p])/(Kmwr + [Wee1p]) nM/minute
   parameters: kwr = 0.1 nM/minute
               Kmwr = 1 nM
      species: Wee1p = 1 nM (inactive)
```

```
                Wee1 = 0 nM (active)
```

Reaction 40, Activation of Intermediate Enzyme by Active MPF

The inactive intermediate enzyme (IE) is activated by phosphorylation with active M-phase promoting factor (MPFp).

```
      reaction: IE + (MPFp) -> IEp + (MPFp)
reaction rate: (kie*[MPFp]*[IE])/(Kmie + [IE])
   parameters: kie = 0.02 1/minute
               Kmie = 0.01nM
      species: IE = 1 nM (inactive)
               IEp = 0 nM (active)
```

Reaction 41, Deactivation of IE

The active intermediate enzyme (IE) is deactivated by dephosphorylation.

```
      reaction: IEp -> IE
reaction rate: (kier*[IEp])/(Kmier + [IEp])
   parameters: kier = 0.15 nM/minute
               Kmier = 0.01 nM
      species: IE = 1 nM (inactive)
               IEp = 0 nM (active)
```

Reaction 42, APC Activation by IEp

Anaphase-promoting complex (APC) is activated by an active intermediate enzyme (IEp).

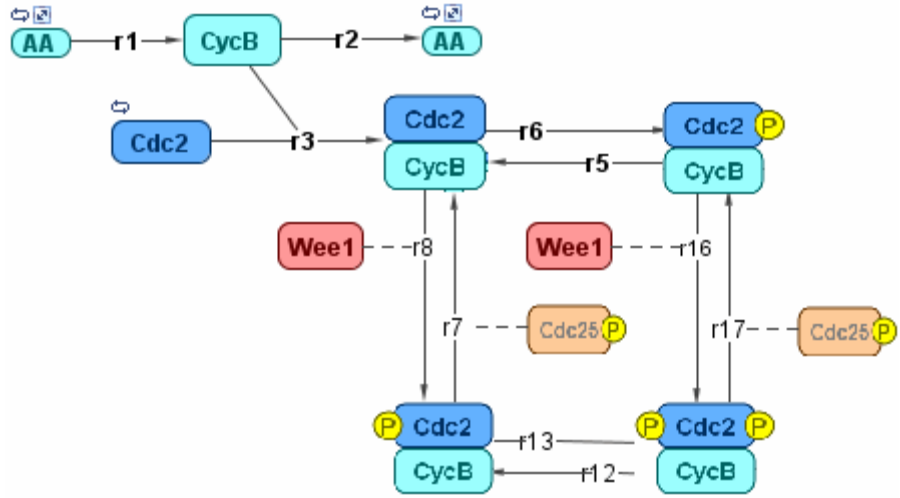
```
      reaction: APCi + IEp -> APCa + IEp
reaction rate: (kap*[IEp]*[APCi])/(Kmap + [APCi])
   parameters: kap = 0.13 1/minute
               Kmap = 0.01 nM
     species : APCi = 1 nM
               APCa = 0 nM
```

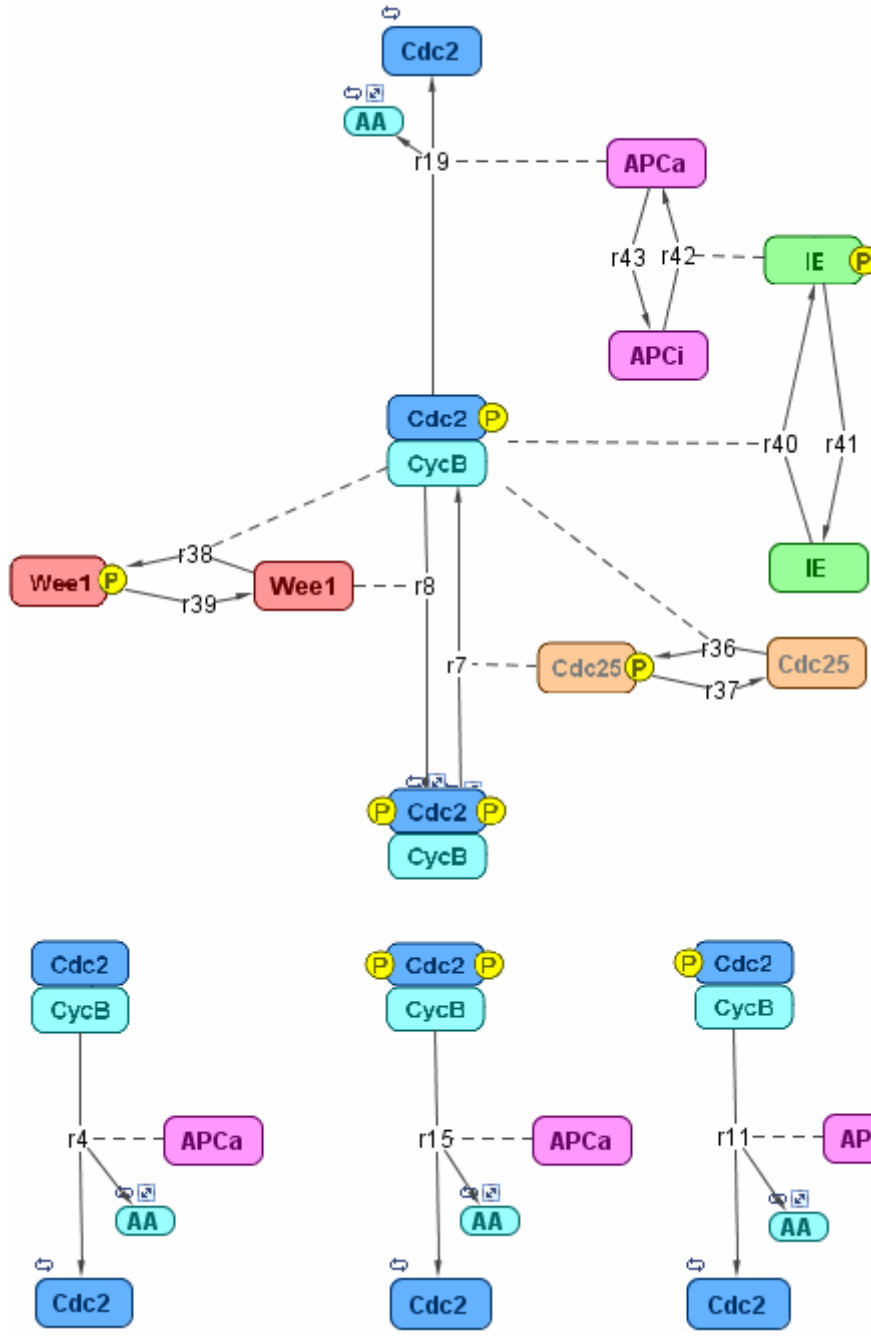
Reaction 43, APC Deactivation

Anaphase-promoting complex (APC) is deactivated.

```
      reaction: APCa -> APCi
reaction rate: (kapr*[APCa])/(Kmapr + [APCa])
   parameters: kapr = 0.13 nM/minute
               Kmapr = 1 nM
     species : APCi = 1 nM
               APCa = 0 nM
```

Block Diagram of the M-Phase Control Model with Reactions




Cdc2
AA
r19
APCa
r43
r42
IE
P
APCi
Cdc2
P
CycB
r40
r41
IE
r38
Wee1
P
r39
Wee1
r8
r7
r36
Cdc25
P
r37
Cdc25
P
Cdc2
P
CycB
Cdc2
CycB
r4
APCa
AA
Cdc2
P
Cdc2
P
CycB
r15
APCa
AA
Cdc2
P
Cdc2
CycB
r11
APC
AA
Cdc2

References

[1] Borisuk M, Tyson J (1998), "Bifurcation analysis of a model of mitotic control in frog eggs," Journal of Theoretical Biology, 195(1):69–85, PubMed 9802951.

[2] Marlovits G, Tyson C, Novak B, Tyson J (1998), "Modeling M-phase control in Xenopus oocyte extracts: the surveillance mechanism for unreplicated DNA," Biophysical Chemistry, 72(1-2):169–184, PubMed 9652093.

[3] Novák B, Tyson J (1993), "Numerical analysis of a comprehensive model of M-phase control in Xenopus oocyte extracts and intact embryos," Journal of Cell Science, 106(4):1153–1168, PubMed 8126097.

Index