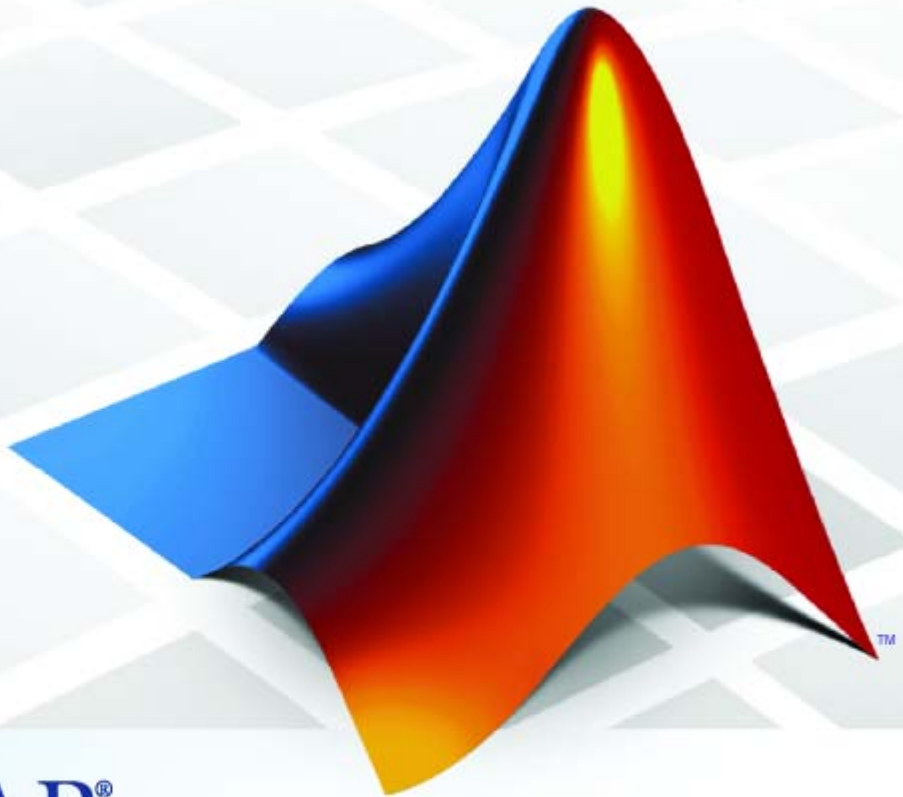


SimBiology[®] 3

User's Guide



MATLAB[®]

How to Contact The MathWorks



www.mathworks.com
comp.soft-sys.matlab
www.mathworks.com/contact_TS.html

Web
Newsgroup
Technical Support



suggest@mathworks.com
bugs@mathworks.com
doc@mathworks.com
service@mathworks.com
info@mathworks.com

Product enhancement suggestions
Bug reports
Documentation error reports
Order status, license renewals, passcodes
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*

© COPYRIGHT 2005–2009 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

The 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)

1

Modeling

Defining Reaction Rates with Mass Action Kinetics . . .	1-3
Definition of Mass Action Kinetics	1-4
Zero-Order Reactions	1-4
First-Order Reactions	1-6
Second-Order Reactions	1-7
Reversible Mass Action	1-9
Defining Reaction Rates with Enzyme Kinetics	1-10
Simple Model for Single Substrate Catalyzed Reactions . .	1-10
Enzyme Reactions with Differential Rate Equations	1-10
Enzyme Reactions with Mass Action Kinetics	1-12
Enzyme Reactions with Irreversible Henri-Michaelis-Menten Kinetics	1-13
Evaluation of Reaction Rate	1-15
Reaction Rate Dimensions	1-15
Reactions Spanning Multiple Compartments	1-15
Examples	1-15
Using Constant Amount and Boundary Condition for Species	1-17
Definition of Constant and Boundary Properties	1-17
Changing Species Amounts with Reactions or Rules	1-18
Keeping Species Amount Unchanged	1-18
Constant = NO, Boundary = YES	1-19
Constant = YES, Boundary = YES	1-20
Model Edges	1-21
Scoping Parameters for Reactions, Rules, and Events	1-22
Definition of Parameter Scope	1-22
Using a Parameter in Events and Rules	1-23
Changing the Scope of a Parameter	1-23

Changing Model Component Values Using Rules	1-24
What Is a Rule?	1-24
What Is an initialAssignment Rule?	1-25
What Is a repeatedAssignment Rule?	1-25
What Is an Algebraic Rule?	1-26
What Is a Rate Rule?	1-27
Typical Uses of Rate Rules	1-27
Changing Model Component Values Using Events	1-32
What Is an Event?	1-32
How Events Are Evaluated	1-33
Evaluation of Simultaneous Events	1-38
Evaluation of Multiple Event Functions	1-39
When One Event Triggers Another Event	1-39
Cyclical Events	1-40
Desktop Example — Changing Species Amounts Using an Event	1-41
Overview	1-41
Prerequisites	1-42
Adding an Event to the Example Model	1-42
Simulating the Modified Model	1-44
Storing and Applying Alternate Model Values Using Variants	1-48
What Are Variants?	1-48
Using Variants	1-49
Applying Multiple Variants in a Model	1-50
Desktop Example — Applying Changes to Parameter Value Using a Variant	1-51
Overview	1-51
Prerequisites	1-52
Applying Alternate Values Using Variants	1-53
Simulation Results for the Model of the Mutant Strain ...	1-53
Verifying that the Model Has No Warnings or Errors ..	1-56
Verifying the Model in the SimBiology Desktop	1-56
Verifying the Model at the Command Line	1-58

Desktop Example — Using User-Defined Functions in Expressions	1-59
Prerequisites	1-59
Overview	1-59
Creating an M-File Function	1-61
Calling the Function in a Rule Expression	1-62
See Also	1-67
Importing and Exporting Model Component Data	1-68
Importing Model Component Data	1-68
Exporting Model Component Data	1-69

Simulation

2

Performing Simulations	2-2
Performing Simulations at the Command Line	2-2
Performing Simulations in the SimBiology Desktop	2-2
About Simulation Solvers	2-5
How Solvers Work	2-5
Stiff Versus Nonstiff Models	2-5
Selecting a Solver	2-6
Nonstiff Deterministic Solvers	2-8
When to Use Nonstiff Deterministic Solvers	2-8
ode45 (Dormand-Prince)	2-8
ode23 (Bogacki-Shampine)	2-8
ode113 (Adams)	2-8
See Also	2-9
Stiff Deterministic Solvers	2-10
When to Use Stiff Deterministic Solvers	2-10
ode15s (stiff/NDF)	2-10
ode23s (stiff/Mod. Rosenbrock)	2-10
ode23t (Mode. stiff/Trapezoidal)	2-10
ode23tb (stiff/TR-BDF2)	2-11
See Also	2-11

Stochastic Solvers	2-12
When to Use Stochastic Solvers	2-12
Stochastic Simulation Algorithm (SSA)	2-12
Explicit Tau-Leaping Algorithm	2-13
Implicit Tau-Leaping Algorithm	2-13
Ensemble Runs of Stochastic Simulations	2-14
References	2-16
Sundials Solvers	2-17

Analysis

3

Scanning Analysis	3-2
Scanning Overview	3-2
Creating a Scan Task	3-3
Setting Options to Scan Using User-Defined Values	3-3
Options For Monte Carlo Methods	3-6
Setting Options to Scan Using Monte Carlo Methods	3-7
Desktop Example — Scanning	3-10
Overview	3-10
Prerequisites	3-12
Setting Options to Scan with One Parameter	3-13
Results of Scanning with One Parameter	3-15
Scanning with Multiple Parameters	3-20
Results of Scanning with Multiple Parameters	3-21
References	3-24
Sensitivity Analysis	3-25
About Sensitivity Analysis	3-25
Performing Sensitivity Analysis Using the Command	
Line	3-26
Performing Sensitivity Analysis Using the Desktop	3-27
Reference	3-28
Desktop Example — Calculating Sensitivities	3-29
Overview	3-29
Prerequisites	3-32

Setting Options for Sensitivity Analysis	3-33
Getting Results for Sensitivity Analysis	3-34
References	3-38
Command-Line Example — Calculating Sensitivities ..	3-39
Overview	3-39
Loading and Configuring the Model for Sensitivity Analysis	3-40
Performing Sensitivity Analysis	3-41
Extracting and Plotting Sensitivity Data	3-41
See Also	3-44
Parameter Estimation	3-45
About Parameter Estimation	3-45
SimBiology Parameter Estimation	3-45
Command-Line Example — Parameter Estimation	3-46
About the Example Model	3-46
Importing Target Experimental Data	3-47
Simulating the G Protein Model	3-47
Estimating a Parameter (kGd) in the G Protein Model ...	3-50
Simulating and Plotting Results Using the Estimated Parameter	3-53
Estimating Other Parameters in the G Protein Model	3-54
Moiety Conservation	3-58
Introduction to Moiety Conservation	3-58
Algorithms for Conserved Cycle Calculations	3-58
Command-Line Examples — Determining Conserved Moieties	3-61
G Protein Example	3-61
Mitotic Oscillator Example	3-64
Desktop Example — Creating Custom Analysis	3-68
About Custom Analysis	3-68
Open the Example Project	3-68
Setting Up a Custom Task	3-69
Parameter Estimation Using Custom Task	3-69
Visualizing Results Using Plot Types	3-73

About Plot Types	3-73
How Plot Types Work	3-75
Summary of Built-In Plot Types	3-79
When to Use User-Defined Plot-Types	3-80
Desktop Example — Creating and Using Plot Types ..	3-81
Goal	3-81
Workflow	3-81
Why Use User-Defined Plot Types?	3-82
Creating a User-Defined Plot Type in the Library	
Explorer	3-84
Using the Plot Type in a Task	3-87

Pharmacokinetic Modeling

4

Pharmacokinetic Modeling Functionality	4-2
Overview	4-2
Required and Recommended Products	4-2
How This Product Supports Pharmacokinetic Modeling ..	4-3
Using the Command Line Versus the SimBiology	
Desktop	4-5
Accessing a Pharmacokinetic Modeling Demo	4-5
Importing Data	4-6
Supported Files and Data Types	4-6
Importing Data at the Command Line	4-7
Data Import in the SimBiology Desktop	4-8
Importing Data from the MATLAB Workspace into the	
SimBiology Desktop	4-9
Importing Data from a Text File into the SimBiology	
Desktop	4-12
Importing Data from an Excel File into the SimBiology	
Desktop	4-14
Importing Data from a MAT-File into the SimBiology	
Desktop	4-15
Resources for Working with Imported Data	4-17

Working with Imported Data in the SimBiology Desktop	4-19
Accessing Imported Data	4-19
Identifying Group and Independent Variables in the Data	4-20
Visualizing Data Using Plots in the SimBiology Desktop ..	4-21
Excluding Rows of Data	4-23
Creating Additional Data Columns Using Derived Data ..	4-25
Calculating Statistics for Imported Data	4-26
Saving Your Work as a SimBiology Project File	4-27
Creating Pharmacokinetic Models	4-28
Overview	4-28
How Pharmacokinetic Models Are Represented by SimBiology Models	4-29
Creating PK Models at the Command Line	4-30
Creating PK Models in the SimBiology Desktop Using a Wizard	4-32
About Dosing Types	4-34
About Elimination Types	4-37
About Intercompartmental Clearance	4-39
Unit Conversion for Imported Data and the Model	4-40
Parameter Fitting Using Custom SimBiology Models ..	4-42
Overview	4-42
Defining Model Components for Observed Response, Dose, Dosing Type, and Parameters to be Estimated	4-42
Simulating a Model Containing Dosing Information	4-45
Parameter Fitting in Pharmacokinetic Models	4-47
Parameter Fitting Functionality	4-47
Prerequisites for Parameter Fitting	4-48
Fitting Pharmacokinetic Model Parameters at the Command Line	4-49
Fitting Parameters	4-49
Specifying and Classifying the Data to Fit	4-50
Setting Initial Estimates	4-51
Specifying the Covariance Pattern of Random Effects and a Covariate Model	4-52
Performing Population Fitting Using sbionlmeft	4-56
Performing Individual Fitting Using sbionlinfit	4-60

About Simulation Settings and Specifying Alternate Values for Initial Estimates	4-61
--	------

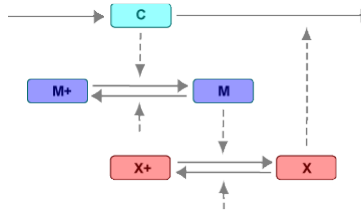
Fitting Pharmacokinetic Model Parameters in the

SimBiology Desktop	4-63
Fitting Parameters	4-63
Opening a SimBiology Project	4-64
Adding a Model Task to Fit Parameters	4-64
Performing Population Fitting	4-64
Performing Individual Fitting	4-71
About Simulation Settings and Specifying Alternate Values for Initial Estimates	4-75
Visualizing Parameter Fitting Results and Generating Diagnostic Plots	4-76

Index

Modeling

This chapter is a collection of topics relevant to modeling in general, but is presented in the context of using SimBiology® software to model biological processes.



- “Defining Reaction Rates with Mass Action Kinetics” on page 1-3
- “Defining Reaction Rates with Enzyme Kinetics” on page 1-10
- “Evaluation of Reaction Rate” on page 1-15
- “Using Constant Amount and Boundary Condition for Species” on page 1-17
- “Scoping Parameters for Reactions, Rules, and Events” on page 1-22
- “Changing Model Component Values Using Rules” on page 1-24
- “Changing Model Component Values Using Events” on page 1-32
- “Desktop Example — Changing Species Amounts Using an Event” on page 1-41
- “Storing and Applying Alternate Model Values Using Variants” on page 1-48
- “Desktop Example — Applying Changes to Parameter Value Using a Variant” on page 1-51
- “Verifying that the Model Has No Warnings or Errors” on page 1-56
- “Desktop Example — Using User-Defined Functions in Expressions” on page 1-59

- “Importing and Exporting Model Component Data” on page 1-68

Defining Reaction Rates with Mass Action Kinetics

In this section...
“Definition of Mass Action Kinetics” on page 1-4
“Zero-Order Reactions” on page 1-4
“First-Order Reactions” on page 1-6
“Second-Order Reactions” on page 1-7
“Reversible Mass Action” on page 1-9

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.

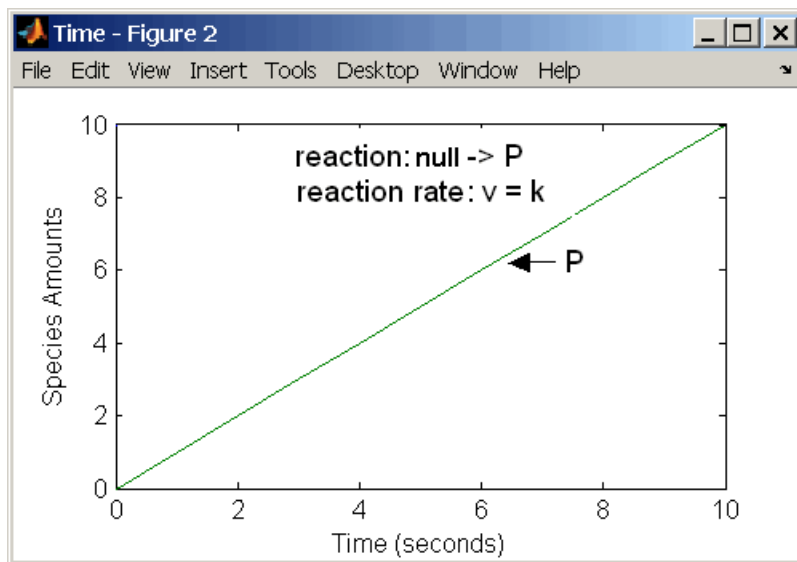
Zero-Order Reactions

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

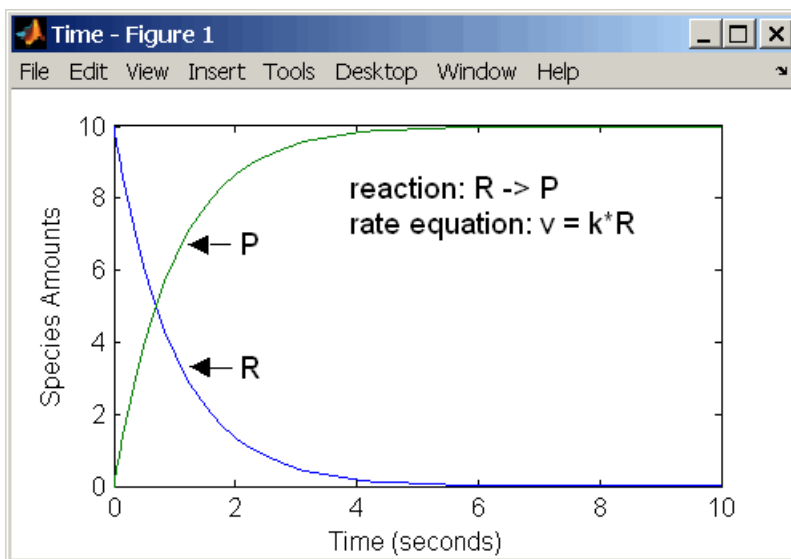
Note If the amount of a reactant with zero-order kinetics reaches zero before the end of a simulation, then the amount of reactant can go below zero regardless of the solver or tolerances you set.

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 = 11/second
```

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



First-Order Mass Action Kinetics

Second-Order Reactions

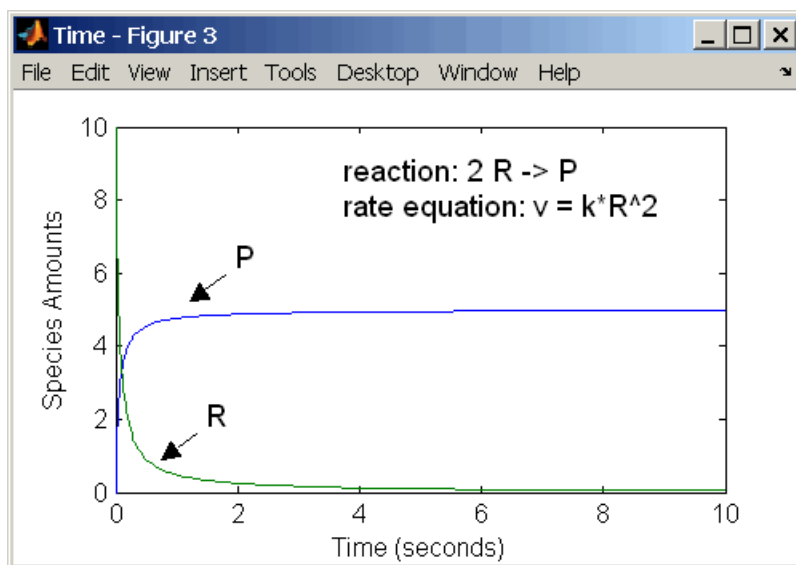
A second-order reaction has a reaction rate that is proportional to the square or the concentration of a single reactant or proportional to two reactants. Notice the space between the reactant coefficient and the name of the reactant. Without the space, 2R would be considered the name of a species.

```

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.

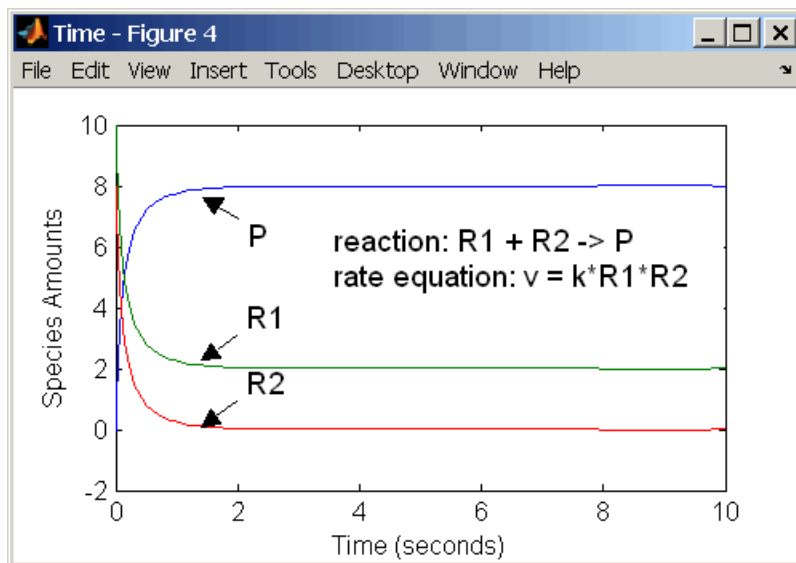
```

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

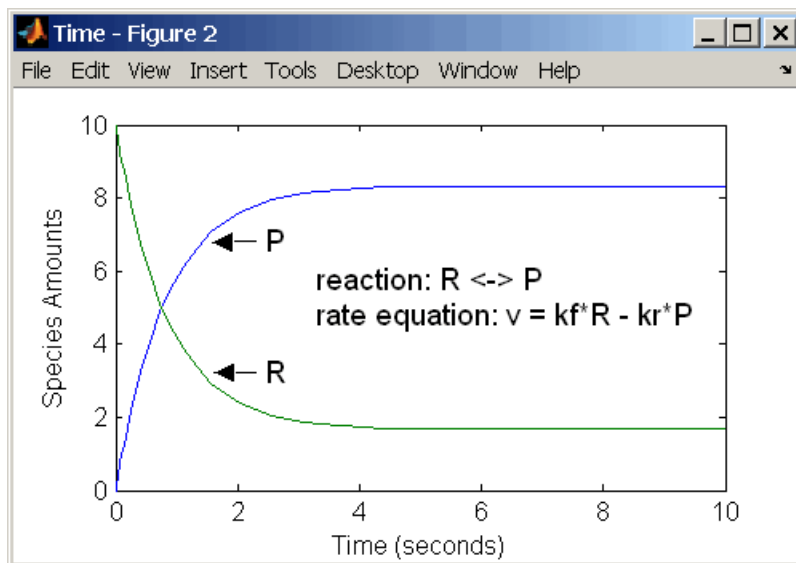
You can model reversible reactions with two separate reactions or with one reaction. With a single reversible reaction, the reaction rates for the forward and reverse reactions are combined into one expression. Notice the angle brackets before and after the hyphen to represent a reversible reaction.

```

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 $P/R = kf/kr$.



Defining Reaction Rates with Enzyme Kinetics

In this section...

“Simple Model for Single Substrate Catalyzed Reactions” on page 1-10

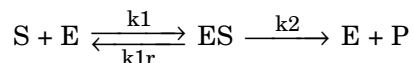
“Enzyme Reactions with Differential Rate Equations” on page 1-10

“Enzyme Reactions with Mass Action Kinetics” on page 1-12

“Enzyme Reactions with Irreversible Henri-Michaelis-Menten Kinetics” on page 1-13

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.



$$v_1 = k_1[S][E], \quad v_{1r} = k_{1r}[ES], \quad v_2 = k_2[ES]$$

This simple model can be defined with

- Differential rate equations. See “Enzyme Reactions with Differential Rate Equations” on page 1-10.
- Reactions with mass action kinetics. See “Enzyme Reactions with Mass Action Kinetics” on page 1-12.
- Reactions with Henri-Michaelis-Menten kinetics. See “Enzyme Reactions with Irreversible Henri-Michaelis-Menten Kinetics” on page 1-13.

Enzyme Reactions with Differential Rate Equations

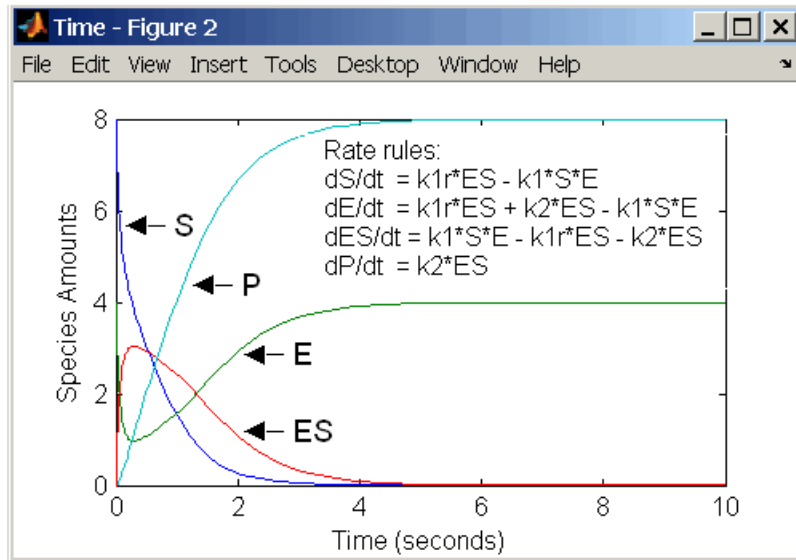
The reactions for a single-substrate enzyme reaction mechanism (see “Simple Model for Single Substrate Catalyzed Reactions” on page 1-10) can be described with differential rate equations. You can enter the differential rate equations into the software as rate rules.

```

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 “What Is a Rate Rule?” on page 1-27.



Alternatively, you could remove the rate rule for ES, add a new species E_{total} for the total amount of enzyme, and add an algebraic rule $0 = E_{total} - E - ES$, where the initial amounts for E_{total} 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 uses 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 1-10.

```

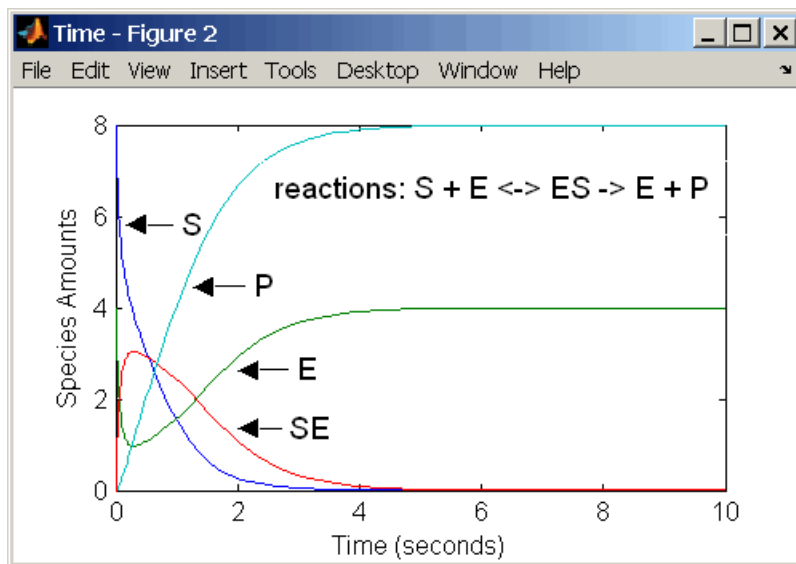
reaction: S + E -> ES
reaction rate: k1*S*E (binding)

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

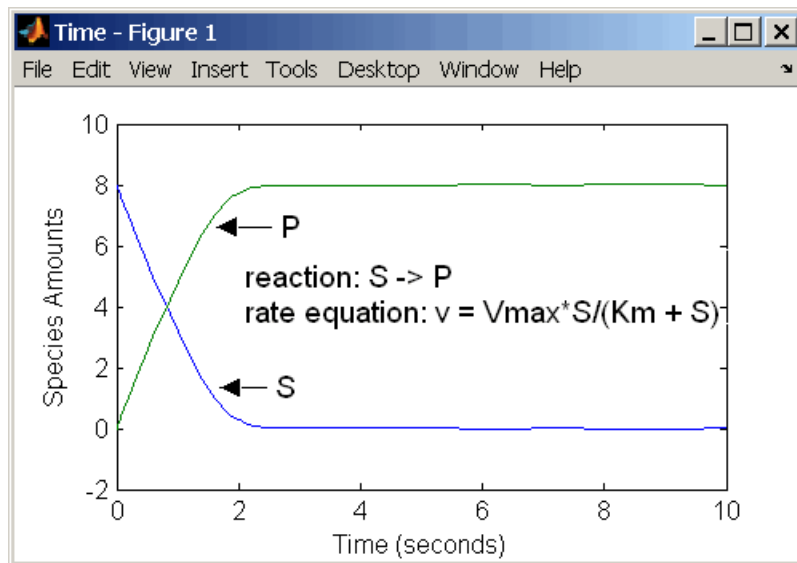
Representing an enzyme-catalyzed reaction with mass action kinetics requires you to know the rate constants k_1 , k_{1r} , and k_2 . However, these rate constants are rarely reported in the literature. It is more common to give the rate constants for Henri-Michaelis-Menten kinetics with the maximum velocity $v_m = k_2 \cdot E$ and the constant $K_m = (k_{1r} + k_2) / k_1$. 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 1-10.

$$v = \frac{V_{\max}[S]}{K_m + [S]}$$

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.



Evaluation of Reaction Rate

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 $a + b \rightarrow c$. Using mass action kinetics, the reaction rate is $k \cdot a \cdot b$, 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 $1/(\text{molarity} \cdot \text{second})$. If you specify k with another equivalent unit definition, for example, $1/((\text{moles/liter}) \cdot \text{second})$, 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 ($1/(\text{mole} \cdot \text{second})$ for `concentration/time`, or $1/(\text{molarity} \cdot \text{second})$ for `amount/time`).

Using Constant Amount and Boundary Condition for Species

In this section...

“Definition of Constant and Boundary Properties” on page 1-17

“Changing Species Amounts with Reactions or Rules ” on page 1-18

“Keeping Species Amount Unchanged” on page 1-18

“Constant = NO, Boundary = YES” on page 1-19

“Constant = YES, Boundary = YES” on page 1-20

“Model Edges” on page 1-21

Definition of Constant and Boundary Properties

There are two properties (constant amount, boundary condition) to specify how the amount of a species changes or does not change during a simulation. Based on the conditions of your model you can decide how to use these properties.

The SBML specification (Level 2, Version 1) added the property `BoundaryCondition` to the model definition.

Species with `BoundaryCondition = Yes` — The species amount is either constant or determined by a rule, but in either case the amount is 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.

Species with `ConstantAmount = No` — The species amount is determined by a reaction or a rule, but not both.

Species with `ConstantAmount = Yes` — The species amount does not change during a simulation. The species can be in a reaction or rule, but it cannot have a rule that changes its amount.

Changing Species Amounts with Reactions or Rules

Set Constant = NO, Boundary = NO

The value of a species can change, and it can change with either a reaction or rule, but not both

Constant	Boundary	Reaction	Rule	Changed By
NO	NO	YES	NO	Reaction
NO	NO	NO	YES	Rule

Example 1 — Species **A** is in 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 in 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 in a reaction, and it is not in a rate equation. **G** varies with an algebraic rule or rate rule.

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

Keeping Species Amount Unchanged

Set Constant = YES, Boundary = NO

The value of a species cannot change. When a species has its `ConstantValue` selected and `BoundaryCondition` not selected, it acts like a parameter. It cannot be in a reaction and it cannot be varied by a rule.

Constant	Boundary	Reaction	Rule	Changed By
YES	NO	NO	NO	Never

Example — Species **E** is not in the reaction, but it is in the reaction rate equation. **E** is constant and could be replaced with the constant $V_m = k_2 * E$.

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

Constant = NO, Boundary = YES

The value of a species can change, and it is in a reaction, but a differential rate term from the reaction is not created. The value of the species change with a rule and a differential rate term is created from the rule.

Constant	Boundary	Reaction	Rule	Changed By
NO	YES	YES	YES	Rule

From the SBML specification (Level 2, Version 1), “By default, when a species is a product or reactant of one or more reactions, its concentration is determined by those reactions. In SBML, it is possible to indicate that a given species’ concentration is not determined by the set of reactions even when that species occurs as a product or reactant; i.e., the species is on the boundary of the reaction system but is a component of the rest of the model.”

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)
```

Constant = YES, Boundary = YES

The value of the species cannot change. It is in a reaction, but a differential rate term is not created from the reaction. The differential rate term is created from a rule.

Constant	Boundary	Reaction	Rule	Changed By
YES	YES	YES	NO	Never

During simulation, a differential rate equation is not created for the species. $d\text{Species}/dt$ 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 $k \cdot A$ 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)
```


Model Edges

As you build complex models from simpler pathways, there are edges in the model that you need to define before simulating the model. Knowing where the model edges are located is important because a species that is initially constant or unregulated can later vary as you add details to your model. The concept of a model edge overlaps with SBML boundaries, but not always.

Model edge — Species with constant amounts that might or might not be modeled in the reaction and reaction rate equations. Examples are cofactors, NAD^+ , ATP, and DNA.

Model edge — Enzymes with constant amounts that are not regulated. For example, a Michaelis-Menten rate equation with V_{\max} specified as a parameter assumes that the amount of enzyme catalyzing the reaction remains constant.

$$v = \frac{V_{\max} * [\text{Substrate}]}{K_m + [\text{Substrate}]}$$

You may want to temporarily model a regulated enzyme in a rate equation. If the amount of enzyme is constant, then this species is a model edge. After adding the reaction(s) that change the amount of the enzyme,

$$v = \frac{k * [\text{Enzyme}] * [\text{Substrate}]}{K_m + [\text{Substrate}]}$$

Model edge — Null or source species that synthesizes another species at a constant rate (zero order reaction). Mass is added to the system.

Model edge — Degradation of a species to a null or sink species (first-order reaction). Mass is taken away from the system.

Scoping Parameters for Reactions, Rules, and Events

In this section...

“Definition of Parameter Scope” on page 1-22

“Using a Parameter in Events and Rules” on page 1-23

“Changing the Scope of a Parameter” on page 1-23

Definition of Parameter Scope

Parameters are quantities used to define the behavior of a modeled system. Parameters can either be constant or change over time. SimBiology parameters are generally used to define rate constants.

A SimBiology parameter is defined either globally at the model level or locally at the kinetic law level. *Scope* refers to this definition of the parameter at the model or kinetic law level.

- If the scope of the parameter is global in the model, it can be used by any event or rule, or by any reaction rate expression in the model.
- If the scope of the parameter is at the kinetic law level, it can be used only by the reaction rate expression for which it was defined.

If you create a new parameter in the **Project Settings-Parameters** pane, the scope is set by default to the `model`. When you create a new parameter to define a reaction rate equation in the **Project Settings-Reactions** pane’s **Kinetic Law** tab, you can choose whether to assign the parameter locally to the kinetic law or globally to the model.

SimBiology parameters are resolved hierarchically:

- For reaction rate, the software hierarchically uses the value of the parameter at the kinetic law level first. If no such parameter is at the kinetic law level, the software looks for the parameter at the model level.
- If two parameters have the same name, one at the model level and the other at the kinetic law level, the software uses the value of the parameter at the kinetic law level for the reaction rate. The software uses the value

of the parameter at the model level for any rules or events that reference the parameter.

Note If you want to vary a parameter that is being referenced in a reaction rate equation, that parameter must have a unique name, and have scope at the model level.

Using a Parameter in Events and Rules

When you want to refer to a parameter in an event or rule expression, or in more than one reaction rate equation, the parameter scope must be at the model level.

If you want to vary a parameter that is being referenced in a reaction rate equation, that parameter must have a unique name, and have scope at the model level. See “Definition of Parameter Scope” on page 1-22 for more information.

Note To vary a parameter with a rule or an event, clear the **ConstantValue** check box in the **Project Settings-Parameters** pane, **Settings** tab.

Changing the Scope of a Parameter

When you want to refer to a parameter in an expression for a rule, or in more than one reaction rate equation, the parameter scope must be at the model level.

To change the scope of a parameter in the SimBiology desktop:

- 1** In the **Project Explorer**, click **Parameters**, to open the **Parameters** pane.
- 2** In the **Parameters** table, right-click a parameter row select **Change Parameter Scope** to change the scope of the selected parameter from kinetic law to model, or the reverse.

Changing Model Component Values Using Rules

In this section...

“What Is a Rule?” on page 1-24

“What Is an initialAssignment Rule?” on page 1-25

“What Is a repeatedAssignment Rule?” on page 1-25

“What Is an Algebraic Rule?” on page 1-26

“What Is a Rate Rule?” on page 1-27

“Typical Uses of Rate Rules” on page 1-27

What Is a Rule?

A *rule* is a model component that specifies a mathematical relationship between model component values during a simulation. Rules let you specify mathematical relationships involving the following model components values:

- parameter value
- species amount
- compartment capacity

Examples of using a rule include:

- 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.

The types of rules in SimBiology are as follows:

- `initialAssignment` — Lets you specify the initial value of a parameter, species, or compartment capacity, as a function of other model component values in the model.
- `repeatedAssignment` — Lets you specify a value that holds at all times during simulation, and is a function of other model component values in the model.
- `algebraic` — Lets you specify mathematical constraints on one or more parameters, species, or compartments that must hold during a simulation.
- `rate` — Lets you specify the time derivative of a parameter value, species amount, or compartment capacity.

What Is an `initialAssignment` Rule?

`initialAssignment` rules are evaluated once at the beginning of a simulation. `initialAssignment` rules are expressed as `Variable = Expression`.

For example, use the rule to set the initial amount of `species1` to be proportional to `species2`. This lets you change the initial conditions for both species by updating the value for one.

```
species1 = k*species2
```

What Is a `repeatedAssignment` Rule?

`repeatedAssignment` rules are evaluated at every time step during a simulation. `repeatedAssignment` rules are 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`), use the rule to set the capacity of `cytoplasm` to be proportional to `x`.

```
cytoplasm = k*x
```

Where `k` is a specified constant.

`repeatedAssignment` 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 configuration set used while performing simulations.

Tip

- If you can solve for the variable, use a `repeatedAssignment` rule instead of an algebraic rule.
 - In `repeatedAssignment` 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 1-26 in the SimBiology User’s Guide).
-

What Is an Algebraic Rule?

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 configuration set used while performing simulations.

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,

$$x \cdot \log(x) - 3$$

Considerations When Imposing Constraints

Consider the mathematical constraint $y = m \cdot x - c$. In the software this rule is written as $m \cdot x - c - y$. 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.

What Is 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

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

Typical Uses of 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 .

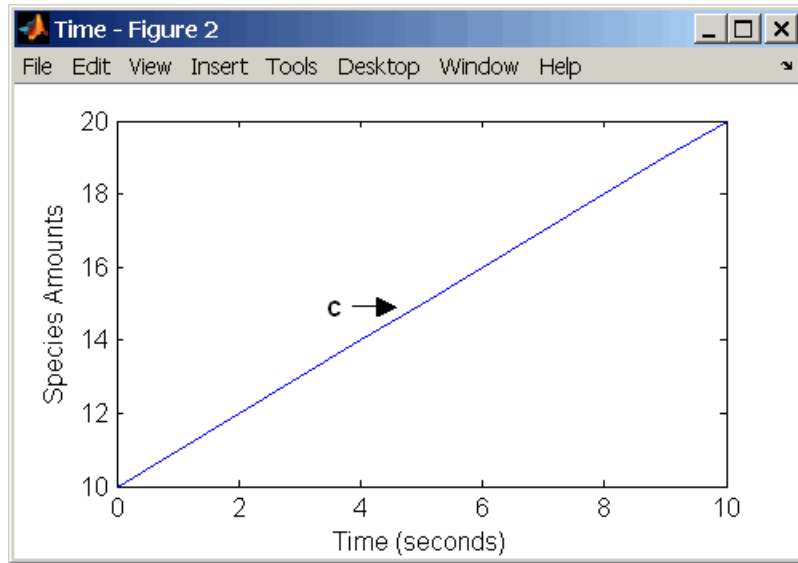
```

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 $c = kt + c_0$, where c_0 is the initial amount or concentration of the species c .

Enter the rule described above as $c = k$. From the **RuleType** list, select **rate**, enter the values for c and k , and then simulate.



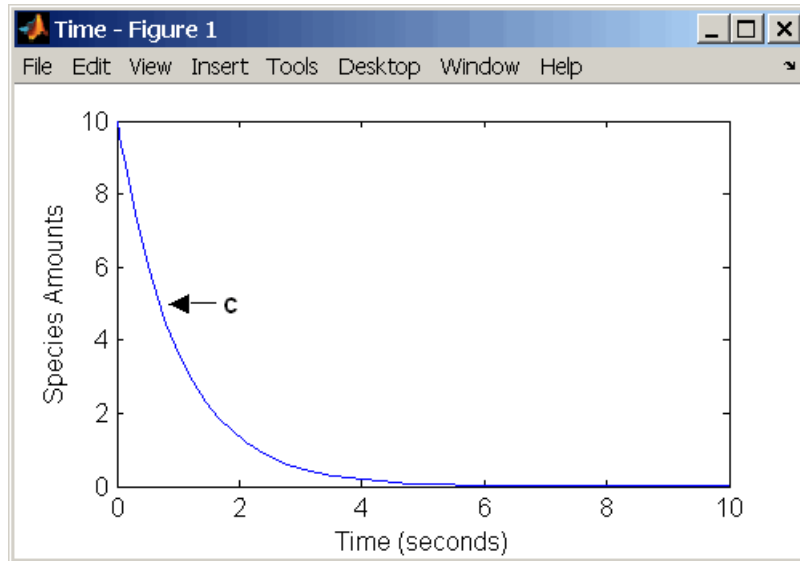
Alternatively, you could model a constant increase in a species using Mass Action reaction $\text{null} \rightarrow C$.

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 $dc/dt = -k \cdot c$ 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 $a \rightarrow c$ and a rate rule $dc/dt = k \cdot c$, select the `BoundaryCondition` 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 boundary condition is not selected, you will get the following error message:

```
Invalid rule variable 'in a reaction or another rule'.
```

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

```

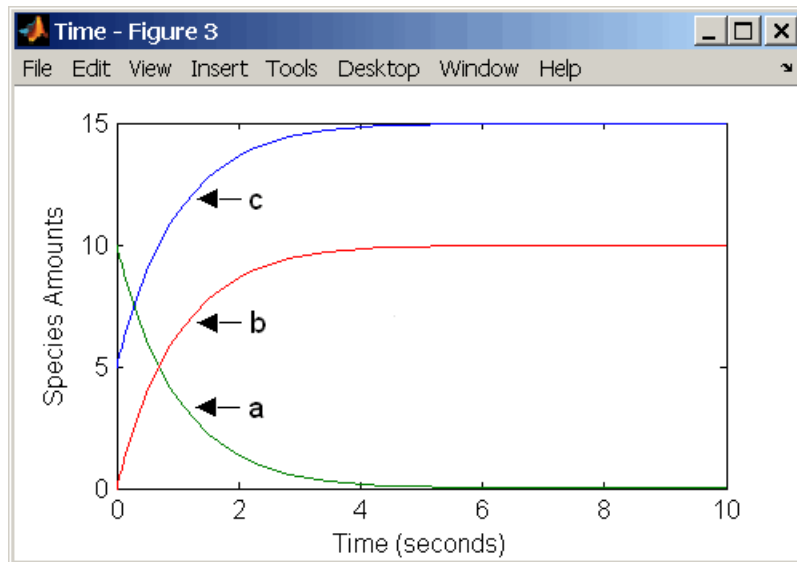
The solution for the species in the reaction are:

$$a = a_0 e^{-k_1 t} \quad \text{and} \quad b = a_0 (1 - e^{-k_1 t})$$

With the rate rule $dc/dt = k_2 * a$ dependent on the reaction, $dc/dt = k_2 (a_0 e^{-k_1 t})$, and the solution is:

$$c = c_0 + k_2 a_0 / k_1 (1 - e^{-k_1 t})$$

Enter the reaction and rule described above and simulate.



Expressing 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:

$$C = v_i - (v_d * X * C) / (K_c + C) - k_d * C$$

Changing Model Component Values Using Events

In this section...

“What Is an Event?” on page 1-32

“How Events Are Evaluated” on page 1-33

“Evaluation of Simultaneous Events” on page 1-38

“Evaluation of Multiple Event Functions” on page 1-39

“When One Event Triggers Another Event” on page 1-39

“Cyclical Events” on page 1-40

What Is an Event?

Events are used to describe sudden changes in model behavior. An event lets you specify discrete transitions in model component values that occur when a user-specified condition becomes true. You can specify that the event occurs at a particular time, or specify a time-independent condition.

For example, you can use events to activate or deactivate certain species (activator or inhibitor species), change parameter values based on external signals, or change reaction rates in response to addition or removal of species. You can also use an event in a model when you want to replicate an experimental condition, for example, to replicate the addition or removal of an activating agent (such as a drug) to a sample. This section describes events and how they are evaluated.

Use events to define events that occur when a condition becomes true. When you specify a condition in the **Trigger** you are specifying that the event should be executed when the condition becomes true. Typical triggers are:

- Cause an event to occur at 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.
- Cause an event to occur 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.

The event that is executed when the `Trigger` becomes true is called an event function (`EventFcn`). Event functions could 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.

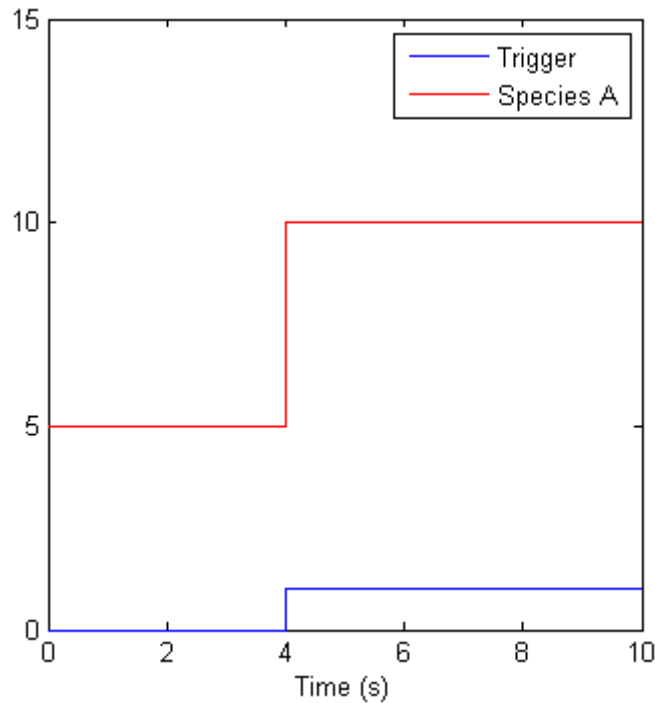
To simulate SimBiology models containing events, use the deterministic `sundials` solver or the stochastic `ssa` solver; other solvers do not support events. See “Sundials Solvers” on page 2-17 and “Stochastic Solvers” on page 2-12 for more information.

Procedures

- For an example using events in the SimBiology desktop, see “Desktop Example — Changing Species Amounts Using an Event” on page 1-41 in the *SimBiology User’s Guide*
 - For an example of using events at the command line, see `addevent` in the *SimBiology Reference*
-

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 the trigger becomes true and the event is executed. 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 10. In theory, with a perfect solver, the event would be executed exactly at time = 4.00. 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`.

Trigger	EventFcn
<code>time >= 4</code>	<code>A = 10</code>

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 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.

Specifying Event Triggers

A Trigger is a condition that must become true for an event to be executed. Typically, the condition uses a combination of relational and logical operators to build a trigger expression.

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

Trigger	Explanation
'((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.

For more information on triggers see `Trigger` in the SimBiology Reference Guide.

Specifying Event Functions

The event that is executed when a `Trigger` condition has a rising edge is called an event function (`EventFcn`). You can use an event function to change the value of a species or a parameter, or you can specify complex tasks by calling an M-file containing 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. For more information see also `EventFcns` in the SimBiology Reference Guide. Some examples of event functions include:

EventFcn	Explanation
'speciesA = speciesB'	When the event is executed set the amount of <code>speciesA</code> equal to that of <code>speciesB</code> .
'k = k/2'	When the event is executed halve the value of the rate constant <code>k</code> .

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

Evaluation of Simultaneous Events

When two or more trigger conditions simultaneously become true, the solver executes the events in the order in which they are on the model. You can reorder events using the `reorder` method at the command-line. Alternatively, in the SimBiology desktop, arrange the rows of events in the order you desire, then right-click and select **Reorder Events as Shown in Table**. 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	<code>SpeciesA >= 4</code>	<code>SpeciesC = 10</code>
2	<code>time >= 15</code>	<code>SpeciesB = SpeciesC</code>

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 stores the value of `SpeciesC` at the rising edge, before any event functions are executed and uses this stored value to assign `SpeciesB` its value. In the above example if `SpeciesC = 15` when the events are triggered, after the events are executed `SpeciesB = 15`, and `SpeciesC = 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
<code>time >= 4</code>	<code>{'SpeciesA = 10, SpeciesB = SpeciesA'}</code>

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'}

Desktop Example – Changing Species Amounts Using an Event

In this section...

“Overview” on page 1-41

“Prerequisites” on page 1-42

“Adding an Event to the Example Model” on page 1-42

“Simulating the Modified Model” on page 1-44

Overview

This example shows you how to add an event to a model to trigger a time-based change. For information on events and how they are evaluated see “Changing Model Component Values Using Events” on page 1-32.

This example shows you how to add an event that modifies amount of ligand (L), thus modeling a delay in the addition of α -factor to the cell culture.

About the Example Model

This example uses the model from “Modeling a G Protein Cycle” in the SimBiology Model Reference documentation.

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	Rate Parameters
1	Receptor-ligand interaction	$L + R \leftrightarrow RL$	k_{RL}, k_{RLm}
2	Heterotrimeric G protein formation	$G_d + G_{bg} \rightarrow G$	k_{G1}
3	G protein activation	$RL + G \rightarrow Ga + G_{bg} + RL$	k_{Ga}

No.	Name	Reaction	Rate Parameters
4	Receptor synthesis and degradation	$R \leftrightarrow \text{null}$	kRdo, kRs
5	Receptor-ligand degradation	$RL \rightarrow \text{null}$	kRD1
6	G protein inactivation	$Ga \rightarrow Gd$	kGd

Prerequisites

Opening and Saving the Example Model

- 1 Load the example project by typing the following at the command line:

```
sbioloadproject gprotein
```

The model is stored in a variable called m1.

- 2 Open the SimBiology desktop with the model loaded by typing:

```
simbiology(m1)
```

The desktop opens with **Model Session-Heterotrimeric_G_Protein_wt**.

- 3 Select **File > Save Project As**. The Save SimBiology Project dialog box opens.
- 4 Specify a name (for example, gprotein_ex) and location for your project, and click **Save**.

Adding an Event to the Example Model

- 1 In the **Project Explorer** expand **SimBiology Model** and click **Events** to open the **Events** pane.
- 2 In the **Enter Trigger** box, type the following expression and press **Enter**:

```
time >= 100
```

- 3** In the **EventFcns** box, type the following expression and press **Enter**:

$$L = 6.022E17$$

In the **Settings** tab, note that the species L is available.

- 4** In the row containing the species L, double-click the **InitialAmount** column, type 0, and then press **Enter**.

The **InitialAmount** of L is set to be 0.0 when the simulation starts.

The Events pane should now resemble the following:

Enter Trigger: Add Delete


	Name	Trigger	EventFcns
1		time >= 100	L = 6.022E17

Settings | **Description**

Active (Select if the event can be generated during the simulation.)

Name:

Trigger:

EventFcns: 

Parameters being used by Event:

Name	Scope	Value	ValueUnits

Species being used by Event:

	Name	Scope	InitialAmount	InitialAmountUnits
1	L	unnamed	0.0	<input type="text"/>

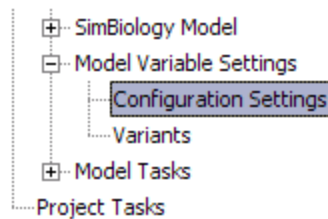
Compartments being used by Event:

Name	Owner	Capacity	CapacityUnits

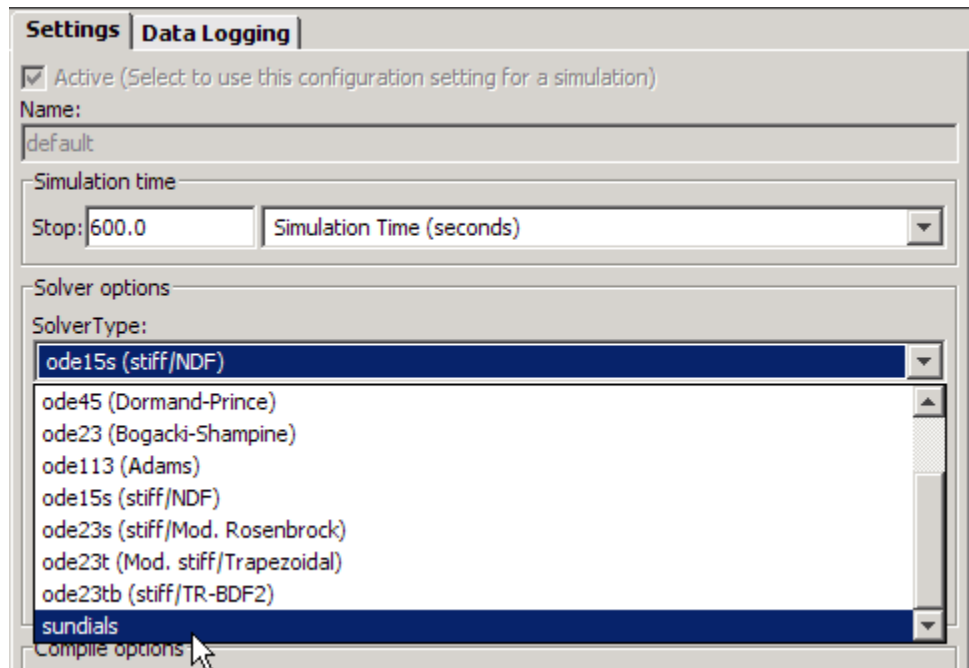
5 Save the project by selecting **File > Save Project**.

Simulating the Modified Model

1 In the **Project Explorer**, expand **Model Variable Settings** and click **Configuration Settings** to open the pane that contains solver settings.

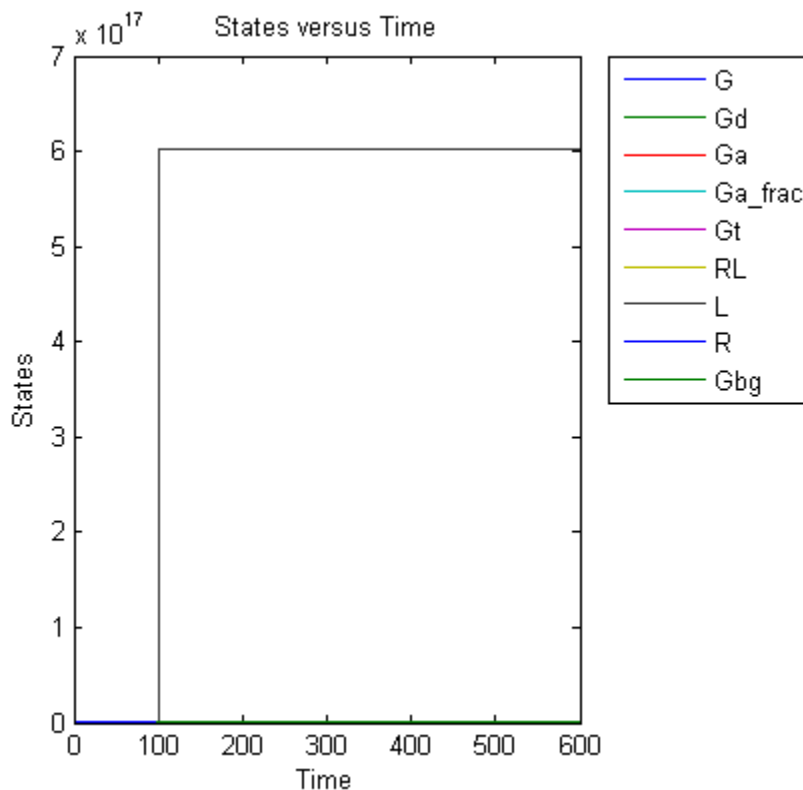


2 In the **Settings** tab, from the **SolverType** list, select **sundials**. This solver lets you simulate models with events.




- 3** In the **Project Explorer**, right-click **Model Session-Heterotrimeric_G_Protein_wt** and select **Run Simulation**.

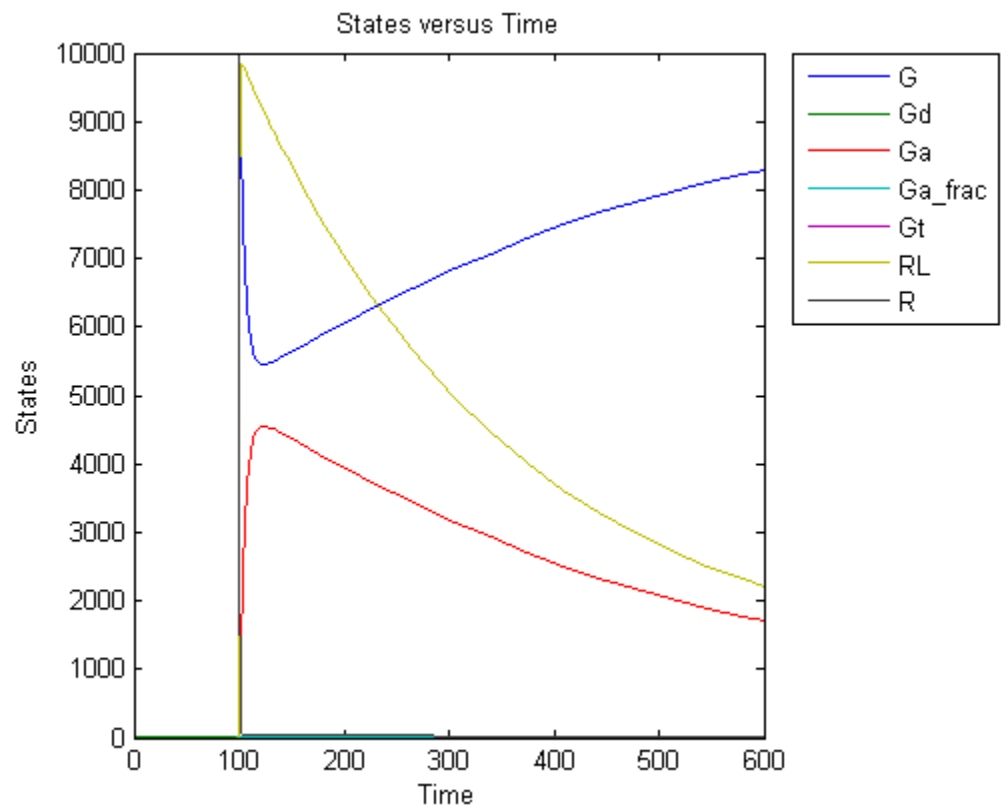
The simulation runs to completion and plots the result in a figure. Notice that the plot shows that the ligand amount increases when the event is executed.



The plot does not show the species of interest due to a wide range in species amounts. Follow the next steps to view the species of interest.

- 4** In the **Project Explorer**, under **Simulation**, right-click **Data** and select **Save Data**. The Save Data dialog box opens.

- 5** Specify a name for the saved data, for example, `event_ex`, and click **Save**. The **Project Explorer** shows a new item with the saved data name under **Simulation**.
- 6** In the **Project Explorer**, click the saved data, for example, `event_ex`, to open the **Data** pane for the saved data.
- 7** Click the **Plots** tab.
- 8** In the **Plot Type** box, select **Time** and click **Add Plot Type**.
- 9** Select the new plot (second row), and in the **Arguments** section, click . The Select Values for y dialog box opens.
- 10** Clear the check boxes for the species **L** and **Gbg**.
- 11** Click **OK**.
- 12** (Optional) Clear the **Create Plot** check box for the first plot.
- 13** Click **Plot**. Your plot should resemble the one below. Notice the increase in activation of G protein (species **Ga**, shown in red) after ligand (**L**) is added at `time = 100` (simulation time).



Storing and Applying Alternate Model Values Using Variants

In this section...

“What Are Variants?” on page 1-48

“Using Variants” on page 1-49

“Applying Multiple Variants in a Model” on page 1-50

What Are Variants?

Variants allow you to apply alternate values to model components to represent values for the components. The alternate values apply only during a simulation and do not otherwise alter the model’s values. Some examples for use of variants are:

- Store and apply parameter values for yeast and mouse models using two separate variants.
- Store parameter estimates from different experimental conditions in several variants and apply them to a model.
- Store component values for mutant strains and apply the values to a model representing the wild-type.

Simulating using a variant does not alter the model component values permanently. The values specified in the variant temporarily apply during simulation. You can store values for:

- Species InitialAmount
- Parameter Value
- Compartment Capacity

Using one or more variants associated with a model allows you to evaluate model behavior during simulation, with different values for the various model components without having to search and replace these values, or having to create additional models with these values. If you determine that the values

in a variant accurately define your model, you can permanently replace the values in your model with the values stored in the variant object.

Using Variants

Creating and Using Variants at the Command Line

- 1** Create the variant and add it to the model using the `addvariant` method.
- 2** (Optional) Set the `Active` property to true if you always want the variant to be applied before simulating the model.
- 3** Enter the variant object as an argument to `sbiosimulate`. This applies the variant only for the current simulation and supersedes any active variant objects on the model.

If you followed step 2, simply calling `sbiosimulate` on the model object applies the variant.

For more information, see the following sources of information in the *SimBiology Reference*:

For information about...	See...
Variant object methods and properties	Variant object
Example of creating and adding a variant	<code>addvariant</code>
Appending contents to variants	<code>addcontent</code>
Replacing model values permanently with values from a variant	<code>commit</code>

Creating and Using Variants in the SimBiology Desktop

For information about creating variants in the SimBiology desktop and applying them during a simulation, see the following:

- The context-sensitive help (**SimBiology Desktop Help**) in the SimBiology desktop for procedures. To access this help:

- 1** In the SimBiology desktop, from the **Project Explorer** expand **Model Variable Settings** and select **Variants**.

The desktop opens the **Variants** pane. The **SimBiology Desktop Help** updates to show you how to work with variants.

- 2** If the context-sensitive help is not open, select **Help > SimBiology Desktop Help**.

- “Desktop Example — Applying Changes to Parameter Value Using a Variant” on page 1-51

Applying Multiple Variants in a Model

When multiple variants are used 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 (at the command line) or the last variant created.

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.

Desktop Example — Applying Changes to Parameter Value Using a Variant

In this section...

“Overview” on page 1-51

“Prerequisites” on page 1-52

“Applying Alternate Values Using Variants” on page 1-53

“Simulation Results for the Model of the Mutant Strain” on page 1-53

Overview

For a description of variants, see “Storing and Applying Alternate Model Values Using Variants” on page 1-48.

About the Example Model

This example uses the model from “Modeling a G Protein Cycle” in the SimBiology Model Reference documentation.

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	Rate Parameters
1	Receptor-ligand interaction	$L + R \leftrightarrow RL$	k_{RL}, k_{RLm}
2	Heterotrimeric G protein formation	$Gd + Gbg \rightarrow G$	k_{G1}
3	G protein activation	$RL + G \rightarrow Ga + Gbg + RL$	k_{Ga}
4	Receptor synthesis and degradation	$R \leftrightarrow \text{null}$	k_{Rdo}, k_{Rs}

No.	Name	Reaction	Rate Parameters
5	Receptor-ligand degradation	RL -> null	kRD1
6	G protein inactivation	Ga -> Gd	kGd

About the Variant Created in This Example

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

The value of the parameter `kGd` is 0.11 for the wild-type and 0.04 for the mutant. To represent the mutant, the alternate value of the parameter `kGd` is stored as 0.004 in a variant and applied during simulation.

Prerequisites

Opening and Saving the Example Model

- 1 Load the example project by typing the following at the command line:

```
sbioloadproject gprotein
```

The model is stored in a variable called `m1`.

- 2 Open the SimBiology desktop with the model loaded by typing:

```
simbiology(m1)
```

The desktop opens with **Model Session-Heterotrimeric_G_Protein_wt**.

- 3 Select **File > Save Project As**. The Save SimBiology Project dialog box opens.
- 4 Specify a name (for example, `gprotein_ex`) and location for your project, and click **Save**.

Applying Alternate Values Using Variants


- 1 In the **Project Explorer**, under **Model Session-Heterotrimeric_G_Protein_wt**, expand **Model Variable Settings**, and click **Variants** to open the **Variants** pane.

For this example, ignore the preexisting variant that appears in the table.

- 2 In the **Enter Name** box, type a name for the variant, and then click **Add** or press **Enter**. For example:

```
mut_value_ex
```

- 3 Add content to the variant:

- a In the **Settings** tab, click  (Add table row). The table updates with a row containing information about a model component.
- b From the **Type** list, select **parameter**. The **Property** list updates to show the property available for changing.
- c In the **Name** cell, type the name of the component.

```
Gprotein Inactivation.kGd
```

The parameter kGd is at the kinetic law level, and not the model level. Thus, you must specify the parameter in the format *ReactionName.ParameterName*.

- d In the **Value** cell, type a value to apply using the variant.

```
0.004
```

See Also

“Storing and Applying Alternate Model Values Using Variants” on page 1-48 in the *SimBiology User’s Guide*.

Simulation Results for the Model of the Mutant Strain

To simulate the model of the mutant strain, apply the variant and simulate as follows:

- 1** If the **Simulation** pane is not already open, in the **Project Explorer** , in **Model Session-Heterotrimeric_G_Protein_wt**, under **Model Tasks**, select **Simulation** to open the pane.

Before simulating, set options to remove species L during plotting because large amounts of L are present in the model and this affects the scaling of the plot.


- 2** Click the **Plots** tab.

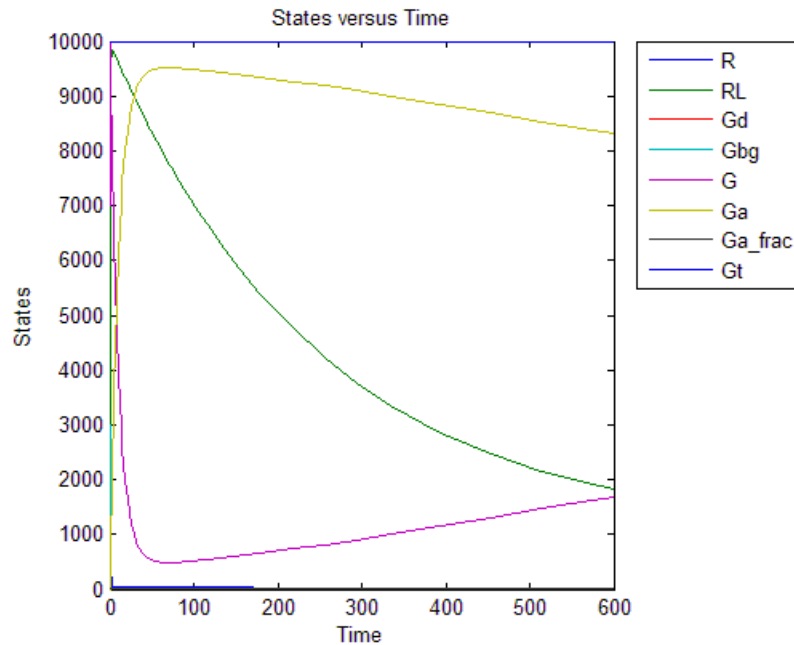
- 3** Under **Arguments**, click . The Select Values for y dialog box opens.

- 4** Clear the check box for L (unnamed.L) and click **OK**.

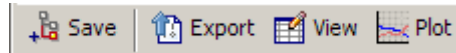
- 5** Click the **Simulation Settings** tab.

- 6** In the **Variants** table, select the **Use in Task** check box for `mut_value_ex`.

- 7** Click  (**Run**). Your plot should resemble the following figure.



- 8** In the **Data** pane for the latest simulation, click **Save** to open the Save Data dialog box.



- 9** Specify a name for your data, and then click **Save**.

mut_model_run1

The desktop adds the saved data under the **Simulation** task in the **Project Explorer**.

The simulation results for the wild-type strain are described in “Simulation Results for the Wild-Type Strain Model”.

Verifying that the Model Has No Warnings or Errors

In this section...

“Verifying the Model in the SimBiology Desktop” on page 1-56

“Verifying the Model at the Command Line” on page 1-58

The SimBiology desktop and MATLAB command line have 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.

You can check your model for errors and warnings at any time in your workflow of building or working with your model, this check includes dimensional analysis or unit conversion issues. For example, verify your model during construction to ensure that the model is always complete, or verify the model or configuration set after changing simulation settings or settings for dimensional analysis or unit conversion.

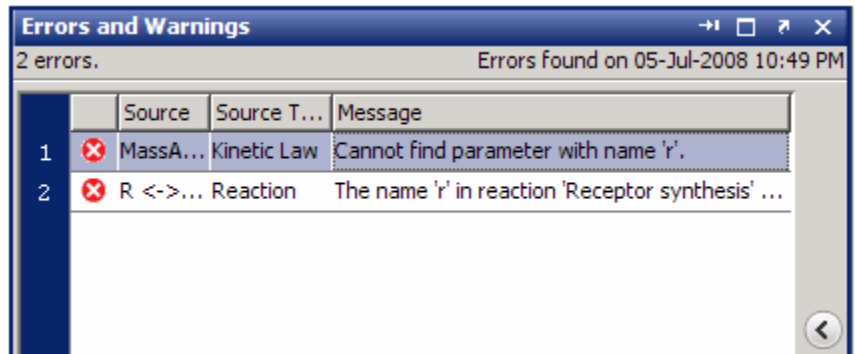
Analysis such as simulation, scanning, and parameter fitting also causes a model to be verified before the task runs. Repeatedly running a task using different variants or setting different values for the `InitialAmount` of species, the `Capacity` of compartments, and the `Value` of parameters, generates warnings only the first time. Use the verification functionality described in this section to display warnings again.


Verifying the Model in the SimBiology Desktop

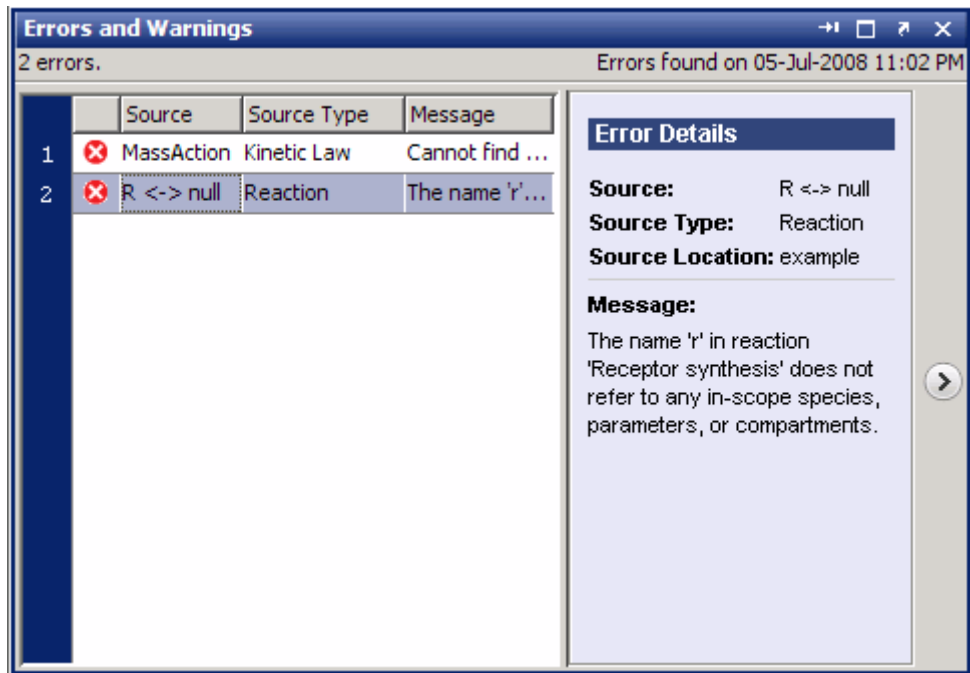
While you are building your model in the SimBiology desktop, you can click



at any time to generate a list of any errors and warnings in the model. The errors and warnings appear in the **Errors and Warnings** pane. The following is an example of the error generated when the reaction rate of a reaction is set to a parameter that you have not defined.



To see details of the error or warning, click  to expand the pane showing the details.



Double-click the error row to move to the location of the error.

You can interrupt and stop the verification process by clicking **Stop** at any time during verification.

Troubleshooting Using Colored Indicators

Model component tables and the items in the **Project Explorer** have colored indicators that let you see if the specified properties are valid. The error or warning icons in the **Project Explorer** allow you to quickly assess which model components have errors or warnings associated with them.

You can turn off these indicators by selecting **File > Preferences** to open the SimBiology Preferences dialog box. Select **Explorers** to find this preference.

In the tables, move the mouse over the indicator to see more details about the warning or error. This table shows the indicators and their significance.

Colored Indicator	Description
Green	Components have valid properties.
Yellow	There are warnings in the associated rows.
Red	There are errors in the associated rows.

- Click **Verify** in the **Toolbar** to see if any errors are listed in the **Errors and Warnings** pane. You must fix any reported errors before you can simulate a model. Warnings may not need to be resolved before simulating, but might result in unexpected answers.
- Click the colored square, if present, in the **Settings** tab to open a dialog box with tips on how to fix the warning or error.

Verifying the Model at the Command Line

Use the `verify` method to see a list of warnings and errors in your model. This method checks a model or a configuration set, depending on the argument you give to the method, and returns a list of warnings and errors encountered.

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

Desktop Example — Using User-Defined Functions in Expressions

In this section...

“Prerequisites” on page 1-59

“Overview” on page 1-59

“Creating an M-File Function” on page 1-61

“Calling the Function in a Rule Expression” on page 1-62

“See Also” on page 1-67

Prerequisites

To work through the example, these sections assume you have a working knowledge of the following:

- MATLAB desktop
- Creating and saving M-files
- SimBiology desktop

Overview

You can use custom defined functions in reaction rate, rule, and event expressions. When you use a function in a SimBiology expression, the solver calls the function every time the expression is evaluated during simulation.

Therefore, if you have complex reaction rate expressions you can define the reaction rate with a function.

Requirements For Using User-Defined Functions in SimBiology Expressions

- Create an M-file function. To find out more about M-file functions, see function in *MATLAB Function Reference*. For an example of a function to be used in a SimBiology model, see “Creating an M-File Function” on page 1-61 in this topic.

- Change the current folder to the folder containing your M-file using the `cd` command or use the Current Folder field in the MATLAB desktop toolbar. Alternatively, add the path to the folder containing your M-file using `addpath` or see “Adding Folders to the Search Path” in the *MATLAB Desktop Tools and Development Environment* documentation.
- Call the function in a SimBiology reaction, rule, or event expression.

The requirements do not have to be executed in any specific order, but you might find it more convenient to start with the first two items before calling the function from within the SimBiology expression. This is because colored cues for model verification in the SimBiology desktop will show the expression as invalid if the function has not yet been defined, or is not on the path or the working directory.

The example below shows how to create and call user-defined functions in SimBiology expressions. More specifically, the example shows how to use a user-defined function in a rule expression. You can use user-defined functions similarly in event functions (`EventFcns` property), event triggers (`Trigger` property) and in reaction rate expressions (`ReactionRate` property). For example, you can define a function that gives the reaction rate expression as the output, and use the function in a reaction rate expression.

About the Example Model

This example uses the model from “Modeling a G Protein Cycle” in the SimBiology Model Reference documentation.

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	Rate Parameters
1	Receptor-ligand interaction	$L + R \rightleftharpoons RL$	k_{RL}, k_{RLm}
2	Heterotrimeric G protein formation	$Gd + Gbg \rightarrow G$	k_{G1}
3	G protein activation	$RL + G \rightarrow Ga + Gbg + RL$	k_{Ga}

No.	Name	Reaction	Rate Parameters
4	Receptor synthesis and degradation	$R \leftrightarrow \text{null}$	kRdo, kRs
5	Receptor-ligand degradation	$RL \rightarrow \text{null}$	kRD1
6	G protein inactivation	$Ga \rightarrow Gd$	kGd

Assumptions

For the purpose of this example, assume that:

- An inhibitor (Inhib) slows the inactivation of the active G protein (reaction 6 above, $Ga \rightarrow Gd$).
- The variation in the amount of Inhib is defined in a function.
- The effect on the reaction is through a change in the rate parameter kGd.

Creating an M-File Function

1 In the MATLAB desktop, select **File > New > M-File**, to open a new M-file in 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 define the input arguments in a
% SimBiology rule expression.
% For example in the rule, define:
% 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 M-file (name the file `inhibvalex.m`) in a directory that you can access, or that is on the path.
- 4** If the location of the M-file is not on the path, change the working directory to the M-file location.

Calling the Function in a Rule Expression

- “Opening and Saving the Example Model” on page 3-68
- “Adding User-Defined Functions to the Example Model” on page 1-63
- “Defining a Rule to Affect Parameter Value” on page 1-64
- “Simulating the Modified Model” on page 1-65

Opening and Saving the Example Model

- 1** Load the example project by typing the following at the command line:

```
sbioloadproject gprotein
```

The model is stored in a variable called `m1`.

- 2** Open the SimBiology desktop with the model loaded by typing:

```
simbiology(m1)
```

The desktop opens with **Model Session-Heterotrimeric_G_Protein_wt**.

- 3** Select **File > Save Project As**. The Save SimBiology Project dialog box opens.
- 4** Specify a name (for example, `gprotein_ex`) and location for your project, and click **Save**.

Adding User-Defined Functions to the Example Model

The previously defined function `inhibvalex` in “Creating an M-File Function” on page 1-61 lets you specify how the inhibitor amount changes over time. This section shows you how to specify the input values for the function in a rule expression. As defined in the function, the output value is the amount of inhibitor.

Define a new rule to assign the inhibitor value.

1 In the **Project Explorer**, expand **SimBiology Model** and click **Rules** to open the **Rules** pane.

2 In the **Enter Rule** box, type the following expression and press **Enter**:

```
Inhib = inhibvalex(time, Cpo, Kel)
```

The Rule Variables dialog box opens for you to define the rule variables.

3 From the **Type** list, select **species** for `Inhib`, and parameter for `Cpo` and `Kel`. The SimBiology desktop creates the two species and the parameter.

Note If `inhibvalex` is on the list of undefined variables in the Rule Variables dialog box, this means that you have not yet put the M-file on the path or changed the working directory to the location of the M-file.

Leave `inhibvalex` undefined in the Rule Variables dialog box and click **OK**. In the MATLAB desktop, change the **Current Directory** field to the location of the M-file.

4 Click **OK**.

5 In the **Rules** pane from the **RuleType** list, select `repeatedAssignment`.

6 In the **Settings** tab, in **Parameters being used by Rule**, find the row containing the parameter `Kel`. Double-click the **Value** column, type `0.01`, and press **Enter**.

7 Find the row containing the parameter `Cpo`. Double-click the **Value** column, type `250` and press **Enter**.

Note You do not have to set a value for the species `Inhib` because it is being specified by a `repeatedAssignment` rule.

8 Save the project by selecting **File > Save Project**.

Defining a Rule to Affect Parameter Value

As described in “About the Example Model” on page 3-29 , the parameter `kGd` should be affected by the amount of inhibitor present in the system. Add a rule to describe this action, but first change the `Scope` and `ConstantValue` properties of the parameter `kGd` so that it can be varied by a rule.

Note Although the model has a previously defined parameter called `kGd`, this parameter’s scope is currently at the kinetic law level. The parameter must be scoped to the model for it to be varied by a rule or an event.

- 1** In the **Project Explorer**, expand **SimBiology Model** and click **Parameters** to open the **Parameters** pane.
- 2** Select the row containing the parameter `kGd`.
- 3** Right-click and select **Change Parameter Scope**. Notice that the **Scope** column now shows the model name for this parameter.
- 4** In the **Settings** tab, clear the **ConstantValue** check box for `kGd`, as this parameter is being varied by a rule.
- 5** In the **Project Explorer**, click **Rules** to open the **Rules** pane.
- 6** In the **Enter Rule** box, type the following expression and press **Enter**:

`kGd = 1/Inhib`


In the **Settings** tab you should see a green square indicating that the rule variables have been previously defined, and that there are no other warnings or errors associated with this rule.

- 7 For the new rule, in the **Rules** pane from the **RuleType** list, select **repeatedAssignment**.
- 8 Save the project by selecting **File > Save Project**.

Simulating the Modified Model


- 1 In the **Project Explorer**, right-click **SimBiology Model** and select **Run Simulation**.

The simulation runs to completion and plots the result in a figure. The plot does not show the species of interest due to a wide range in species amounts/concentrations. Follow the next steps to view the species of interest.

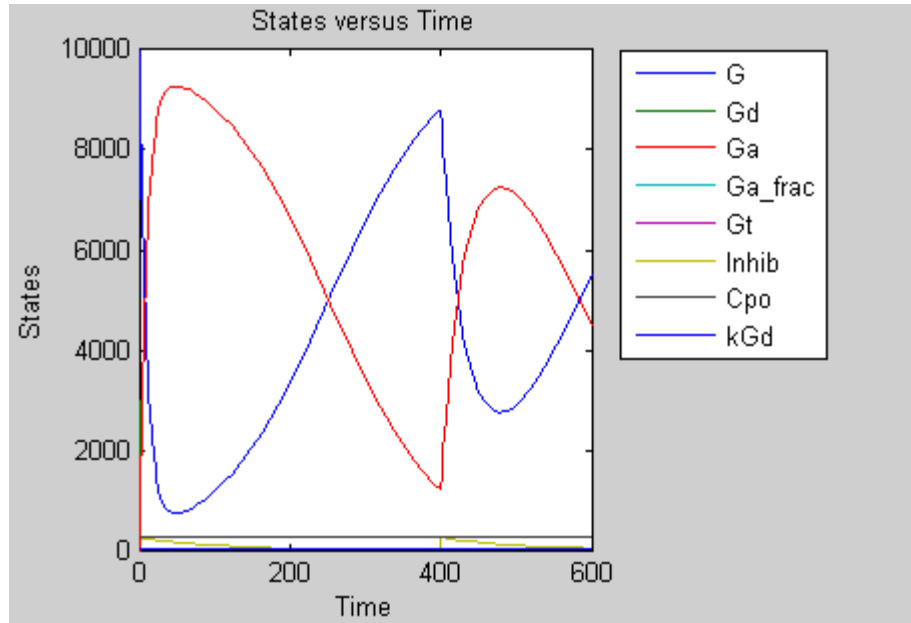
- 2 In the **Project Explorer**, under **Simulation**, click **Data** to open the **Data** pane for the most recent simulation run.
- 3 Click the **Plots** tab.
- 4 In the **Arguments** section, click . The Select Values for y dialog box opens.
- 5 Clear the check box for the following species:

RL
L
R
Gbg

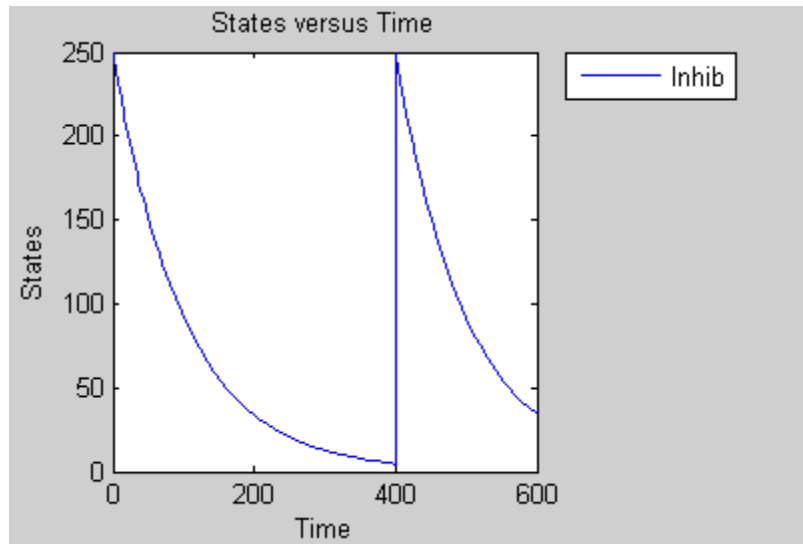
Note Species names are prefixed with the name of the compartment to which the species belongs. The default compartment is 'unnamed'.

- 6 In the **Plot Type** box, select **Time** and click **Add Plot Type**.
- 7 Select the row containing the new plot, and then in the **Arguments** section, click . The Select Values for y dialog box opens.

- 8 Click **Clear All**, and then select the check box for the species Inhib.
- 9 Click **OK**.
- 10 Click **Plot**. Your plots should resemble the following:



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



See Also

To learn about...	Refer to...
The SimBiology desktop	“Getting Started in the SimBiology Desktop” in the <i>SimBiology Getting Started Guide</i> .
M-file functions	<code>function</code> in the <i>MATLAB Function Reference</i> .
Changing the working directory to the directory containing your M-file	<code>cd</code> command in the <i>MATLAB Function Reference</i> .
Adding the directory containing your M-files to the MATLAB search path	<code>addpath</code> in the <i>MATLAB Function Reference</i> or “Adding Folders to the Search Path” in <i>MATLAB Desktop Tools and Development Environment</i> .

Importing and Exporting Model Component Data

In this section...

“Importing Model Component Data” on page 1-68

“Exporting Model Component Data” on page 1-69

Importing Model Component Data



You can import lists of species, reactions, parameters, and rules to and from the SimBiology desktop.

You can import the data from an Excel spreadsheet, or from a comma-separated or tab-separated text file using the **Load Model Components from File** menu item. The Excel[®] option is only supported on the Windows[®] platform.

- 1 From the **File** menu, point to **Load Model Components from File** and select the component type, for example, **Reaction**. The Load Reaction from File dialog box opens.

Note When there are multiple **Model Sessions**, components are imported into the **Active** model. To specify the **Active** model, select **Task > Active Model > *name of model***.

- 2 From the **File Type** list, select Excel, comma-separated text file, or tab-separated text file.
- 3 In the **File Name** box, enter a file path and name. Alternatively browse to select a file name and click **Open**.
- 4 If the first row in the file contains header information, select the **First row contains header information** check box.
- 5 If your model and the file have some identical names, clear the **Overwrite current property values** check box to preserve the values in the model.

- 6 Select the properties to import. There are required properties based on the component type. For example, **Reaction** is a required property. Specify column order using the  and  arrows. The first property selected corresponds to the first column in the Excel spreadsheet or text file.
- 7 Click **OK**. The data from your file is entered into the model.

Note

- If you have preexisting species in the model, the software appends nonidentical species names.
 - If you want a species to remain constant throughout a simulation, you can specify this using the Boolean operator **TRUE** in a column of the Excel file or line of text file (you do not have to specify the property name, **ConstantAmount**). While importing the data, the software will select the **ConstantAmount** check box for that species. The default is unchecked.
-



Exporting Model Component Data

You can export lists of species, reactions, parameters, and rules to and from the SimBiology desktop.

You can export data to an Excel spreadsheet, or to a comma-separated or tab-separated text file using the **Export Model Components to File** menu item. The Excel option is only supported on the Windows platform.

- 1 From the **File** menu, point to **Export Model Components to File** and select the component type, for example, **Reaction**. The Export Reactions to File dialog box opens.

Note When there are multiple **Model Sessions**, components of the **Active** model are exported. To specify the **Active** model, select **Task > Active Model > name of model**.

- 2** From the **File Type** list, select Excel, comma-separated text file, or tab-separated text file.
- 3** In the **File Name** box, enter a file path and name. Alternatively, if you browse to select a file name, select the file name and click **Save**.
- 4** If the first row in the generated file should contain the property names, select the **Write property names to first row in file** check box.
- 5** Select the properties to export. There are required properties based on the component type. For example, **Reaction** is a required property. Specify column order using the  and  arrows.

The first property selected corresponds to the first column in the Excel spreadsheet or text file.
- 6** Click **OK**. The data from your model is entered into the file.

Simulation

- “Performing Simulations” on page 2-2
- “About Simulation Solvers” on page 2-5
- “Nonstiff Deterministic Solvers” on page 2-8
- “Stiff Deterministic Solvers” on page 2-10
- “Stochastic Solvers” on page 2-12
- “Sundials Solvers” on page 2-17

Performing Simulations

In this section...
“Performing Simulations at the Command Line” on page 2-2
“Performing Simulations in the SimBiology Desktop” on page 2-2

Performing Simulations at the Command Line

The function `sbiosimulate` allows you to perform simulations at the command line.

At the command line, you can create a configuration set (`configset` object) that contains information about:

- The type of solver to use, associated tolerances, and maximum step size for deterministic (ODE) solvers
- The simulation stop time
- The options to use in sensitivity analysis

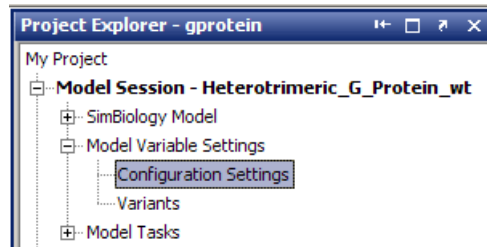
You can add many configuration sets to a model, each containing a different combination of simulation settings. To use a configuration set during a simulation you can either set the object’s `Active` property to true, or specify the object as an argument to `sbiosimulate`.

Performing Simulations in the SimBiology Desktop

The SimBiology integrated desktop environment provides convenient access to the configuration settings for simulations.

To access your configuration settings:

- On any model task pane, on the **Simulation Settings** tab, next to **Configuration Settings**, click **View**.
- Alternatively, in the **Project Explorer**, under the **Model Variable Settings** node, select **Configuration Settings**.



The **Configuration Settings** pane appears, where you can set, change, and save simulation parameters, configure data logging, and compile options.

Where to Find Configuration Settings Controls

Use the following controls on the **Configuration Settings** pane:

- Use the **Settings** tab to set the simulation solver and timing options for the currently selected model, and compile options for unit checking.
- Use the **Data Logging** tab to choose which species to log and how often.

Where to Find Simulation Controls

To set up and run your simulation, use the following controls on any model task pane:

- Use the **Simulation Settings** tab to select your configuration settings, or use the default. After you set up custom configuration sets, you can select them by name from the **Configuration Settings** list.

Note Use the **Run** button in the **Simulation Toolbar** to run the simulation with your currently selected configuration settings.

If you are using **Variants**, you can view them and apply them to tasks on the **Simulation Settings** tab.

- Use the **Export** tab to export simulation data to the MATLAB Workspace and/or to file every time you run a simulation.

- Use the **Plots** tab to configure what plots to generate when you run a simulation.

About Simulation Solvers

In this section...
“How Solvers Work” on page 2-5
“Stiff Versus Nonstiff Models” on page 2-5
“Selecting a Solver” on page 2-6

How Solvers Work

In order to simulate a model, the model is converted to a set of differential equations. The solver functions are used to compute solutions for those equations at different time intervals, giving the model's states and outputs over a span of time. You can then plot these outputs from your simulation.

The MATLAB ODE solvers are designed to handle *ordinary differential equations*. An ordinary differential equation contains one or more derivatives of a dependent variable y with respect to a single independent variable t , usually referred to as time.

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.

Stiff Versus Nonstiff Models

An ordinary differential equation problem is stiff if the solution being sought is varying slowly, but there are nearby solutions that vary rapidly, so the numerical method must take small steps to obtain satisfactory results. Many biological models are numerically stiff because they include species amounts that are changing quickly and others that change slowly.

Stiffness is an efficiency issue. In principle, nonstiff methods can solve stiff problems; they just take a long time to do it.

As an illustration, imagine trying to find the quickest descent through a canyon. An explicit algorithm, which is normally used for nonstiff models, would sample the local gradient to find the descent direction. But following the gradient on either side of the trail will send you bouncing back and forth from wall to wall — the descent will be found but it will take a long time. An implicit algorithm used for stiff models can anticipate where each step is taking you, keep you on the trail with fewer steps, and so save time. Using a stiff solver for a stiff problem can save thousands of solver steps and function evaluations compared to a nonstiff solver.

Methods intended to solve stiff problems efficiently do more work per step, but can take much bigger steps. Stiff methods are implicit. At each step they use MATLAB matrix operations to solve a system of simultaneous linear equations that helps predict the evolution of the solution.

Not all difficult problems are stiff, but all stiff problems are difficult for solvers not specifically designed for them. Solvers for stiff problems can be used exactly like the other solvers.

For an illustrative code example you can run to plot the effects of numerical stiffness on different solvers, see MATLAB News & Notes - May 2003 Cleve's Corner: Stiff Differential Equations.

Selecting a Solver

Choice of solver depends on the problem and time available for computation. There are trade-offs to be made between speed and accuracy. In general, `ode45` is the best function to apply as a "first try" for most problems, or `ode15s` if you suspect that a problem is stiff. As you find out more about the problem you can try other solvers. Experimentation is generally required to determine the best solver for a particular model. As a general guide:

- 1 Models with either all fast or all slow changing variables are nonstiff problems:

Use "Nonstiff Deterministic Solvers" on page 2-8.

- `ode45` — Best first guess.
- Sundials — Alternative best first guess. May be faster.

- ode23 — May be more efficient than ode45 with crude tolerances and mild stiffness.
- ode113 — May be more efficient than ode45 with stringent tolerances.

2 Models with both fast and slow changing variables are stiff problems:

Use “Stiff Deterministic Solvers” on page 2-10.

- ode15s — Try first if you suspect that a problem is stiff, or if ode45 failed or was very inefficient.
- Sundials — Alternative best first guess. May be faster.
- ode23s — May be more efficient than ode15s at crude tolerances, and can solve some stiff problems that ode15s cannot.
- ode23t — Use this solver if the problem is only moderately stiff and you need a solution without numerical damping.
- ode23tb — Like ode23s, this solver may be more efficient than ode15s at crude tolerances.

3 Models with a small number of molecules:

Use “Stochastic Solvers” on page 2-12.

- Stochastic — Most accurate, may be too slow if the initial number of molecules for a reactant species is large.
- Explicit Tau — Speeds up the simulation at the cost of some accuracy; can be orders of magnitude faster than Stochastic. Can be used for large problems (provided the problem is not numerically stiff).
- Implicit Tau — May be the fastest, at the cost of some accuracy. Can be used for large problems and also for numerically stiff problems. For nonstiff systems may not be a good choice because it adds computational overhead.

If you use a stochastic solver to simulate a model, the software ignores any rate, assignment, or algebraic rules if present in the model.

Nonstiff Deterministic Solvers

In this section...
“When to Use Nonstiff Deterministic Solvers” on page 2-8
“ode45 (Dormand-Prince)” on page 2-8
“ode23 (Bogacki-Shampine)” on page 2-8
“ode113 (Adams)” on page 2-8
“See Also” on page 2-9

When to Use Nonstiff Deterministic Solvers

If you have models with either all fast or all slow changing variables, these may not be numerically stiff; nonstiff deterministic solvers are appropriate to try.

ode45 (Dormand-Prince)

Based on an explicit Runge-Kutta (4,5) formula: the Dormand-Prince pair, ode45 is a one-step solver in computing $y(t_n)$. It needs only the solution at the immediately preceding time point $y(t_{n-1})$. In general, ode45 is the best function to apply as a "first try" for most problems.

ode23 (Bogacki-Shampine)

Based on an explicit Runge-Kutta (2,3) pair of Bogacki and Shampine, ode23 may be more efficient than ode45 at crude tolerances and in the presence of mild stiffness. Like ode45, ode23 is a one-step solver.

ode113 (Adams)

A variable order Adams-Bashforth-Moulton PECE solver, ode113 may be more efficient than ode45 at stringent tolerances and when the ODE function is particularly expensive to evaluate. ode113 is a multistep solver; it normally needs the solutions at several preceding time points to compute the current solution.

See Also

“ODEs” in *MATLAB Mathematics*.

Stiff Deterministic Solvers

In this section...

“When to Use Stiff Deterministic Solvers” on page 2-10

“ode15s (stiff/NDF)” on page 2-10

“ode23s (stiff/Mod. Rosenbrock)” on page 2-10

“ode23t (Mode. stiff/Trapezoidal)” on page 2-10

“ode23tb (stiff/TR-BDF2)” on page 2-11

“See Also” on page 2-11

When to Use Stiff Deterministic Solvers

If you have models with a mixture of fast and slow changing variables, such models are numerically stiff. Stiff deterministic solvers are the best choice.

ode15s (stiff/NDF)

A variable order solver based on the numerical differentiation formulas (NDFs), ode15s optionally uses the backward differentiation formulas, BDFs (also known as Gear’s method). Like ode113, ode15s is a multistep solver. If you suspect that a problem is stiff or if ode45 failed or was very inefficient, try ode15s.

ode23s (stiff/Mod. Rosenbrock)

The ode23s solver is based on a modified Rosenbrock formula of order 2. Because it is a one-step solver, it may be more efficient than ode15s at crude tolerances. It can solve some kinds of stiff problems for which ode15s is not effective.

ode23t (Mode. stiff/Trapezoidal)

The ode23t solver is an implementation of the trapezoidal rule using a “free” interpolant. Use this solver if the problem is only moderately stiff and you need a solution without numerical damping.

ode23tb (stiff/TR-BDF2)

The `ode23tb` is an implementation of TR-BDF2, an implicit Runge-Kutta formula with a first stage that is a trapezoidal rule step and a second stage that is a backward differentiation formula of order 2. Like `ode23s`, this solver may be more efficient than `ode15s` at crude tolerances.

See Also

“ODEs” in *MATLAB Mathematics*.

Stochastic Solvers

In this section...
“When to Use Stochastic Solvers” on page 2-12
“Stochastic Simulation Algorithm (SSA)” on page 2-12
“Explicit Tau-Leaping Algorithm” on page 2-13
“Implicit Tau-Leaping Algorithm” on page 2-13
“Ensemble Runs of Stochastic Simulations” on page 2-14
“References” on page 2-16

When to Use Stochastic Solvers

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. The stochastic simulation algorithms provide a practical method for simulating reactions which are stochastic in nature. Depending on the model, stochastic simulations may take more computation time than deterministic simulations.

If you use a stochastic solver to simulate a model, the software ignores any rate, assignment, or algebraic rules if present in the model.

Stochastic Simulation Algorithm (SSA)

Using the stochastic simulation algorithm for a system is equivalent to solving the Chemical Master Equation for the system. The Chemical Master Equation is otherwise impossible to solve for most practical problems. Thus, the stochastic simulation algorithm provides a practical method for simulating stochastic systems. The algorithm simulates one reaction at a time based on the propensity function for each reaction.

Advantage:

- This algorithm is exact.

Disadvantages:

- Since it evaluates one reaction at a time, it may be too slow for large problems.
- If the number of molecules of any of the reactants is huge, it may take a long time to complete the simulation.

Explicit Tau-Leaping Algorithm

Since the stochastic simulation algorithm may be too slow for a lot of practical problems, this algorithm has been designed to speed up the simulation at the cost of some accuracy. The algorithm treats each reaction channel 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 the user-specified error tolerance. After selecting the time interval, the algorithm computes the number of times each reaction channel fires 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.
- This algorithm can be used for large problems (provided the problem is not numerically stiff).

Disadvantages

- Some accuracy is sacrificed for speed.
- Not good for stiff models.
- The error tolerance needs to be specified in such a manner 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 and the “fast modes” are stable. 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 channel as being independent of others. It automatically chooses 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, the algorithm computes the number of times each reaction channel fires 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.
- It can be used 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

- Some accuracy is sacrificed 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 damps out the perturbations off the slow manifold leading to a reduced state variance about the mean.

Ensemble Runs of Stochastic Simulations

Ensemble runs are ensemble simulations that you can use in conjunction with the stochastic solvers to gather data from multiple stochastic runs of the model. Ensemble runs let you investigate fluctuations in the behavior of a stochastic model over repeated simulations.

In contrast, scans are multiple simulations of the model performed with varying values of parameters or initial amounts of species. You can specify the range for the parameter or the species, and each simulation is performed with a different value of the parameter or species amount within the specified range. Scans let you see changes in the model's behavior with respect to changes in species amounts, or parameter values.

You can perform ensemble simulations using the stochastic solvers to gather data from multiple stochastic runs of the model.

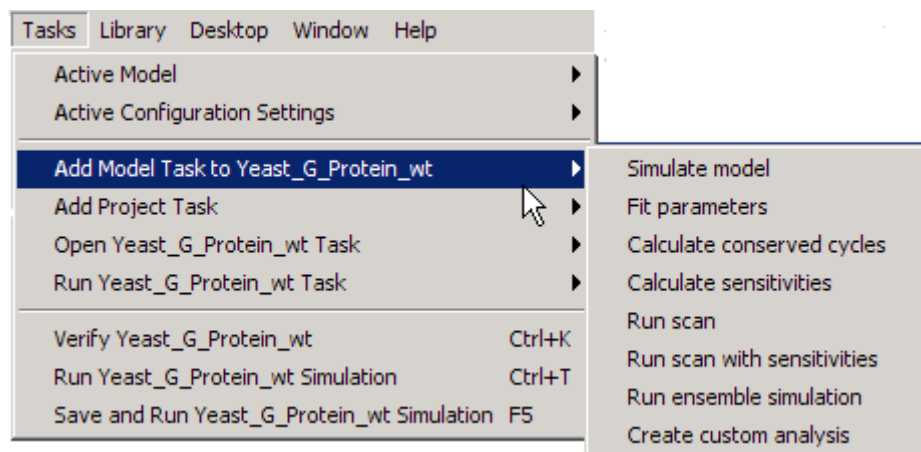
Running Ensemble Simulations at the Command Line

The following functions let you perform ensemble runs at the command line:

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

Running Ensemble Simulations in the Desktop

- 1 In the SimBiology desktop, from the **Tasks** menu select **Add Model Task to *model_name* > Run ensemble simulation**.



The desktop adds **Ensemble Run** in the **Project Explorer** and opens the **Ensemble Run** pane.

- 2 See the context-sensitive

SimBiology Desktop Help for more information on how to set up ensemble runs. To access **SimBiology Desktop Help**, select **Help > SimBiology Desktop Help**.

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.

Sundials Solvers

The Sundials solvers are part of a freely available third-party package developed at Lawrence Livermore National Laboratory. All the other ODE solvers used for simulation of SimBiology models, such as `ode45`, and `ode15s`, are part of the MATLAB ODE suite. 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 in the same way, as described in “How Solvers Work” on page 2-5

When you select the **SolverType** `Sundials`, the software automatically chooses one of two Sundials solvers as appropriate for your model: `CVODE` or `IDA`. `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 the web site <http://www.llnl.gov/casc/sundials/description/description.html>.

Analysis

You can perform sensitivity analysis on your model, look for conserved moieties, estimate parameters, and gather data with ensemble stochastic runs.

- “Scanning Analysis” on page 3-2
- “Desktop Example — Scanning” on page 3-10
- “Sensitivity Analysis” on page 3-25
- “Desktop Example — Calculating Sensitivities” on page 3-29
- “Command-Line Example — Calculating Sensitivities” on page 3-39
- “Parameter Estimation” on page 3-45
- “Command-Line Example — Parameter Estimation” on page 3-46
- “Moiety Conservation” on page 3-58
- “Command-Line Examples — Determining Conserved Moieties” on page 3-61
- “Desktop Example — Creating Custom Analysis” on page 3-68
- “Visualizing Results Using Plot Types” on page 3-73
- “Desktop Example — Creating and Using Plot Types” on page 3-81

Scanning Analysis

In this section...

“Scanning Overview” on page 3-2

“Creating a Scan Task” on page 3-3

“Setting Options to Scan Using User-Defined Values” on page 3-3

“Options For Monte Carlo Methods” on page 3-6

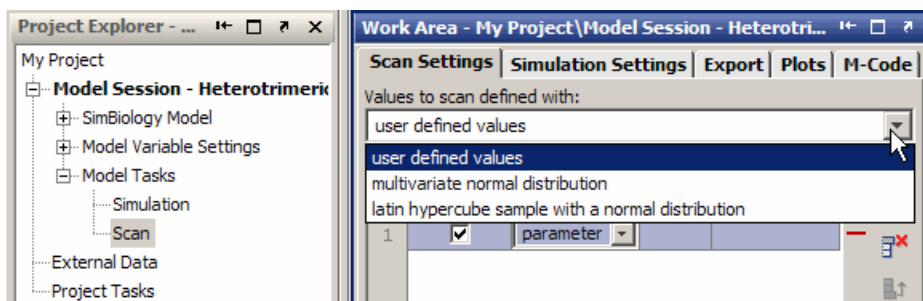
“Setting Options to Scan Using Monte Carlo Methods” on page 3-7

Scanning Overview

Scanning using parameter values, species initial amounts, or compartment capacities in a model lets you observe model behavior over a range of values and in combination with other parameter, species, and compartment ranges.

Perform scanning analysis of your model using the **Run scan** task in the SimBiology desktop. The **Run scan** task lets you:

- Define the range to scan (user defined values option)
- Use Monte Carlo methods, and specify sampling using the following options:
 - multivariate normal distribution
 - latin hypercube sample with a normal distribution



Creating a Scan Task

- 1 Open a model in the SimBiology desktop to enable scanning.
- 2 From the **Tasks** menu select **Add Model Task to *model_name* > Run scan**.

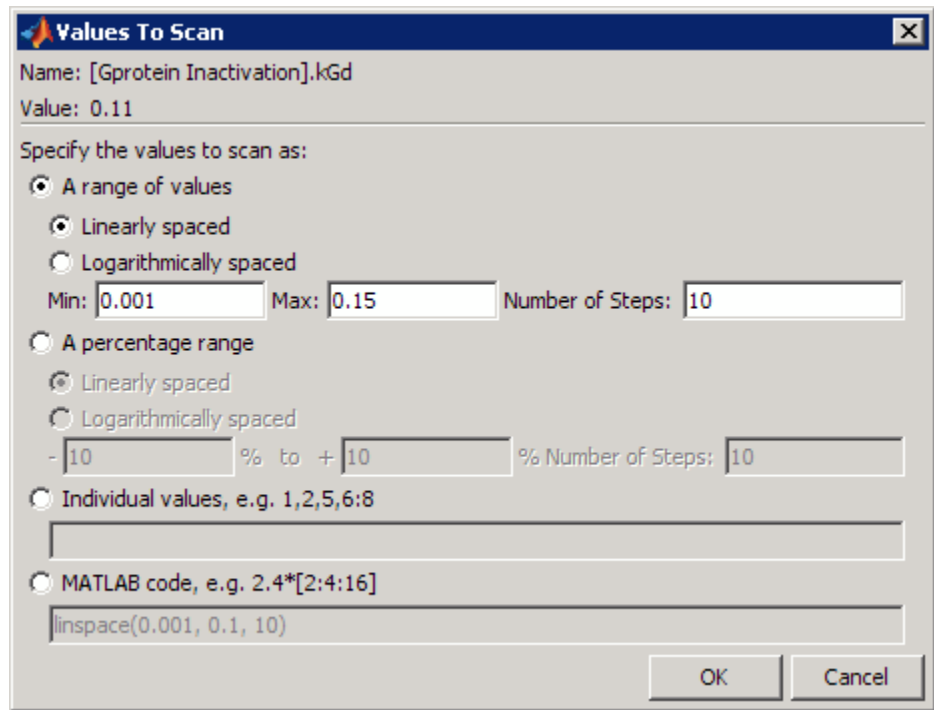
Setting Options to Scan Using User-Defined Values

- 1 If user defined values is not already specified, select it from the **Values to scan defined with** list. By default, the table contains one row specifying scan options.
- 2 From the **Type** list, select Parameter, Species, or Compartment.
- 3 Double-click a cell in the **Name** column.
- 4 Enter a component name. For example:

```
transcription.k  
nucleus  
membrane.receptor
```

Tip Double-click the **Name** cell and press the down arrow for a list of available components. Select a name from the list.

- 5 Double-click a cell in the **Values To Scan** column. The Values To Scan dialog box opens. The **Name** and the **Value**, **InitialAmount**, or **Capacity** of the selected parameter, species, or compartment appears at the top.



- 6 Select one of the options below and expand the option for the next steps:

A range of values

- a Select either **Linearly spaced** or **Logarithmically spaced** based on how you want to space the scan intervals.
- b In the **Min** and **Max** boxes, enter the minimum and maximum value of the scan interval (if logarithmically spaced, specify the minimum and maximum of the exponent).
- c In the **Number of Steps** box, enter the total number of iterations of the simulation to be performed, including the simulations using the minimum and maximum values.

A percentage range

- a Select either **Linearly spaced** or **Logarithmically spaced** based on how you want to space the scan intervals.

- b** In the – % and + % boxes, enter the percent minimum and maximum value of the scan interval. The percentages are relative to the **Value**, **InitialAmount**, or **Capacity** of a parameter, species, or compartment, which is listed at the top. In other words, if **Value** of a parameter is 0.11 and you enter -10% to +10%, then the values scanned are between 90% and 110% of 0.11.
- c** In the **Number of Steps** box, enter the total number of iterations of the simulation to be performed, including the simulations using the minimum and maximum percentage values.

Individual values

In the **Individual values** box, specify the values to use. For example:

1,2,5 or 5:10 or 5:0.5:10

MATLAB code


In the **MATLAB code** box, specify any valid MATLAB code, for example,

```
10*[2,3,5:10]
```

Use functions from MathWorks™ products. For example, perform more complex Monte Carlo analysis using additional functions from the Statistics Toolbox™, read in values from a spreadsheet, or use a custom function written to generate scan intervals.

- 7** Click **OK** to apply the settings and close the dialog box.
- 8** (Optional) In the **Scan Settings** tab, select the **Scan along the diagonal** check box.

When scanning over multiple parameters, if you have the same number of values to scan for each parameter, you can specify that the scan occur along the diagonal. That is, specify that the first parameter value be used with the first specified value of the next parameter, the second parameter value be used with the second specified value of the next parameter and so on. This reduces the number of simulations whilst scanning over multiple parameters.

- 9 (Optional) to scan with multiple parameters, click  (Add table row) to add more components to scan. A new row appears in the table. Set the options as described in the previous steps.
- 10 Click **Run**. The figure window opens with two plots. One plot has each run plotted in separate subplots. The other plot shows the results of all the runs in one plot.

Options For Monte Carlo Methods

Note Requires Statistics Toolbox

The SimBiology desktop supports Monte Carlo analysis by providing functionality to perform scanning using parameter values sampled from a probability distribution.

The multivariate normal distribution and latin hypercube sample with a normal distribution options use Statistics Toolbox functions to generate the sample values used in the simulation.

Multivariate Normal Distribution Option

This option uses the `mvnrnd` function to generate the parameter values chosen from a multivariate normal distribution.

Multivariate normal distribution is a generalization of the univariate normal to two or more variables. It is a distribution for random vectors of correlated variables, each element of which has a univariate normal distribution. In the simplest case, there is no correlation among variables, and elements of the vectors are independent univariate normal random variables.

The multivariate normal distribution is parameterized with a mean vector, μ , and a covariance matrix, Σ . These are analogous to the mean μ , and the variance (square of the standard deviation σ), of a univariate normal distribution.

By default, the mean is defined by the specified parameter value, species initial amount, or compartment capacity. You can specify alternate values

for the means and the covariance matrix. Use a diagonal matrix to specify no covariance between the parameters.

Latin Hypercube Sample with a Normal Distribution Option

This option uses the `lhsnorm` function to generate the parameter values chosen from a multivariate normal distribution. Sampling resembles multivariate normal distribution, however, this option additionally adjusts the marginal distribution of each column such that its sample marginal distribution is close to its theoretical normal distribution.

Use Latin Hypercube sampling to maximize the distribution of samples while minimizing the number of scans to perform. This functionality is useful to scan over many parameters at once.

Setting Options to Scan Using Monte Carlo Methods

Statistics Toolbox is required for this functionality.

1 In the **Scan Settings** tab, select either of the following from the **Values to scan defined with** list :

- multivariate normal distribution
- latin hypercube sample with a normal distribution

By default, the table contains one row specifying scan options.

2 From the **Type** list, select **Parameter**, **Species**, or **Compartment**.


3 Double-click a cell in the **Name** column.

4 Enter a component name. For example:

```
transcription.k  
nucleus  
membrane.receptor
```

Tip Double-click the **Name** cell and press the down arrow for a list of available components. Select a name from the list.

The **Mean Value** column is populated with the values entered in the model.

- 5** (Optional) to scan with multiple parameters, click  (Add table row) to add more components to scan. A new row appears in the table. Set the options as described in the previous steps.
- 6** Enter the covariance matrix. The covariance matrix must be a symmetric positive semi-definite matrix. Under **Define the covariance matrix below or define with MATLAB code** do one of the following:
 - Double-click a matrix cell to enter the variance or covariance values.

Note When you enter the covariance for one pair of parameters, the corresponding cell automatically updates.

- Specify MATLAB code as follows:
 - 1** Click **Editor** to open the Covariance Matrix Editor to calculate and enter covariance values using MATLAB code. For example, to enter σ^2 (square of the standard deviation) values in the covariance matrix enter:

```
% Perform a Monte Carlo simulation with 2 independent
% parameters
% Specify the number of parameters being scanned
n = 2;
% Specify the standard deviations for each scanned
% parameter
stddev = [3 2];
% Construct the covariance matrix
covarianceMatrix = diag(stddev.^2) ;
```

Note Use the keyword `covarianceMatrix` as the variable name in the editor.

- 2** Click **OK** to enter the covariance matrix.

- 7 Click **Run**. The figure window opens with two plots. One plot has each run plotted in a separate subplot. The other plot shows the results of all the runs in one plot.

Desktop Example – Scanning

In this section...

“Overview” on page 3-10

“Prerequisites” on page 3-12

“Setting Options to Scan with One Parameter” on page 3-13

“Results of Scanning with One Parameter” on page 3-15

“Scanning with Multiple Parameters” on page 3-20

“Results of Scanning with Multiple Parameters” on page 3-21

“References” on page 3-24

Overview

This example shows you how to set up and run simple scans using user-defined values in the SimBiology desktop. You can run parameter, species, or compartment scans.

For information on how to run scans at the command line, type the following at the command line to open a demo that includes scanning:

```
edit gprotein
```

About the Example Model

This example uses the model from “Modeling a G Protein Cycle” in the SimBiology Model Reference documentation.

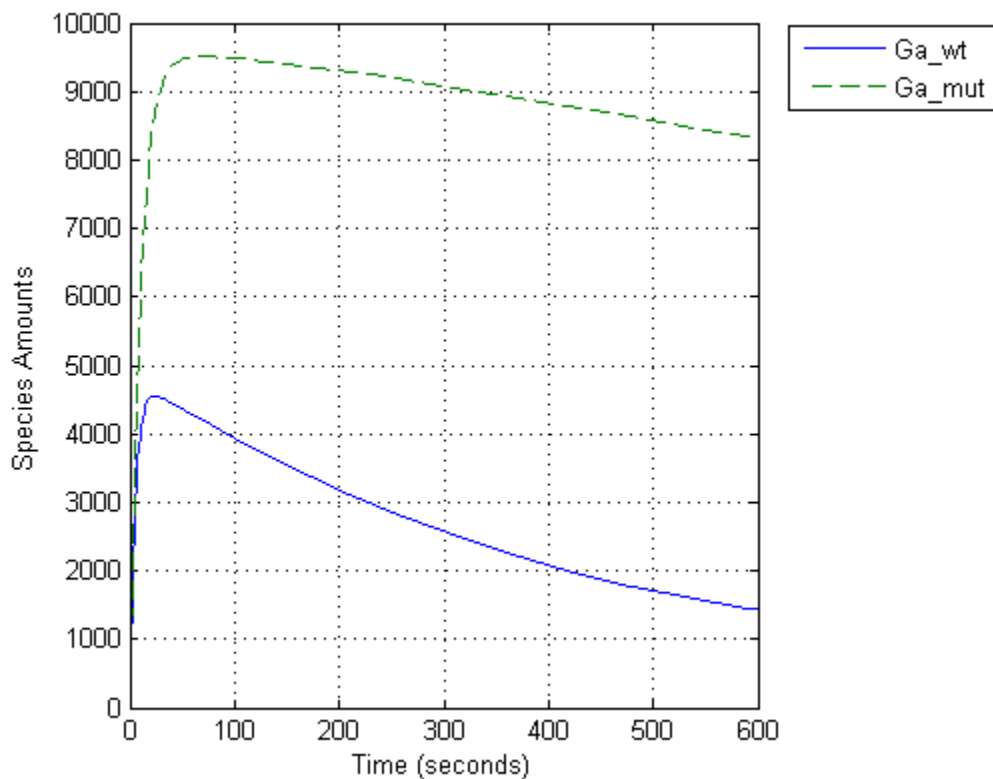
This table shows the reactions used to model the G protein cycle and the corresponding rate parameters (rate constants) for each reaction. For reversible reactions, the forward rate parameter is listed first.

No.	Name	Reaction	Rate Parameters
1	Receptor-ligand interaction	$L + R \rightleftharpoons RL$	k _R L, k _R L _m

No.	Name	Reaction	Rate Parameters
2	Heterotrimeric G protein formation	$G_d + G_{\beta\gamma} \rightarrow G$	kG1
3	G protein activation	$RL + G \rightarrow Ga + G_{\beta\gamma} + RL$	kGa
4	Receptor synthesis and degradation	$R \leftrightarrow \text{null}$	kRdo, kRs
5	Receptor-ligand degradation	$RL \rightarrow \text{null}$	kRD1
6	G protein inactivation	$Ga \rightarrow G_d$	kGd

About This Example

The rate of G protein inactivation is much lower in the mutant strain ($k_{Gd} = 0.004$) relative to the wild-type strain ($k_{Gd} = 0.11$) [Yi et al. (2003)]. This rate explains the higher levels of activated G protein (Ga) over time as shown in the following figure.



For a more detailed look at how the variation of k_{Gd} affects levels of Ga, perform a parameter scan of several simulations in which the value of k_{Gd} varies over a range of values. This example illustrates a scan over 10 values of the parameter k_{Gd} .

Prerequisites

Opening and Saving the Example Model

- 1 Load the example project by typing the following at the command line:


```
sbioloadproject gprotein
```

The model is stored in a variable called `m1`.

- 2 Open the SimBiology desktop with the model loaded by typing:

```
simbiology(m1)
```

The desktop opens with **Model Session-Heterotrimeric_G_Protein_wt**.

- 3 Select **File > Save Project As**. The Save SimBiology Project dialog box opens.
- 4 Specify a name (for example, `gprotein_ex`) and location for your project, and click **Save**.

Adding a Scanning Task

To scan a parameter, species, or compartment in a model, add a model task for scanning:

- 1 In the **Project Explorer** under **Model Session-Heterotrimeric_G_Protein_wt**, right-click **Model Tasks**.
- 2 Select **Add Task > Run scan**. The **Scan** pane opens.

Setting Options to Scan with One Parameter

- 1 In the **Scan Settings** tab, from the **Values to scan defined with** list, select **user defined values**, if not already specified. By default, the table contains one row specifying scan options.
- 2 In the table, from the **Type** list, select **Parameter**, **Species**, or **Compartment**.
- 3 In the **Scan** pane, the first row contains a default **Parameter**.
- 4 Double-click the **Name** cell and enter the name of the parameter to scan.

```
[Gprotein Inactivation].kGd
```

Tip Double-click the **Name** cell and press the down arrow for a list of available parameters. Select a parameter from the list.

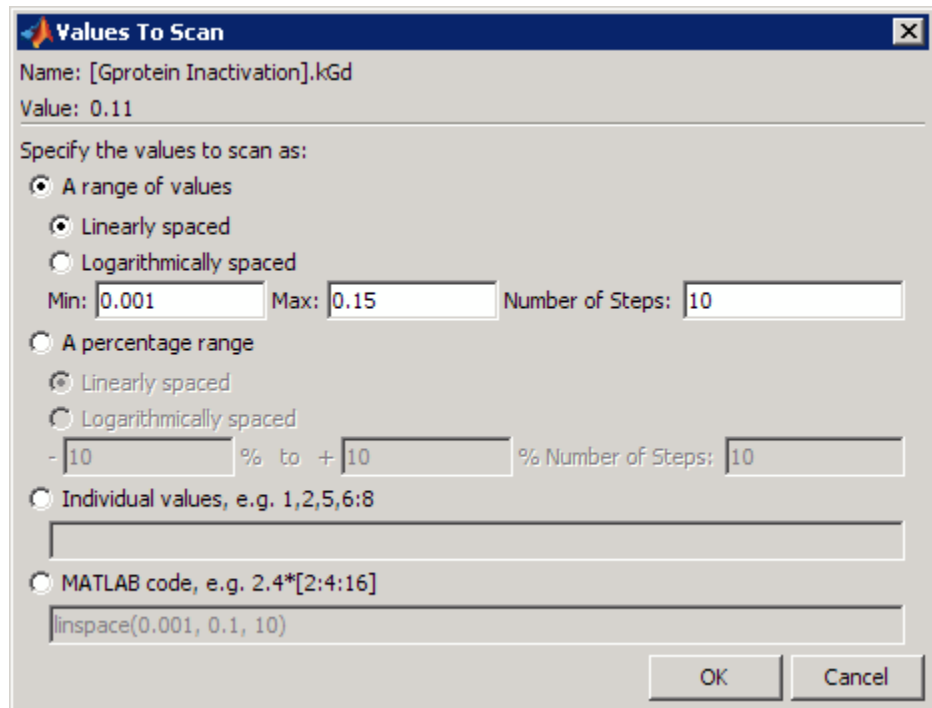
- 5** Double-click the **Values To Scan** cell. The Values To Scan dialog box opens. You see the current value of the parameter, and can select a range of values to scan.

Leave these default options selected:

- **A range of values**
- **Linearly spaced**

- 6** In the **Min** box enter 0.001 and in the **Max** box enter 0.15.

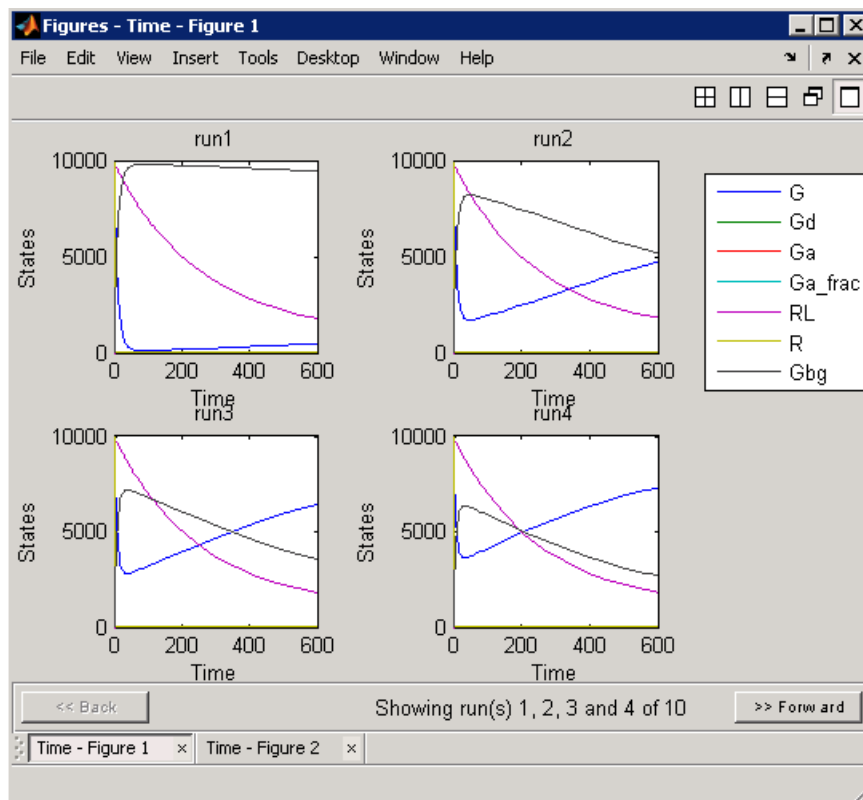
- 7** Leave the **Number of Steps** as the default (10). **Number of Steps** specifies the number of values applied during scanning. Thus, 10 means that 10 values will be applied between **Min** and **Max** and this results in 10 simulations; each with one value of the parameter applied to the model.



- 8 Click **OK**.
- 9 Select **File > Save Project**.

Results of Scanning with One Parameter

- 1 Click **Run**. The figure window opens with two plots. One plot has each run plotted in a separate subplot. The other plot shows the results of all the runs in one plot.




Tip View the range of plots by clicking **Forward**. Click **Time - Figure2** to view the consolidated plot.

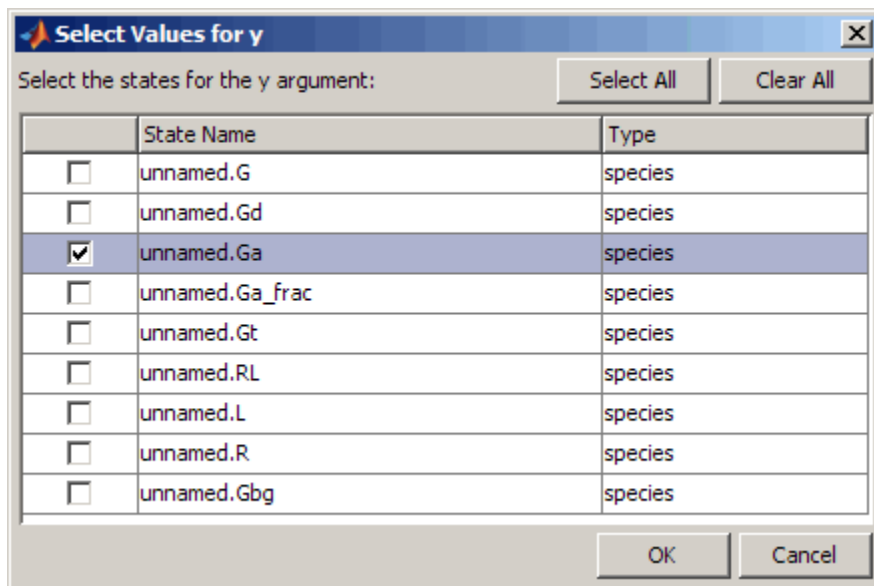
- 2 Close the figure window. The **Data** pane showing the most recent scan data is now available in the **Project Explorer**. In the **Project Explorer** under **Scan**, click **Data (Date)**.

In the **Data** pane you can plot the results for the species that are of interest in this scan, such as Ga. First, it is useful to save the data separately so that future scans do not overwrite existing data.

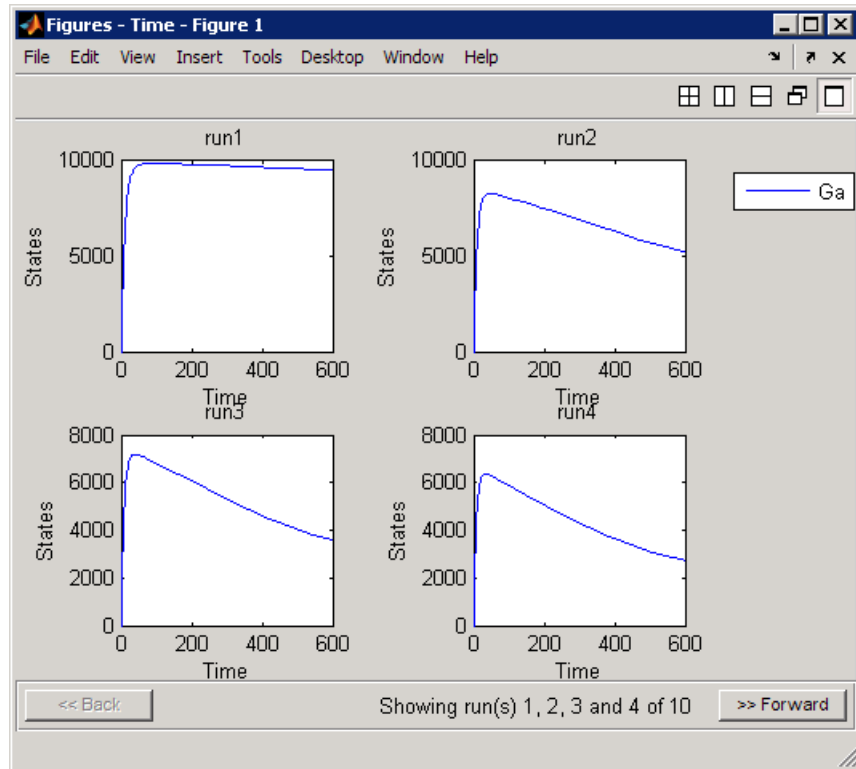
- 3** In the **Project Explorer** under **Scan**, right-click **Data** and select **Save Data**. The Save Data dialog box opens.
- 4** Specify a name for the saved data, for example, `scan_ex1`, and click **Save**. The **Project Explorer** shows a new item with the saved data name under **Scan**.
- 5** In the **Project Explorer**, click the saved data, for example, `scan_ex1`, to open the **Data** pane for the saved data.
- 6** Click the **Plots** tab.
- 7** In the **Plot Type** box, select the first row. This row specifies a subplot.

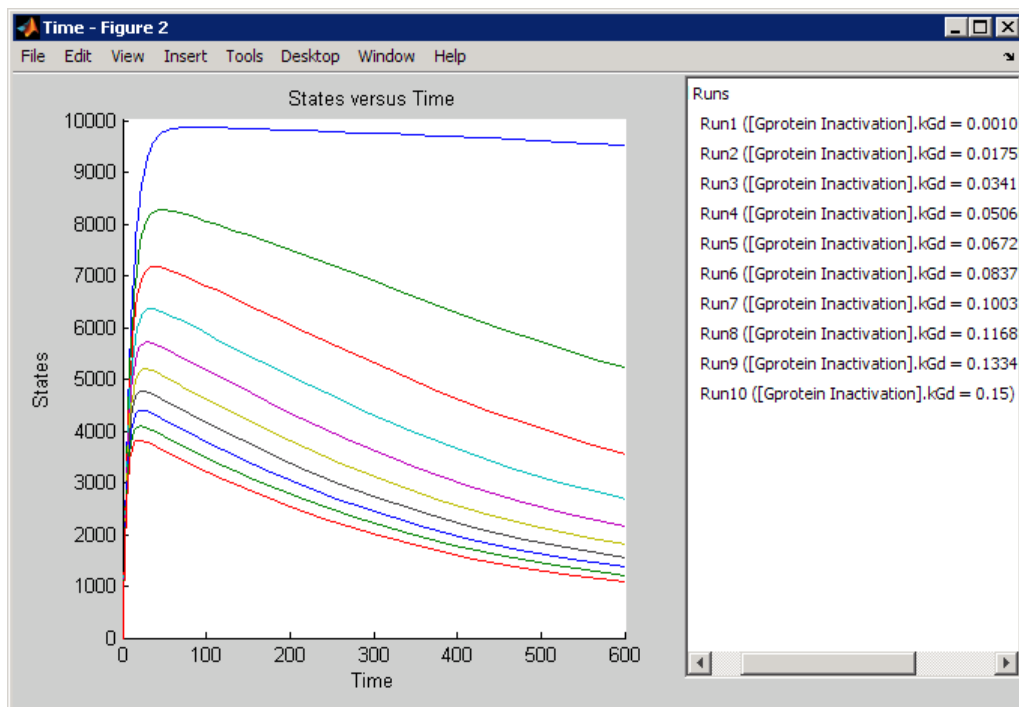
Note In the **Arguments** section, the **plotStyle** list option is `subplot`.

- 8** In the **Arguments** section, click . The Select Values for y dialog box opens.
- 9** Click **Clear All**, and then select the check box for the species **Ga**.



- 10 Click **OK**.
- 11 In the plots table, select the second row containing the Time plot and then choose to plot Ga by repeating steps 8 to 10 for this plot.
- 12 Click **Plot**. Your plots should resemble the following. Depending on the values selected automatically for scanning, your results may not exactly match the following plots.






Scanning with Multiple Parameters

If evidence shows that multiple parameters have an effect on the species of interest, you might want to scan over a range of values for those parameters.

For example, in this model previous sensitivity analysis shows that the amount of active G protein (G_a) is also sensitive to the parameter k_{RL} . This result follows from the model because parameter k_{RL} governs the rate of formation of the receptor-ligand complex which then facilitates the formation of active G protein.

Thus, it might be interesting to study the effect of these two parameters in conjunction with each other on the levels of active G protein. Perform a scan that iterates through each value of parameter k_{Gd} with each value of parameter k_{RL} .

To scan using multiple parameter:

- 1 In the **Project Explorer**, expand **Model Tasks** and click **Scan**.
- 2 In the **Scan** pane, click  (Add table row) to add more components to scan.

A second row containing the type of scan to run appears in the table.

	Run Scan	Type	Name	Values To Scan
1	<input checked="" type="checkbox"/>	Parameter	[Gprotein Inactivation].kGd	linspace(0.001, 0.15, 10)
2	<input checked="" type="checkbox"/>	Parameter		

- 3 Double-click the **Name** cell and press the down arrow for a list of available parameters. Select **kRL** and click **OK**.
- 4 Double-click the **Values To Scan** cell. The Values To Scan dialog box opens. You see the current value of the parameter, and can select a range of values to scan.
- 5 Click the **Individual values** option and enter these values:

3.32E-16, 3.32E-17, 3.32E-18, 3.32E-19, 3.32E-20

Entering individual values lets you control precise values to use in the scan. This is useful when there are fixed concentrations of species, values of parameters, and compartment capacities that you want to specify for the scan.

- 6 Click **OK**.
- 7 Select **File > Save Project**.

Notice that you specified five values for **kRL** in step 5. Thus, there will be 50 iterative simulations when the task runs because the simulations must loop through each value of **kGd** (there are 10 values), against each value of **kRL**.

Results of Scanning with Multiple Parameters

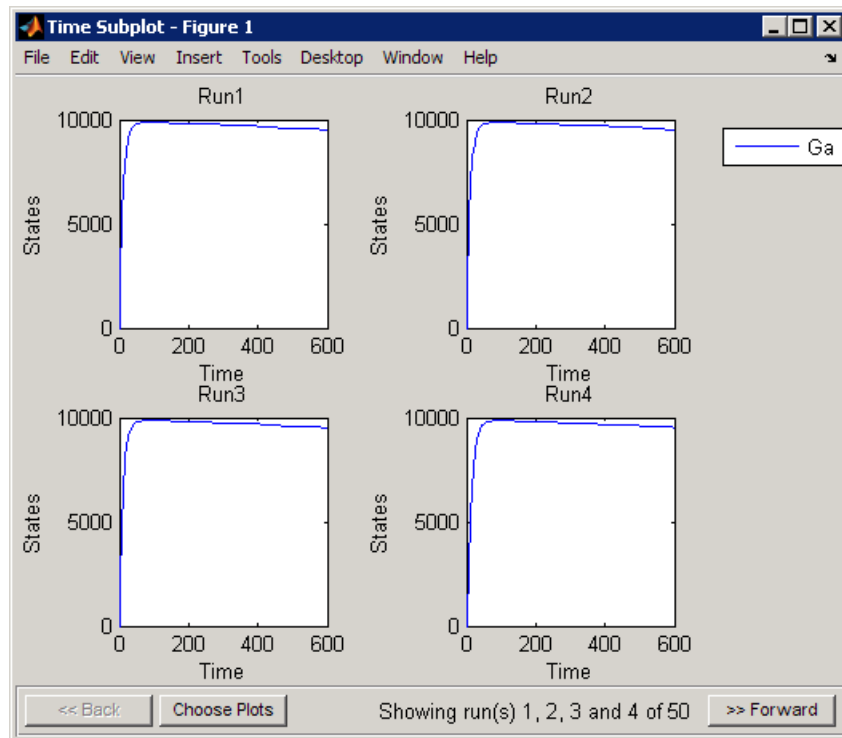
Previously, in “Results of Scanning with One Parameter” on page 3-15 the results for **Ga** were plotted from the **Data** pane after scanning.

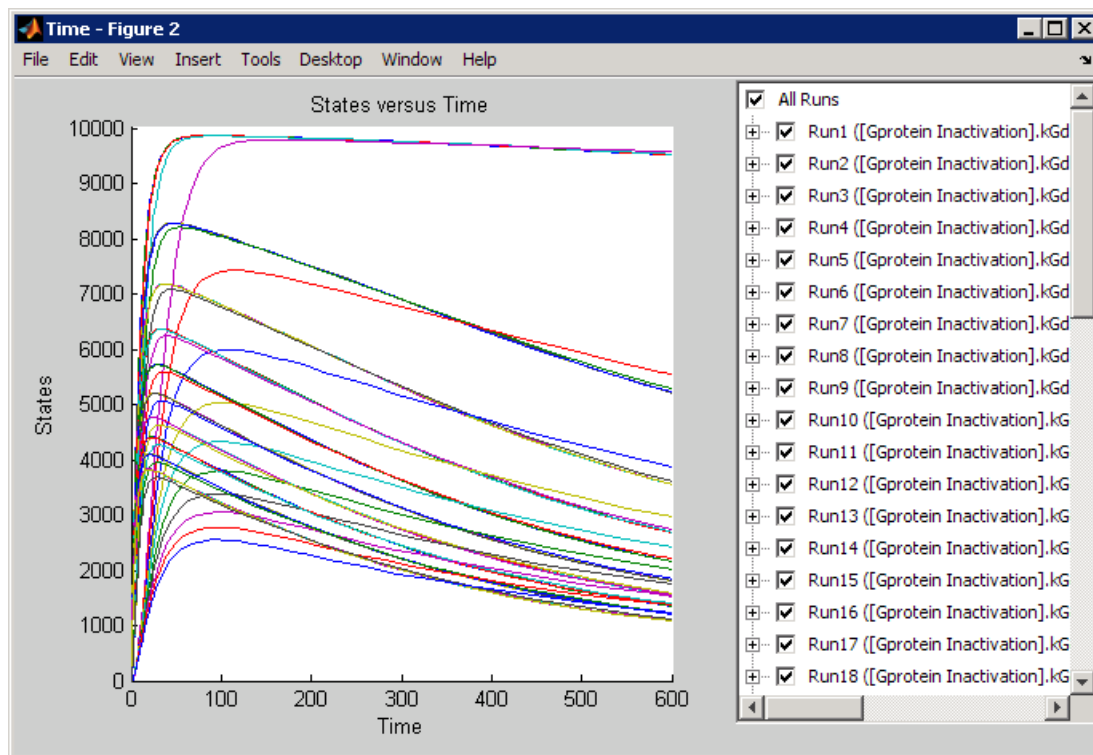
This time, before running the scan, it might be useful to change the settings in the **Plots** tab to modify the plots generated by the scan to show the results just for active G protein (Ga) and ignore the other species. This workflow is particularly useful when you know the modifications you want to make in the visualization of the results from a task.

- 1 In the **Scan** pane, click **Plots**.
- 2 In the **Project Explorer**, click the saved data for the previous scan, scan_ex.
- 3 Press **Shift+** click to select both rows, and copy the rows using the context menu or **Ctrl+C**.
- 4 In the **Project Explorer**, click **Scan** to open the **Scan** pane and paste the rows into the **Plots** tab.
- 5 Clear the **Create Plot** check box for the default plots in the first two rows that specify $y = \langle \text{all} \rangle$ in the **Arguments** column.

	Create Plot	Plot Behavior	Plot Type	Arguments
1	<input type="checkbox"/>	New figure ▾	Time	tobj = Simulation Results; y = <all>; plotStyle = subplot
2	<input type="checkbox"/>	New figure ▾	Time	tobj = Simulation Results; y = <all>; plotStyle = one axes
3	<input checked="" type="checkbox"/>	New figure ▾	Time	tobj = Simulation Results; y = unnamed.Ga; plotStyle = subplot
4	<input checked="" type="checkbox"/>	New figure ▾	Time	tobj = Simulation Results; y = unnamed.Ga; plotStyle = one axes

- 6 Click **Run**. Your plots should resemble the following. Depending on the values selected automatically for scanning, your results may not exactly match the following plots.





Tip In the Time plot (second figure above), select or clear the **Run#** check boxes to view each run individually or in combination with other runs.

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.

Sensitivity Analysis

In this section...

“About Sensitivity Analysis” on page 3-25

“Performing Sensitivity Analysis Using the Command Line” on page 3-26

“Performing Sensitivity Analysis Using the Desktop” on page 3-27

“Reference” on page 3-28

About Sensitivity Analysis

Sensitivity analysis lets you calculate 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.

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 the "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 non-analytic 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 non-analytic 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 the MathWorks Technical Support group for additional information.

Note Models containing rules and events do not support sensitivity analysis.

For more information on the calculations performed, see “Reference” on page 3-28

Performing Sensitivity Analysis Using the Command Line

You can perform sensitivity analysis at the command line by setting the following properties:

- **SensitivityAnalysis** – Lets you 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 summarized below:
 - **SpeciesOutputs** – Specify the species for which you want to compute the sensitivities. Sensitivities are calculated with respect to the initial conditions of the specified species. This is the numerator as described in “About Sensitivity Analysis” on page 3-25.
 - **SpeciesInputFactors** – Specify the species with respect to which you want to compute the sensitivities of the species outputs in your model. Sensitivities are calculated with respect to the initial conditions of the specified species. This is the denominator as described in “About Sensitivity Analysis” on page 3-25.

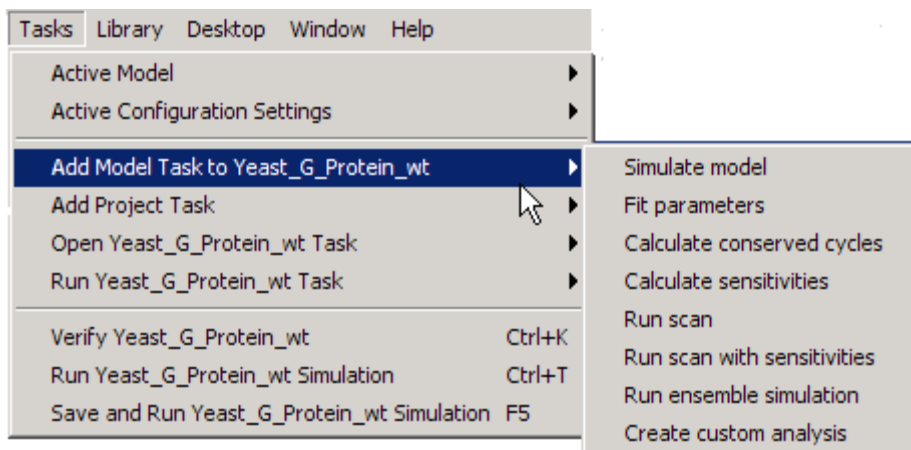
- **ParameterInputFactors** – Specify the parameters with respect to which you want to compute the sensitivities of the species outputs in your model. Sensitivities are calculated with respect to the values of the specified parameters. This is the denominator as described in “About Sensitivity Analysis” on page 3-25.
- **Normalization** – Specify the normalization for the calculated sensitivities.
 - 'None' specifies no normalization.
 - 'Half' specifies normalization relative to the numerator (species output) only.
 - 'Full' specifies full dedimensionalization.

For an example, see: “Command-Line Example — Calculating Sensitivities” on page 3-39

Performing Sensitivity Analysis Using the Desktop

You must have a model open in the desktop for this feature to be enabled. After opening a model, to get started with calculating sensitivities, do the following:

- 1 In the SimBiology desktop, from the **Tasks** menu select **Add Model Task to *model_name* > Calculate sensitivities**.



The desktop adds **Sensitivity Analysis** in the **Project Explorer** and opens the **Sensitivity Analysis** pane.

- 2 See the context-sensitive **SimBiology Desktop Help** for more information on how to set up sensitivity analysis. To access **SimBiology Desktop Help**, select **Help > SimBiology Desktop Help**.

For an example, see: “Desktop Example — Calculating Sensitivities” on page 3-29

Reference

Ingalls, B. P. , and H. M. Sauro. “Sensitivity analysis of stoichiometric networks: an extension of metabolic control analysis to non-steady state trajectories.” *Journal of Theoretical Biology* Vol. 222, 2003, pp. 23–36.

Desktop Example — Calculating Sensitivities

In this section...
“Overview” on page 3-29
“Prerequisites” on page 3-32
“Setting Options for Sensitivity Analysis” on page 3-33
“Getting Results for Sensitivity Analysis” on page 3-34
“References” on page 3-38

Overview

This example shows you how to set up and calculate sensitivities in the SimBiology desktop. For information on how to calculate sensitivities at the command line, see “Command-Line Example — Calculating Sensitivities” on page 3-39.

About the Example Model

This example uses the model from “Modeling a G Protein Cycle” in the SimBiology Model Reference documentation.

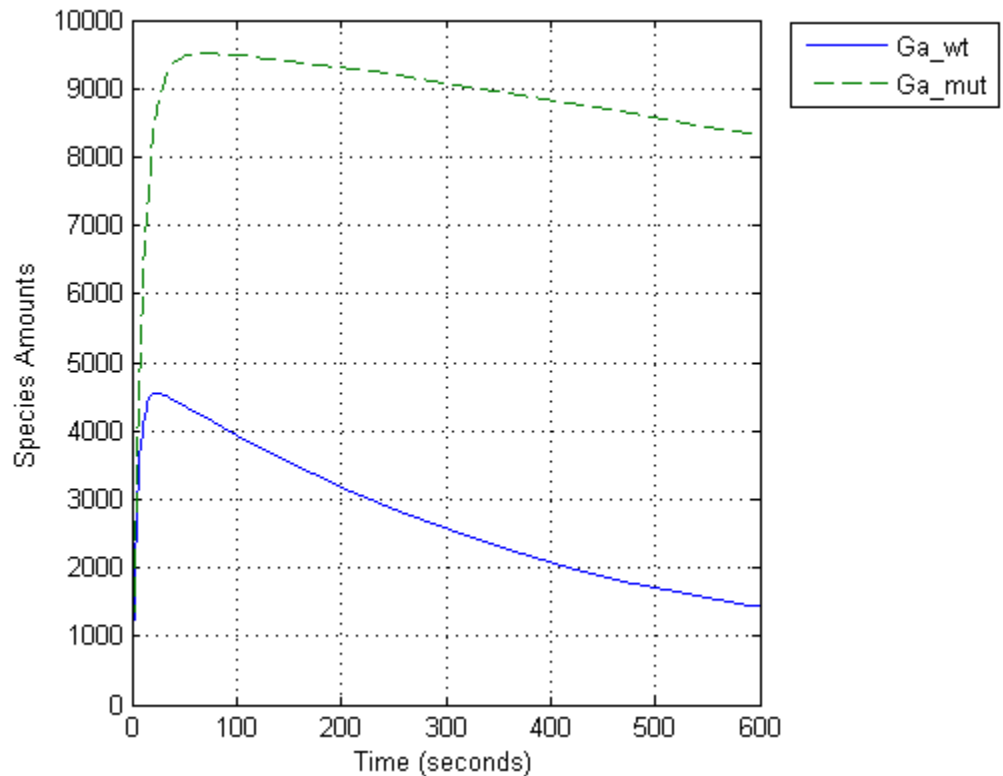
This table shows the reactions used to model the G protein cycle and the corresponding rate parameters (rate constants) for each reaction. For reversible reactions, the forward rate parameter is listed first.

No.	Name	Reaction	Rate Parameters
1	Receptor-ligand interaction	$L + R \leftrightarrow RL$	k_{RL}, k_{RLm}
2	Heterotrimeric G protein formation	$G_d + G_{bg} \rightarrow G$	k_{G1}
3	G protein activation	$RL + G \rightarrow G_a + G_{bg} + RL$	k_{Ga}
4	Receptor synthesis and degradation	$R \leftrightarrow \text{null}$	k_{Rdo}, k_{Rs}

No.	Name	Reaction	Rate Parameters
5	Receptor-ligand degradation	RL -> null	kRD1
6	G protein inactivation	Ga -> Gd	kGd

About This Example

Yi et al. (2003) show that the rate of G protein inactivation is much lower in the mutant strain ($k_{Gd} = 0.004$) relative to the wild-type strain ($k_{Gd} = 0.11$), which explains the higher levels of activated G protein (Ga) over time as shown in the following figure.



Thus, the active G protein, Ga, is sensitive to the value of the parameter, k_{Gd} . Other species or parameters in the model can also affect levels of active G protein. To study the sensitivity of a species to other species or parameters in a model, you can perform sensitivity analysis. Sensitivity analysis lets you compute the time-dependent derivatives of one or more species (**Output**) relative to either model parameter values or species initial conditions (**Input**).

First, it might be useful to explore the sensitivities of every species with respect to every parameter in the model. You can later narrow down the results to visualize the sensitivity results for Ga using plots. Thus, you want to calculate the time-dependent derivatives:

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

Prerequisites

- “Opening and Saving the Example Model” on page 3-32
- “Adding a Task to Calculate Sensitivities” on page 3-32

Opening and Saving the Example Model

- 1 Load the example project at the command line by typing

```
sbioloadproject gprotein_norules
```

The model is stored in a variable called `m1`.

- 2 Open the SimBiology desktop with the model loaded by typing:

```
simbiology(m1)
```

The SimBiology desktop opens with **Yeast_G_Protein_wt**.

- 3 Select **File > Save Project As**. The Save SimBiology Project dialog box opens.
- 4 Specify a name (for example, `gprotein_ex`) and location for your project and click **Save**.

Adding a Task to Calculate Sensitivities

To perform sensitivity analysis in a model, first add a model task to calculate sensitivities:

- In the **Project Explorer**, under **Model Session-Yeast_G_Protein_wt**, right-click **Model Tasks** and select **Add Task > Calculate Sensitivities**.

The **Sensitivity Analysis** pane opens.

Setting Options for Sensitivity Analysis

- 1 In the **Sensitivity Settings** tab, right-click the table and select **Define all species as outputs**.
- 2 Right-click the table and select **Define all parameters as inputs**.
- 3 Under **Normalization**, select **Full** to facilitate full normalization so the sensitivities can be compared with each other. The **Sensitivity Settings** tab should resemble the following.

Sensitivity Settings | Simulation Settings | Export | Plots | M-Code

Normalization

None (no normalization)
 Half (normalization relative to numerator - species quantity)
 Full (full dedimensionalization)

Specify the input and output factors. To calculate the sensitivity of X with respect to Y, X is an output factor and Y is an input factor, i.e. $d[X]/d[Y]_0$.

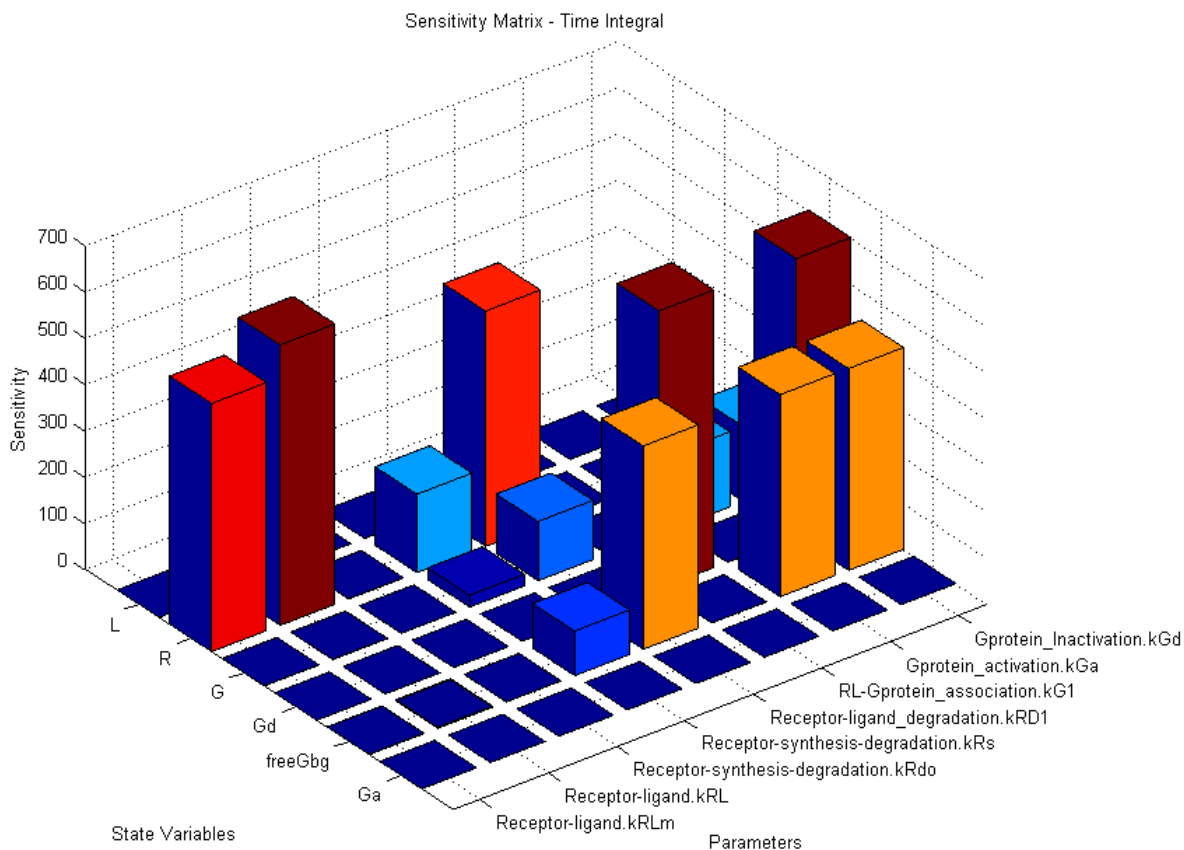
	Output	Input	Type	Name	Scope
1	<input checked="" type="checkbox"/>	<input type="checkbox"/>	species	L	unnamed
2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	species	R	unnamed
3	<input checked="" type="checkbox"/>	<input type="checkbox"/>	species	G	unnamed
4	<input checked="" type="checkbox"/>	<input type="checkbox"/>	species	Gd	unnamed
5	<input checked="" type="checkbox"/>	<input type="checkbox"/>	species	freeGbg	unnamed
6	<input checked="" type="checkbox"/>	<input type="checkbox"/>	species	Ga	unnamed
7	<input checked="" type="checkbox"/>	<input type="checkbox"/>	species	RL	unnamed
8	<input type="checkbox"/>	<input checked="" type="checkbox"/>	parameter	kRLm	Yeast_G_Protein_wt.[Receptor-ligand]
9	<input type="checkbox"/>	<input checked="" type="checkbox"/>	parameter	kRL	Yeast_G_Protein_wt.[Receptor-ligand]
10	<input type="checkbox"/>	<input checked="" type="checkbox"/>	parameter	kRdo	Yeast_G_Protein_wt.[Receptor-synthesis-degradation]
11	<input type="checkbox"/>	<input checked="" type="checkbox"/>	parameter	kRs	Yeast_G_Protein_wt.[Receptor-synthesis-degradation]
12	<input type="checkbox"/>	<input checked="" type="checkbox"/>	parameter	kRD1	Yeast_G_Protein_wt.[Receptor-ligand_degradation]
13	<input type="checkbox"/>	<input checked="" type="checkbox"/>	parameter	kG1	Yeast_G_Protein_wt.[RL-Gprotein_association]
14	<input type="checkbox"/>	<input checked="" type="checkbox"/>	parameter	kGa	Yeast_G_Protein_wt.Gprotein_activation
15	<input type="checkbox"/>	<input checked="" type="checkbox"/>	parameter	kGd	Yeast_G_Protein_wt.Gprotein_inactivation

- 4 Save the project by selecting **File > Save Project**.

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

Getting Results for Sensitivity Analysis

- 1 Click **Run**. The **Sensitivity Matrix Subplot**, which is the default plot for sensitivity analysis, opens.




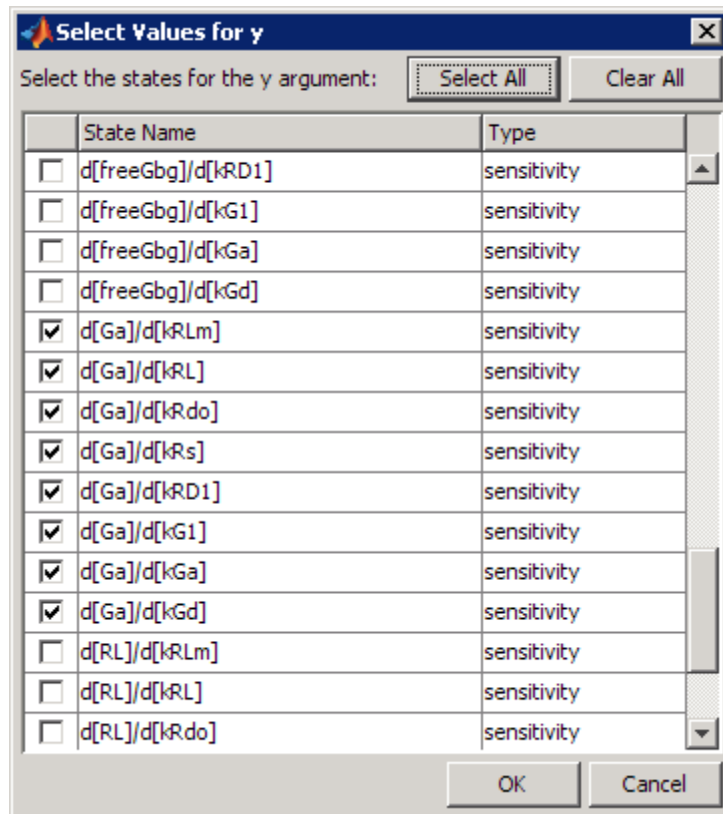
The plot results show for example that as expected, receptor R is sensitive to values of parameters involved in the receptor-ligand complex formation and receptor degradation. The sensitivity of Ga to the parameters in the model is not visible in the plot because there are other higher values for the time integral for the sensitivities.

To see the sensitivities for **Ga**, follow the next steps.

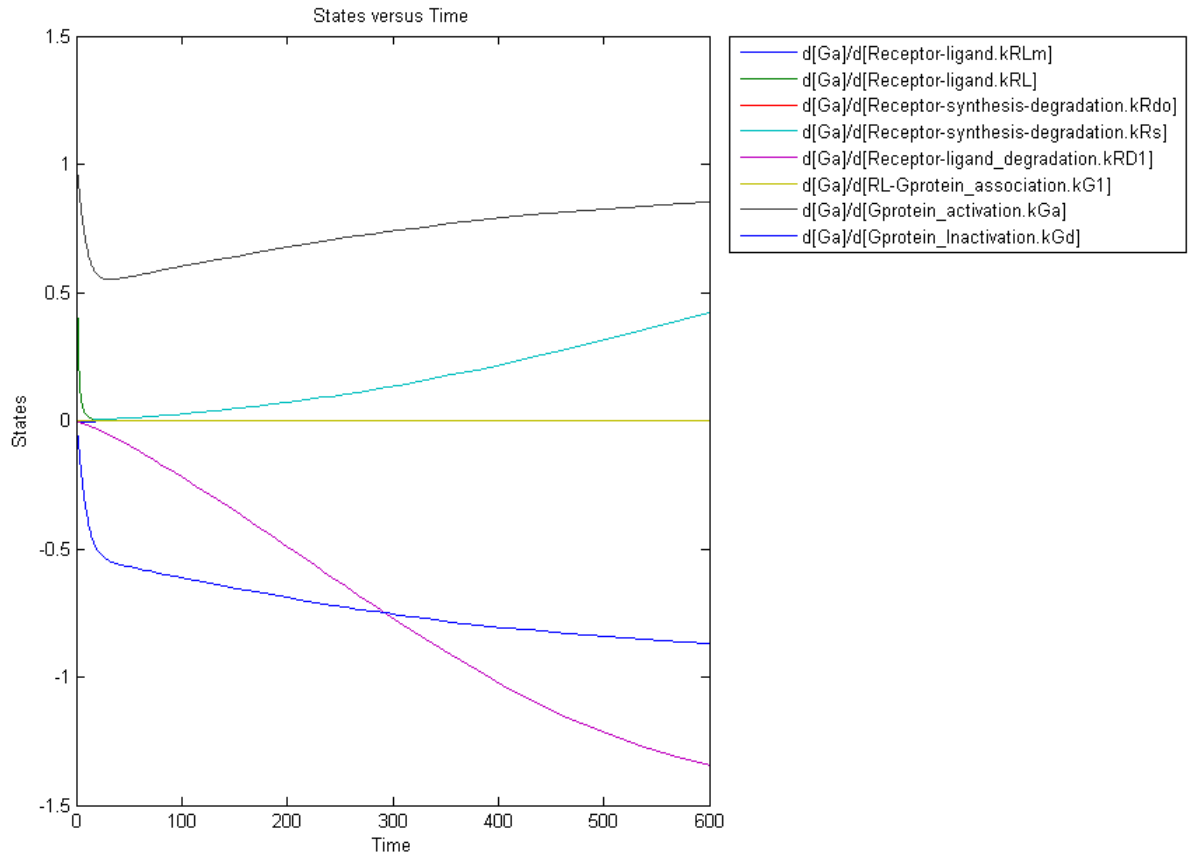
- 2** Close the figure window. Notice that the **Data** pane showing the most recent data is now open in the SimBiology desktop . If the **Data** pane is not open:
 - In the **Project Explorer**, under **Sensitivity Analysis**, click **Data (date)**.

In the **Data** pane you can plot the results for the species that are of interest, for example, **Ga**. But first, it is useful to save the data separately so that any future runs do not over write the existing data.

- 3** In the **Project Explorer**, under **Sensitivity Analysis**, right-click **Data** and select **Save Data**. The Save Data dialog box opens.
- 4** Specify a name for the saved data, for example, `sensitivity_ex1`, and click **Save**. The **Project Explorer** shows a new item with the saved data name under **Sensitivity Analysis**.
- 5** In the **Project Explorer**, click the saved data, for example, `sensitivity_ex1`, to open the **Data** pane for the saved data.
- 6** Click the **Plots** tab. Here, you can plot the results using the default **Sensitivity Matrix Subplot**, the other relevant plots for these results might be a **Time plot**, or the **Sensitivity Matrix Subplot Max** which plots the sensitivity matrix using maximum values of the sensitivities for the specified input and output factors.
- 7** In the **Plot Type** box, select **Time** and click **Add Plot Type**.
- 8** Select the new plot (second row), and in the **Arguments** section click . The Select Values for y dialog box opens.
- 9** Click **Clear All** and then select the check box for the rows containing the time-dependent derivatives of **Ga** with respect to each parameter.



- 10 Click **OK**.
- 11 Clear the **Create Plot** check box for the Sensitivity Matrix Subplot plot.
- 12 Click **Plot**. Your plot should following one.



From the previous plot you can see that Ga 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

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.

Command-Line Example — Calculating Sensitivities

In this section...

“Overview” on page 3-39

“Loading and Configuring the Model for Sensitivity Analysis” on page 3-40

“Performing Sensitivity Analysis” on page 3-41

“Extracting and Plotting Sensitivity Data” on page 3-41

“See Also” on page 3-44

Overview

This example uses the G protein model from the “Model of the Yeast Heterotrimeric G Protein Cycle” example to illustrate SimBiology sensitivity analysis options.

This table lists 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	$L + R \rightleftharpoons RL$	kRRL, kRRLm
2	Heterotrimeric G protein formation	$Gd + Gbg \rightarrow G$	kG1
3	G protein activation	$RL + G \rightarrow Ga + Gbg + RL$	kGa
4	Receptor synthesis and degradation	$R \rightleftharpoons \text{null}$	kRdo, kRS
5	Receptor-ligand degradation	$RL \rightarrow \text{null}$	kRD1
6	G protein inactivation	$Ga \rightarrow Gd$	kGd

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:

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

To calculate these sensitivities:

- 1** Load the model and set the `SpeciesOutputs` property to *Ga*.
- 2** Set the `ParameterInputFactors` property to all the parameters in the model.
- 3** Set the `SensitivityAnalysis` property to `true` and simulate the model.
- 4** Plot the data.

The following sections explain the details of the procedure.

You can also view a demo that shows sensitivity analysis of this model by typing the following at the command line:

```
edit gprotein
```

Loading and Configuring the Model for Sensitivity Analysis

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

```
sbioloadproject gprotein_norules m1
```

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

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

- 3** Set the `SpeciesOutputs` property to calculate the sensitivities for the species `Ga` in `m1`.

```
Ga = m1.Compartments(1).Species(6);
set(csObj.SensitivityAnalysisOptions, 'SpeciesOutputs', Ga);
```

- 4** Retrieve all the parameters in the model and store the vector in a variable.

```
% The function sbioselect allows you to query by Type
pif = sbioselect(m1, 'Type', 'parameter');
```

- 5** Set the `ParameterInputFactors` property of the `SensitivityAnalysisOptions` object to the variable containing the parameters.

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

Performing Sensitivity Analysis

- 1** Enable sensitivity analysis in the configuration set object (`csObj`) by setting the `SensitivityAnalysis` option to `true`.

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

- 2** Set the `Normalization` property of the `SensitivityAnalysisOptions` object to perform `'Full'` normalization.

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

- 3** Simulate the model and return the data to a `SimData` object (`simDataObj`).

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

For more information about normalization see [Normalization](#) in the *SimBiology Reference*.

Extracting and Plotting Sensitivity Data

You can extract sensitivity results using `sbiogetsensmatrix`. 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).

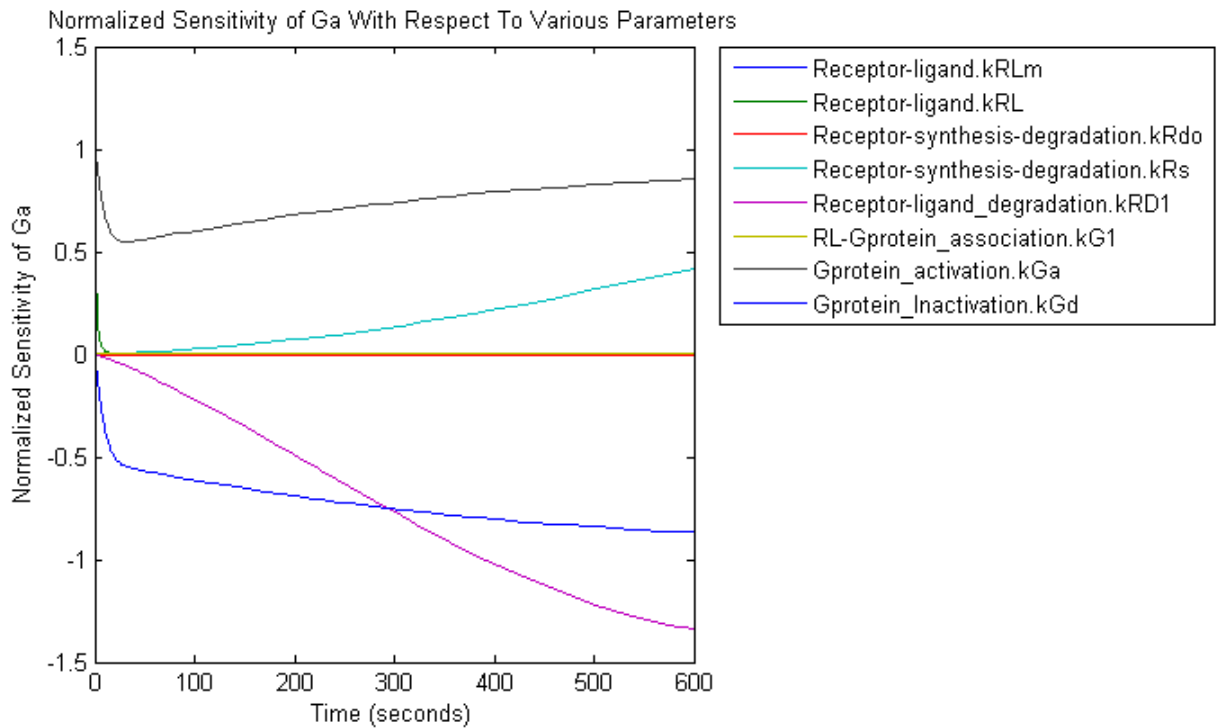
```
[T, R, snames, ifacs] = getsensmatrix(simDataObj);
```

- 2** Reshape `R` into columns of input factors to facilitate visualization and plotting.

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

- 3** After extracting the data and reshaping the matrix, you can now 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 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

See Also

For information about...	See...
Configuring simulation settings	“Performing Simulations at the Command Line” on page 2-2
Normalizing the data	Normalization in the SimBiology Reference
Selecting model component objects by querying the model as shown in Step 4 in “Loading and Configuring the Model for Sensitivity Analysis” on page 3-40	<code>sbioselect</code>

Parameter Estimation

In this section...
“About Parameter Estimation” on page 3-45
“SimBiology Parameter Estimation” on page 3-45

About Parameter Estimation

Parameter estimation lets you estimate the values of unknown parameters in a model. This is especially useful when some parameters cannot be measured experimentally .

SimBiology Parameter Estimation

You can estimate a single parameter or all parameters in your model using the `sbioparamestim` function. Parameter estimation uses the optimization functions in MATLAB, Optimization Toolbox™, and Genetic Algorithm and Direct Search Toolbox™ to enable estimation.

Optimization Toolbox, and Genetic Algorithm and Direct Search Toolbox are not required for you to use `sbioparamestim`. If you have these products installed, you can specify optimization methods from these toolboxes as arguments for the `sbioparamestim` function. If you do not have these products installed, `sbioparamestim` uses the MATLAB function `fminsearch` by default.

For more information, see `sbioparamestim` in the SimBiology Reference. For an example, see “Command-Line Example — Parameter Estimation” on page 3-46

Command-Line Example – Parameter Estimation

In this section...

“About the Example Model” on page 3-46

“Importing Target Experimental Data” on page 3-47

“Simulating the G Protein Model” on page 3-47

“Estimating a Parameter (kGd) in the G Protein Model” on page 3-50

“Simulating and Plotting Results Using the Estimated Parameter” on page 3-53

“Estimating Other Parameters in the G Protein Model” on page 3-54

About the Example Model

This example uses a G protein model built in the “Model of the Yeast Heterotrimeric G Protein Cycle” tutorial to illustrate parameter estimation. The study used to build this model (Yi et al., 2003) reported the estimated value of parameter kGd as 0.11 for the wild-type strain.

In “Desktop Example — Calculating Sensitivities” on page 3-29, the analysis showed that Ga is sensitive to parameters kGd, kRS, kRD1, and kGa.

This example first shows you the estimation of the parameter kGd and how it affects the model. Next the same example shows how you can estimate parameters kGd, kRS, kRD1, and kGa to obtain a better fit to the experimental data.

You can also access a demo that shows you parameter estimation in this model by typing the following at the command line:

```
gprotein
```

Loading the Model

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

```
sbioloadproject gprotein_norules m1
```

2 Display reaction information.

```
m1.Reactions
```

```
SimBiology Reaction Array
```

Index:	Reaction:
1	L + R <-> RL
2	R <-> null
3	RL -> null
4	Gd + freeGbg -> G
5	RL + G -> Ga + freeGbg + RL
6	Ga -> Gd

Importing Target Experimental Data

For this example, you will store the experimental data in a variable in the MATLAB workspace.

The study used for this example (Yi et al., 2003) reports the experimental data in a plot as the fraction of active G (Ga). Calculate and store the amount of Ga in a variable.

- 1 The initial amount of total G protein is 1000 molecules. The values for the fraction of active G are stored in Ga_frac. Ga_target contains the values of Ga over time.

```
Gt = 10000;
Ga_frac = [0 0.35 0.4 0.36 0.39 0.33 0.24 0.17 0.2]';
Ga_target = Ga_frac * Gt;
```

- 2 The time data for the experimental results is stored in t_expt.

```
t_expt = [0 10 30 60 110 210 300 450 600]';
```

Simulating the G Protein Model

Display the configuration set that is loaded with the G protein cycle model and simulate the model.

- 1 Display the configuration set options in the model.

```
getconfigset(m1)

Configuration Settings - default (active)
  SolverType:          ode15s
  StopTime:            600.000000

  SolverOptions:
    AbsoluteTolerance: 1.000000e-006
    RelativeTolerance: 1.000000e-003
    SensitivityAnalysis: false

  RuntimeOptions:
    StatesToLog:        6

  CompileOptions:
    UnitConversion:    false
    DimensionalAnalysis: false

  SensitivityAnalysisOptions:
    InputFactors:      0
    Outputs:           0
```

The model configuration set has `StopTime` set to 600 seconds.

- 2 Simulate the model and return the results to a `SimData` object.

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

- 3 Retrieve the time and state data.

```
[t_orig, Ga_orig] = selectbyname(simDataObj, 'Ga');
```

Calculating R-Square for the G Protein Model

R-square measures how successful the fit is in explaining the variation of the data. In other words, R-square is the square of the correlation between the response values and the predicted response values.

- 1 Calculate the sum of squares about the mean (SST).

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

- 2 Interpolate the data to get time points that match the time points in the experimental data with the cubic interpolation method.

```
Ga_resampled = interp1(t_orig, Ga_orig, t_expt, 'cubic');
```

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

```
sse = norm(Ga_target - Ga_resampled)^2;
```

- 4 Calculate R-square for the simulation data before parameter estimation.

```
rsquare_orig = 1-sse/sst
```

```
rsquare_orig =
```

```
0.8967
```

For more information about R-square, see “Goodness-of-Fit Statistics” in the Curve Fitting Toolbox documentation. For more information about the functions used here, see `interp1`, `norm`.

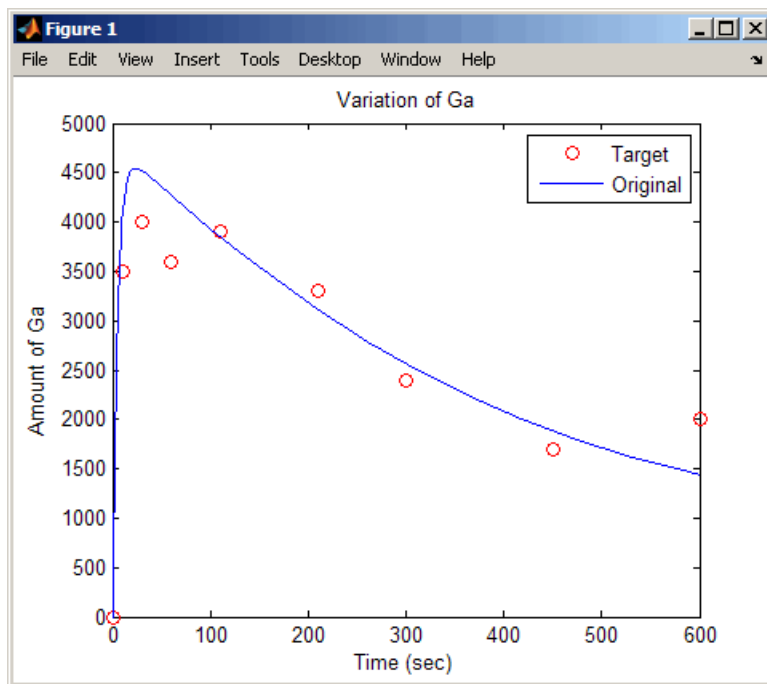
Plotting the Experimental Results and Simulation Data

- 1 Plot the experimental data for Ga.

```
plot(t_expt, Ga_target, 'ro');
title('Variation of Ga');
xlabel('Time (sec)');
ylabel('Amount of Ga');
legend('Target');
```

- 2 Plot the simulation data in the same plot.

```
hold on;
plot(t_orig, Ga_orig);
legend('Target', 'Original');
```



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

Estimating a Parameter (kGd) 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 kGd and calculates the R-square value with the new estimate.

- 1 Set up the parameter to estimate and the state to match.

```
param_to_tune = sbioselect(m1,'Type',...
    'parameter','Name','kGd');
Ga = sbioselect(m1,'Type','species','Name','Ga');
```

- 2 Switch on information about iterations in the display to see how optimization is progressing.

```
opt1 = optimset('Display','iter');
```

- 3** Use the current values of parameters in the model as the starting values for optimization. Use the default optimization method ('lsqcurvefit' if you have Optimization Toolbox installed).

```
[k_new1, result1] = sbioparamestim(m1, t_expt, ...
    Ga_target, Ga, param_to_tune, {}, {'lsqcurvefit',opt1});
```

The results from optimization should look similar to this, but may not match exactly:

Iteration	Func-count	f(x)	step	optimality	CG-iterations
0	2	1.4264e+006		2.84e+007	
1	4	1.11306e+006	0.0105776	8.23e+006	1
2	6	1.11306e+006	0.0045504	8.23e+006	1
3	8	1.11306e+006	0.0011376	8.23e+006	0
4	10	1.11183e+006	0.0002844	6.93e+005	0
5	12	1.11183e+006	0.0002844	6.93e+005	1
6	14	1.11183e+006	7.10999e-005	6.93e+005	0
7	16	1.11183e+006	1.7775e-005	6.93e+005	0
8	18	1.11183e+006	4.44375e-006	6.93e+005	0
9	20	1.11183e+006	1.11094e-006	6.93e+005	0
10	22	1.11183e+006	2.77734e-007	6.93e+005	0

Optimization terminated: norm of the current step is less than OPTIONS.TolX.

Alternatively, if you do not have Optimization Toolbox, the following command lets you use 'fminsearch' in MATLAB.

```
[k_new1, result1] = sbioparamestim(m1, ...
    t_expt, Ga_target, Ga, param_to_tune, {}, {'fminsearch',opt1});
```

The results from optimization should look similar to this, but may not match exactly:

Iteration	Func-count	min f(x)	Procedure
0	1	1194.32	
1	2	1091.86	initial simplex
2	4	1054.2	reflect
3	6	1054.2	contract outside

4	8	1054.2	contract inside
5	11	1054.2	shrink
6	13	1054.2	contract outside
7	15	1053.95	contract outside
8	17	1053.95	contract inside
9	19	1053.86	reflect
10	21	1053.86	contract inside
11	23	1053.86	contract inside
12	25	1053.84	reflect
13	27	1053.84	contract inside
14	29	1053.82	reflect
15	31	1053.82	contract inside
16	33	1053.34	reflect
17	36	1053.34	shrink
18	38	1053.32	reflect
19	40	1053.32	contract inside
20	42	1053.32	contract inside
21	44	1053.32	contract inside
22	46	1053.32	contract outside
23	48	1053.32	contract inside
24	51	1053.32	shrink
25	53	1053.32	contract outside

Optimization terminated:

the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-004
and F(X) satisfies the convergence criteria using OPTIONS.TolFun of 1.000000e-004

- 4** Calculate the R-Square value with the new estimate obtained with 'lsqcurvefit'. The fval field in result1 contains the value of SSE.

```
sse = result1.fval;  
rsquare1 = 1-sse/sst
```

```
rsquare1 =
```

```
0.9195
```


Simulating and Plotting Results Using the Estimated Parameter

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

- 1 Use a model `Variant` to store the estimated value of `kGd`.

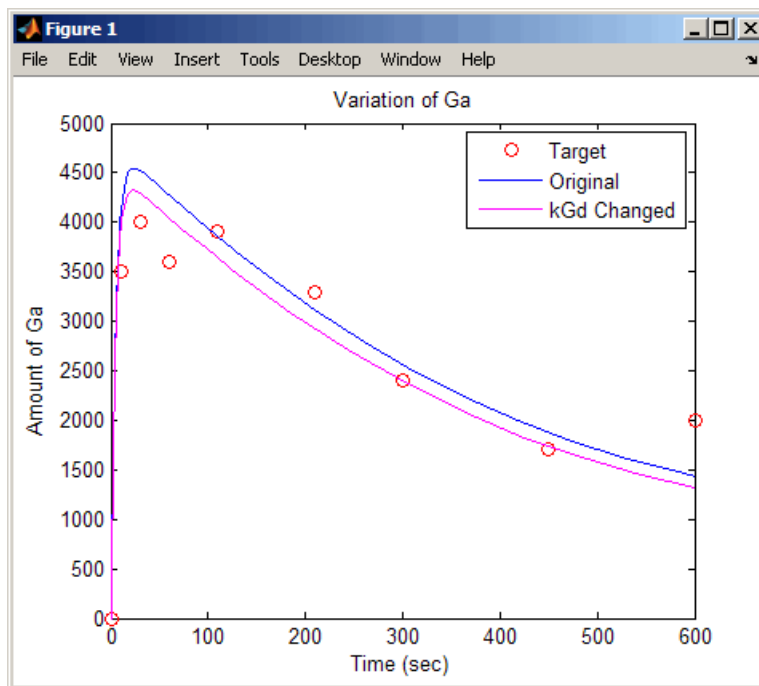
```
estvarObj = addvariant (m1, 'Optimized kGd');  
addcontent (estvarObj, {'parameter', 'Gprotein_Inactivation.kGd', ...  
    'Value', k_new1});
```

- 2 Apply the value stored in the `Variant`, simulate the model, and get the results.

```
simDataObj1 = sbiosimulate(m1, estvarObj );  
[t1, Ga1] = selectbyname(simDataObj1, 'Ga');
```

- 3 Plot the data and compare. If you have left the previous figure open, since `hold` is on, this plot will appear in that figure to facilitate the comparison.

```
plot(t1, Ga1, 'm-');  
legend('Target', 'Original', 'kGd Changed');
```



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 (“Command-Line Example — Calculating Sensitivities” on page 3-39) showed that G_a is sensitive to parameters kGd , kR_s , $kRD1$, and kG_a . Based on this data, this tutorial shows you how to estimate these parameters. The sensitivity data is presented in “Extracting and Plotting Sensitivity Data” on page 3-41.

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 Find the indices for each of the parameters to estimate.

```
params = sbioselect(m1, 'Type', 'parameter')
```

SimBiology Parameter Array

Index:	Name:	Value:	ValueUnits:
1	kRLm	0.01	
2	kRL	3.32e-018	
3	kRdo	0.0004	
4	kRs	4	
5	kRD1	0.004	
6	kG1	1	
7	kGa	1e-005	
8	kGd	0.11	

Note that the required parameter indices are 4, 5, 7, and 8.

- 2 Set the parameter array for estimation.

```
param_to_tune = params([4 5 7 8]);
```

- 3 Switch on information about iterations in the display to see how optimization is progressing.

```
opt2 = optimset('Display','iter');
```

Note `fminsearch` performs many more iterations and therefore takes more time in the next step.

- 4 Estimate the parameters. Use the current values of parameters in the model as the starting values for optimization. Use the default optimization method ('lsqcurvefit') if you have Optimization Toolbox installed. Note that the `param_to_tune` argument now contains the array of parameters to be estimated.

```
[k_new2, result2] = sbioparamestim(m1, t_expt,...
    Ga_target, Ga, param_to_tune, {}, {'lsqcurvefit',opt2});
```

The results from optimization should look similar to this, but may not match exactly:

Norm of First-order

Iteration	Func-count	f(x)	step	optimality	CG-iterations
0	5	1.4264e+006		3.56e+011	
1	10	588669	2.48973	8.5e+010	0
2	15	555508	0.638157	1.68e+011	0
3	20	555508	5.45966	1.68e+011	0
4	25	555508	1.36491	1.68e+011	0
5	30	555508	0.341229	1.68e+011	0
6	35	555508	0.0853071	1.68e+011	0
7	40	555508	0.0213268	1.68e+011	0
8	45	555508	0.0053317	1.68e+011	0
9	50	555508	0.00133292	1.68e+011	0
10	55	555508	0.000333231	1.68e+011	0
11	60	555508	8.33078e-005	1.68e+011	0
12	65	555508	2.08269e-005	1.68e+011	0
13	70	555508	5.20673e-006	1.68e+011	0
14	75	555508	1.30168e-006	1.68e+011	0
15	80	555508	3.90253e-007	1.68e+011	0

Local minimum possible.

lsqnonlin stopped because the size of the current step is less than the default value of the step size tolerance.

Alternatively, if you do not have Optimization Toolbox the following command lets you use 'fminsearch' in MATLAB:

```
[k_new2, result2] = sbioparamestim(m1, t_expt, ...
    Ga_target, Ga, param_to_tune, {}, {'fminsearch',opt2});
```

- 5** Compare original parameter values and the estimated parameter values obtained with 'lsqcurvefit'.

```
% Original parameter values.
param_to_tune
```

```
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.
```

```
k_new2 =
```

```
5.8515
```

```
0.0033
```

```
0.0000
```

```
0.1293
```

- 6** Calculate the R-Square value with the new estimates obtained with 'lsqcurvefit'.

```
sse = result2.fval;
```

```
rsquare2 = 1-sse/sst
```

```
rsquare2 =
```

```
0.9598
```

Moiety Conservation

In this section...

“Introduction to Moiety Conservation” on page 3-58

“Algorithms for Conserved Cycle Calculations” on page 3-58

Introduction to Moiety Conservation

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

Consider the 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 and real-world example of a conserved moiety is adenine in its various forms ATP, ADP, AMP, etc. Finding and analyzing conserved moieties may yield insights into the structure and function of a biological network. In addition, for the quantitative modeler, conserved moieties represent dependencies which can be removed to reduce a system's dimensionality, or number of dynamic variables. In the simple network above, for example, in principle, it is only necessary to calculate the time courses for A and B; once this is done, C is fixed by the conservation relation.

Algorithms for Conserved Cycle Calculations

The `sbioconsmoiety` function lets you calculate a complete set of linear conservation relations for the species in a SimBiology model object.

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

- When you specify 'qr', `sbioconsmoiety` uses an algorithm based on QR factorization. From a numerical standpoint, this is the most efficient and reliable approach.
- When you specify 'rreduce', `sbioconsmoiety` uses an algorithm based on row reduction, which yields better numbers for smaller models. This is the default.
- When you specify 'semipos', `sbioconsmoiety` returns conservation relations in which all the coefficients are greater than or equal to 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 may 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 the "Introduction to Moiety Conservation" on page 3-58 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 .

The matrix N can be split up into the following independent and dependent parts:

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

where R in the independent submatrix N_R denotes 'reduced'; the $(n-k)$ -by- k link matrix L_0 is defined so that $N_D = L_0 * N_R$. 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.

Command-Line Examples — Determining Conserved Moieties

In this section...

“G Protein Example” on page 3-61

“Mitotic Oscillator Example” on page 3-64

G Protein Example

- 1 Load the project `gprotein_norules.sbproj`

```
sbioloadproject gprotein_norules
```

MATLAB populates the workspace with the model objects from the project and lists the objects as `m1` and `m2`.

The project contains two models, one for the wild-type strain (stored in variable `m1`), and one for the mutant strain (stored in variable `m2`).

- 2 Display the species information.

```
m1.Compartments.Species
```

```
SimBiology Species Array
```

Index:	Compartment:	Name:	InitialAmount:	InitialAmountUnits:
1	unnamed	L	6.022e+017	
2	unnamed	R	10000	
3	unnamed	G	7000	
4	unnamed	Gd	3000	
5	unnamed	freeGbg	3000	
6	unnamed	Ga	0	
7	unnamed	RL	0	

- 3 Display reaction information.

```
m1.Reactions
```

```
SimBiology Reaction Array
```

```

Index:   Reaction:
1        L + R <-> RL
2        R <-> null
3        RL -> null
4        Gd + freeGbg -> G
5        RL + G -> Ga + freeGbg + RL
6        Ga -> Gd

```

- 4 Use the simplest form of the `sbioconsmoiety` function and display the results.

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

```
g =
```

```

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

```

```
sp =
```

```

'L'
'R'
'G'
'Gd'
'freeGbg'
'Ga'
'RL'

```

The columns in `g` are labeled by the species `sp`. Thus the second row describes the conserved relationship, $G + Gd + Ga$.

- 5 Use the semipositive algorithm to explore conservation relations in the model. The `'p'` specifies that the output should be in the form of a printed cell array.

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

```
ans =
```

```
'G + freeGbg'
'G + Gd + Ga'
```

As expected, the function predicts the conservation relationship for the different forms of the G protein complex.

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

```
[SI,SD,L0,NR,ND] = sbioconsmoiety(m1, 'link');
```

- 7** Show the list of independent species.

```
SI
```

```
SI =
```

```
'R'
'G'
'RL'
'Gd'
'L'
```

- Show the list of dependent species.

```
SD
```

```
SD =
```

```
'freeGbg'
'Ga'
```

- Show the link matrix relating SD and SI.

```
L0
```

```
L0 =
```

```
(1,2)    -1
(2,2)    -1
(2,4)    -1
```

- Show the independent stoichiometry matrix, N_R .

NR

NR =

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

- Show the dependent stoichiometry matrix, N_D .

ND

ND =

(1,4)	-1
(1,5)	1
(2,5)	1
(2,6)	-1

Mitotic Oscillator Example

- 1 Load the Goldbeter Mitotic Oscillator model.

```
sbioloadproject Goldbeter_Mitotic_Oscillator_with_reactions
```

MATLAB populates the workspace with the model object from the project and lists the object as m1.

- 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 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.

```
[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, this is because `AA` is constant (`ConstantAmount = 1`).

- 5** Use the semipositive algorithm to explore conservation relations in the model.

```
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.

L0

L0 =

```
(1,2)    -1
(2,3)    -1
```

10 Show the independent stoichiometry matrix, N_R .

NR

NR =

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

11 Show the dependent stoichiometry matrix, N_D .

ND

ND =

```
(1,4)   -1
(1,5)    1
(2,6)   -1
(2,7)    1
```

Desktop Example – Creating Custom Analysis

In this section...

“About Custom Analysis” on page 3-68

“Open the Example Project” on page 3-68

“Setting Up a Custom Task” on page 3-69

“Parameter Estimation Using Custom Task” on page 3-69

About Custom Analysis

You can perform custom analysis by setting up **Custom Tasks**, which are user-defined elements that let you script tasks in combination with each other and with other data processing functions. You can use functions from any of the products in your license. Custom tasks let you work within the context of the SimBiology desktop and use features like plotting, and exporting data within the desktop, while being able to specify custom processing and analysis of the model data.

Open the Example Project

This example shows you how to set up parameter estimation in the desktop for the G protein model shown in the following section:

“Model of the Yeast Heterotrimeric G Protein Cycle”.

Opening and Saving the Example Model

- 1 Load the example project by typing the following at the command line:

```
sbioloadproject gprotein
```

The model is stored in a variable called m1.

- 2 Open the SimBiology desktop with the model loaded by typing:

```
simbiology(m1)
```

The desktop opens with **Model Session-Heterotrimeric_G_Protein_wt**.

- 3** Select **File > Save Project As**. The Save SimBiology Project dialog box opens.
- 4** Specify a name (for example, gprotein_ex) and location for your project, and click **Save**.

Setting Up a Custom Task

- 1** From the **Tasks** menu select **Add Model Task to Heterotrimeric_G_Protein_wt > Create custom analysis**. The SimBiology desktop adds the custom task to the **Project Explorer**, and opens the task pane.
- 2** In the **Custom Settings** tab you can define your own script that you can run in the desktop.

The next section shows an example of how to add the script for custom parameter analysis.

Parameter Estimation Using Custom Task

- 1** In the **Custom Settings** tab, replace the default function declaration statement with the following:

```
function data = custom(modelobj)
%Parameter Estimation of a G Protein Model

%Target Data

%Preprocess the experimental data

% The estimated amount of total G protein (Gt) is 10000
Gt = 10000;
t_expt = [0 10 30 60 110 210 300 450 600]';
Ga_frac = [0 0.35 0.4 0.36 0.39 0.33 0.24 0.17 0.2]';
Ga_target = Ga_frac * Gt;

fh = figure;
plot(t_expt, Ga_target, 'ro');
grid on;
```

```

title('Variation of Ga');
xlabel('Time (sec)');
ylabel('Amount of Ga');
lgnd0 = 'Target';
legend(lgnd0);

% Simulate the model as is to see how Ga in the model varies with
% time. Also calculate the original R-square value
simdata_orig = sbiosimulate(modelobj);
[t_orig, Ga_orig] = selectbyname(simdata_orig, 'Ga');
sst = norm(Ga_target - mean(Ga_target))^2;
Ga_resampled = interp1(t_orig, Ga_orig, t_expt, 'cubic');
sse = norm(Ga_target - Ga_resampled)^2;
rsquare_orig = 1-sse/sst;

% Plot the original species data
figure(fh);
hold on;
plot(t_orig, Ga_orig);
str_orig = sprintf('R^2 = %6.4f', rsquare_orig);
grid on;
lgnd0 = 'Target';
lgnd_orig = ['Wild-Type model, ', str_orig];
legend(lgnd0, lgnd_orig);

% Parameter to Estimate
param_to_tune = sbioselect(modelobj, 'Type', 'parameter', 'Name', 'kGd');

% Match experimental (target) species data to model species data
Ga = sbioselect(modelobj, 'Type', 'species', 'Name', 'Ga');

% Estimate the parameter
[k_new1, result1] = sbioparamestim(modelobj, t_expt, ...
    Ga_target, Ga, param_to_tune);

% Simulation Results of Using the Estimated Parameter Value
%
% Use the estimated value of kGd and see how it affects simulation
% results. Before changing the value, save it to reset it later. We
% will also compute the new R-square value.

```

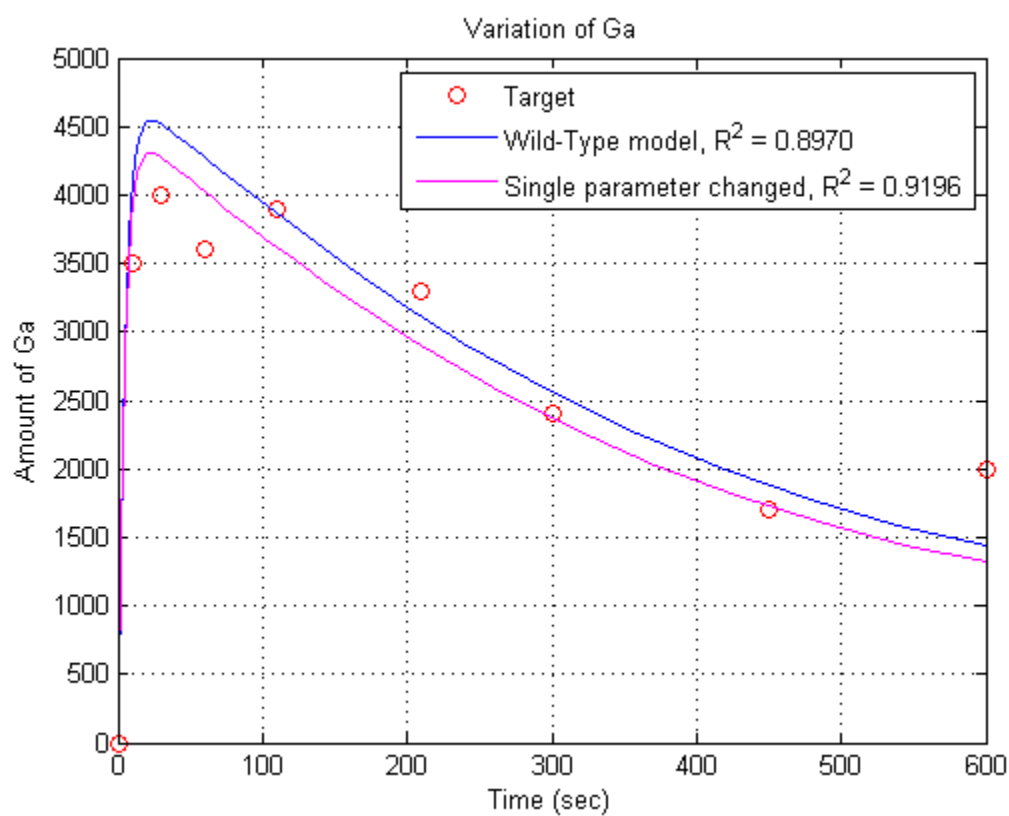
```
%
kGd0 = get(param_to_tune, 'Value');
set(param_to_tune, 'Value', k_new1);
simdata1 = sbiosimulate(modelobj);
[t1, Ga1] = selectbyname(simdata1, 'Ga');
sse = result1.fval;
rsquare1 = 1-sse/sst;

% Plot the data and compare
figure(fh);
plot(t1, Ga1, 'm-');
str1 = sprintf('R^2 = %6.4f', rsquare1);
lgnd_kGd = ['Single parameter changed, ', str1];
legend(lgnd0, lgnd_orig, lgnd_kGd);

%Reset the value of the parameter
set(param_to_tune, 'Value', kGd0);

% Reference
%
% Tau-Mu Yi, Hiroaki Kitano, and Melvin I. Simon. PNAS (2003) vol.
% 100, 10764-10769.
```

- 2** Click  to run the task. The plot should look similar to this:



Visualizing Results Using Plot Types

In this section...

“About Plot Types” on page 3-73

“How Plot Types Work” on page 3-75

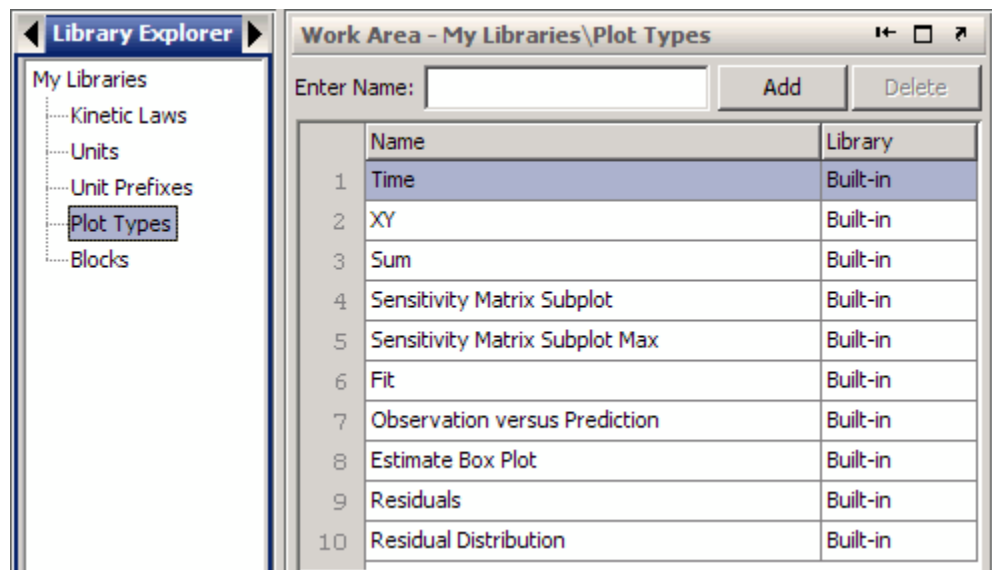
“Summary of Built-In Plot Types” on page 3-79

“When to Use User-Defined Plot-Types” on page 3-80

About Plot Types

Plot types are MATLAB functions for visualization, which you access on the SimBiology desktop. Plot types take input arguments specifying the data source and model quantities, for example, species, parameters, or compartments that you want to visualize. Plot types can also include additional inputs that specify, for example, line style or markers.

The **Library Explorer** contains built-in and user-defined plot types.



Visualize analysis task results with any plot type in the library. The desktop displays various default plots in the context of each task. For example:

Task	Default Plot
Simulation	Time
Sensitivity Analysis	Sensitivity Matrix Subplot
Scan	Time plots showing results for each scan in a separate subplot, and one Time plot with each scan in the same axes

The analysis tasks are also function M-files that the SimBiology desktop runs when you click the **Run** button. Each analysis task outputs some form of data (for example, a `SimData` object, and for some tasks such as parameter fitting, additional statistics). So, when you run a task such as simulation or sensitivity analysis, the desktop runs the analysis task M-files and feeds the relevant data output to the plot M-files. The default plots appear after task completion. You can see this configuration in the **M-Code** tab on any given task pane.

You can add or change the plots that are generated at the end of a task. If needed, either extend the built-in plot types or write your own M-code to create new plot types.

How Plot Types Work

Consider the built-in **Time** plot. The following figure shows the **Time** plot in the **Library Explorer**.

The screenshot shows the 'Work Area - My Libraries\Plot Types' window. At the top, there is an 'Enter Name:' field with 'Add' and 'Delete' buttons. Below this is a table listing plot types:

Name	Library
13 Time	Built-in

Below the table is the 'Plot Type Code:' section, which contains the following MATLAB code:

```
function Time(tobj, y, plotStyle)
%TIME Plots states versus time.
%
% TIME(TOBJ, Y, PLOTSTYLE) plots the results of the simulation for the
% species with the specified Y versus time. If PLOTSTYLE is 'one axes'
% then data from each run is plotted into one axes. If PLOTSTYLE is
% 'subplot' or 'trellis' then data from each run is plotted into its own
% subplot. If Y is '<all>' then all data will be plotted.
%
% See also GETDATA, SELECTBYNAME.

if (length(tobj) > 1)
    switch (plotStyle)
        case 'one axes'
            sbioplot(tobj, @timeplotdata, [], y);
        case 'subplot'
            % Include a legend with the subplot.
            sbiosubplot(tobj, @timesubplotdata, [], y, true);
        case 'trellis'
            sbiotrellis(tobj, @timesubplotdata, [], y);
    end
else
```

At the bottom of the window is the 'List of Arguments Passed to Plot Type Code:' table:

Variable Name	Argument Type	Default Value
tobj	Data Source	
y	Data Names {Source = tobj; Types = Column names, Sensitivities, State names}	<all>
plotStyle	Enumerations{one axes, subplot, trellis}	one axes

- The function line states that **Time** takes the input arguments **tobj**, **y**, and **plotStyle**.
- The remaining M-code specifies how to retrieve and plot the data.
- The **List of Arguments Passed to Plot Type Code** table specifies the accepted values for each input argument. When you use a plot type in a

model task, the SimBiology desktop uses the values defined in this table to prompt you for the accepted values when using the plot-type.

For a built-in plot type like `Time`, the values in the table are read-only. For a user-defined plot type, define these values when you define the plot type (dialog boxes guide you through this process). Refer to “Desktop Example — Creating and Using Plot Types” on page 3-81 for more information on configuring user-defined plot types.

For example, consider the accepted value for each input argument in the `Time` plot type:

- `tobj` — `Data Source`; the results from an analysis, or an imported data set.
- `y` — `Data Names {Source = tobj; Types = Column names, Sensitivities, State names}`; accepts the names of the states, or the names of the sensitivities, or the headers of the columns of imported data. The source of these names is `tobj`, or in other words the simulation results or an imported data set.
- `plotStyle` — `Enumerations{one axes, subplot, trellis}`; accepts one of three values.

The accepted value for each argument appears when you use the plot. For example, for `Time`, the input argument `y` accepts `Data Names {Source = tobj; Types = Column names, Sensitivities, State names}`. So when you use the plot in a task (here a **Simulation** task), the **Plots** tab shows available values as follows.

Simulation Settings | Export | **Plots** | M-Code

Plot Type:

Create...	Plot Behavior	Plot Type	Arguments
1 <input checked="" type="checkbox"/>	<input type="text" value="New figure"/>	Time	tobj = Simulation Results; y = <all>; ...

Arguments

tobj:
Simulation Results

y (Select the button or click the field below to configure the value):
<all>

plotStyle:
one axes

How to Use t

TIME Plots states

TIME(TOBJ, Y, P

versus time. If P

PLOTSTYLE is 's

'<all>' then all da

See also GETDATA, SELECTDNAME.

Select Values for y

Select the states for the y argument:

	State Name	Type
<input checked="" type="checkbox"/>	unnamed.G	species
<input checked="" type="checkbox"/>	unnamed.Gd	species
<input checked="" type="checkbox"/>	unnamed.Ga	species
<input checked="" type="checkbox"/>	unnamed.Ga_frac	species
<input checked="" type="checkbox"/>	unnamed.RI	series

Click , to open the Select Values for y dialog.

Types of Accepted Values for Plot Type Arguments

Option	Description
Data Source	Use this option when the argument represents simulation results or imported data (External Data).
Task Results	Use this option when an analysis task generates a structure to store the results, and the plotting function accesses these fields. When an analysis task returns multiple outputs (for example parameter fitting), M-code for the analysis task stores the results in a <i>taskResults</i> structure containing a field for each output argument from the task.
Numeric Value	Use this option when the argument accepts numeric values in a specified range, or from a specified set of values.
Data Names	Use this option when the argument accepts names of states that are logged in the Configuration Settings pane, parameter names, sensitivities, or columns from data. when you use the plot in a task, the desktop lists the available names.
Enumerations	Use this option when the argument accepts an enumerated list of values that appear in a list box when you use the plot type.
Any Value	Use this option when the argument accepts a string. When you use the plot in a task pane, any value entered is valid.

Summary of Built-In Plot Types

Plot Type	Description
Time	<p>Specify one or more states or column names to plot against time. By default Time plots all states or column names against time, specified by the string '<all>'.</p> <p>By default Time plots the data from all runs into a single set of axes. Choose an option from the plotStyle list to plot the results in one plot, subplots, or trellis plots.</p>
XY	<p>Specify two states or column names to plot against each other.</p> <p>By default XY plots the data from all runs into a single set of axes. Choose an option from the plotStyle list to plot the results in one plot, subplots, or trellis plots.</p>
Sum	<p>Plot the sum of the simulation results of one or more states against time.</p> <p>By default Sum plots the data from all runs into a single set of axes. Choose an option from the plotStyle list to plot the results in one plot, subplots, or trellis plots.</p>
Sensitivity Matrix Subplot	Plot the sensitivity matrix using the time integral for the sensitivities for the specified input and output factors.
Sensitivity Matrix Subplot Max	Plot the sensitivity matrix using maximum values of the sensitivities for the specified input and output factors.
Fit	Plot observed and predicted values versus time.
Observation versus Prediction	Plot observed values versus predicted values.
Estimate Box Plot	Create a box plot for the estimated parameters.
Residuals	Plot the residuals versus time, group, or predicted value.
Residual Distribution	Create a normal probability plot of the residuals.

When to Use User-Defined Plot-Types

Create your own plot types using the **Library Explorer**.

Use user-defined plot types when:

- The visualization of results from a model requires plotting that extends or customizes built-in plot types.
- You want to use MATLAB functions that are not available in the built-in plot types.

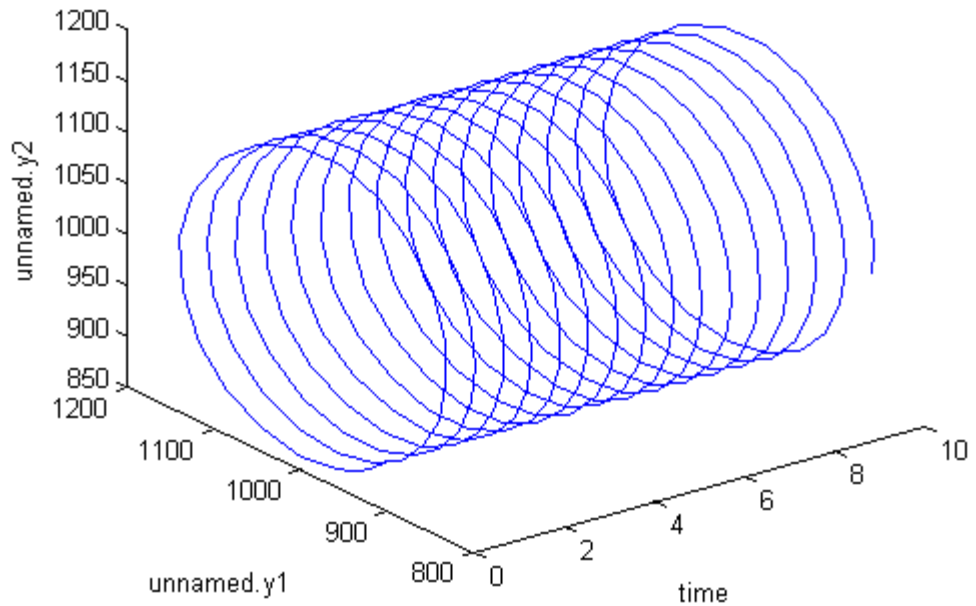
For more information about user-defined plot types see “Desktop Example — Creating and Using Plot Types” on page 3-81.

Desktop Example – Creating and Using Plot Types

Goal

This example shows how to create a 3-D plot to display the time varying relationship between predator (y1) and prey (y2) in the Lotka-Volterra model.

unnamed.y1 versus unnamed.y2 over time



Workflow

To use a user-defined plot type in a task or **Data** pane:

- 1 Create a user-defined plot type containing your plot function in the **Library Explorer**.

- 2** Add the plot type to the list of plots created after the task runs or in the **Data** pane.
- 3** Run the task or click **Plot** in the **Plots** tab on the **Data** pane, to plot the results.

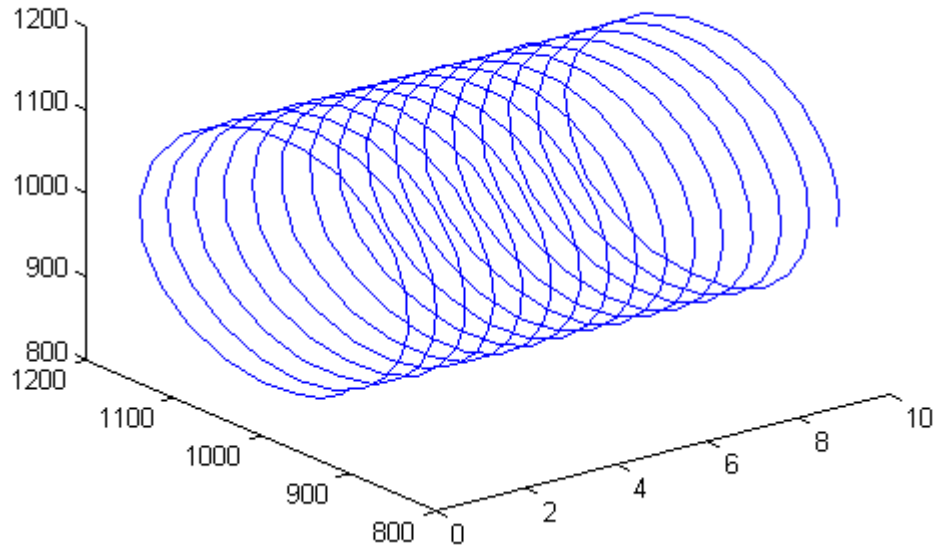
Why Use User-Defined Plot Types?

Consider a specific example of plotting. Suppose you want a 3-D plot to display the time varying relationship between predator (y1) and prey (y2) in the Lotka-Volterra model.

```
% Load, simulate, and plot results from
% the Lotka-Volterra model
sbioloadproject lotka.sbproj
simDataObj = sbiosimulate(m1);

% Get the data associated with the species y1.
[time, dataY1] = selectbyname(simDataObj, 'y1');

% Get the data associated with species y2.
[~, dataY2] = selectbyname(simDataObj, 'y2');
plot3(time,dataY1,dataY2)
```



To use this plot for any model, create a function that takes the results returned by the simulation (`simDataObj`) and creates a 3-D plot showing the relationship between two species over time.

```
function simplot3(simDataObj, x, y)
% Get the data associated with the species specified by x.
[time, dataX] = selectbyname(simDataObj, x);

% Get the data associated with y.
[~, dataY] = selectbyname(simDataObj, y);

% Plot.
plot3(time, dataX, dataY);
```

Now to create the above figure, save the function definition as an M-file on the path (or change directories to the location of the M-file), and type the following at the command line:

```
% Define the x argument
```

```
x = 'y1'  
% Define the y argument  
y = 'y2'  
simplot3(simDataObj,x, y);
```

It would be convenient to use `simplot3` in the SimBiology desktop, where you can interactively apply the plot to any model, species, or task.

To use the plot in the desktop, create the plot type in the **Library Explorer**, and then add it to the list of plots created after a task runs or in the **Data** pane as shown below.

Creating a User-Defined Plot Type in the Library Explorer

- 1** In the SimBiology desktop, select **Desktop > Library Explorer**. The **Library Explorer** opens.
- 2** Select **Plot Types**. The **Plot Types** pane appears in the **Work Area**.
- 3** In the **Enter Name** box, enter a name for the plot and click **Add**.

```
simplot3
```

The desktop adds the new plot to the table listing plots and lists the plot type as **User-defined** in the **Library** column.

- 4** In the section to enter code for the plot type, the desktop fills in the function line with the name of the plot type as the function name, and `tobj`, which is the data object from the simulation, as the input. Replace the function line with the `simplot3` function that was previously defined:

```
function simplot3(simDataObj, x, y)  
% Get the data associated with the species specified by x.  
[time, dataX] = selectbyname(simDataObj, x);  
  
% Get the data associated with y.  
[~, dataY] = selectbyname(simDataObj, y);  
  
% Plot.
```



```
plot3(time, dataX, dataY);
```

The desktop updates the **List of Arguments Passed to Plot Type Code** table with the list of arguments.

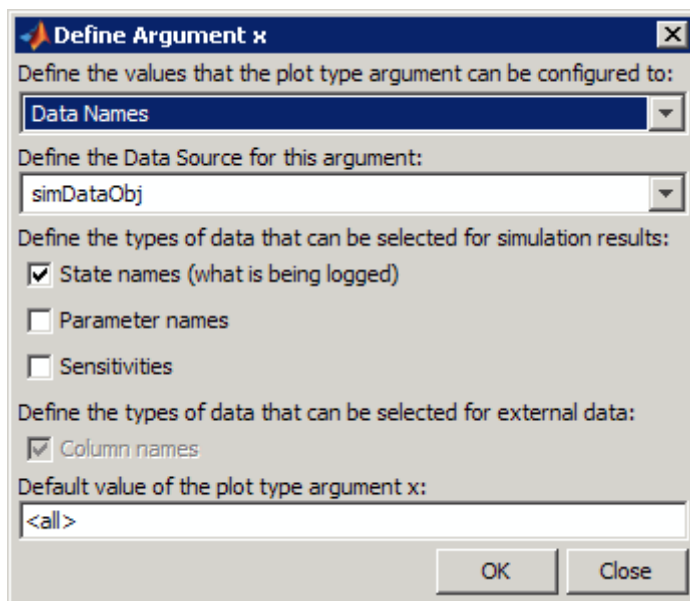
- 5 Specify valid values for an input argument by double-clicking its row in the **List of Arguments Passed to Plot Type Code** table. The desktop uses these values to prompt you for the accepted values when using the plot type.

Double-click `simDataObj`. In the Define Argument `simDataObj` dialog box select **Data Source** and click **OK**. When you use the plot type in a task, the desktop lists the data sets available for plotting.

- 6 Double-click `x`. The Define Argument `x` dialog box opens
- 7 From the **Define the values that the plot type argument can be configured to** list, select **Data Names**.

The dialog box updates to show other options to configure.

- 8 For this example, in the function line the data source is already defined as `simDataObj`. If you have more than one argument in the function line designated as a data source. Select the data source in which the data names are defined.
- 9 Define the data types to choose for simulation results by selecting **State names** and leaving the default value as `<all>`. When you use the plot type in a task, the SimBiology desktop shows you all the species available for plotting.



- 10 Click **OK**.
- 11 Repeat steps 7 to 9 for the *y* argument to produce the following.

List of Arguments Passed to Plot Type Code:		
Variable Name	Argument Type	Default Value
simDataObj	Data Source	
x	Data Names {Source = simDataObj; Types = Column names, State names}	<all>
y	Data Names {Source = simDataObj; Types = Column names, State names}	<all>

For a complete description of the types of values you can specify for each argument see “Types of Accepted Values for Plot Type Arguments” on page 3-78

- 12 Click **Save Now** to save your new plot type. Plot types are automatically saved at regular intervals. If **Save Now** is disabled, then the desktop automatically saved the plot type.

Using the Plot Type in a Task

Opening and Saving the Example Model

- 1 In the MATLAB Command Window, type

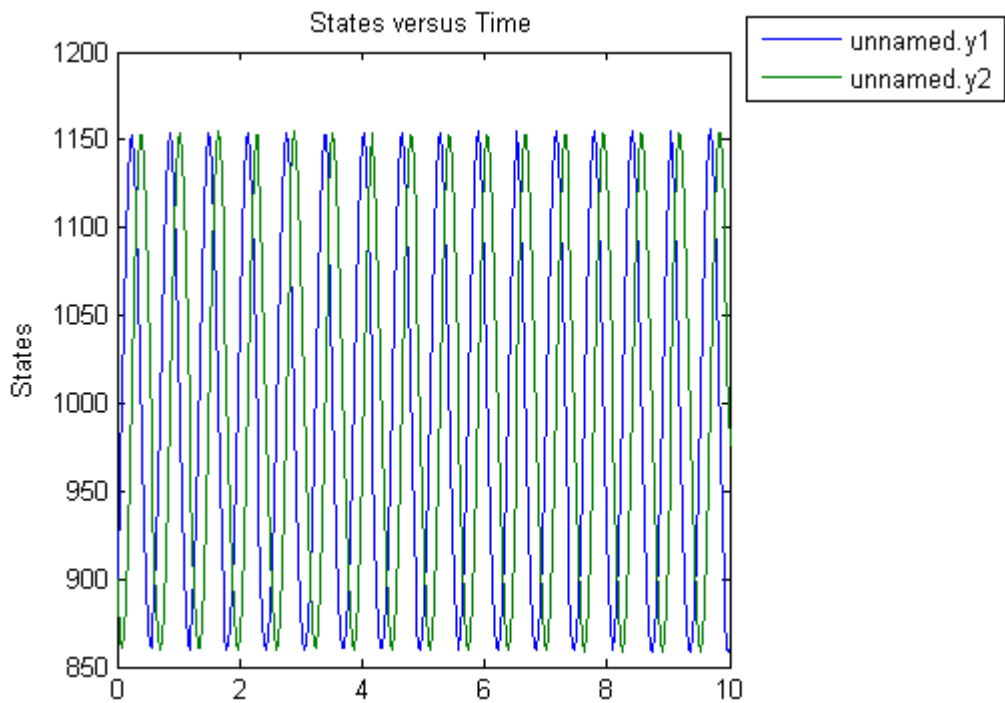
```
simbiology
```

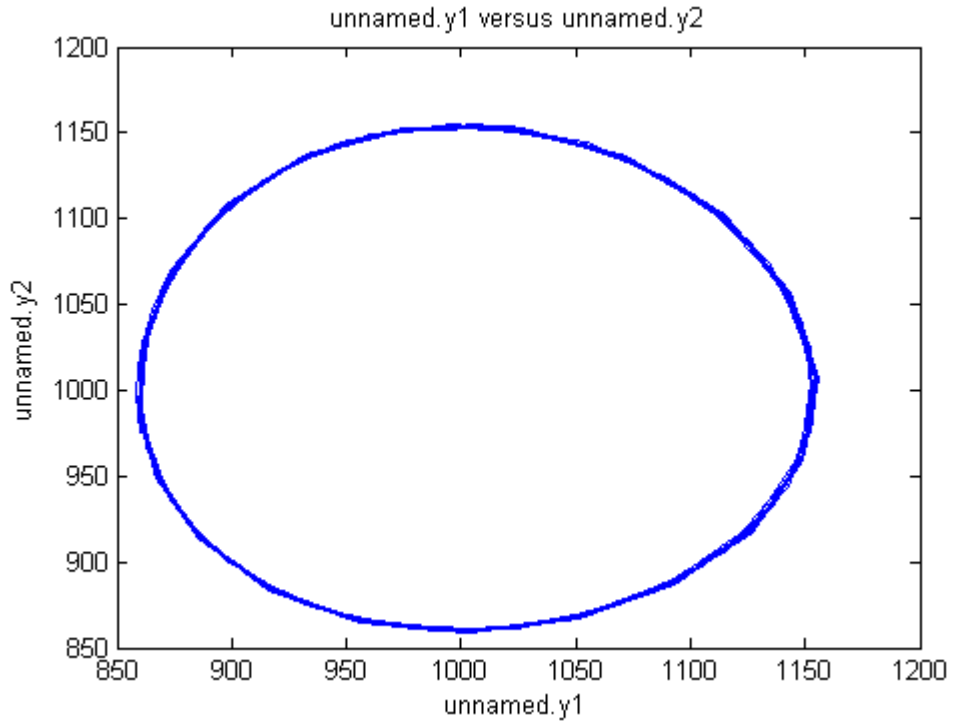
The SimBiology desktop opens.

- 2 Load the example project by navigating to the `simbiodeemos` folder and selecting `lotka.sproj`. The `simbiodeemos` folder is in `matlab/toolbox/simbio/simbiodeemos` relative to where MATLAB is installed.
- 3 Select **File > Save Project As**. The Save SimBiology Project dialog box opens.
- 4 Specify a name (for example, `lotka_ex`) and location for your project, and click **Save**.

Simulating the Model

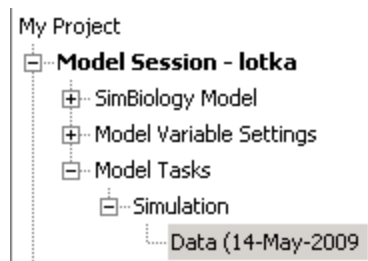
Simulate the model with the default plots and look at the results. To simulate the model, click **Run**.






Plotting the Results with the Plot Type

- 1 In the **Project Explorer** under **Model Tasks**, expand **Simulation** and select **Data**.



- 2 In the **Data** pane, click **Plots**.

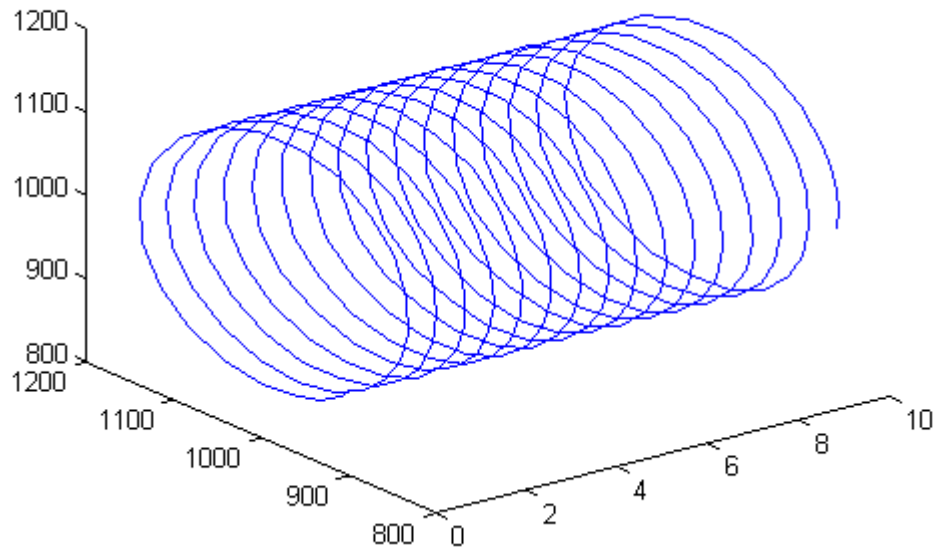
3 From the **Plot Type** list, select `simplot3` and click **Add Plot Type**.

4 In the **Arguments** section, for the x argument, click .

5 Clear all the states except `unnamed.y1` and click **OK**.

6 Similarly for the y argument, clear all the states except `unnamed.y2` click **OK**.

7 Click .



8 Add a legend to the plot.

a Select **Desktop > Library Explorer**. The **Library Explorer** opens.

b Select **Plot Types**. The **Plot Types** pane appears in the **Work Area**.

c If it is not already selected, select `plot3`.

d After the `plot3` syntax add the following:

```
% Label plot
title ([x ' versus ' y ' over time']);
xlabel('time');
ylabel(x);
zlabel(y);
```

- e Click **Save Now** to save the changes.

Note If you want to make changes to the function line, you must first delete the plot from any model tasks where it is being used.

- 9 In the **Project Explorer**, click **Data**.

- 10 Click  **Plot**

Pharmacokinetic Modeling

- “Pharmacokinetic Modeling Functionality” on page 4-2
- “Importing Data” on page 4-6
- “Working with Imported Data in the SimBiology Desktop” on page 4-19
- “Creating Pharmacokinetic Models” on page 4-28
- “Parameter Fitting Using Custom SimBiology Models” on page 4-42
- “Parameter Fitting in Pharmacokinetic Models” on page 4-47
- “Fitting Pharmacokinetic Model Parameters at the Command Line” on page 4-49
- “Fitting Pharmacokinetic Model Parameters in the SimBiology Desktop” on page 4-63

Pharmacokinetic Modeling Functionality

In this section...
“Overview” on page 4-2
“Required and Recommended Products” on page 4-2
“How This Product Supports Pharmacokinetic Modeling” on page 4-3
“Using the Command Line Versus the SimBiology Desktop” on page 4-5
“Accessing a Pharmacokinetic Modeling Demo” on page 4-5

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 4-3 for more information.
- Fit data — Fit a nonlinear mixed-effects models to data, and estimate the fixed and random effects, or fit the data using nonlinear least squares. See “Analyze Data Using Models” on page 4-4 for more information.
- Generate diagnostic plots — See “Analyze Data Using Models” on page 4-4 for more information.

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 Products

Required Product

Statistics Toolbox (Version 7.0 or later) - This product provides fitting tools including functions used to analyze nonlinear mixed effects.

Recommended Products

- Optimization Toolbox - Lets you specify additional methods for likelihood maximization. If you do not have this product `fminsearch` provided by MATLAB is used for likelihood maximization.
- Parallel Computing Toolbox™. If you have large models or data sets, then Parallel Computing Toolbox enables you to use MATLAB and Simulink® to harness multicore and multiprocessor computers for solving computationally and data-intensive problems.

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`.

In the SimBiology desktop you can filter the raw data to suppress outliers, visualize data using common plots such as `plot`, `semilog`, `scatter`, and `stairs`, and perform basic statistical analysis. You can also 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, see “Creating Pharmacokinetic Models” on page 4-28.

Analyze Data Using Models

You can perform both individual and population fits to grouped longitudinal data.

- Individual fit — Fit data using 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

For more information about each of these methods see `nlmefit` 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

Unsupported Functionality

In this release, SimBiology software does not support the following:

- Categorical covariates
- Observation-dependent covariates are not supported in SimBiology. Use `nlmefit` in the Statistics Toolbox for this functionality.

- Covariate models are not supported in the SimBiology desktop. Use the command-line features (`sbionlmeFit`) to specify covariate models.
- Error models — Only a constant error model is supported
- Parameter transformations — Only log normal parameter transformations are supported

Using the Command Line Versus the SimBiology Desktop

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

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

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.

Further, if you are using the SimBiology desktop and want to learn about using the command line, the M-code capture feature in the desktop lets you see the commands and export M-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 Modeling the Population Pharmacokinetics of Phenobarbital in Neonates to open the demo in MATLAB.

Importing Data

In this section...

“Supported Files and Data Types” on page 4-6

“Importing Data at the Command Line” on page 4-7

“Data Import in the SimBiology Desktop” on page 4-8

“Importing Data from the MATLAB Workspace into the SimBiology Desktop” on page 4-9

“Importing Data from a Text File into the SimBiology Desktop” on page 4-12

“Importing Data from an Excel File into the SimBiology Desktop” on page 4-14

“Importing Data from a MAT-File into the SimBiology Desktop” on page 4-15

“Resources for Working with Imported Data” on page 4-17

Supported Files and Data Types

To be able to fit data to a model, the data must be in a `dataset` array, constructed by the `dataset` function. You can construct the `dataset` array at the command line, or import the data into the SimBiology desktop which constructs the `dataset` array for you.

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

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 4-40.

Importing Data at the Command Line

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. 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. See `dataset` for more information.

Examples

```
% text files
data = dataset('file', 'Theophylline.txt', 'TreatAsEmpty', '.')

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

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 might have created new variables. You can export these 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

The “Importing Data” section of the MATLAB documentation provides detailed information about supported data formats and the functions for

importing data into the MATLAB Workspace. You can also import data using the MATLAB Import Wizard (see “Using the Import Wizard” in the MATLAB documentation). With the Import Wizard, you can 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 and recognizes data delimiters, as well as row or column headers, to facilitate the process of data selection and import into the MATLAB Workspace. You can import the data into the SimBiology desktop from the MATLAB Workspace.

Data Import in the SimBiology Desktop

In the SimBiology desktop, the import process constructs the `dataset` array for you, using one of the following:

- A tabular text file
- A Microsoft® Excel® spreadsheet
- A MAT-file containing variables that are of `double` data type, or `dataset` object, time series objects, or a `SimData` object.
- Variables that are in the MATLAB Workspace (for example, `double`, `dataset` object, time series objects, or a `SimData` object).
- SimBiology project (`.sbproj` file) containing data

During import, you can specify whether the first row contains header information, and whether any characters in the file should be interpreted as missing values. You can also edit column headings and see a preview of the data (first 20 rows).

Note If you want to change the names of the column headers you must do so in the import dialog box. Once the data is imported you cannot change the names of the columns.

Tips for Importing Data from Text Files or Excel Spreadsheets

Use the following tips to prepare the data before importing. This helps the import process to construct the `dataset` array in the SimBiology desktop.

- **Headers** — The data file must contain only one row of headers. If the first row contains headers, it should immediately precede numerical data. Remove any comments from the file.
- **Header names** — If the file contains header names, then the number of header names should equal the number of columns.
- **Delimiters** — Text files must contain a consistent delimiter. Each row and column should have the same type and number of delimiters.
- **Missing data** — In text files, specify missing data with a character, for example, a period. In Microsoft Excel files, leave cells representing missing data empty.

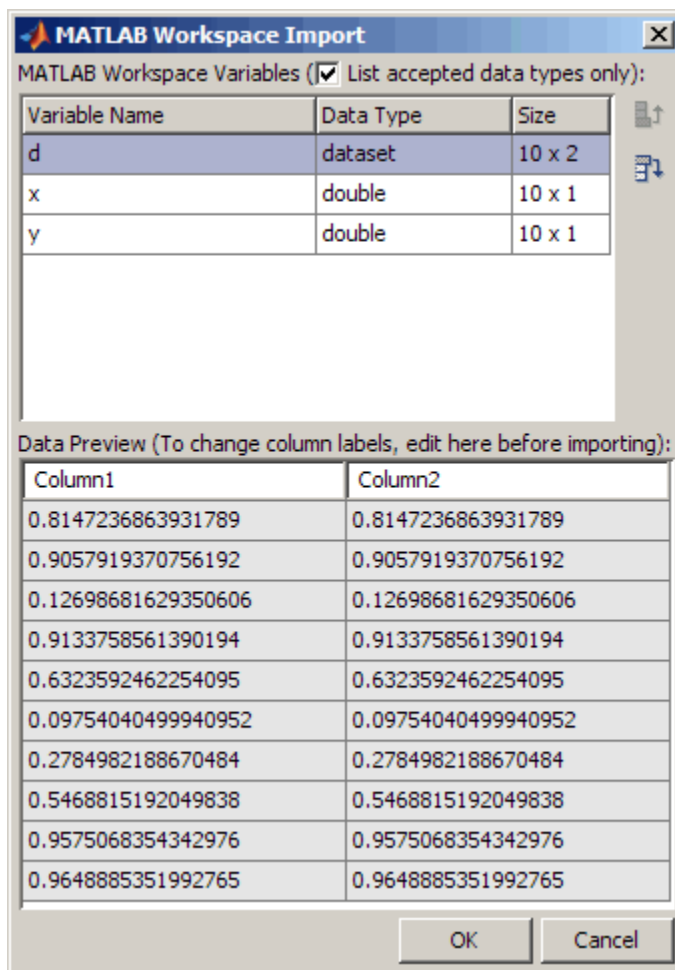
Importing Data from the MATLAB Workspace into the SimBiology Desktop

If you have already imported data into MATLAB, you can either continue to work at the command line or import the data into the SimBiology desktop for analysis. For information on when to use the command line or the desktop, see “Using the Command Line Versus the SimBiology Desktop” on page 4-5.

- 1 If the SimBiology desktop is not open, at the MATLAB command line type:

```
simbiology
```

- 2 Select **File > Add Data > From MATLAB Workspace**. The MATLAB Workspace Import dialog box opens with a list of variables that contain the accepted data types, and a preview of the data.



- 3 In the **MATLAB Workspace Variables** table, select the variables to import.
- If the **Data Type** of the variable you wish to import is **dataset**, time series, or **SimData** object, you can select one variable containing the object.
 - If the **Data Type** of the variable you wish to import is **double**, you can select multiple variables to create the data set. Note the following:

- The variables must each contain the same number of rows.
- During import, the selected variables are concatenated column-wise.
- The variables are concatenated in the order that they appear in the table, from top to bottom resulting in columns ordered from left to right.
- If necessary, a matrix is transposed to create a matrix with compatible size.

The **Size** column shows the *number of rows x number of columns* in the variable.

If your data file contains missing values, the **Data Preview** table shows the missing values as NaN.

- 4 Double-click the cells in the first row of the **Data Preview** table to edit the names of the column headers.

Note After the data is imported into the SimBiology desktop, you cannot rename the headers.

The SimBiology desktop uses the names you specify in the column headers to:

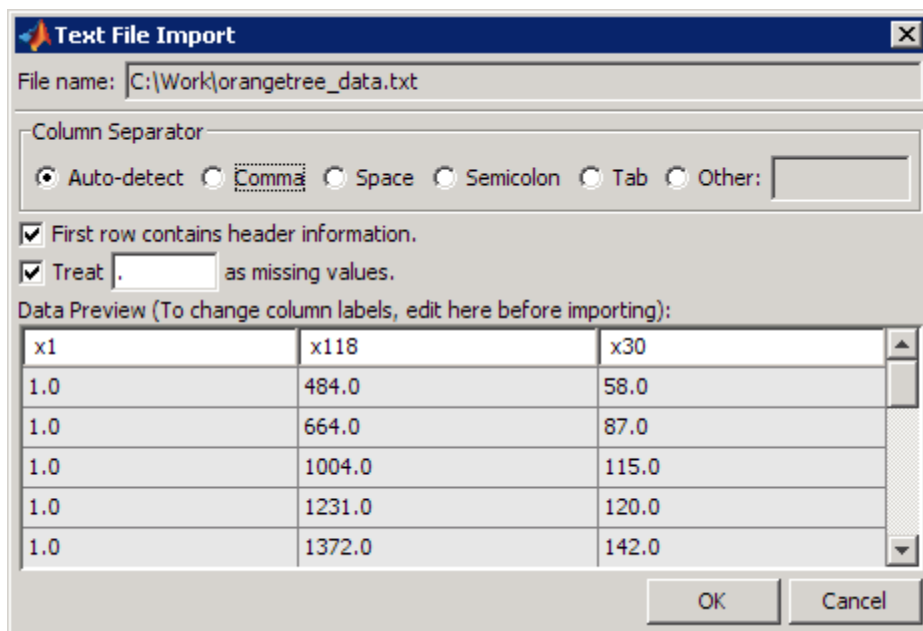
- Identify columns containing data used in fitting
 - Plot one variable against another during data exploration
 - Identify names in expressions when working with your data
- 5 Click **OK**. The SimBiology desktop adds the data to the **Project Explorer** under **External Data**, and opens the pane containing the data.
 - 6 (Optional) Rename the pane containing the imported data set, in the **Project Explorer**:
 - a Right-click the item representing the data set and select **Rename Data**. The Rename Data Set dialog box opens.
 - b Specify a name for the data set and click **Save**.

Importing Data from a Text File into the SimBiology Desktop

- 1 If the SimBiology desktop is not open, at the MATLAB command line type:

```
simbiology
```

- 2 Select **File > Add Data > From File**. The Import Data from File dialog box opens.
- 3 Select a .txt file and click **Open**. The Text File Import dialog box opens. This dialog box lets you preview the data.



- 4 Under **Column Separator**, select the type of separator in the text file:
 - Select **Auto-detect** to let the SimBiology desktop detect and choose the separator.
 - If **Auto-detect** does not select the correct option, select one of the following options:

Comma
Space
Semicolon
Tab
Other

If you select **Other**, enter a character that represents the column separator.

- 5** If your data file does not contain header information in the first row, clear the **First row contains header information** check box.
- 6** If your data file contains missing values, specify the character used to identify missing values in the file. The **Data Preview** table shows missing values as NaN.
- 7** Double-click the cells in the first row of the **Data Preview** table to edit the names of the column headers.

Note After the data is imported into the SimBiology desktop you cannot rename the headers.

The SimBiology desktop uses the names you specify in the column headers to:

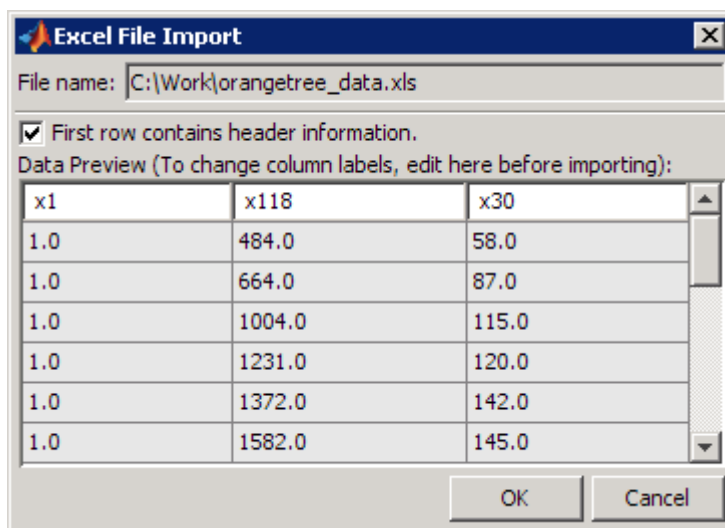
- Identify columns containing data used in fitting
 - Plot one variable against another during data exploration
 - Identify names in expressions when working with your data
- 8** Click **OK**. The SimBiology desktop adds the data to the **Project Explorer** under **External Data**, and opens the pane containing the data.
 - 9** (Optional) Rename the pane containing the imported data set, in the **Project Explorer**:
 - a** Right-click the item representing the data set and select **Rename Data**. The Rename Data Set dialog box opens.
 - b** Specify a name for the data set and click **Save**.

Importing Data from an Excel File into the SimBiology Desktop

- 1 If the SimBiology desktop is not open, at the MATLAB command line type:

```
simbiology
```

- 2 Select **File > Add Data > From File**. The Import Data from File dialog box opens.
- 3 Select a .xls file and click **Open**. The Excel File Import dialog box opens. This dialog box lets you preview the data.



- 4 If your data file does not contain header information in the first row, clear the **First row contains header information** check box.

If your data file contains missing values, the **Data Preview** table shows missing values as NaN.

- 5 Double-click the cells in the first row of the **Data Preview** table to edit the names of the column headers.

Note After the data is imported into the SimBiology desktop you cannot rename the headers.

The SimBiology desktop uses the names you specify in the column headers to:

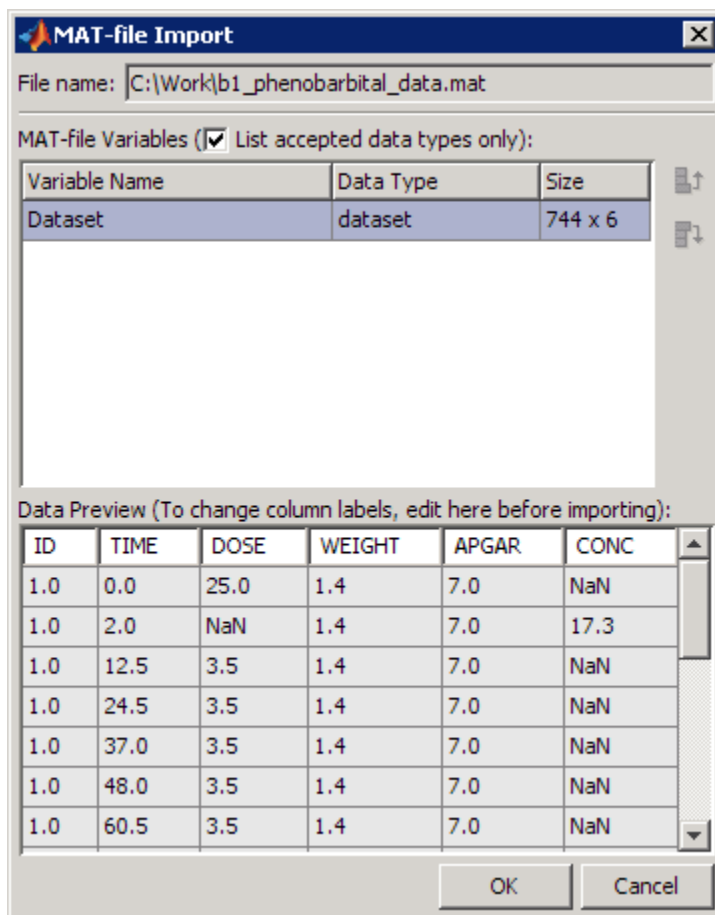
- Identify columns containing data used in fitting
 - Plot one variable against another during data exploration
 - Identify names in expressions when working with your data
- 6** Click **OK**. The SimBiology desktop adds the data to the **Project Explorer** under **External Data**, and opens the pane containing the data.
 - 7** (Optional) Rename the pane containing the imported data set, in the **Project Explorer**:
 - a** Right-click the item representing the data set and select **Rename Data**. The Rename Data Set dialog box opens.
 - b** Specify a name for the data set and click **Save**.

Importing Data from a MAT-File into the SimBiology Desktop

- 1** If the SimBiology desktop is not open, at the MATLAB command line type:

```
simbiology
```

- 2** Select **File > Add Data > From File**. The Import Data from File dialog box opens.
- 3** Select a **.mat** file and click **Open**. The MAT File Import dialog box opens with a list of the importable variables in the MAT-file along with their data type and size.



- 4 In the MAT-file Variables table, select the variables to import.
- If the **Data Type** of the variable you wish to import is dataset, time series, or SimData object, you can select one variable containing the object.
 - If the **Data Type** of the variable you wish to import is double, you can select multiple variables to create the data set. Note the following:
 - The variables must each contain the same number of rows.
 - During import, the selected variables are concatenated column-wise.

- The variables are concatenated in the order that they appear in the table, from top to bottom resulting in columns ordered from left to right.
- If necessary, a matrix is transposed to create a matrix with compatible size.

The **Size** column shows the *number of rows x number of columns* in the variable.

If your data file contains missing values, the **Data Preview** table shows missing values as NaN.

- 5 Double-click the cells in the first row of the **Data Preview** table to edit the names of the column headers.

Note After the data is imported into the SimBiology desktop you cannot rename the headers.

The SimBiology desktop uses the names you specify in the column headers to:

- Identify columns containing data used in fitting
 - Plot one variable against another during data exploration
 - Identify names in expressions when working with your data
- 6 Click **OK**. The SimBiology desktop adds the data to the **Project Explorer** under **External Data**, and opens the pane containing the data.
 - 7 (Optional) To rename the pane containing the imported data set, in the **Project Explorer**, right-click the item representing the data set and select **Rename Data**. The Rename Data Set dialog box opens.
 - 8 Specify a name for the data set and click **Save**.

Resources for Working with Imported Data

See the following sections for help on working with data in the SimBiology desktop.

For information on ...	See ...
Identifying group and independent variables	“Identifying Group and Independent Variables in the Data” on page 4-20
Plotting data	“Visualizing Data Using Plots in the SimBiology Desktop” on page 4-21
Cleaning up the imported data	“Excluding Rows of Data” on page 4-23
Calculating statistics	“Calculating Statistics for Imported Data” on page 4-26

See the following sections for help with creating models and fitting data at the command line or the desktop

For information on ...	See ...
Creating PK models	<ul style="list-style-type: none">• “Creating PK Models at the Command Line” on page 4-30• “Creating PK Models in the SimBiology Desktop Using a Wizard” on page 4-32
Fitting data	<ul style="list-style-type: none">• “Fitting Pharmacokinetic Model Parameters at the Command Line” on page 4-49• “Fitting Pharmacokinetic Model Parameters in the SimBiology Desktop” on page 4-63

Working with Imported Data in the SimBiology Desktop

In this section...

“Accessing Imported Data” on page 4-19

“Identifying Group and Independent Variables in the Data” on page 4-20

“Visualizing Data Using Plots in the SimBiology Desktop” on page 4-21

“Excluding Rows of Data” on page 4-23

“Creating Additional Data Columns Using Derived Data” on page 4-25

“Calculating Statistics for Imported Data” on page 4-26

“Saving Your Work as a SimBiology Project File” on page 4-27

Accessing Imported Data

- 1 If the SimBiology desktop is not open, at the MATLAB command line type:

```
simbiology
```

- 2 In the **Project Explorer**, under **External Data**, click the name of the data set to open.

The screenshot displays the SimBiology software interface. On the left, the Project Explorer shows a tree view with 'My Project' expanded to 'Model Session - PKModel', which includes 'SimBiology Model', 'Model Variable Settings', 'Model Tasks', and 'External Data'. Under 'External Data', 'DataSet3' is selected.

The main Work Area window, titled 'Work Area - My Project \External Data \DataSet3', shows a data table with the following columns: ID, TIME, DOSE, WEIGHT, APGAR, and CONC. The data is organized into 12 rows, with the first column labeled 'group variable' and the second column labeled 'independent variable'.

ID	TIME	DOSE	WEIGHT	APGAR	CONC
1	0.0	25.0	1.4	7.0	NaN
2	2.0	NaN	1.4	7.0	17.3
3	12.5	3.5	1.4	7.0	NaN
4	24.5	3.5	1.4	7.0	NaN
5	37.0	3.5	1.4	7.0	NaN
6	48.0	3.5	1.4	7.0	NaN
7	60.5	3.5	1.4	7.0	NaN
8	72.5	3.5	1.4	7.0	NaN
9	85.3	3.5	1.4	7.0	NaN
10	96.5	3.5	1.4	7.0	NaN
11	108.5	3.5	1.4	7.0	NaN
12	112.5	NaN	1.4	7.0	31.0

Below the table, the 'Plot' tab is active, showing options for 'Exclude', 'Derived Data', 'Statistics', and 'Description'. The 'Group data by column:' dropdown is set to 'ID'. The 'Line Plots' section includes 'plot(x,y)', 'semilogx(x,y)', 'semilogy(x,y)', and 'loglog(x,y)'. The 'Scatter Plots' section is also visible. The 'Plots Remembered' section shows a table with 'Create' and 'Plot Command' columns, with a checked box and 'Click to ent...' in the 'Create' column.

Identifying Group and Independent Variables in the Data

After importing data into the SimBiology desktop, you can identify the group and independent variables in the data set. SimBiology uses these settings to calculate statistics such as max, min, and mean for each group.

In addition, SimBiology also populates the independent and group variable settings for the parameter fitting task.

If you have not imported any data, see “Importing Data” on page 4-6.

If you do not have the data set open, open the pane containing the data set as shown in “Accessing Imported Data” on page 4-19.

To specify which columns represent the group and independent variables in the data:

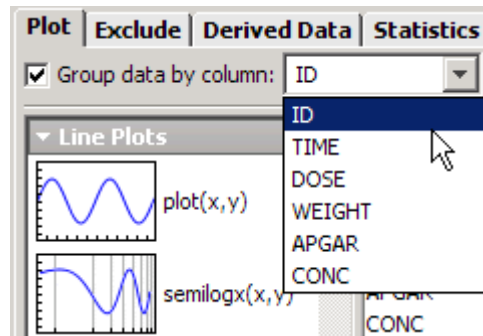
- Under the column header from the list, select **independent variable** or **group variable**. You can only identify a single column as independent or group variable at any given time.

Visualizing Data Using Plots in the SimBiology Desktop

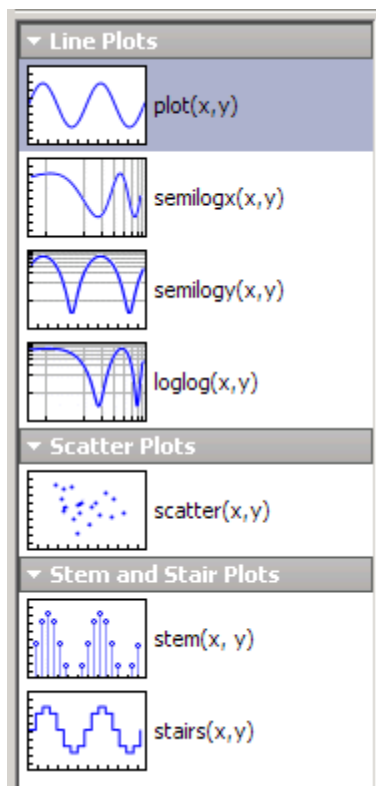
If you do not have the data set open, open the pane containing the data as shown in “Accessing Imported Data” on page 4-19.

To plot the data:

- 1 If you want to plot the data by group do the following, if not go to step 2.
 - a In the **Plot** tab, select the **Group data by column** check box.
 - b From the list of columns, select the column that contains the groups.



- 2 From the list of plots, select the type of plot.



- 3 In the **x** table, select the column to plot in the x-axis
- 4 In the **y** table, select the column to plot in the y-axis. A figure opens with the selected plot.
- 5 (Optional) Click **Add** to add the plot to the **Plots Remembered** table. **Plots Remembered** lets you add plots that you want to create when you click **Plot**.

Plots Remembered:	
Create	Plot Command
<input checked="" type="checkbox"/>	plot(Column1, Column2)
<input checked="" type="checkbox"/>	Click to enter plot or select to the left

- 6 (Optional) In the **Plots Remembered** table, you can also enter any plot command. If the command is invalid and the **Create** check box is selected, you will see a red indicator. If the **Create** check box is clear, you will see a yellow indicator for invalid commands.
- 7 (Optional) By default the **Plot when selection changes** check box is selected. Clear this check box if you do not want plotting to occur automatically. Click **Plot** to create the selected plot.

Excluding Rows of Data

You can exclude rows of data to trim outliers; use MATLAB expressions to filter the data, or select a row in the table containing the data and use the context menu.

If you do not have the data open, first open the pane containing the data as shown in “Accessing Imported Data” on page 4-19.

To exclude specific rows of data from the table:

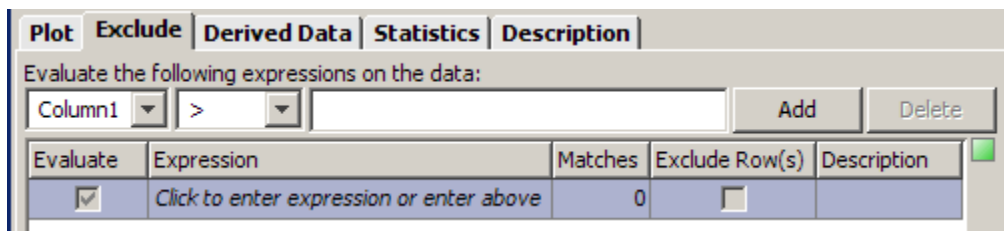
- 1 In the data table, select a row or multiple rows.
- 2 Right-click and select **Exclude Selected Row from Data**, or, press **Delete**.

The excluded rows are grayed out in the table and the **Expression** cell updates to show the row numbers excluded.

Summary of Matches to the right of the table shows you the sum of the matches from the evaluated expressions.

You can exclude rows of data based on a filter criterion. Directly enter an expression to filter the data or have the expression built for you. To directly enter the expression:

- 1 In the pane containing the data, click the **Exclude** tab.



- 2 Enter an expression in the **Expression** cell. For example:

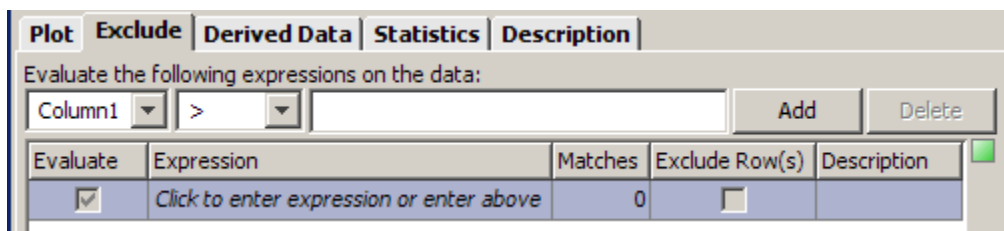
WEIGHT < (1.4)

Summary of Matches to the right of the table shows the all the matches from the evaluated expressions.

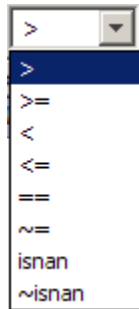
- 3 To exclude the rows that matched the expression, select the **Exclude Row(s)** check box. The excluded rows are grayed out in the table. Notice that the table containing the data shows black markers for the rows that matched the expression.

To have the expression built for you:

- 1 Open the pane containing the data as shown in “Accessing Imported Data” on page 4-19.
- 2 Click the **Exclude** tab.



- 3 From the list under **Evaluate the following expressions on the data**, select a column heading. For example in the previous figure, the column heading is Column1.
- 4 Select a relational operator from the list.



- 5 In the text box, enter a value to compare against and click **Add** or press **Enter**. The table in the **Exclude** tab updates with the **Expression** and the number of **Matches** in the data. Notice that the table containing the data shows black markers for the rows that matched the expression.

Summary of Matches to the right of the table shows you all the matches from the evaluated expressions.

- 6 To exclude the matched data, select the **Exclude Row(s)** check box. The excluded rows are grayed out in the table.

Creating Additional Data Columns Using Derived Data

You can create new columns of derived data from one or more columns using MATLAB expressions. If you do not have the data set open, first open the pane containing the data set as shown in “Accessing Imported Data” on page 4-19.

To create a new column of derived data:

- 1 Click the **Derived Data** tab.
- 2 In the **Expression** box, enter a MATLAB expression to evaluate. For example, assume that the data contains columns specifying drug dose (DOSE) and dose time (DOSE_TIME). To calculate dose rate, enter:

DOSE/DOSE_TIME

- 3 In the **Column Label** box, enter a name for the new column. For example:

DOSE_RATE

4 Click **Add**. The table updates with the new column and the resulting data.

Calculating Statistics for Imported Data

You can calculate statistics for the grouped data in the **Statistics** tab. If you do not have the data set open, first open the pane containing the data set as shown in “Accessing Imported Data” on page 4-19.

1 Click the **Statistics** tab.

2 Select a variable from the **Select the variable to calculate statistics on** list.

The table in the **Statistics** tab updates to show the results (max, min, mean, tmax).

3 To calculate auc, aumc, and mrt, right-click the table containing the statistics, and select **Statistics to Calculate**.

4 Click **OK**.

5 In the **Statistics to Calculate** dialog box, you can specify the additional statistics to view.

6 (Optional) If you change any excluded rows, click **Refresh**.

The statistical measures available are as follows.

Statistical Measure	Description
max	The maximum value of the dependent variable for each group.
min	The minimum value of the dependent variable for each group.

Statistical Measure	Description
mean	The mean of the dependent variable for each group.
tmax	Time of maximum concentration for each group.
auc	Area under the curve.
aumc	Area under the first moment curve.
mrt	Mean residence time

Saving Your Work as a SimBiology Project File

A SimBiology project is the file format that the software uses to save one or more model sessions; the file extension is `.sbproj`. Projects let you save custom settings, notes, and data associated with your models.

Save your work as a project so that you can access this file later. If you have models in the SimBiology desktop, the model and the data are saved together in the project.

- 1** Select **File > Save Project as** to open the Save SimBiology Project dialog box.
- 2** Browse to the folder in which you want to save your projects, and enter a name for the project file.
- 3** Click **Save**.

Creating Pharmacokinetic Models

In this section...
“Overview” on page 4-28
“How Pharmacokinetic Models Are Represented by SimBiology Models” on page 4-29
“Creating PK Models at the Command Line” on page 4-30
“Creating PK Models in the SimBiology Desktop Using a Wizard” on page 4-32
“About Dosing Types” on page 4-34
“About Elimination Types” on page 4-37
“About Intercompartmental Clearance” on page 4-39
“Unit Conversion for Imported Data and the Model” on page 4-40

Overview

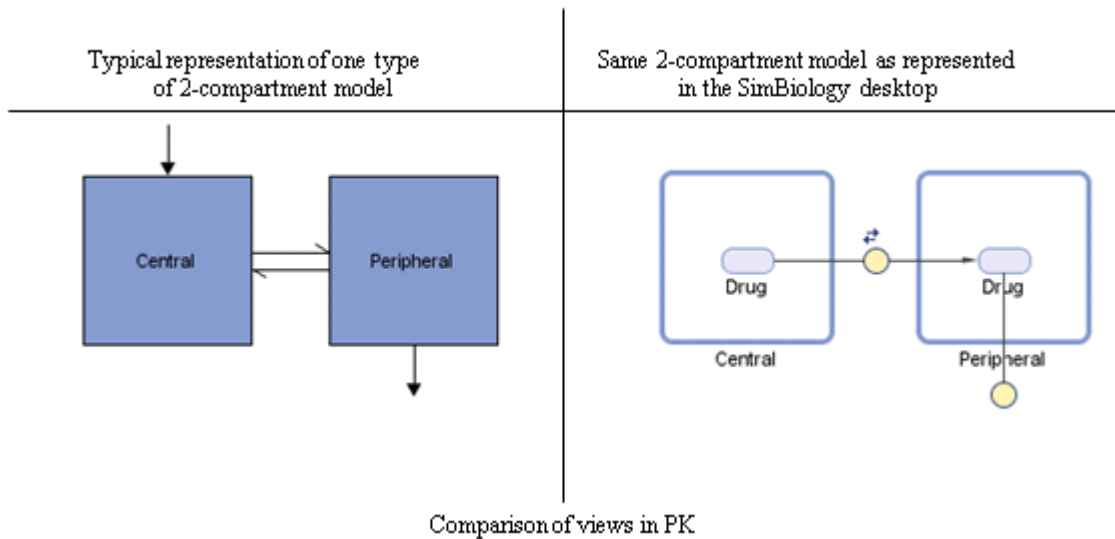
As described in “Model” on page 4-3, you can do any of the following to start modeling:

- 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.

Whether you plan on using the wizard to build a PK model or building your own model, you might want to understand how models are represented in the software, as described in “How Pharmacokinetic Models Are Represented by SimBiology Models” on page 4-29.

How Pharmacokinetic Models Are Represented by SimBiology 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 a drug using a two-compartment model with bolus input and zero-order elimination kinetics.



- The concentration or amount of a drug in a given compartment or volume is represented in SimBiology by a species *object* contained within a compartment.
- The exchange or flow of the drug between compartments and the elimination of the drug is represented by reactions in SimBiology.
- Intercompartmental clearance is represented by the parameter (Q) representing clearance between the compartments.

Another view of this model could be 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 at the Command Line

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, rules, and events.

```
pkm = PKModelDesign;
```

- 2 Add a compartment specifying the name, the type of dosing and method of elimination, and specify whether the data contains a response variable measured in this compartment. For example, specify a compartment named `Central`, with `FirstOrder` for the `DosingType` property, `linear-clearance` for the `EliminationType` property and `true` for the `HasResponseVariable` property.

```
pkm.addCompartment('Central', 'FirstOrder', ...  
                  'linear-clearance', true);
```

For a description of the dosing and elimination types, see “About Dosing Types” on page 4-34 and “About Elimination Types” on page 4-37.

Note You can only specify one compartment in a model with a dosing type other than `''` (empty). Similarly, you can only specify one compartment as containing a response variable.

- 3 Add a second compartment named `Peripheral`, no dosing, no elimination, and specify (using `false`) that there is no response variable measured in this compartment.

```
pkc = pkm.addCompartment('Peripheral', '', '', false);
```

The model construction process adds the necessary parameters, including a parameter representing intercompartmental clearance Q . The “About Dosing Types” on page 4-34 and “About Elimination Types” on page 4-37 lists the parameters added for each option. 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 in the model.

Use the handle to the compartment to change compartment properties.

- 4 Construct a SimBiology model object with the previous specifications.

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

The `construct` method returns a SimBiology model object (`modelObj`) and a `PKModelMap` object which contains the mapping of the model components to the elements of the regression function. See “Defining Model Components for Observed Response, Dose, Dosing Type, and Parameters to be Estimated” on page 4-42 for more information about the `PKModelMap` object.

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 at the Command Line” on page 4-49.

(Optional) Before parameter fitting you can set parameter values and apply dosing information to simulate the model object and visualize the results as shown in “Simulating a Model Containing Dosing Information” on page 4-45.

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

For information on ...	See ...
Dosing types	“About Dosing Types” on page 4-34
Elimination types	“About Elimination Types” on page 4-37

For information on ...	See ...
Parameter fitting	“Fitting Pharmacokinetic Model Parameters at the Command Line” on page 4-49
Simulating the model and a description of configuration sets	<ul style="list-style-type: none"> • “About Simulation Settings and Specifying Alternate Values for Initial Estimates” on page 4-61 • “Performing Simulations” on page 2-2

Creating PK Models in the SimBiology Desktop Using a Wizard

In the Add Model wizard, you can specify the number of compartments, route of administration, and method of clearance to create a PK model.

You can access the Add Model wizard when you create a new project or add a model to an existing project.

- 1** Select **File > Add Model**. The Add Model wizard opens.
- 2** From the **Select a model to add** list, select **Create PK model**. The Add Model wizard updates to show you options for selecting the number of compartments, the type of administration, and elimination.
- 3** In the **Model Name** box, enter a name for the model. For example,


```
one_compartment_bolus
```
- 4** From the **Number of Compartments** list, select from 1 to 4 compartments. The **Model Description** table updates to show each compartment.
- 5** Double-click a **Compartment Name** cell to edit the name of the compartment.
- 6** From the **Dosing** list, select one of the following:
 - None
 - bolus

infusion
zero-order
first-order

Note Only one compartment in a model can contain a dosing type other than None.

See “About Dosing Types” on page 4-34 for more information on each of the above options.

7 From the **Elimination** list, select one of the following:

None
Linear{Clearance, Volume}
Linear{Elimination Rate, Volume}
Enzymatic(Michaelis-Menten)

See “About Elimination Types” on page 4-37 for more information on the above options.

8 (Optional) Click the **SimBiology View** tab to see how the model is represented in SimBiology. For more information, see “How Pharmacokinetic Models Are Represented by SimBiology Models” on page 4-29.

9 Click **OK**.

The SimBiology desktop adds the model in the **Project Explorer** and opens the model in the **Work Area** pane.

Saving Your Work as a SimBiology Project File

A SimBiology project is the file format that the software uses to save one or more model sessions. The file extension is `.sbproj`. Projects let you save custom settings, notes, and data associated with your models.

Save your work as a project so that you can access this file later. If you have data in the SimBiology project, the model and the data are saved together in the project.

- 1 Select **File > Save Project as** to open the Save SimBiology Project dialog box.
- 2 Browse to the folder in which you want to save your projects, and enter a name for the project file.
- 3 Click **Save**.

About Dosing Types

In SimBiology models, you establish dosing in two steps involving the type of dosing and the dosing schedule.

In the first step, the process of generating a SimBiology model via the wizard (in the SimBiology desktop) or the `PKModelDesign` object's `construct` method (at the command line) creates model components related to a particular route of administration. These model components represent the general kinetics of a particular route of administration and they are invariant with respect to a particular dosing schedule.

The second step (specified by the function `sbiosetdosingprofile`) adds components related to a particular dosing schedule, and the specified dosing type. SimBiology automatically uses `sbiosetdosingprofile` during the fitting process.

The following table describes the dosing types, the default parameters to estimate, and lists the model components created in the first step. For dosing types that have a fixed infusion or absorption duration (`infusion` and `zero-order`), you can use overlapping doses; the doses are additive. You can only dose one compartment in the model.

Dosing Type	Description	SimBiology Model Components Created	Default Parameters to Estimate
' '(None or empty)	No dose	<ul style="list-style-type: none"> • A species (Drug) for each compartment 	None

Dosing Type	Description	SimBiology Model Components Created	Default Parameters to Estimate
<p>SimBiology desktop — bolus</p> <p>Command line — Bolus</p>	<p>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.</p>	<ul style="list-style-type: none"> • A species (Drug) for each compartment 	<p>None</p>
<p>SimBiology desktop — infusion</p> <p>Command line — Infusion</p>	<p>Assumes that the infused drug amount increases at a constant known absorption (or infusion) rate over a known duration. In the SimBiology model, the drug concentration is specified by a reaction whose rate represents the infusion rate, and by the volume of the compartment containing the drug.</p> <p>The imported data set must contain the rate and not an infusion time.</p>	<ul style="list-style-type: none"> • A species (Drug) for each compartment • A parameter (kInf) representing the rate of infusion • A mass-action reaction (null -> CompartmentName.Drug) representing the infusion of drug over time, with forward rate parameter kInf 	<p>None</p>

Dosing Type	Description	SimBiology Model Components Created	Default Parameters to Estimate
SimBiology desktop — zero-order Command line — ZeroOrder	Assumes that the drug is added at a constant rate over fixed, but unknown duration. Similar to infusion, but with absorption duration as an estimated parameter. In the SimBiology model, the drug concentration is specified by a reaction whose rate is defined as [dose amount/absorption duration], and by the volume of the compartment containing the drug.	<ul style="list-style-type: none"> • A species (Drug) for each compartment • A parameter (Tk0) representing the duration of drug absorption • A parameter (ka) representing the absorption rate of the drug • A MassAction reaction (null > CompartmentName.Drug) with forward rate parameter (ka) 	<ul style="list-style-type: none"> • Tk0 (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 to be governed by mass-action kinetics.	<ul style="list-style-type: none"> • A species (Dose) representing the dose amount before it is absorbed • A species (Drug) for each compartment • A parameter (ka) representing the absorption rate of the drug • A MassAction reaction (Dose > CompartmentName.Drug) with forward rate parameter (ka) 	<ul style="list-style-type: none"> • ka (absorption rate)

As described previously, in the second step (which automatically takes place during fitting), the function `sbiosetdosingprofile` adds components related to a particular dosing schedule and the specified dosing type. `sbiosetdosingprofile` uses this step is used to update the correct drug amount or rate for each dose defined in the dosing table (specified by the

data set). These model components exist only on the model temporarily during fitting.

If you are using a custom model, or want to simulate a model with the dosing schedule applied, you must use `sbiosetdosingprofile`. See the following additional sources of information:

For information on ...	See ...
Preparing the model before simulating	“Defining Model Components for Observed Response, Dose, Dosing Type, and Parameters to be Estimated” on page 4-42
Applying dosing information	“Simulating a Model Containing Dosing Information” on page 4-45
Model components added temporarily in preparation for fitting	<code>sbiosetdosingprofile</code>

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 elimination rate constant specified by the forward rate parameter (<code>ke</code>).	<ul style="list-style-type: none"> A parameter representing the elimination rate (<code>ke_CompartmentName</code>) A <code>MassAction</code> reaction (<code>drug > null</code>) with forward rate parameter (<code>ke_CompartmentName</code>) 	<ul style="list-style-type: none"> Compartment volume (Capacity property) Elimination rate constant (<code>ke_CompartmentName</code>) Inter-compartmental clearance (<code>Q</code>) when there is more than one compartment. See “About Intercompartmental Clearance” on page 4-39.

Elimination Type	Description	SimBiology Model Components Created	Default Parameters to Estimate
<p>SimBiology desktop — Linear {Clearance, Volume}</p> <p>Command line — 'linear-clearance'</p>	<p>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, $Cl = k_e * volume$.</p>	<ul style="list-style-type: none"> • A parameter representing the clearance (Cl_CompartmentName) • A parameter representing the elimination rate constant (k_e_CompartmentName) • An InitialAssignment rule that initializes k_e_CompartmentName based on the initial values for Cl_CompartmentName and compartment volume • A MassAction reaction (drug > null) with forward rate parameter (k_e_CompartmentName) 	<ul style="list-style-type: none"> • Compartment volume (Capacity property) • Clearance (Cl_CompartmentName) • Inter-compartmental clearance (Q) when there is more than one compartment. See “About Intercompartmental Clearance” on page 4-39.
<p>SimBiology desktop — Enzymatic (Michaelis-Menten)</p> <p>Command line — 'enzymatic'</p>	<p>Assumes that elimination is governed by Michaelis-Menten kinetics.</p>	<ul style="list-style-type: none"> • Parameter representing the Michaelis constant, (K_m_CompartmentName) • A parameter for maximum velocity (V_m_CompartmentName) • A reaction with Michaelis-Menten kinetics (drug -> null), with kinetic 	<ul style="list-style-type: none"> • Compartment volume (Capacity property) • Parameter (K_m_CompartmentName) • Parameter (V_m_CompartmentName) • Inter-compartmental clearance (Q) when

Elimination Type	Description	SimBiology Model Components Created	Default Parameters to Estimate
		law parameters Vm_CompartmentName and Km_CompartmentName	there is more than one compartment. See “About Intercompartmental Clearance” on page 4-39.

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 $C1.Drug \leftrightarrow C2.Drug$. 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 (k_{mn}) representing the rate from transfer of the drug from C_m to C_n , where $k_{mn} = Q_{mn}/V_m$.
- A parameter (k_{nm}) representing the rate from C_n to C_m , where $k_{nm} = Q_{mn}/V_n$.
- A reversible mass-action reaction between the two compartments, $C_m.Drug \leftrightarrow C_n.Drug$, with forward rate parameter k_{mn} , and reverse rate parameter k_{nm} .
- An initial assignment rule that initializes the value of the parameter k_{mn} , based on the initial values for C_m and Q_{mn} .
- An initial assignment rule that initializes the value of the parameter k_{nm} , based on the initial values for C_n and Q_{mn} .

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 physical quantities in the model are returned in the user-specified units.

Regardless of whether unit conversion is on or off, dosing data must be expressed in amount. By default, **Unit Conversion** is off, so you must ensure that units for the data are consistent with each other.

If **Unit Conversion** is on, the data file should use the following default units, or change the model components units to match the units in the data file.

Physical Quantity or Model Parameter	Unit
Dose Time	second
Amount (dose)	milligram
Infusion rate	milligram/second

Units of the component representing the dependent (or response) variable in the model should be same as that of the response variable in the data file. The default units are as follows.

Physical Quantity or Model Parameter	Unit
Response (concentration)	milligram/liter

Parameters in the model also 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
K _m — Michaelis constant	milligram/liter
V _m — (V _{max}) Maximum reaction-velocity (Michaelis-Menten kinetics)	milligram/second
Clearance	liter/second
T _{k0} (absorption duration)	second
k _a (absorption rate)	1/second

Use the configuration settings options to turn unit conversion on or off. For more information see, “Performing Simulations” on page 2-2 in the SimBiology User’s Guide.

See “Evaluation of Reaction Rate” on page 1-15 in the SimBiology User’s Guide for more information on dimensional analysis for reaction rates.

Parameter Fitting Using Custom SimBiology Models

In this section...

“Overview” on page 4-42

“Defining Model Components for Observed Response, Dose, Dosing Type, and Parameters to be Estimated” on page 4-42

“Simulating a Model Containing Dosing Information” on page 4-45

Overview

This section provides information that is relevant if you are working with a custom SimBiology model that was not created using the or the `PKModelDesign` object's `construct` method at the command line or the wizard in the SimBiology desktop. If you created a PK model using either of these two methodologies, you can skip this section.

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 Parameters to be Estimated” on page 4-42.

If you want to explore the effects of different dosing schedules on the model, you need to set parameter values, apply dosing information to the model object, and then simulate the model as shown in “Simulating a Model Containing Dosing Information” on page 4-45.

Another point to consider is the solver used 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 4-61, for more information.

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

The `construct` method at the command line and the wizard in the SimBiology desktop automatically create a `PKModelMap` object. 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 was not created using 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 , and can be 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 model component object's name string to specify the corresponding property in the `PKModelMap` object.

Model Component Represents	PKModelMap Object Property
Object that is 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 called `Tk0` to the model. `Tk0` represents the duration of drug absorption and is used when simulating the model with dosing information or during fitting.

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 <code>Dosed</code> , <code>DosingType</code> , <code>Estimated</code> , and <code>Observed</code>
Allowed dosing types	“About Dosing Types” on page 4-34
Parameter scoping	“Scoping Parameters for Reactions, Rules, and Events” on page 1-22
Parameter fitting	“Fitting Pharmacokinetic Model Parameters at the Command Line” on page 4-49

Simulating a Model Containing Dosing Information

The function `sbioetdosingprofile` lets you modify a model object with a specific dosing schedule. `sbioetdosingprofile` modifies the model object with rules and events (as applicable) based on the dosing type and dosing information you give to the function. You can then use the modified model object in the `sbiosimulate` function to simulate the model.

`sbioetdosingprofile` is useful when you want to simulate different dosing schedules with the PK model generated using the `PKModelDesign` object's `construct` method, or when you have a custom model to which you want to apply a dosing pattern and visualize the results.

- 1 Create a model object using the `construct` method. Alternatively, use a custom SimBiology model object and create a corresponding `PKModelMap` object.

```
pkm = PKModelDesign;
pkc = pkm.addCompartment('Central','bolus','linear-clearance',true);
[modelObj PKModelMapObj] = pkm.construct;
```

- 2 Set the initial values of the parameters to be estimated. You can check the order of the parameters using the `PKModelMap` object's `Estimated` property (`PKModelMapObj.Estimated`).

```
modelObj.Parameters(1).Value = 2.59;
modelObj.Compartments.Capacity = 100;
```

- 3 Import or create a *DosingTable*.

```
dosetable = [100 6000; 0 12000];
```

For specifications of the dosing table see `sbioetdosingprofile`.

- 4 Create the modified model object containing dosing information.

```
modModelObj = sbioetdosingprofile(modelObj,...
    PKModelMapObj.Dosed(1), dosetable, 'Bolus');
```

- 5 Set the solver to `sundials`. `sbioetdosingprofile` adds events to the model for dosing information, and `sundials` is the only deterministic solver that supports events.

```
modModelObj.getConfigset.SolverType = 'sundials';
```

- 6 Simulate the model and plot the results.

```
sd = sbiosimulate(modModelObj);  
sbioplot(sd)
```

- 7 Clean up the modified model to remove model components added for dosing information.

```
delete(sbioselect(modModelObj, 'Tag', 'DOSE_COMPONENT'));
```

For more information on how the model is constructed in step 1, see “Creating PK Models at the Command Line” on page 4-30.

Simulations run during fitting using `sbionlmeft` or `sbionlinfit` may differ from simulations run using `sbiosimulate`. During parameter fitting via `sbionlmeft` or `sbionlinfit`, SimBiology will sometimes use a specialized simulation process, optimized for first-order mass-action kinetics.

If you do not fully specify units on the input model, this process can produce simulation results that differ from those produced by `sbiosimulate`. This is a consequence of differences in the dimensions implicitly assigned to the reaction rates of mass-action reactions. If this situation occurs, you will see a warning from `sbionlmeft` or `sbionlinfit`. The results of the parameter fitting will be correct, subject to the appropriate interpretation for the dimensions of the reaction rates of mass-action reactions. You cannot reproduce the fitting results by running simulations with `sbiosimulate`. Simulations run using `sbiosimulate` will differ from those computed internally during the fitting process. The best way to work around this issue is to set units for all model components when using `sbionlmeft` or `sbionlinfit`. Contact the MathWorks Technical Support group for additional information.

Parameter Fitting in Pharmacokinetic Models

In this section...

“Parameter Fitting Functionality” on page 4-47

“Prerequisites for Parameter Fitting” on page 4-48

Parameter Fitting Functionality

SimBiology lets you perform individual fits and population fits on grouped data. This functionality uses features in Statistics Toolbox (Version 7.0 or later).

There are two classes of methods available for population fitting:

- Methods that directly estimate parameters using the likelihood function namely, LME and ReLME
- Methods that linearize the likelihood function, namely, first-order approximation (FO) and first-order approximation at the conditional estimates 'FOCE'

The following results are returned:

- 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

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

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” on page 4-6 for more information)
- A model fit (see “Creating Pharmacokinetic Models” on page 4-28 for more information)

Depending on whether you plan to use the command line or the SimBiology desktop, see the following for more information.

- “Fitting Pharmacokinetic Model Parameters at the Command Line” on page 4-49
- “Fitting Pharmacokinetic Model Parameters in the SimBiology Desktop” on page 4-63

Fitting Pharmacokinetic Model Parameters at the Command Line

In this section...

“Fitting Parameters” on page 4-49

“Specifying and Classifying the Data to Fit” on page 4-50

“Setting Initial Estimates” on page 4-51

“Specifying the Covariance Pattern of Random Effects and a Covariate Model” on page 4-52

“Performing Population Fitting Using `sbionlmeFit`” on page 4-56

“Performing Individual Fitting Using `sbionlinfit`” on page 4-60

“About Simulation Settings and Specifying Alternate Values for Initial Estimates” on page 4-61

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 at the Command Line” on page 4-7.
- 2** Create a PK model as shown in “Creating PK Models at the Command Line” on page 4-30. Alternatively, if you have a SimBiology model that you want to use in fitting, see “Parameter Fitting Using Custom SimBiology Models” on page 4-42.
- 3** (Optional) Simulate the model containing the dosing information to visualize the effects of dosing on the model as shown in “Simulating a Model Containing Dosing Information” on page 4-45.
- 4** Specify the data set to use in fitting, and classify the data columns containing group identifiers, the independent variable (for example, time), the dependent variable (for example, the measured response), the dose, the rate of infusion (if applicable), and the columns that represent the covariates. See “Specifying and Classifying the Data to Fit” on page 4-50.

- 5 Specify the initial guesses for the parameters to be estimated as shown in “Setting Initial Estimates” on page 4-51.
- 6 Perform individual or population fits:
 - For population fits:
 - Specify the covariance matrix and the covariate model. See “Specifying the Covariance Pattern of Random Effects and a Covariate Model” on page 4-52.
 - (Optional) Set tolerances and specify maximum iterations as shown in “Setting Tolerance and Maximum Iteration for Population Fits” on page 4-69.
 - For individual fits:
 - (Optional) Set tolerances and specify maximum iterations as shown in “Setting Tolerance and Maximum Iteration for Individual Fits” on page 4-74.
- 7 Run the task and visualize results as shown in “Visualizing Parameter Fitting Results and Generating Diagnostic Plots” on page 4-76.

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 at the Command Line” on page 4-7.

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 (such as age, gender, weight)	CovariateLabels

For example:

```
pkDataObject.GroupLabel      = 'ID';
pkDataObject.IndependentVarLabel = 'TIME';
pkDataObject.DependentVarLabel  = 'Y';
pkDataObject.DoseLabel        = 'DOSE';
pkDataObject.CovariateLabels   = {'Age', 'Wt'};
```

When you assign a column containing group identification labels to the GroupLabel property, PKData sets the following 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 that are to be estimated, first identify the sequence of the parameters in the model by querying the PKModelMap object, and then construct a vector (beta0) containing the initial conditions. For information about PKModelMap, see step 4 in “Creating PK Models at the Command Line” on page 4-30),

- 1 Query the Estimated property of the PKModelMap object.

```
PKModelMapObj.Estimated
```

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

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

- 2 Set the initial guesses for the parameters. For example:

```
beta0 = [1.5, 0.0398, 1.5669];
```

For information on ...	See ...
The parameters added to the model	<ul style="list-style-type: none"> • “About Dosing Types” on page 4-34 • “About Elimination Types” on page 4-37
Default units for the above parameters	“Unit Conversion for Imported Data and the Model” on page 4-40

Specifying the Covariance Pattern of Random Effects and a Covariate Model

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

$$\begin{aligned}\varphi_i &= A_i\beta + B_ib_i \\ y_i &= f(\varphi_i, X_i) + \varepsilon_i \\ b_i &\sim N(0, \Psi) \\ \varepsilon_i &\sim N(0, \sigma^2)\end{aligned}$$

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 A vector of group-specific errors, assumed to be independent, identically, normally distributed, and independent of b_i
- 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:

$$\begin{pmatrix} Cl \\ V \\ k_a \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \beta_{cl} \\ \beta_v \\ \beta_{ka} \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} b_{cl} \\ b_v \\ b_{ka} \end{pmatrix}$$

In SimBiology, B_i is an identity matrix. That is, `sbionlmeFit` assumes the random effects are not correlated, and 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 tradeoffs between the number of model parameters and the goodness of fit.

Specifying the Covariance Matrix Pattern of Random Effects

By default, the function used to perform population fits (`nlmeFit`) 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 C_I 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`.

```
[results, simdataI, simdataP] = sbionlmeFit(modelObj,...
    PKModelMapObj, pkDataObject, beta0, options)
```

For more information, see `nlmeFit` in the Statistics Toolbox User's Guide.

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. Likewise, relatively

small estimates for covariances among certain random effects suggests 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 the Covariate Model

If the model in the above example assumes a group-dependent covariate such as weight (w) the model becomes:

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

Thus, the parameter for clearance (Cl) for an individual is $Cl_i = \beta_{Cl} + b_{Cl} + \beta_{Cl/w} * w_i$.

To specify a covariate model, use the 'FEGroupDesign' option.

'FEGroupDesign' is a p-by-q-by-m array specifying a different p-by-q fixed-effects design matrix for each of the m groups. Using the previous example the array would resemble the following:

$$\begin{bmatrix} \begin{bmatrix} 1 & 0 & 0 & W_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 & W_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 & W_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} & \dots & \begin{bmatrix} 1 & 0 & 0 & W_m \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{bmatrix}$$

1 Create the array as follows:

```
%Number of parameters in the model (Phi)
num_params = 3;
%Number of covariates
num_cov = 1;
%Assuming number of groups in the data set is 7
num_groups = 7;
% Array of covariate values
covariates = [75; 52; 66; 55; 70; 58; 62 ];
A = repmat(eye(num_params, num_params+num_cov), [1,1,num_groups]);
A(1,num_params+1,1:num_groups) = covariates(:,1)
```

2 Create a struct with the specified design matrix.

```
options.FEGroupDesign = A;
```

3 Specify the arguments for `sbionlmeft` as shown in “Performing Population Fitting Using `sbionlmeft`” on page 4-56.

For more information, see “Nonlinear Regression Models” in the Statistics Toolbox User’s Guide.

Performing Population Fitting Using `sbionlmeft`

The function `sbionlmeft` lets you specify a SimBiology model that you want to use in fitting. `sbionlmeft` uses the `nlmeft` function from the Statistics Toolbox to fit data with both fixed and random sources of variation using nonlinear mixed-effects and returns the estimates. `nlmeft` 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 the random effects.

1 (Optional) Set the tolerance and maximum iteration options. Use an options structure that is an argument for `sbionlmeft`.

```
options.Options.TolX = 1.0E-4;
options.Options.TolFun = 1.0E-4;
```



```
options.Options.MaxIter = 200;
```

- 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, simdataP] = sbionlmeft(modelObj,...
      PKModelMapObj, pkDataObject, beta0, options);
```

`sbionlmeft` returns the following:

- A `results` structure containing
 - `beta` — Estimates for the fixed effects
 - `psi` — Estimated covariance matrix of the random effects
 - `stats` — Structure with statistics such as AIC, and BIC. For a complete list see `nlmeft` in the Statistics Toolbox User's Guide
 - `b` — Estimated random effects for each group in `observedData`
 - `simdataI` contains the data from simulating the model using the estimated parameter values for individuals. That is, the fixed plus the random effects.
 - `simdataP` contains the data from simulating the model using the estimated parameter values for the population. That is the fixed effects only.
- Plot the data from the data set used in fitting. For example, `data` is the imported data set used for fitting, `ID`, `TIME` and `Y` are the column headers for the column containing group IDs, time, and the response variable respectively.

```
p = sbiotrellis(data, 'ID', 'TIME', 'Y')
```

- Plot the simulation results on the same plot. You can leave the second and third arguments for `sbiotrellis` empty for this example since the second argument requires a function handle which is not applicable here, and the third argument requires the column for the x-axis which is by default `time` and thus need not be specified.

```
p.plot = (simdataP, '', '', PKModelMapObj.Observed)
```

For a description of the results, see `sbionlmeft` in the SimBiology Reference.

For more information, see the following in the Statistics Toolbox User's Guide:

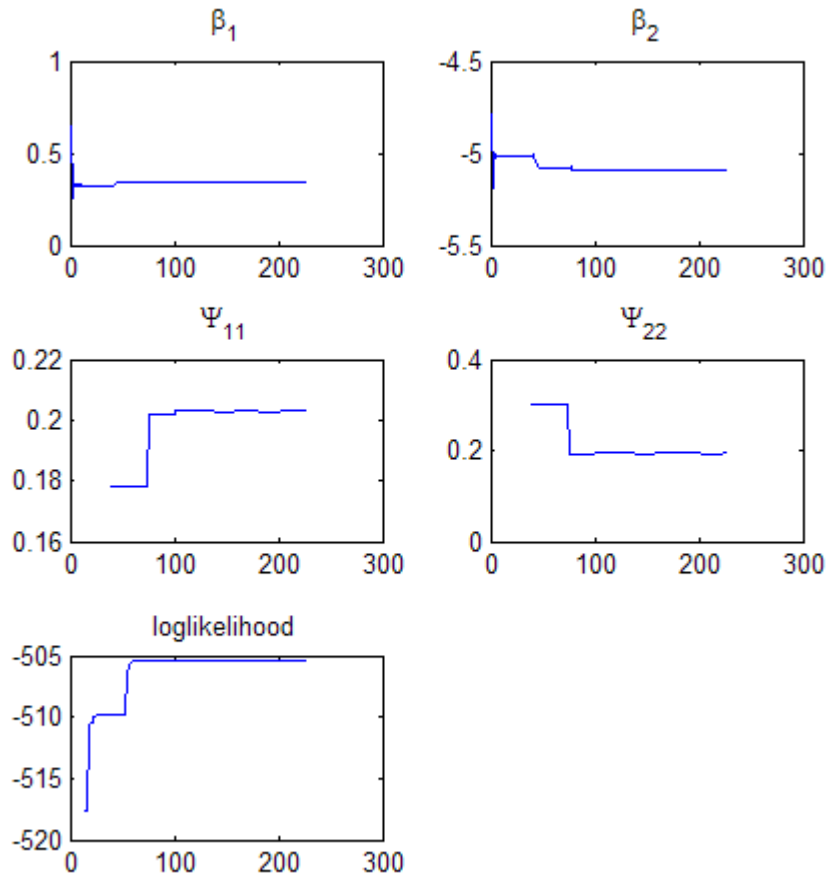
- “Nonlinear Regression Models”
- “Mixed-Effects Models”
- `nlmefit`

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 (β), 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 time taken to return the results is expected to be longer than a few minutes. Use the options structure that is an argument for `sbionlmefit`.

```
% 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. See “Performing Population Fitting Using sbionlmeft” on page 4-56, for information on how to set the maximum iterations.

- Plots for the fixed effects (β) 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 Reference documentation.

Performing Individual Fitting Using `sbionlinfit`

The function `sbionlinfit` 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, pkData, beta0, options);
```

`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 used in fitting. For example, *data* is the imported data set used for fitting, *ID*, *TIME* and *Y* are the column headers for the column containing group IDs, time, and the response variable respectively.

```
p = sbiotrellis(data, 'ID', 'TIME', 'Y')
```

- 4** Plot the simulation results on the same plot. You can leave the second and third arguments for *sbiotrellis* empty for this example since the second argument requires a function handle which is not applicable here, and the third argument requires the column for the x-axis which is by default *time* and thus need not be specified.

```
p.plot = (simdataI, '', '', PKModelMapObj.Observed)
```

For more information, see “Nonlinear Regression Models” and *nlinfit* in the Statistics Toolbox User’s Guide.

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 “Storing and Applying Alternate Model Values Using Variants” on page 1-48.

Note The fitting functions, *population fit* (NLMEFIT) and *individual fit* (NLINFIT), use the initial estimate values specified in “Setting Initial Estimates” on page 4-51, as the initial parameter estimates when fitting parameters (overriding the values in the variant).

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 being used, during fitting there is a temporary change to `sundials` to support events in the model. The change is reversed 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.

Fitting Pharmacokinetic Model Parameters in the SimBiology Desktop

In this section...

“Fitting Parameters” on page 4-63

“Opening a SimBiology Project” on page 4-64

“Adding a Model Task to Fit Parameters” on page 4-64

“Performing Population Fitting” on page 4-64

“Performing Individual Fitting” on page 4-71

“About Simulation Settings and Specifying Alternate Values for Initial Estimates” on page 4-75

“Visualizing Parameter Fitting Results and Generating Diagnostic Plots” on page 4-76

Fitting Parameters

The following steps show you one of the workflows you can use in the SimBiology desktop to fit a PK model and estimate parameters.

- 1 Open a project containing the data and the model as described in “Opening a SimBiology Project” on page 4-64. To learn more about how to add data to a project, and how to create a PK model, see “Importing Data” on page 4-6 and “Creating Pharmacokinetic Models” on page 4-28.
- 2 Add a task that lets you fit parameters as shown in “Adding a Model Task to Fit Parameters” on page 4-64.
- 3 Choose the type of fitting:
 - To perform population fitting see “Performing Population Fitting” on page 4-64.
 - To perform individual fitting see “Performing Individual Fitting” on page 4-71.
- 4 (Optional) Specify alternate values for initial guesses of the parameter values and apply the values during parameter fitting. For more information

on apply alternate values and to learn about constraints on simulation settings, see “About Simulation Settings and Specifying Alternate Values for Initial Estimates” on page 4-75.

- 5 Run the task and visualize results as shown in “Visualizing Parameter Fitting Results and Generating Diagnostic Plots” on page 4-76.

Opening a SimBiology Project

Follow these steps to open a project if you have already saved the project containing the data and the model.

- 1 In the MATLAB Command Window, type:

```
simbiology
```

The SimBiology desktop opens.

- 2 Select **File > Open Project** to open the Open SimBiology Project dialog box.
- 3 Browse to and select the `.sbproj` format file containing the project.
- 4 Click **Open**. The project opens in the SimBiology desktop.

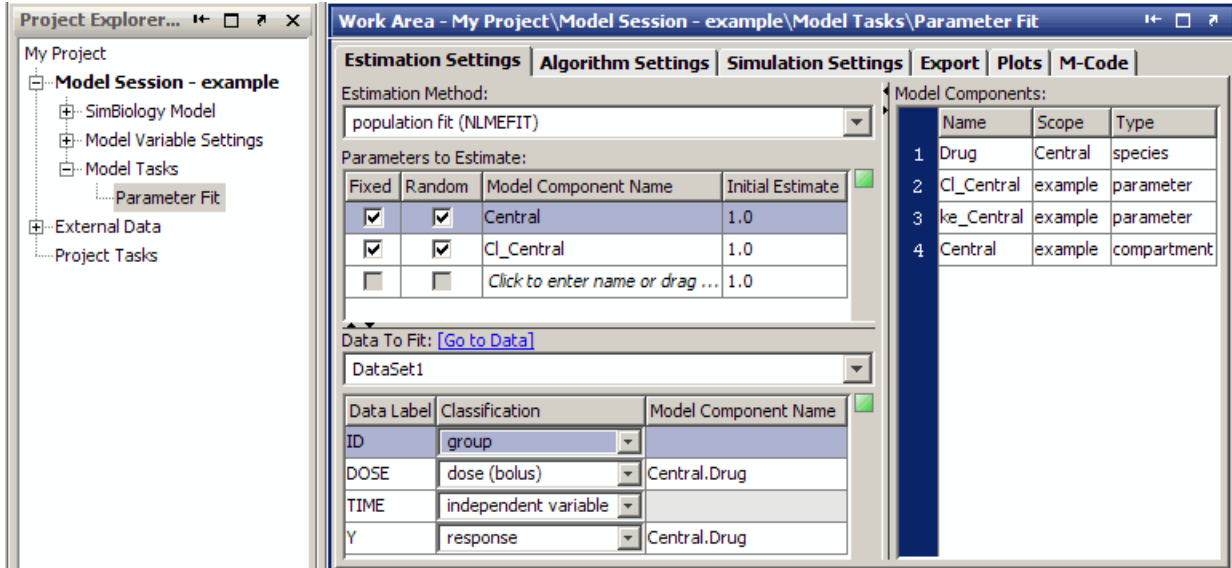
Adding a Model Task to Fit Parameters

- Select **Tasks > Add Task to *ModelName* > Fit Parameters**. The Project Explorer updates with the **Parameter Fit** task and the new task pane opens.

Performing Population Fitting

Specifying the Type of Fitting, the Data to Fit, and Parameters to Estimate

These settings are in the **Parameter Fit** task pane as shown.



- 1 Click the **Estimation Settings** tab if needed.
- 2 From the **Estimation Method** list, select population fit (NLMEFIT). This method uses the `nlmefit` function from the Statistics Toolbox and lets you specify whether the parameters being estimated have random effects. The task returns the fixed effects for the population fit as well as the individual random effects.
- 3 The **Parameters to Estimate** table shows the SimBiology parameters, species, and compartments to estimate. To add parameters to this table, enter a name. Alternatively, from the **Model Components** table (on the right), click and drag the model component and drop it on the **Parameters to Estimate** table.

Tip If the name entered in the table is invalid, you see a red indicator next to the row.

- 4** In the **Parameters to Estimate** table, select **Fixed** to estimate the parameter. In addition, select the **Random** check box if you want to calculate random effects as well.

Note You must specify that at least one component has a random effect. If however, you select all the components as having random effects, this might affect performance depending on the number of components to be estimated.

- 5** The **Initial Estimate** column contains the initial guess for the value of the parameter (**Value** property), the compartment volume (**Capacity** property), or the species initial amount (**InitialAmount** property). Double-click a cell to edit the value.

- 6** From the **Data To Fit** list select the data set to use in fitting.

The **Data Label** column lists the column headers found in the data set.

- 7** The **Classification** list is populated with the selection that the column headings are presumed to represent in the software. If the classification is correct, go to the next step. If the classification is missing or incorrect you can change it as follows:

From the **Classification** list, select the mapping that applies to each column of the data set.

For the column in the data set that represents ...	Select ...
The group ID	group
Time of dosing and sampling	independent variable
Bolus dose amount	dose (bolus)
Infusion dose amount	dose (infusion)
Dose exhibiting zero-order absorption	dose (zero-order)


For the column in the data set that represents ...	Select ...
Dose exhibiting first-order absorption	dose (first-order)
The measured amount or concentration of the compound.	response
The rate of infusion (results of calculation from infusion amount and infusion time)	infusion rate

Note You can select each classification type only once. For example, you cannot classify one column as **dose (bolus)** and another as **dose (infusion)**. Similarly, you cannot select two different columns as **response**.

- 8** The **Model Component Name** column specifies the component in the model that represents the column in the data set. The **Model Component Name** column is only available to columns in the data that represent the dose received and the observed response.

If the **Model Component Name** column lists the correct component, skip to the next step. If the name of the component is incorrect, correct it as follows:

From the **Model Components** table (on the right), click and drag the correct model component and drop it on the **Model Component Name** row that you want to change.

- 9** You can run the parameter fitting task with the default settings by clicking  (Run).

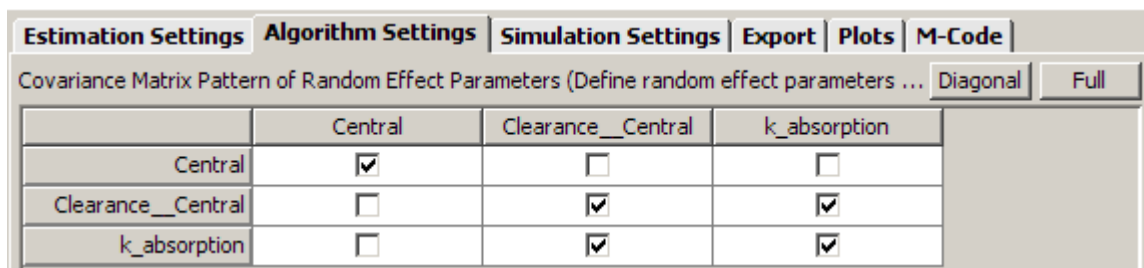
Specifying the Covariance Matrix for Population Fits


For population fitting (population fit (NLMEFIT)) you can specify the covariance matrix pattern of random effect parameters and select the method used to approximate the NLME model likelihood, as shown below.

- 1 Click the **Algorithm Settings** tab.
- 2 In the **Covariance Matrix Pattern of Random Effect Parameters** section select one of the following:

- Click **Diagonal** (default) to specify a diagonal covariance matrix (no covariance relationship between the random effects).
- Click **Full** to specify that there is a covariance relationship between each of the random effects.
- Select specific check boxes to specify block diagonals indicating covariance between a subset of the random effect parameters.

For example, the figure below indicates covariance between the parameters `k_absorption` and `Clearance_Central`.



- 3 From the **Method to Approximate the Non-linear Mixed Effects Model Likelihood** list, select one of the following:
 - 'LME' — (Default) Use the likelihood for the linear mixed-effects model at the current conditional estimates of the parameters.
 - 'RELME' — Use the restricted likelihood for the linear mixed-effects model at the current conditional estimates of the parameters.
 - 'FO' — First-order Laplacian approximation without random effects.
 - 'FOCE' — First-order Laplacian approximation at the conditional estimates of the random effects.
- 4 To run the parameter fitting task with the default tolerance settings, click  **Run**. The parameter fitting task runs and generates the plots and

results discussed in “Visualizing Parameter Fitting Results and Generating Diagnostic Plots” on page 4-76.

Also see `nlmefit` in the Statistics Toolbox documentation for more information.

Setting Tolerance and Maximum Iteration for Population Fits

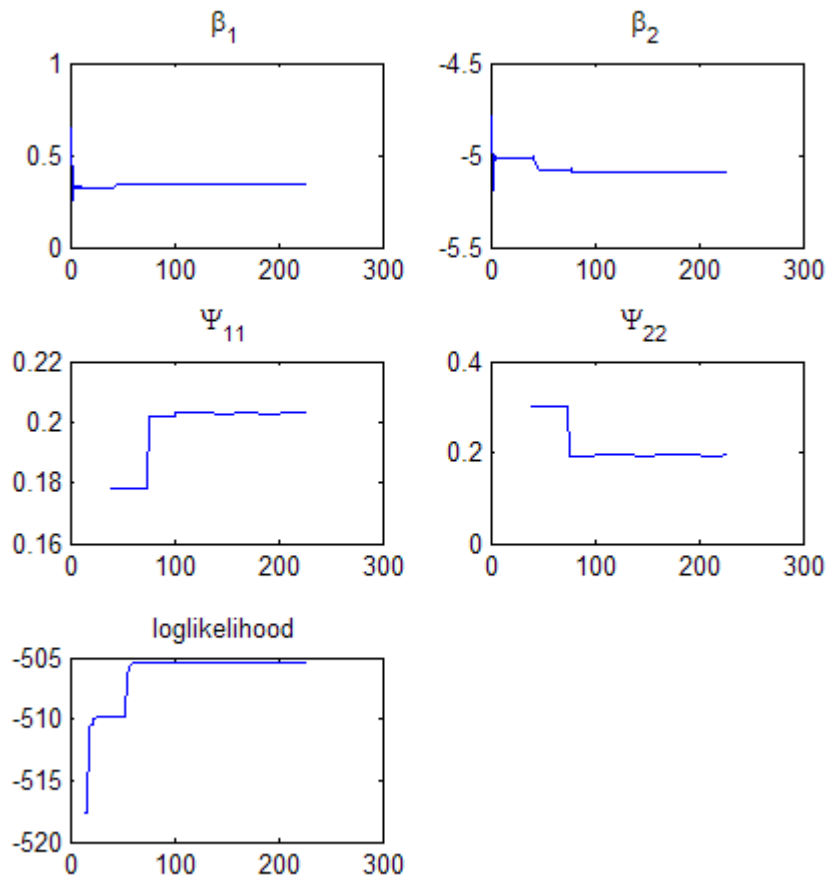
You can optionally set tolerances and specify maximum iterations as follows:

- 1 Click the **Algorithm Settings** tab.
- 2 Under **Advanced Settings**, from the **Optimization Function** list, select `fminsearch` (default) or `fminunc` (available if Optimization Toolbox is installed).
- 3 From the Covariance Matrix Parameterization list, specify the algorithm used internally for the scaled covariance matrix. Select **Cholesky factorization of the matrix logarithm** or **Cholesky factorization**.
- 4 In the **Termination Tolerance on Estimated Fixed and Random Effect Parameters** box, specify a tolerance; the default is $1.0E-4$.
- 5 In the **Termination Tolerance on Log-Likelihood Function** box, specify a tolerance; the default is $1.0E-4$.
- 6 In the **Maximum Iterations** box, specify the maximum number of iterations allowed. The default is 200.

Obtaining the Status of Fitting

By default the parameter fitting task dynamically plots the progress of the fitting task. The task displays plots of the fixed effects (β), 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 the time taken to return the results is expected to be longer than a few minutes.

The following figure shows the type of plot obtained.



To turn the status reporting functionality on or off:

- 1 Click the **Algorithm Settings** tab.
- 2 Select or clear the **Show progress of the Parameter Fit task**.

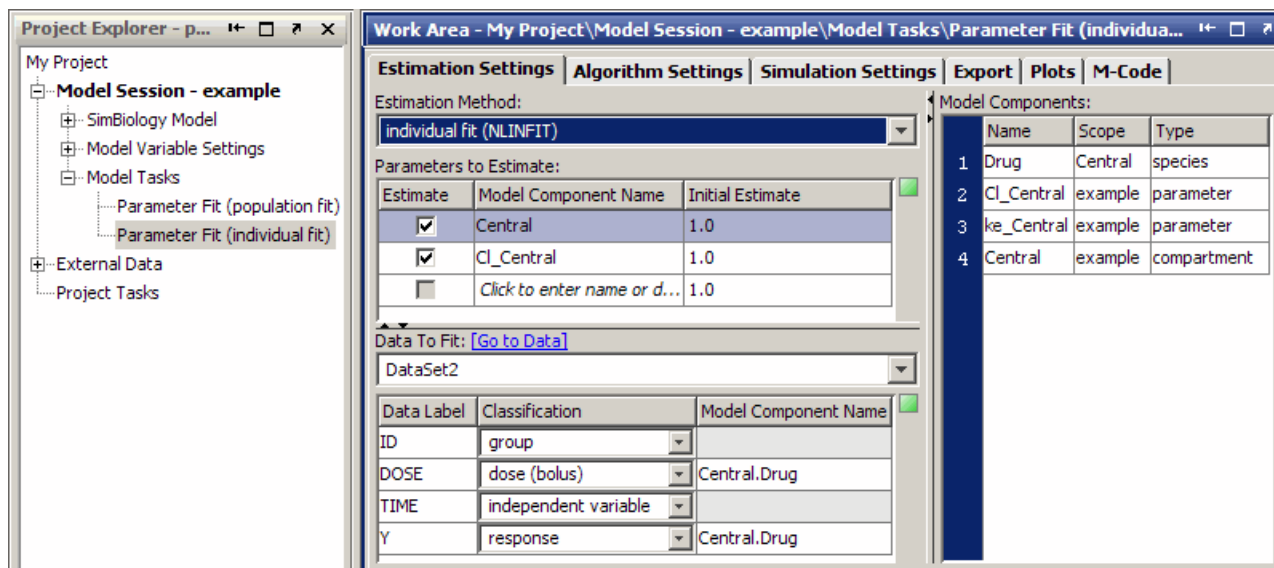
Tips for interpreting status plots:

- 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. See “Performing Population Fitting Using sbionlmeFit” on page 4-56, for information on how to set the maximum iterations.
- Plots for the fixed effects (β) 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

Performing Individual Fitting

Specifying the Type of Fitting, the Data to Fit, and the Parameters to Estimate

These settings are in the **Parameter Fit** task pane as shown.



- 1 Click the **Estimation Settings** tab if needed.
- 2 From the **Estimation Method** list, select individual fit (NLINFIT). individual fit (NLINFIT) uses the `nlinfit` function from the Statistics Toolbox.
- 3 The **Parameters to Estimate** table shows the SimBiology parameters, species, and compartments to estimate. To add parameters to this table, enter a name, or from the **Model Components** table (on the right), click and drag the model component and drop it on the **Parameters to Estimate** table.

Tip If the name entered in the table is invalid you see a red indicator next to the row.

- 4 In the **Parameters to Estimate** table select or clear the **Estimate** check box to estimate a parameter.
- 5 The **Initial Estimate** column contains the initial guess for the value of the parameter (**Value** property), the compartment volume (**Capacity**

property), or the species amount (**InitialAmount** property). Double-click a cell to edit the value.

- 6 From the **Data To Fit** list, select the data set to use in fitting.

The **Data Label** column lists the column headers found in the data set.

- 7 The **Classification** list is populated with the selection that the column headings are presumed to represent in the software. If the classification is correct go to the next step. If the classification is missing or incorrect, you can change it as follows:

From the **Classification** list, select the mapping that applies to each column of the data set.


For the column in the data set that represents ...	Select ...
The group ID	group
Time of dosing and sampling	independent variable
Bolus dose amount	dose (bolus)
Infusion dose amount	dose (infusion)
Dose exhibiting zero-order absorption	dose (zero-order)
Dose exhibiting first-order absorption	dose (first-order)
The measured amount or concentration of the compound.	response
the rate of infusion (results of calculation from infusion amount and infusion time)	infusion rate

Note You can only select each classification type once. For example, you cannot classify one column as dose (bolus) and another as dose (infusion). Similarly, you cannot select two different columns as response.

- 8** The **Model Component Name** column specifies the component in the model that represents the column in the data set. The **Model Component Name** column is only available to columns in the data that represent the dose received and the observed response.

If the **Model Component Name** column lists the correct component, skip to the next step. If the name of the component is incorrect, correct it as follows:

From the **Model Components** table (on the right), click and drag the correct model component and drop it on the **Model Component Name** row that you want to change.

- 9** You can run the parameter fitting task with the default settings by clicking  (Run).

Setting Tolerance and Maximum Iteration for Individual Fits

- 1** Click the **Algorithm Settings** tab.
- 2** In the **Termination Tolerance on the Estimated Coefficients** box, specify a tolerance; the default is 1.0E-8.
- 3** In the **Termination Tolerance on the Residual Sum of Squares** box, specify a tolerance; the default is 1.0E-8.
- 4** In the **Maximum Iterations** box, specify the maximum number of iterations allowed. The default is 100.

About Simulation Settings and Specifying Alternate Values for Initial Estimates

In the **Simulation Settings** tab you can specify **Variants** that let you store and apply alternate values for model components during a simulation. For more information about variants and how to add variants see “Storing and Applying Alternate Model Values Using Variants” on page 1-48.

- In the **Simulation Settings** tab, under **Variants**, select or clear the **Use In Task** check box for a variant to use in the task.

Note The fitting functions (population fit (NLMEFIT) and individual fit (NLINFIT)) use the initial estimate values specified in the **Estimation Settings** tab, as the initial parameter estimates when fitting parameters (overriding the values in the variant).


By default, the **Configuration Settings** list contains a default configuration set which is the active configuration set. You can add other configuration sets or change the settings in the default configuration set as shown in “Performing Simulations in the SimBiology Desktop” on page 2-2.

From the **Configuration Settings** list you can select a different configuration set to use in the parameter fitting task. For the chosen configuration set, you can specify model components for which you want to record the data, in the **Data Logging** tab. The task results include a plot that shows the results of simulating the model using the estimated values returned by the parameter fit task. The result include the components that you selected in the **Data Logging** tab.

The model object’s default configuration set uses the `sundials` solver.

If you change the solver type in the configuration set being used, during fitting there is a temporary change to `sundials` to support events in the model, and the change is reversed 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 (`sbionlmeft` or `sbionlinfit`) will use the tolerance options specified in the deterministic solver.

Visualizing Parameter Fitting Results and Generating Diagnostic Plots

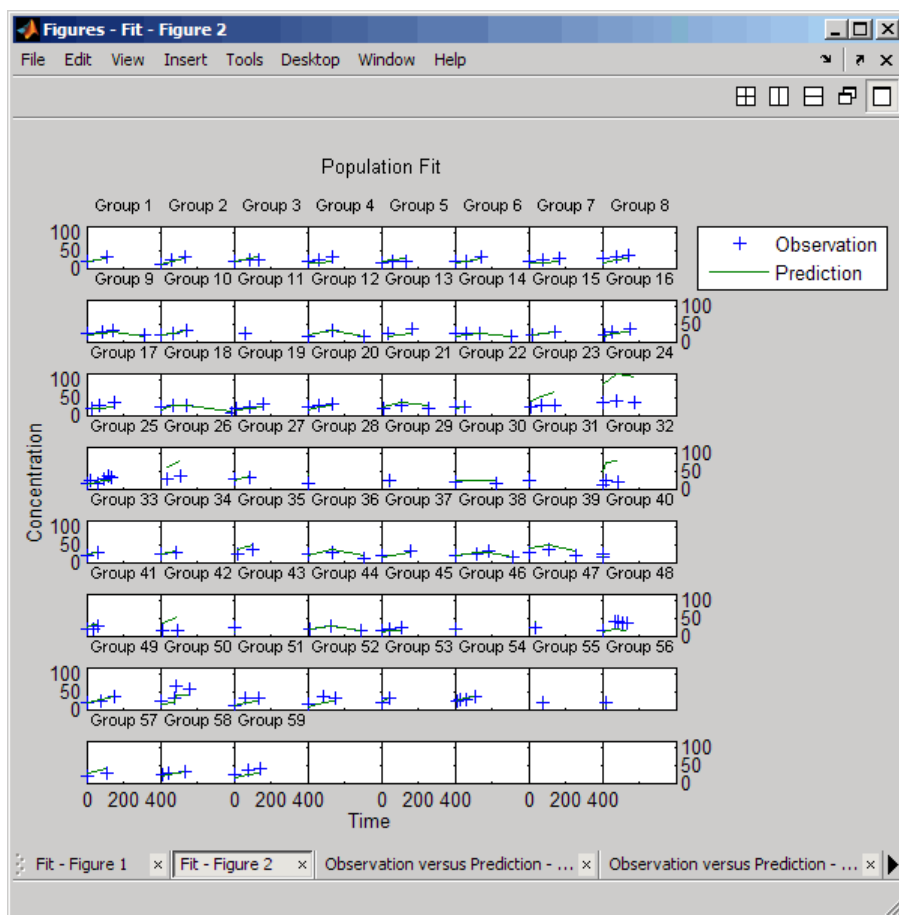
To run the parameter fitting task, click  (Run). The parameter fitting task runs and generates plots and results. If you chose to run the `population fit` task, the task returns results for individual fits and the population fit.

In the **Project Explorer** under **Parameter Fit**, the task adds a **Data** item containing the following results:

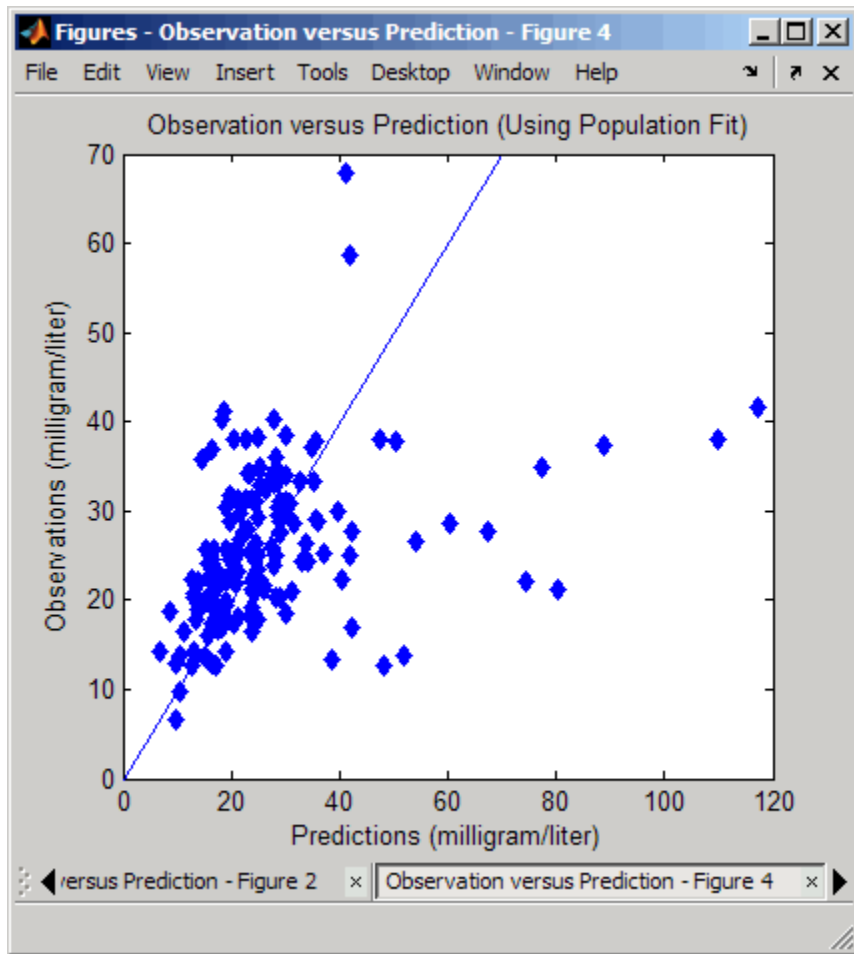
- For individual fits:
 - Estimated parameter values for each group
 - Mean and standard deviation for the estimated values
 - The estimated covariance matrix for each group
- For population fits:
 - Estimated parameter values
 - Estimated random effects
 - The maximized log-likelihood for the fitted model
 - The estimated error variance for the fitted model
 - The Akaike information criterion (AIC) for the fitted model
 - The Bayesian information criterion (BIC) for the fitted model
 - The standard errors for the estimates of the fixed effects
 - The error degrees of freedom for the model
 - Estimated Covariance Matrix of Random Effects

The task also generates the following diagnostic plots for individual and population fits:

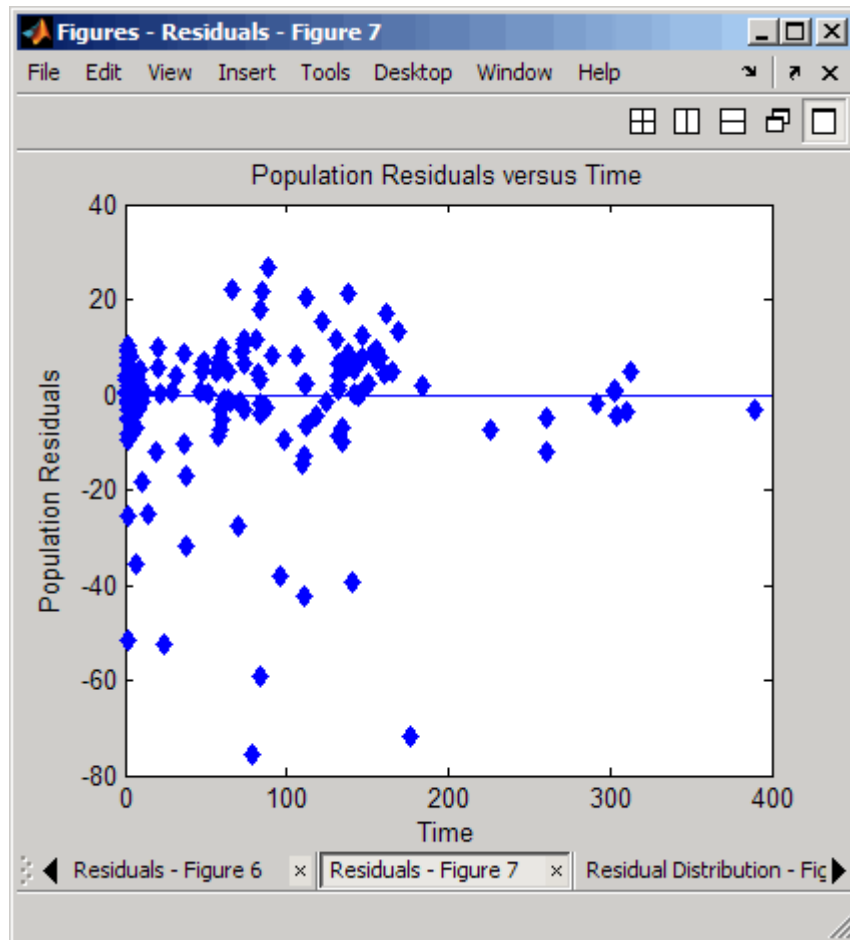
- The predicted time courses and observations for an individual or the population



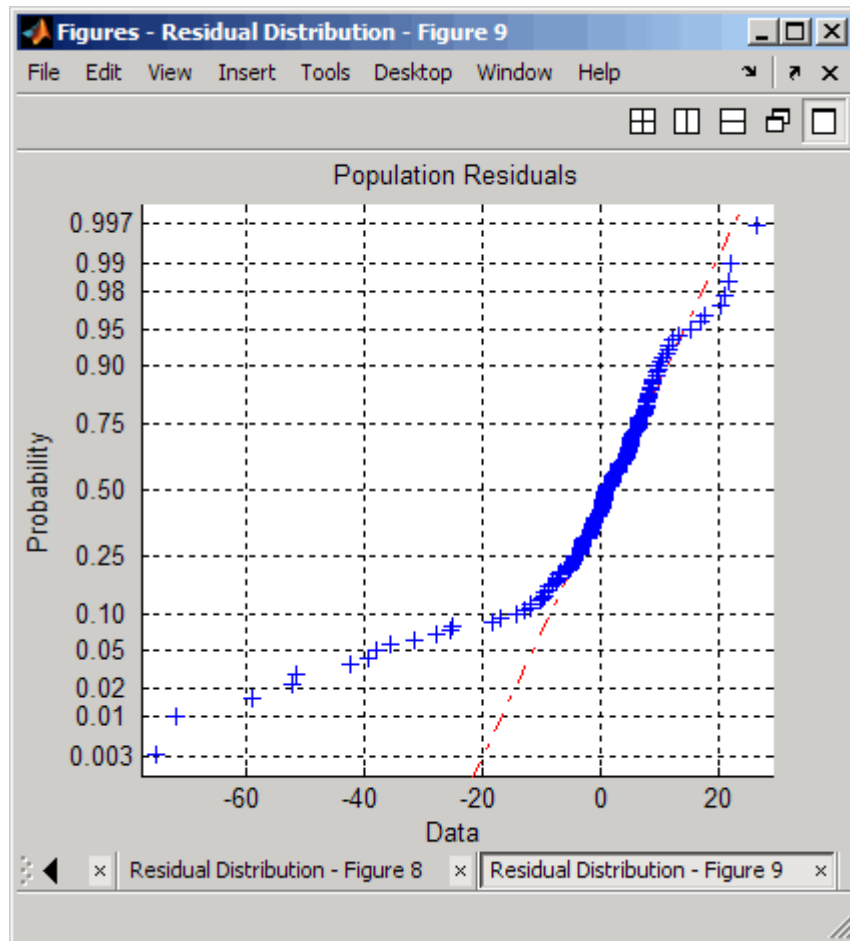
- Observed versus predicted values



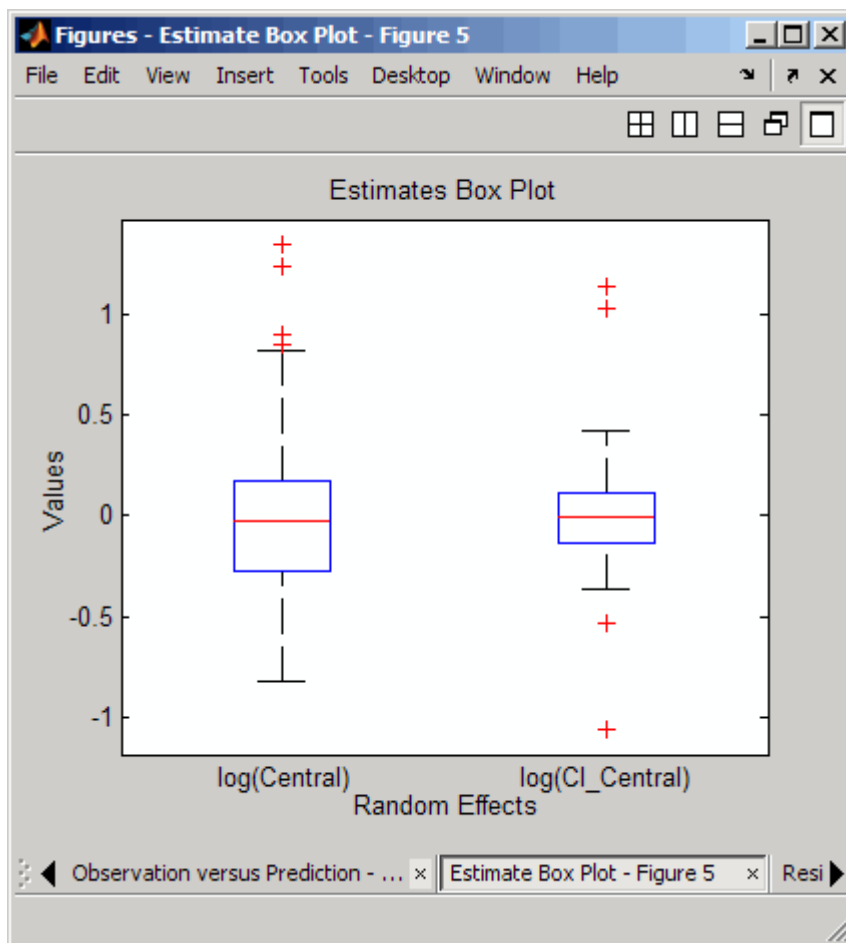
- Residuals versus time, group or predictions



- Distribution of the Residuals



- A box-plot for random effects or parameter estimates from individual fitting



For help on working with plots and a description of each plot type, see the SimBiology context-sensitive help for the **Parameter Fit** pane.

- 1** From the SimBiology desktop, select **Help > SimBiology Desktop Help**.
- 2** In the **Project Explorer**, select **Parameter Fit**.

See also, “Visualizing Results Using Plot Types” on page 3-73

A

algorithm
 explicit tau-leaping 2-12 to 2-13
 implicit tau-leaping 2-13
 SSA 2-12

analysis
 conserved moieties 3-1
 parameter estimation 3-1
 parameter fitting 4-1
 pharmacokinetics 4-1
 sensitivity 3-1

B

boundary condition
 definition of 1-17
 use of 1-17

BoundaryCondition
 property 1-17

C

conserved moieties 3-58

constant amount
 definition of 1-17
 use of 1-17

ConstantAmount
 property 1-17

D

deterministic solvers
 nonstiff 2-8
 stiff 2-10

E

enzyme kinetics
 differential equations in 1-10
 irreversible Henri-Michaelis-Menten
 kinetics 1-10

mass action kinetics 1-10
 single substrate 1-10

explicit tau-leaping algorithm 2-13

export
 model component data 1-68

H

Henri-Michaelis-Menten kinetics
 irreversible 1-10

I

implicit tau-leaping algorithm 2-13

import
 model component data 1-68

Irreversible Henri-Michaelis-Menten kinetics
 example of 1-10

M

mass action kinetics
 example of 1-10
 first-order reactions 1-3
 modeling with 1-3
 reversible 1-3
 second-order reactions 1-3
 zero-order reactions 1-3

Michaelis-Menten kinetics. *See*
 Henri-Michaelis-Menten kinetics

model
 pharmacokinetic 4-1

moiety conservation 3-58

N

NLME 4-2

nonstiff
 ode solvers 2-8

O

ode solvers
 nonstiff 2-8
 stiff 2-10

P

parameter estimation 3-45
parameters
 changing scope of 1-22
 estimation of 3-1 4-1
 scope of 1-22

R

references
 yeast G protein cycle model 3-24 3-38
rules
 rate rules 1-24

S

scope

 definition of 1-22

sensitivity analysis 3-25

SimBiology

 simulation overview 2-2

stiff

 ode solvers 2-10

stochastic (SSA) algorithm 2-12

stochastic solvers

 explicit tau-leaping algorithm 2-12

 implicit tau-leaping algorithm 2-12

 references 2-12

 SSA 2-12

Y

yeast G protein cycle model

 references 3-24 3-38

 simulation results

sst2Δ 1-53