SimBiology® 3
User's Guide
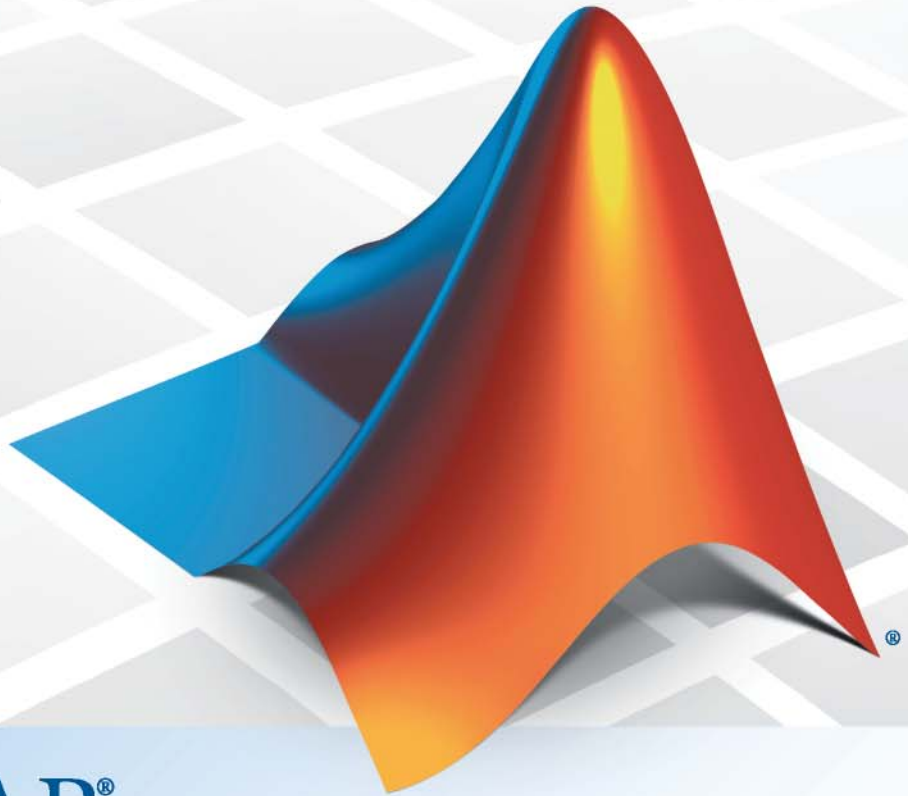
MATLAB®

How to Contact MathWorks

	www.mathworks.com	Web
	comp.soft-sys.matlab	Newsgroup
	www.mathworks.com/contact_TS.html	Technical Support
	suggest@mathworks.com	Product enhancement suggestions
	bugs@mathworks.com	Bug reports
	doc@mathworks.com	Documentation error reports
	service@mathworks.com	Order status, license renewals, passcodes
	info@mathworks.com	Sales, pricing, and general information

☎ 508-647-7000 (Phone)

☐ 508-647-7001 (Fax)

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

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

SimBiology® User's Guide

© COPYRIGHT 2005–2010 by The MathWorks, Inc.

Trademarks

Patents

Revision History

Contents

Simulation

2

Analysis

3

4

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.

- "Desktop Example — Using User-Defined Functions in Expressions" on page 1-70

- "Importing and Exporting Model Component Data" on page 1-79

SimBiology Model Components

Organization of SimBiology Objects

If you plan to work mainly from the SimBiology desktop GUI, you can skip this section and go to "Model" on page 1-6. The GUI allows you to interactively work with SimBiology model components without having prior knowledge of object-oriented programming, or the object implementation thereof in the software product.

If you plan on using the product at the command line or have workflows that span the desktop and the command line, it is useful understand the hierarchical organization of SimBiology objects.

SimBiology objects are organized as a hierarchical list of objects under one `root` object. Models and each model component are represented as objects. Further, there are objects to store simulation settings (`configset` object) and data from simulation and analysis tasks (`SimData` object). Objects created at the command line or the SimBiology desktop are contained in the `root` object.

Models must contain at least one species object and one compartment object. The hierarchy of objects in models is as follows:

```
Model
   Compartment
      Species
   Parameters
   Reaction
      Kinetic Law
          Parameters
   Rules
   Events
   Variant
   Configset

SimData
```

Parameters can be scoped to a kinetic law or the model.

The function `sbioreset` deletes all model and simulation objects at the root level. You should remove the objects from the workspace with the function `clear`. If the SimBiology desktop is open, note that calling `sbioreset` at the command line also deletes model and data objects that are open in the desktop.

All SimBiology objects are all handle objects. A handle is a reference to an object. If you copy an object's handle, MATLAB® copies only the handle and both the original and copy refer to the same object data. If a function modifies a handle object passed as an input argument, the modification affects the original object. For example,

```
%Create a model
Mobj = sbiomodel('cell')

SimBiology Model - cell

   Model Components:
      Compartments:     0
      Events:           0
      Parameters:       0
      Reactions:        0
```

```
      Rules:              0
      Species:            0

%Assign the model object to another variable
newMobj = Mobj;
% Modify the new variabl
compObj = addcompartment(newMobj, 'contents');
set(newMobj, 'Name', 'newcell')
% The new variable has a compartment and a new name
newMobj

   SimBiology Model - newcell

   Model Components:
     Compartments:      1
     Events:            0
     Parameters:        0
     Reactions:         0
     Rules:             0
     Species:           0

% The change is reflected in the original variable
Mobj

SimBiology Model - newcell

   Model Components:
     Compartments:      1
     Events:            0
     Parameters:        0
     Reactions:         0
     Rules:             0
     Species:           0
```

The recommended method to store your SimBiology models is to create and store project files.

Model

A SimBiology *model* is a collection of interrelated reactions, rules, and events that transform, transport, and bind species. The model components are compartments, species, reactions, parameters, rules, and events. After creating a model, add a compartment, then you can add reactions to the model, define a kinetic law for each reaction, and add the parameters for the reaction.

Properties define the characteristics of an object. For example, a reaction object includes properties for reactions, reaction rates, and links to associated species, parameters, and kinetic laws. You can interactively change object properties in the graphical user interface. Use the `get` and `set` commands to list object properties and change their values at the command line.

Compartment

A *compartment* allows you to associate pools of species to physically isolated regions.

Since compartments define physically isolated regions, species are assigned to compartments. Reactions, rules, events, and parameters are assigned to the model.

All models must have at least one compartment and all species in a model must be assigned to a compartment. You can create a compartment within another compartment, but compartment names must be unique within a model. For information on specifying names of compartments, see `Name` in the SimBiology Reference documentation.

Species

A *species* is a chemical or entity that participates in reactions, for example, `DNA`, `ATP`, `Pi`, `creatine`, `G-Protein`, or `Mitogen-Activated Protein Kinase (MAPK)`. Species amounts can vary or remain constant during a simulation.

For information on specifying names of species, see `Name` in the SimBiology Reference documentation.

Reaction

A *reaction* describes a transformation, transport, or binding process that changes one or more species. Typically, the change is the amount of a species.

Examples:

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

Spaces are required before and after species names and stoichiometric values. Stoichiometry values must be greater than zero. If the stoichiometry value is not specified it is assumed to be 1. In a model with a single compartment, the species is specified using its name. In a model with multiple compartments, the species must be specified as compartmentName.speciesName. Enclose names with non-alphanumeric characters in brackets.

Kinetic Law

A *kinetic law* provides a mechanism for applying a specific rate law to multiple reactions. It acts as a template for the reaction rate. The kinetic law is defined by an expression, and the species and parameter variables used in the expression.

The ReactionRate is the result of defining the names of parameters and species variables in a rate law. To get ReactionRate, the species variables and parameter variables that participate in the reaction rate should be clearly assigned in the **Project Settings-Reactions** pane on the **Kinetic Law** tab.

For example, the kinetic law Henri-Michaelis-Menten is expressed in the software as

$$V_m * S / (K_m + S)$$

where Vm and Km are defined as parameters and S is defined as a species. (To see parameter and species definitions for a kinetic law, access the **Kinetic Law Manager** from the **Desktop** menu.)

By applying the kinetic law Henri-Michaelis-Menten to a reaction

```
A -> B
with Vm = Va and Km = Ka
and S = A
```

the rate equation is

Va*A/(Ka + A)

A, Va, and Ka are defined in the **Kinetic Law** tab.

Reaction Rate Equation

The *reaction rate equation* is a mathematical expression that describes the rate at which the reactants combine to form products.

For a reaction following Mass Action kinetics,

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

the rate equation is written as

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

Note If you write your own reaction rate expression, and it is not continuous and differentiable, see "Using Events to Address Discontinuities in Rule and Reaction Rate Expressions" on page 1-50 before simulating your model.

Parameters

A *parameter* is a quantity that can change or can be constant. Parameters are generally used to define rate constants. But, you could also use parameters as constants to modify model component values in rules and events.

For information on specifying names of parameters see Name in the SimBiology Reference documentation.

Scope refers to the definition of a parameter at the model (global) level or kinetic law (local) level.

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

How to Use Scope

For reaction rate, SimBiology hierarchically uses the value of the parameter with scope at the kinetic law level first. If there is no such parameter at the kinetic law level, SimBiology looks for the parameter with scope at the model level.

If there are two parameters with 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.

Rules

A *rule* is a mathematical expression that modifies a species amount, compartment capacity, or a parameter value. There are four types of SimBiology rules:

- `algebraic` — Algebraic rules are evaluated continuously during a simulation. An algebraic rule takes the form `0 = Expression` and the rule is specified as the `Expression`. For example, you could write a mass conservation expression such as `species_total = species1 + species2` where `species_total` is the independent variable. In SimBiology, write the rule as `species1 + species2 - species_total`.

- `initialAssignment` — `initialAssignment` rules are evaluated once at the beginning of a simulation. `initialAssignment` rules are expressed as `Variable = Expression`. For example, you could write an `initialAssignment` rule to set the amount of `species1` to be proportional to `species2`.

  ```
  species1 = k/species2
  ```

- `repeatedAssignment` — `repeatedAssignment` rules are evaluated at every time-step during a simulation. These time steps are determined by

the solver during the simulation process. repeatedAssignment rules are expressed as Variable = Expression. For example, you could use the rule to specify the amount of species1 to always be proportional to species2.

    species1 = k/species2

- rate — Rate rules are evaluated continuously during a simulation. Rate rules are determined by dVariable/dt = Expression, which is expressed in the software as Variable = Expression. For example, to define the rate of change in the quantity of species3 (dspecies3/dt), write the rule in the software as:

    species3 = k * (species1 + species2)

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

Note If your algebraic, repeatedAssignment, or rate rule expression is not continuous and differentiable, see "Using Events to Address Discontinuities in Rule and Reaction Rate Expressions" on page 1-50 before simulating your model.

Tip If you plan to use a parameter in a rule, remember to set the **Scope** of the parameter to model. Further, if you are using an algebraic or rate rule to vary the value of a parameter during the simulation, clear the **ConstantValue** check box for the parameter in the **Parameters** pane, **Settings** tab.

For more information, see "Changing Model Component Values Using Rules" on page 1-34 in the SimBiology User's Guide.

Events

Events are used to describe sudden changes in model behavior. An event lets you specify discrete transitions in model component values that occur when a user-specified condition become true. You can specify that 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, change reaction rates in response to addition or removal of species. For more information about events, evaluation of events, and examples, see "Changing Model Component Values Using Events" on page 1-42 in the SimBiology User's Guide.

Configuration Sets

Configuration sets contain the options that the solver uses during simulation of a model. To add, view or edit configuration sets associated with a model, in the **Project Explorer**, click **Configuration Settings**. The configuration set contains the following options for you to choose:

- Type of solver

- Data to record (**Data Logging** tab)

- Stop time for the simulation

- Solver error tolerances, and for ode solvers – the maximum time step the solver should take

- Whether to perform sensitivity analysis during simulation

- Whether to perform dimensional analysis and unit conversion during simulation

- Species and parameter input and output factors for sensitivity analysis

For more information see "Performing Simulations" on page 2-2 in the *SimBiology User's Guide.*

Variants

Variants let you store the names and values of model components and allow you to use the values stored in the variant as the alternate value to be applied during a simulation. To add, view or edit variants associated with a model, in the **Project Explorer**, click **Variants**. You can store values for species `InitialAmount`, parameter `Value`, and compartment `Capacity`, in variants.

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 by clicking **Commit**.

For more information see "Storing and Applying Alternate Model Values Using Variants" on page 1-59in the *SimBiology User's Guide*.

Simulation Data

The SimBiology `SimData` object contains simulation data. The `SimData` object holds time and state data as well as metadata, such as the types and names for the logged states, or the configuration set used during simulation. For more information see `SimData object`.

Defining Reaction Rates with Mass Action Kinetics

Definition of Mass Action Kinetics

Mass action describes the behavior of reactants and products in an elementary chemical reaction. Mass action kinetics describes this behavior as an equation where the velocity or rate of a chemical reaction is directly proportional to the concentration of the reactants.

Zero-Order Reactions

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

```
      reaction: null -> P
 reaction rate: k mole/second
       species: P =  O 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:
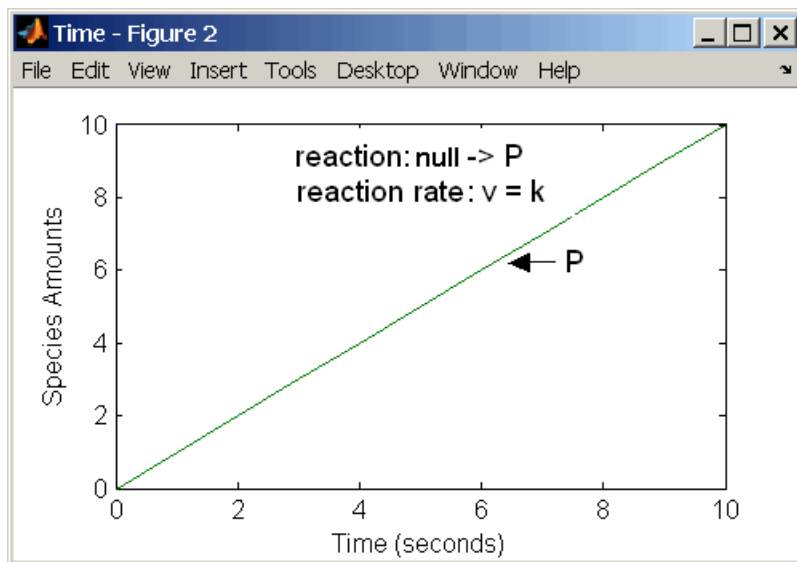
Zero-Order Mass Action Kinetics

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 =  O mole/liter
    parameters: k =  1 1/second
```

Entering the reaction above into the software and simulating produces the following results:
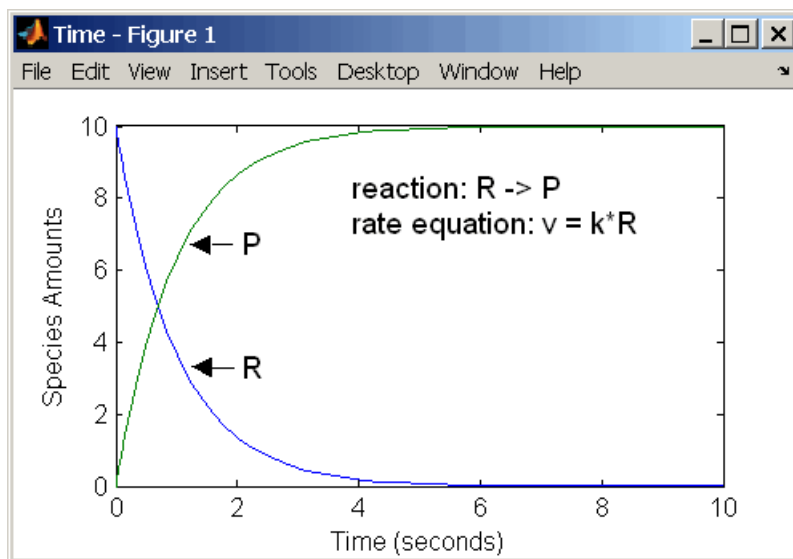
First-Order Mass Action Kinetics

Second-Order Reactions

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

```
      reaction: 2 R -> P
 reaction rate: k*R^2 mole/(liter*second)
       species: R = 10 mole/liter
               P =  0 mole/liter
    parameters: k =  1 liter/(mole*second)
```

Entering the reaction above into the software and simulating produces the following results:
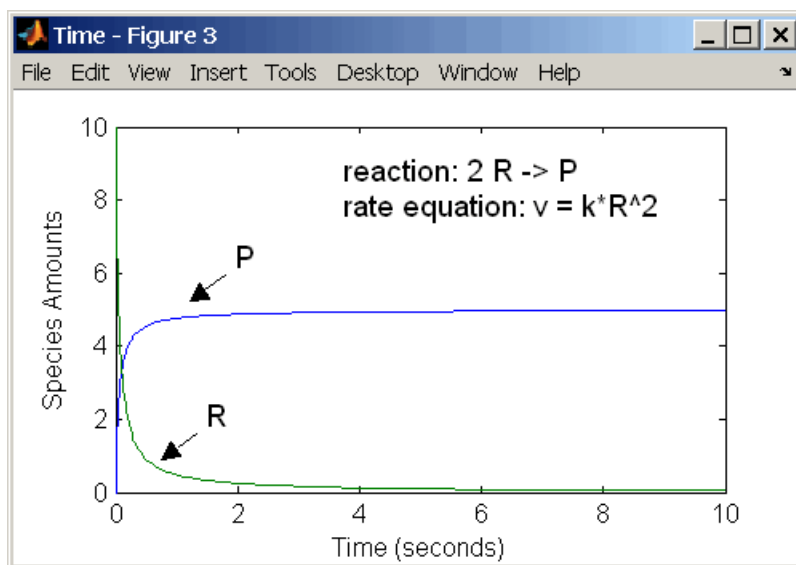
Second-Order Kinetics with Single Reactant

With two reactants, the reaction rate depends on the concentration of two of the reactants.

```
      reaction: R1 + R2 -> P
 reaction rate: k*R1*R2 mole/(liter*second)
```

```
        species: R1 = 10 mole/liter
                 R2 =  8 mole/liter
                  P =  0 mole/liter
     parameters:  k =  1 liter/(mole*second)
```

Enter the reaction above into the software and simulating produces the following results. There is a difference in the final values because the initial amount of one of the reactants is lower than the other. After the first reactant is used up, the reaction stops.
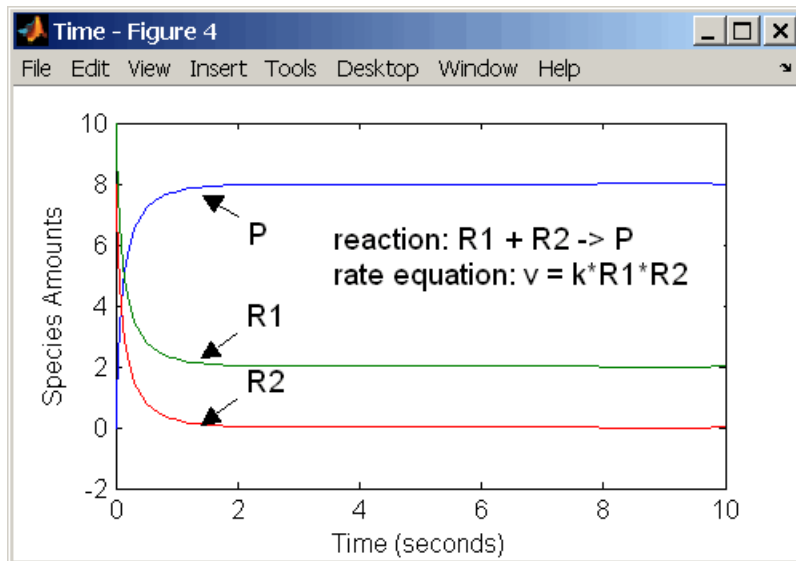
Second-Order Kinetics with Two Reactants

Reversible Mass Action

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

```
      reaction: R <-> P
 reaction rate: kf*R - kr*P mole/(liter*second)
       species: R = 10   mole/liter
                P =  0   mole/liter
    parameters: kf = 1   1/second
                kr = 0.2 1/second
```

Entering the reaction above into the software and simulating produces the following results. At equilibrium when the rate of the forward reaction equals the reverse reaction, `v = kf*R - kr*P = 0` and `P/R = kf/kr`.
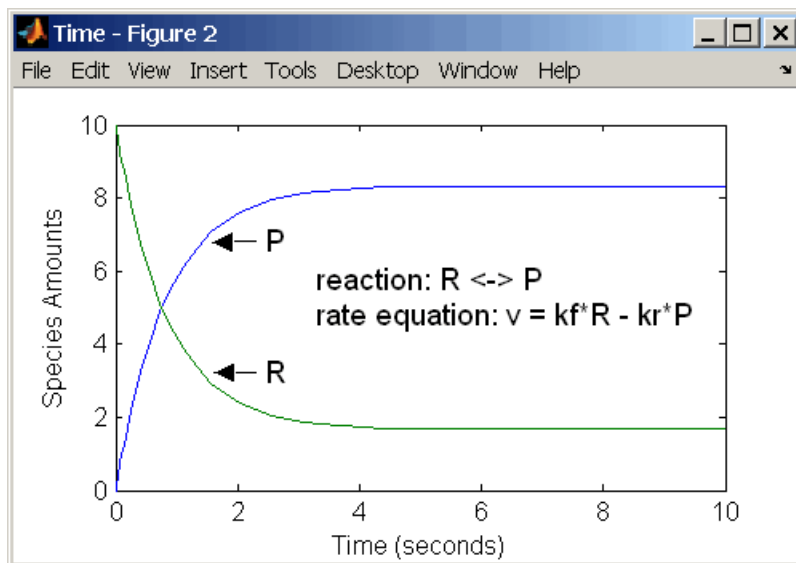
Defining Reaction Rates with Enzyme Kinetics

In this section...

Simple Model for Single Substrate Catalyzed Reactions

A simple model for enzyme-catalyzed reactions starts a substrate S reversibly binding with an enzyme E. Some of the substrate in the substrate/enzyme complex is converted to product P with the release of the enzyme.

$$S + E \underset{k1r}{\overset{k1}{\rightleftharpoons}} ES \xrightarrow{k2} E + P$$
$$v1 = k1[S][E], \quad v1r = k1r[ES], \quad v2 = k2[ES]$$
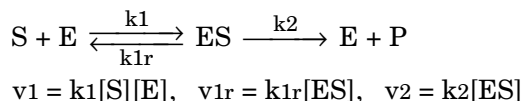
This simple model can be defined with

- Differential rate equations. See "Enzyme Reactions with Differential Rate Equations" on page 1-20.

- Reactions with mass action kinetics. See "Enzyme Reactions with Mass Action Kinetics" on page 1-22.

- Reactions with Henri-Michaelis-Menten kinetics. See "Enzyme Reactions with Irreversible Henri-Michaelis-Menten Kinetics" on page 1-23.

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 1-20) 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 1-37.
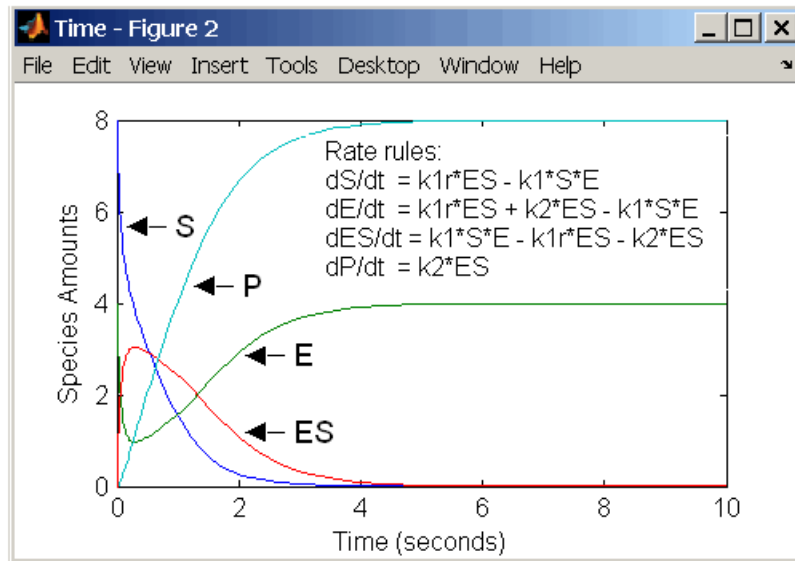
Alternatively, you could remove the rate rule for `ES`, add a new species `Etotal` for the total amount of enzyme, and add an algebraic rule `0 = Etotal - E - ES`, where the initial amounts for `Etotal` and `E` are equal.

```
      reactions: none
 reaction rate: none
     rate rules: dS/dt = k1r*ES - k1*S*E
                 dE/dt = k1r*ES + k2*ES - k1*S*E
                 dP/dt = k2*ES
 algebraic rule: 0 = Etotal - E - ES
        species: S =  8   mole
                 E =  4   mole
                ES =  0   mole
                 P =  0   mole
            Etotal = 4    mole
     parameters: k1 = 2   1/(mole*second)
                 k1r = 1   1/second
                 k2 = 1.5 1/second
```

Enzyme Reactions with Mass Action Kinetics

Determining the differential rate equations for the reactions in a model
is a time-consuming process. A better way is to enter the reactions for a
single substrate enzyme reaction mechanism directly into the software. The
following example using models an enzyme catalyzed reaction with mass
action kinetics. For a description of the reaction model, see "Simple Model for
Single Substrate Catalyzed Reactions" on page 1-20.

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

      reaction: ES -> S + E
 reaction rate: k1r*ES (unbinding)

      reaction: ES -> E + P
 reaction rate: k2*ES (transformation)
       species: S =  8   mole
                E =  4   mole
               ES =  0   mole
                P =  0   mole
    parameters: k1  = 2   1/(mole*second)
                k1r = 1   1/second
                k2  = 1.5 1/second
```

The results for a simulation using reactions are identical to the results from using differential rate equations.
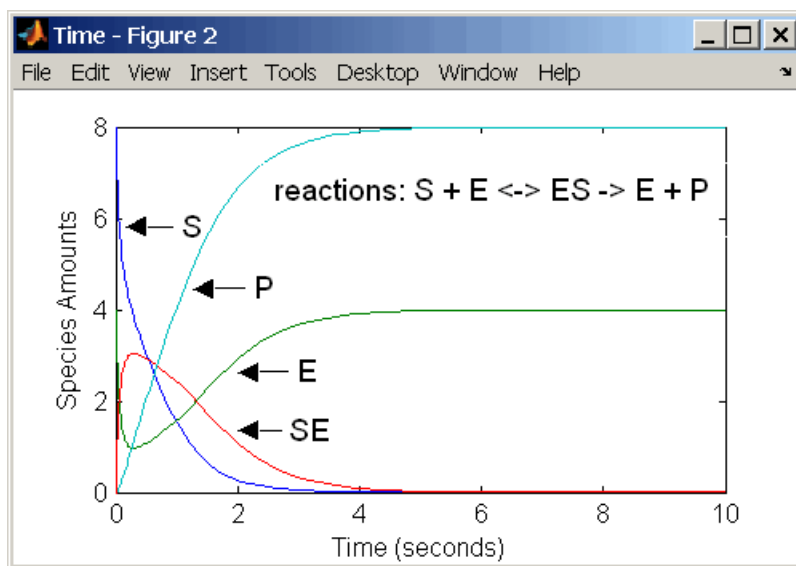
Enzyme Reactions with Irreversible Henri-Michaelis-Menten Kinetics

Representing an enzyme-catalyzed reaction with mass action kinetics requires you to know the rate constants k1, k1r, and k2. However, these rate constants are rarely reported in the literature. It is more common to give the rate constants for Henri-Michaelis-Menten kinetics with the maximum velocity Vm=k2*E and the constant Km = (k1r + k2)/k1. The reaction rate for a single substrate enzyme reaction using Henri-Michaelis-Menten kinetics is given below. For information about the model, see "Simple Model for Single Substrate Catalyzed Reactions" on page 1-20.

$$v = \frac{Vmax[S]}{Km + [S]}$$

The following example models an enzyme catalyzed reaction using Henri-Michaelis-Menten kinetics with a single reaction and reaction rate equation. Enter the reaction defined below into the software and simulate.

```
      reaction: S -> P
 reaction rate: Vmax*S/(Km + S)
       species:    S = 8     mole
                   P = 0     mole
    parameters: Vmax = 6     mole/second
                  Km = 1.25 mole
```

The results show a plot slightly different from the plot using mass action kinetics. The differences are due to assumptions made when deriving the Michaelis-Menten rate equation.
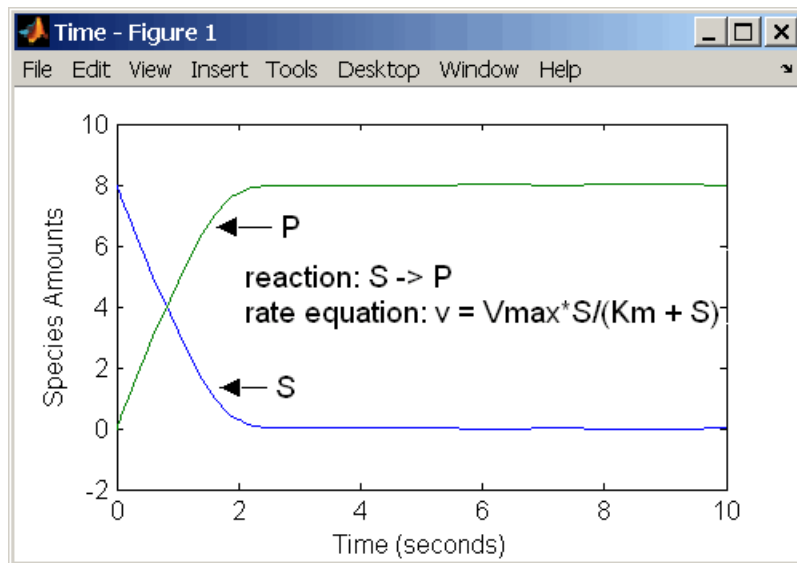
Evaluation of Reaction Rate

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  > c`. Using mass action kinetics, the reaction rate is `k*a*b`, where `k` is the rate constant of the reaction. If you specify that initial amounts of `a` and `b` are `0.01 molarity` and `0.005 molarity` respectively, then the reaction rate is in concentration/time (and units of `molarity/second`)

if the units of k are `1/(molarity*second)`. If you specify k with another equivalent unit definition, for example, `1/((moles/liter)*second)`, SimBiology checks whether the physical quantities match. If the physical quantities do not match, you see an error and the model is not simulated.

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

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

Using Constant Amount and Boundary Condition for Species

In this section...
"Definition of Constant and Boundary Properties" on page 1-27
"Changing Species Amounts with Reactions or Rules " on page 1-28
"Keeping Species Amount Unchanged" on page 1-28
"Constant = NO, Boundary = YES" on page 1-29
"Constant = YES, Boundary = YES" on page 1-30
"Model Edges" on page 1-31

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 Vm = k2*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. dSpecies/dt does not exist.

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

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

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

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

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

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

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

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

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

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 `Vmax` 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,

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

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

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 1-32 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

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. These time steps are determined by the solver during the simulation process. `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.

Note If your `repeatedAssignment` rule expression is not continuous and differentiable, see "Using Events to Address Discontinuities in Rule and Reaction Rate Expressions" on page 1-50 before simulating your model.

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 1-37 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*log(x) - 3
```

Note If your `algebraic` rule expression is not continuous and differentiable, see "Using Events to Address Discontinuities in Rule and Reaction Rate Expressions" on page 1-50 before simulating your model.

Considerations When Imposing Constraints

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

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)
```

Note If your `rate` rule expression is not continuous and differentiable, see "Using Events to Address Discontinuities in Rule and Reaction Rate Expressions" on page 1-50 before simulating your model.

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 $c = kt + c_o$, where c_o 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.
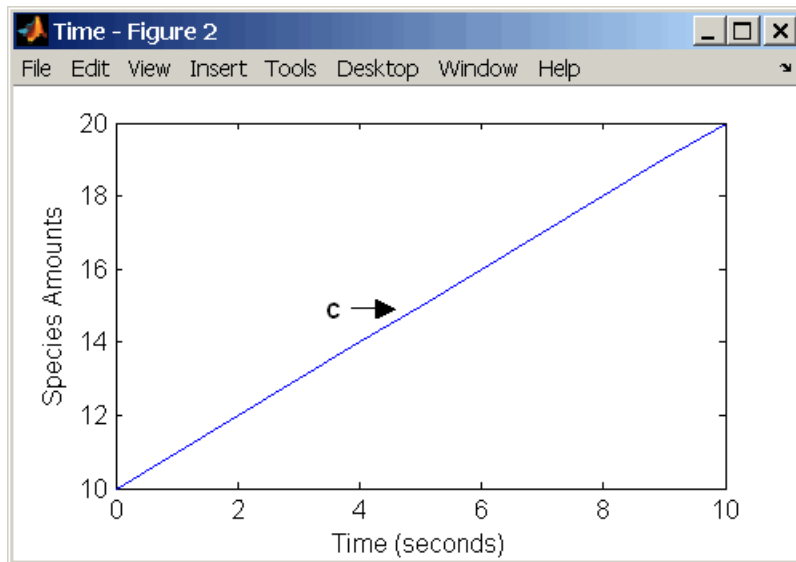
Alternatively, you could model a constant increase in a species using Mass Action reaction null -> C.

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*c is $c = c_0 e^{-kt}$.

Enter the rate rule described above and the simulate.
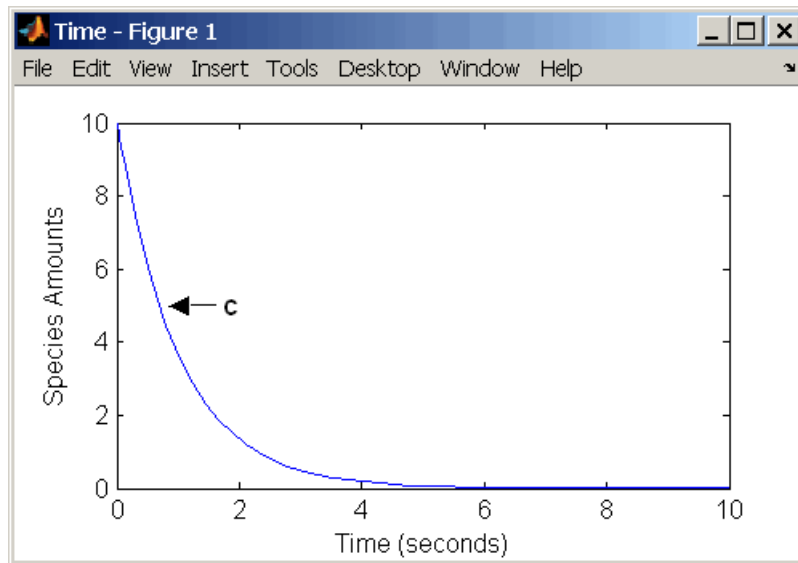
Notice that if the amount of a species c is determined by a rate rule and c is also in a reaction, c must have its BoundaryCondition property set to true. For example, with a reaction a -> c and a rate rule dc/dt = k*c, select the BoundaryCondtion for c so that a differential rate term is not created 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:

$a = a_0 e^{-k1t}$ and $b = a_0(1-e^{-k1t})$

With the rate rule `dc/dt = k2*a` dependent on the reaction, `dc/dt = ` $k_2(a_0 e^{-k1t})$, 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.
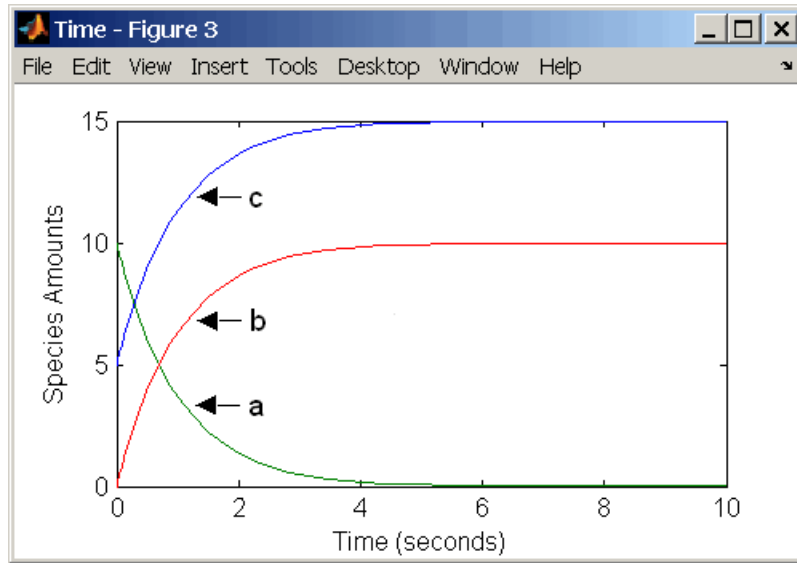
Expressing Differential Rate Equations as Rules

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,

$$\frac{dC}{dt} = vi - vdX\frac{C}{Kc + C} - kdC$$

as a rate rule in SimBiology:

```
C = vi - (vd*X*C)/(Kc + C) - kd*C
```

Changing Model Component Values Using Events

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 2-18 and "Stochastic Solvers" on page
2-13 for more information.

Procedures

- For an example using events in the SimBiology desktop, see "Desktop
  Example — Changing Species Amounts Using an Event" on page 1-52 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.
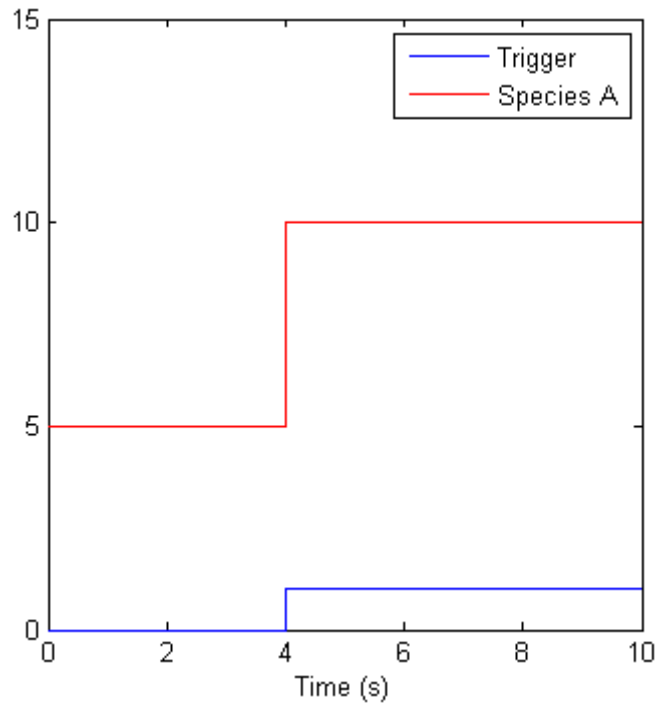
At `time = 4` the trigger becomes true and the event is executed. In the figure above assuming that 0 is false and 1 is true, when the trigger becomes true, the amount of `Species A` is set to 10. In theory, with a perfect solver, the event would be executed exactly at `time = 4.00`. In practice there 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
time >= 4	A = 10

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

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

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

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

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

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)'`	Execute the event when the following condition becomes true:Time is greater than or equal to 5, and speciesA is less than 1000.
	Tip Using a && (instead of &) tells the software to evaluate the first part of the expression for whether the statement is true or false, and skip evaluating the second statement if this statement is false.
`'(time >=5) || (speciesA<1000)'`	Execute the event when the following condition becomes true: Time is greater than or equal to 5, or if speciesA is less than 1000.
`'(s1 >=10.0) || (time>= 250) && (s2<5.0E17)'`	Execute the event when the following condition becomes true: Species, s1 is greater than or equal to 10.0 or, time is greater than or equal to 250 and species s2 is less than 5.0E17.
	Because of operator precedence the expression is treated as if it were `'(s1 >=10.0) || ((time>= 250) && (s2<5.0E17))'`
	Thus, it is always a good idea to use parenthesis to explicitly specify the intended precedence of the statements.

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 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 speciesA equal to that of speciesB.
`'k = k/2'`	When the event is executed halve the value of the rate constant k.

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

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	SpeciesA >= 4	SpeciesC = 10
2	time >= 15	SpeciesB = SpeciesC

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

- If Event 1 and Event 2 do not occur simultaneously, when Event 2 is triggered `SpeciesB` is assigned the value that `SpeciesC` has at the time of the event trigger.

- If Event 1 and Event 2 occur simultaneously, the solver 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
time >= 4	{'SpeciesA = 10, SpeciesB = SpeciesA'}

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

When One Event Triggers Another Event

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

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

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

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

Cyclical Events

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

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

Using Events to Address Discontinuities in Rule and Reaction Rate Expressions

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

- Repeated assignment rule
- Algebraic rule
- Rate rule
- Reaction rate

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

Desktop Example — Changing Species Amounts Using an Event

In this section...
"Overview" on page 1-52
"Prerequisites" on page 1-53
"Adding an Event to the Example Model" on page 1-53
"Simulating the Modified Model" on page 1-56

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 1-42.

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 <-> RL`	kRL,kRLm
2	Heterotrimeric G protein formation	`Gd + Gbg -> G`	kG1
3	G protein activation	`RL + G -> Ga + Gbg + RL`	kGa

No.	Name	Reaction	Rate Parameters
4	Receptor synthesis and degradation	`R <-> null`	kRdo, kRs
5	Receptor-ligand degradation	`RL -> null`	kRD1
6	G protein inactivation	`Ga -> Gd`	kGd

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 Save the project.

   a Select **File > Save Project As**. The Save SimBiology Project dialog box opens.

   b 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, under the **Species being used by Event** section, 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:

5 In the **Project Explorer**, click **Configuration Settings** to open the pane that contains solver settings.

6 Click the **Data Logging** tab.

7 Click **Select All**.

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

Simulating the Modified Model

1 Click **Run**.

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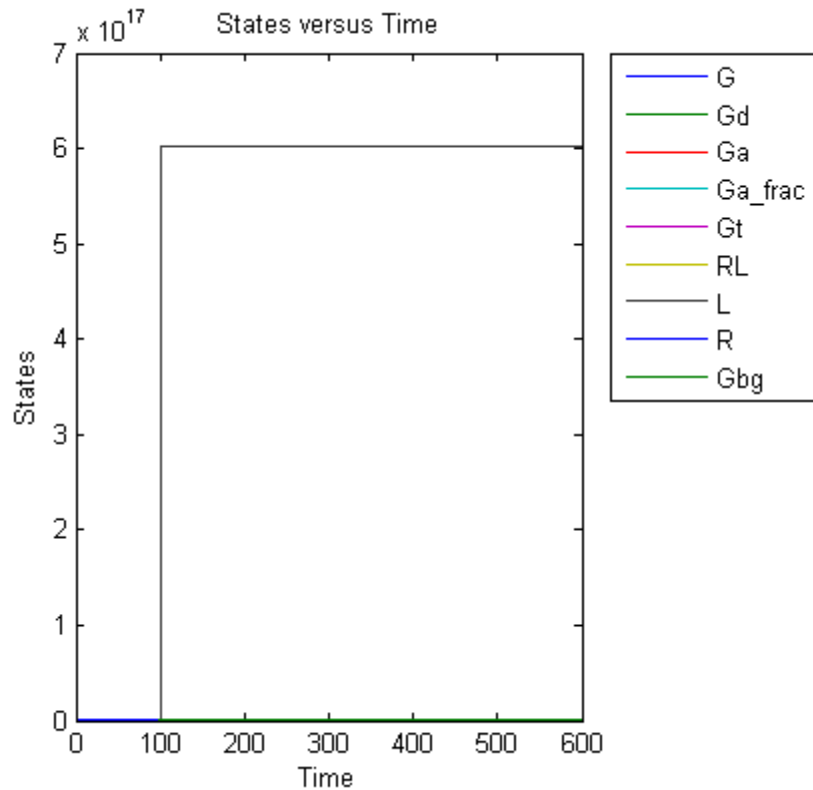
The plot does not show the species of interest due to a wide range in species amounts. Follow the next steps to view the species of interest.

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

3 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**.

4 In the **Project Explorer**, click the saved data, for example, `event_ex`, to open the **Data** pane for the saved data.

5 Click the **Plots** tab.

6 From the **Plot Type** list, select `Time` and click **Add Plot Type**.

7 Select the new plot (second row), and in the **Arguments** section, click ![icon]. The Select Values for y dialog box opens.

8 Clear the check boxes for the species `L` and `Gbg`.

9 Click **OK**.

10 (Optional) Clear the **Create Plot** check box for the first plot.

11 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).
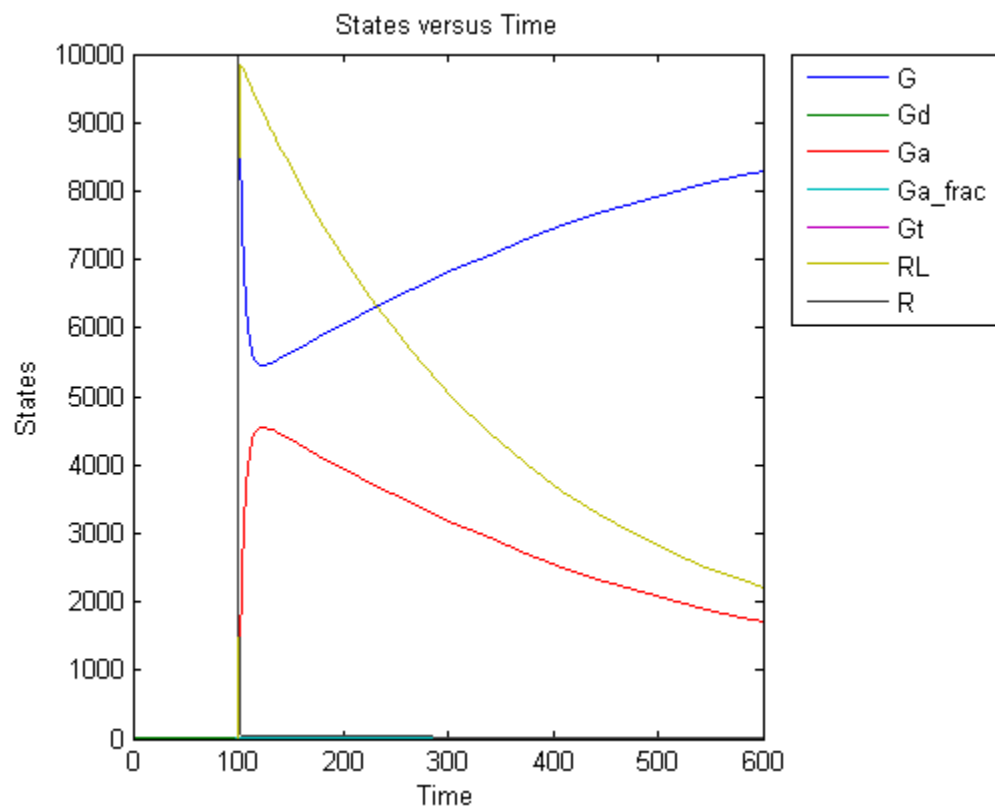
Storing and Applying Alternate Model Values Using Variants

In this section...
"What Are Variants?" on page 1-59
"Using Variants" on page 1-60
"Applying Multiple Variants in a Model" on page 1-61

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 apples 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	addvariant
Appending contents to variants	addcontent
Replacing model values permanently with values from a variant	commit

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**, 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 1-62

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.

If you permanently replace values in a model with values from multiple variants using the **Commit** button in the desktop, the value committed in the model is the last occurrence of the value in the variants.

Desktop Example — Applying Changes to Parameter Value Using a Variant

In this section...
"Overview" on page 1-62
"Prerequisites" on page 1-63
"Applying Alternate Values Using Variants" on page 1-64
"Simulation Results for the Model of the Mutant Strain" on page 1-64

Overview

This example shows you how to apply a variant to a model representing a wild-type strain. The variant represents a parameter value for a G protein cycle in a mutant 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.

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 <-> RL`	`kRL,kRLm`
2	Heterotrimeric G protein formation	`Gd + Gbg -> G`	`kG1`
3	G protein activation	`RL + G -> Ga + Gbg + RL`	`kGa`
4	Receptor synthesis and degradation	`R <-> null`	`kRdo, kRs`

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

The value of the parameter kGd is `0.11` for the wild-type and `0.004` 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.

For a description of variants, see "Storing and Applying Alternate Model Values Using Variants" on page 1-59.

Prerequisites

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 Save the project.

   a Select **File > Save Project As**. The Save SimBiology Project dialog box opens.

   b 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-Yeast_G_Protein_wt**, click **Variants** to open the **Variants** pane.

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, from the **Type** list, select parameter. The **Property** list updates to show the property available for changing.

   b Double-click the **Component Name** cell and type the name of the component.

    Gprotein_Inactivation.kGd

> **Tip** Alternatively, double-click the **Component Name** cell, press the down arrow, select Gprotein_Inactivation.kGd from the list, and press **Enter**.

    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*.

   c 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 1-59 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-Yeast_G_Protein_wt**, under **Model Tasks**, select **Simulation** to open the pane.

2 Click the **Simulation Settings** tab.

3 In the **Variants** table, select the **Use in Task** check box for mut_value_ex.

4 Click ⏵ Run . Your plot should resemble the following figure.
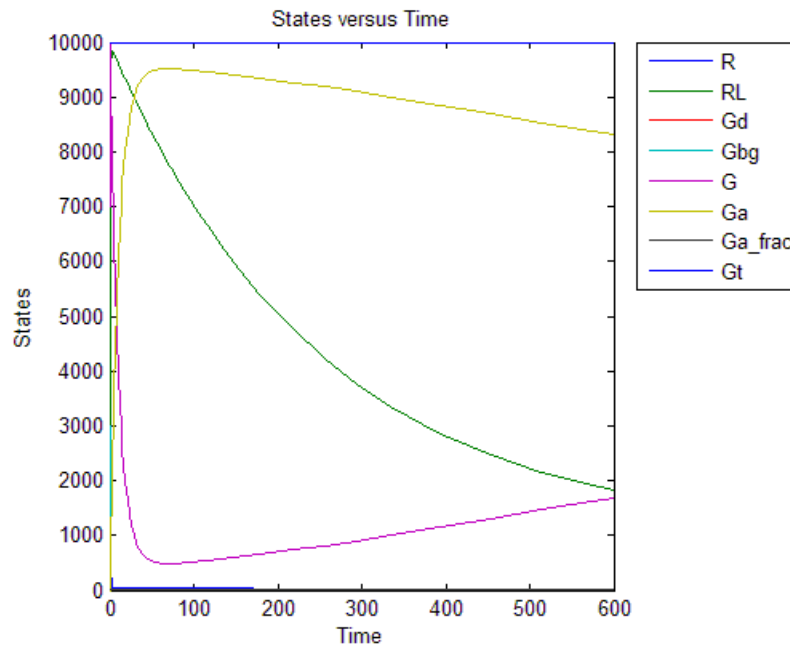
5 In the **Data** pane for the latest simulation, click **Save** to open the Save Data dialog box.
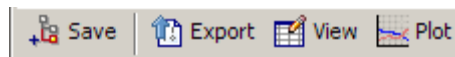
6 Specify a name for your data, and then click **Save**.

```
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 a Model Has No Warnings or Errors

In this section...

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 a Model in the SimBiology Desktop

While you are building your model in the SimBiology desktop, you can click ![Verify] 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.
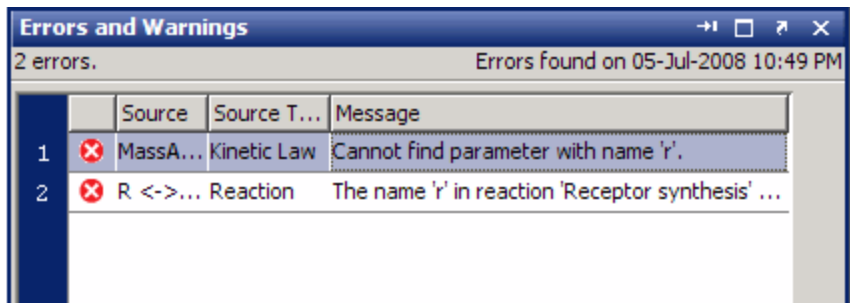
Double-click the error row to move to the location of the error.

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 1-70
"Overview" on page 1-70
"Creating a Function" on page 1-72
"Calling the Function in a Rule Expression" on page 1-73
"See Also" on page 1-78

Prerequisites

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

- MATLAB desktop
- Creating and saving MATLAB programs
- 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 a function. To find out more about functions, see function in *MATLAB Function Reference*. For an example of a function to be used in a SimBiology model, see "Creating a Function" on page 1-72 in this topic.

- Change the current folder to the folder containing your MATLAB 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 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 <-> RL`	`kRL,kRLm`
2	Heterotrimeric G protein formation	`Gd + Gbg -> G`	`kG1`
3	G protein activation	`RL + G -> Ga + Gbg + RL`	`kGa`

No.	Name	Reaction	Rate Parameters
4	Receptor synthesis and degradation	`R <-> null`	kRdo, kRs
5	Receptor-ligand degradation	`RL -> null`	kRD1
6	G protein inactivation	`Ga -> Gd`	kGd

Assumptions

For the purpose of this example, assume that:

- An inhibitor (Inhib) slows the inactivation of the active G protein (reaction 6 above, Ga –> 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 a Function

1 In the MATLAB desktop, select **File > New > Script**, to open a new 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 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 file is not on the path, change the working directory to the file location.

Calling the Function in a Rule Expression

- "Opening and Saving the Example Model" on page 3-62

- "Adding User-Defined Functions to the Example Model" on page 1-74

- "Defining a Rule to Affect Parameter Value" on page 1-75

- "Simulating the Modified Model" on page 1-76

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 Save the project.

   a Select **File > Save Project As**. The Save SimBiology Project dialog box opens.

   b 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 a Function" on page 1-72 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 function file on the path or changed the working directory to the location of the 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 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:
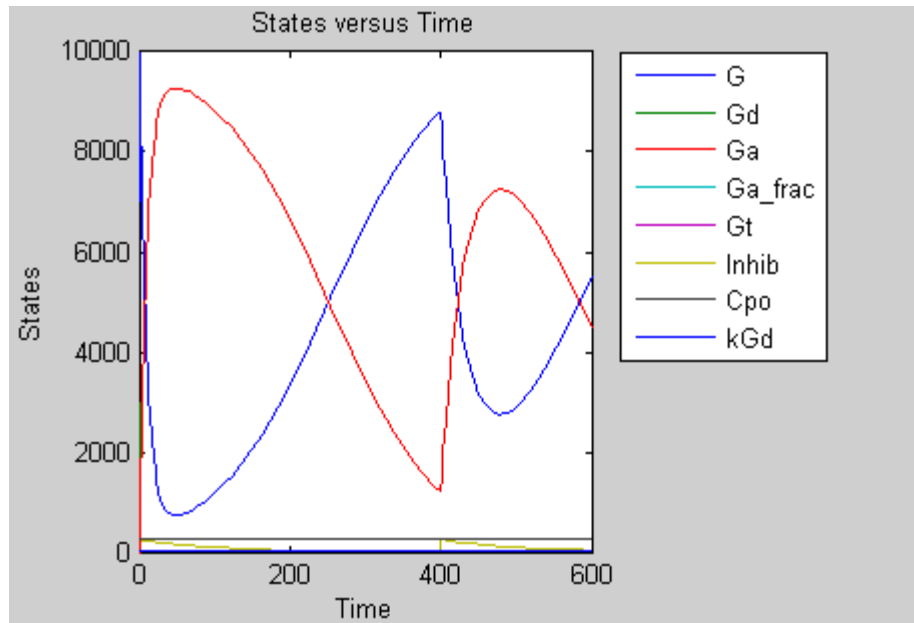
Notice the change in profile of species `Ga` at `time = 400` seconds (simulation time) when the inhibitor amount is changed to reflect the re-addition of inhibitor to the model.
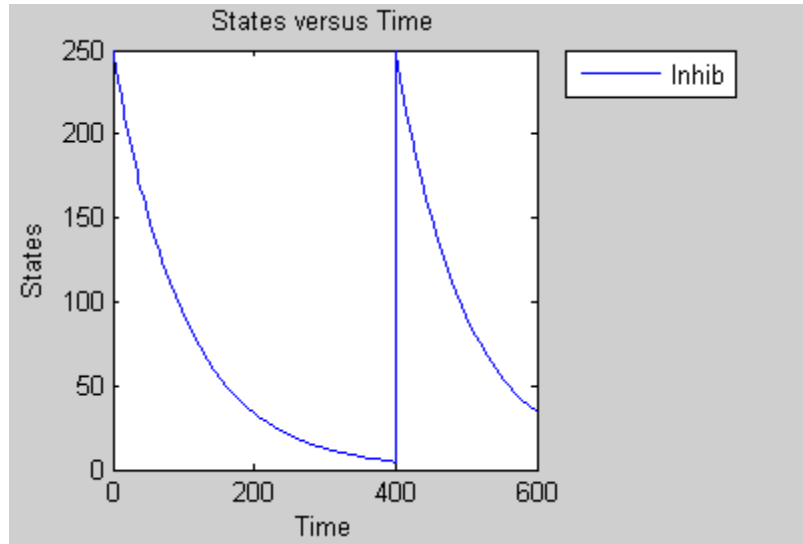
See Also

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

Importing and Exporting Model Component Data

In this section...

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.

2

Simulation

- "Performing Simulations" on page 2-2
- "About Simulation Solvers" on page 2-5
- "Nonstiff Deterministic Solvers" on page 2-9
- "Stiff Deterministic Solvers" on page 2-11
- "Stochastic Solvers" on page 2-13
- "Sundials Solvers" on page 2-18
- "Accelerating Simulations" on page 2-19

Performing Simulations

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:

- In the Simulation pane, on the **Simulation Settings** tab, next to **Configuration Settings**, click **Go to**.

- Alternately, in the **Project Explorer**, select **Configuration Settings**.
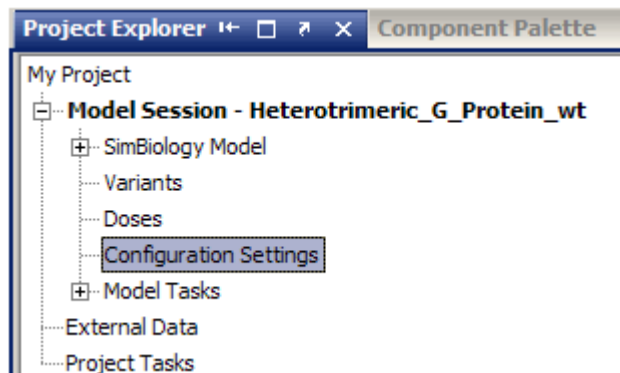
The **Configuration Settings** pane appears, where you can set, change, and save simulation parameters, configure data logging, and compile options.

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 on any Model Task pane 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 on any Model Task pane.
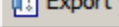
If you are using **Doses**, you can view and apply them to simulation tasks or scan tasks on the **Simulation Settings** tab of the Simulation pane or Scan pane. You can also specify dosing information for parameter fit tasks on the **Estimation Settings** tab of the Parameter Fit pane.

Note If you are using the SimBiology Desktop to perform a simulation using multiple doses, make sure each dose's **TargetName** (Simulation or Scan pane) or **DoseComponentName** (Parameter Fit pane) refers to a different species in the model.

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

Exporting Simulation Data

By default, simulation data is exported to the MATLAB Workspace every time you run a simulation. To disable data exportation, select **File > Preferences > Model Tasks**, then clear the **Export task data when task completes executing** check box. To export when exporting is disabled, or to export using settings other than the default, in the Project Explorer pane, select the Data node under Simulation, then in the Data panel, click ![Export] to display the Export Data dialog box.

Interpreting Simulation Results

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

- Slightly negative due to numerical noise introduced by the simulation process. In this case, you can interpret these values as 0.

- Significantly negative due to the dynamics in your model not being physical, that is, the dynamics in the system are driving a particular species to be negative. In this case, examine your reaction rate expressions for errors.

About Simulation Solvers

In this section...
"How Solvers Work" on page 2-5
"Validating Solver Inputs" on page 2-5
"Stiff Versus Nonstiff Models" on page 2-6
"Selecting a Solver" on page 2-7

How Solvers Work

To simulate a model, the SimBiology software converts a model to a system of differential equations. It then uses the solver functions 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.

Validating Solver Inputs

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

- Repeated assignment rule
- Algebraic rule
- Rate rule

- Reaction rate

Either ensure that the previous expressions are continuous and differentiable or use events to reset the solver at the discontinuity. For more information, see "Using Events to Address Discontinuities in Rule and Reaction Rate Expressions" on page 1-50.

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, sundials is the best function to apply as a "first try" for most problems. 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 2-9.

- Sundials — Best first guess.

- ode45 — 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 2-11.

- Sundials — Try first if you suspect that a problem is stiff, or if ode45 failed or was very inefficient.

- ode15s — 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 2-13.

- 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 2-9
"ode45 (Dormand-Prince)" on page 2-9
"ode23 (Bogacki-Shampine)" on page 2-9
"ode113 (Adams)" on page 2-9
"See Also" on page 2-10

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

"Ordinary Differential Equations" in *MATLAB Mathematics.*

Stiff Deterministic Solvers

In this section...

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

"Ordinary Differential Equations" in *MATLAB Mathematics*.

Stochastic Solvers

In this section...

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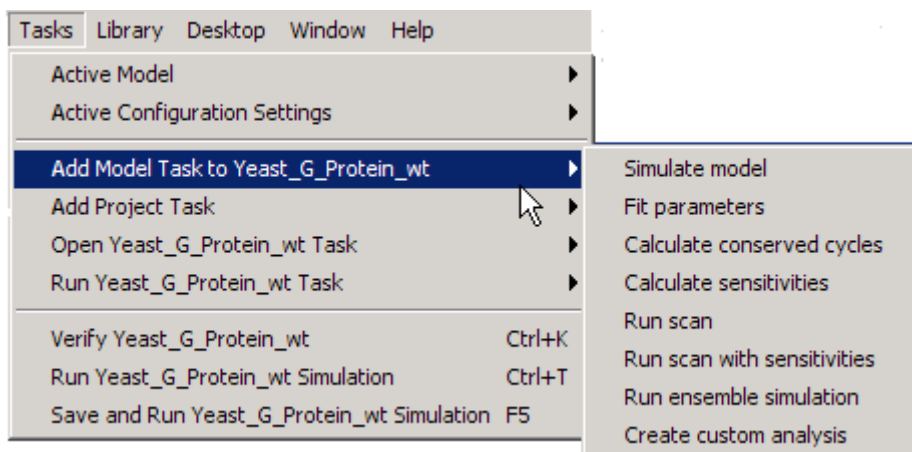
The desktop adds **Ensemble Run** in the **Project Explorer** and opens the **Ensemble Run** pane.

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.

Accelerating Simulations

In this section...
"Prerequisites for Accelerating Simulations" on page 2-19
"When to Accelerate Simulations" on page 2-19
"Accelerating Simulations at the Command Line" on page 2-20
"Accelerating Simulations in the SimBiology Desktop" on page 2-21
"Troubleshooting" on page 2-21

Prerequisites for Accelerating Simulations

SimBiology enables you to prepare your models for accelerated simulations. To use this functionality do the following:

1 Install a C compiler (if one is not already installed on your system). For an up-to-date list of supported compilers, see the Supported and Compatible Compilers web page. For a list of free compilers see the following Technical Solution.

2 Run `mex -setup` once after compiler installation, as described in the documentation for `mex` in the MATLAB Function Reference.

On 32–bit Windows platform, the Lcc complier is automatically installed. However, for optimal performance of the acceleration functionality install a different complier. You can also use `mex -setup` to choose and configure a different C compiler (if one is already installed), as described in "Building MEX-Files" in the MATLAB External Interfaces documentation.

Note For parameter estimations, acceleration functionality is automatically enabled, if available.

When to Accelerate Simulations

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

• Running many simulations with different initial conditions.

• Running very long simulations (for example, simulations that take a minute or longer to run).

Tip

• For models with fewer than 50 species, use the Sundials solver.

• As the size of the model increases beyond 50 species, the ode15s solver may enable better performance.

Accelerating Simulations at the Command Line

Accelerating simulations is a two step process.

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

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

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

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

• Changes to the variants

• Changes to values for the InitialAmount of species

• Changes to the Capacity of compartments

• Changes to the Value of parameters

---

2 Use the sbiosimulate command with the same model, configset, and dose input arguments. For example,

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

Accelerating Simulations in the SimBiology Desktop

You can control whether to accelerate simulations in the **Simulation Settings** tab of any model task pane that performs simulations (for example, **Simulation**, **Sensitivity Analysis**, **Scan** or **Scan with Sensitivities** pane).

1 In the **Project Explorer**, expand **Model Tasks** and select a model task. For example, **Simulation**, **Sensitivity Analysis**, **Scan** or **Scan with Sensitivities** pane.

The selected task pane opens in the **Work Area**.

2 In the task pane, select the **Simulation Settings** tab if it is not already selected.

3 Select the **Prepare the model for accelerated simulation before running** check box.

When you click **Run** SimBiology first prepares the model for acceleration and then runs the task selected.

Note In the **Parameter Fit** task the **Prepare the model for accelerated simulation before running** check box is always selected because the task always uses acceleration.

Troubleshooting

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

If you specify user-defined functions in SimBiology expressions, you might see the following warning if your code is not compliant with the Embedded MATLAB subset.

```
The Expression and any user-defined functions
could not be accelerated. Please check that these expressions
and any user-defined functions are
compliant with the Embedded MATLAB subset.
```

Where *Expression* is any of the following:

- reaction rate/rule expressions

- `repeatedAssignment` rule expressions

- event trigger expressions

- event function expressions

For more information see Making Code Compliant with the Embedded MATLAB Subset.

3

Analysis

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

Scanning Analysis

In this section...

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
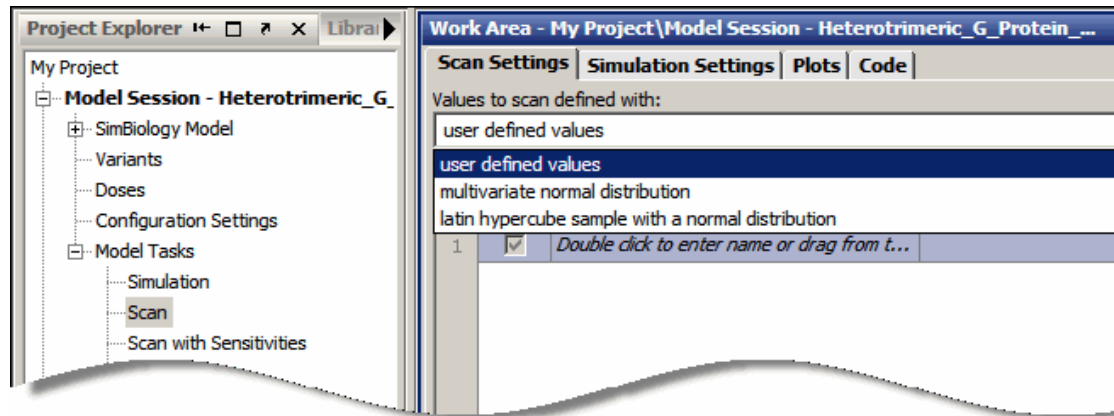
Creating a Scan Task

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 In the **Scan Settings** tab, select user defined values from the **Values to scan defined with** list. By default, the table contains one row specifying scan options.

2 Double-click a cell in the **Component Name** column to enter the name of the component. Examples include transcription.k (parameter), nucleus (compartment), and membrane.receptor (species).

Tip Alternatively, double-click the **Component Name** cell and press the down arrow to display a list of available components. Select a component from the list and press **Enter**.

> **Tip** If the name entered in the table is invalid, a red indicator appears to
> the right of the row.

3 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.

4 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.

5 Click **OK** to apply the settings and close the dialog box.

6 (Optional) In the **Scan Settings** tab, select the **Scan along the diagonal** check box at the bottom of the tab.

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 while scanning over multiple parameters.

7 (Optional) To scan with multiple parameters, select the last (undefined) row and set the options as described in the previous steps.

8 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.

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 Double-click a cell in the **Component Name** column to type the name of the component. Examples include transcription.k (parameter), nucleus (compartment), and membrane.receptor (species).

> **Tip** Alternatively, double-click the **Component Name** cell and press the down arrow to display a list of available components. Select a component from the list and press **Enter**.

> **Tip** If the name entered in the table is invalid, a red indicator appears to the right of the row.

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

3 (Optional) To scan with multiple parameters, select the last (undefined) row and set the options as described in the previous steps.

4 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:

  a 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.

    b Click **OK** to enter the covariance matrix.

5 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

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 <-> RL`	`kRL,kRLm`
2	Heterotrimeric G protein formation	`Gd + Gbg -> G`	`kG1`
3	G protein activation	`RL + G -> Ga + Gbg + RL`	`kGa`
4	Receptor synthesis and degradation	`R <-> null`	`kRdo, kRs`
5	Receptor-ligand degradation	`RL -> null`	`kRD1`
6	G protein inactivation	`Ga -> Gd`	`kGd`

About This Example

The rate of G protein inactivation is much lower in the mutant strain (`kGd = 0.004`) relative to the wild-type strain (`kGd = 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.
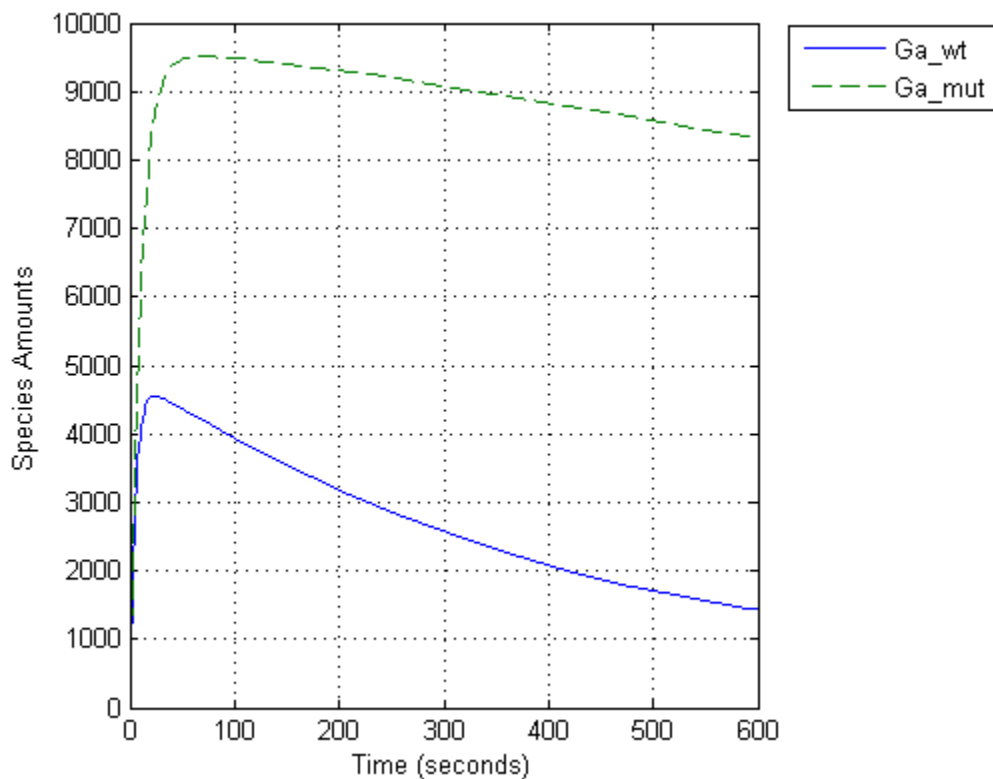
For a more detailed look at how the variation of kGd affects levels of Ga, perform a parameter scan of several simulations in which the value of kGd varies over a range of values. This example illustrates a scan over 10 values of the parameter kGd.

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 Save the project.

a Select **File > Save Project As**. The Save SimBiology Project dialog box opens.

b 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 **Scan** pane, the table contains an undefined row.

3 Double-click the **Component Name** cell and enter the name of the parameter to scan.

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

> **Tip** Alternatively, double-click the **Component Name** cell and press the down arrow to display a list of available parameters. Select a parameter from the list and press **Enter**.

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.

Leave these default options selected:

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

5 In the **Min** box enter `0.001` and in the **Max** box enter `0.15`.

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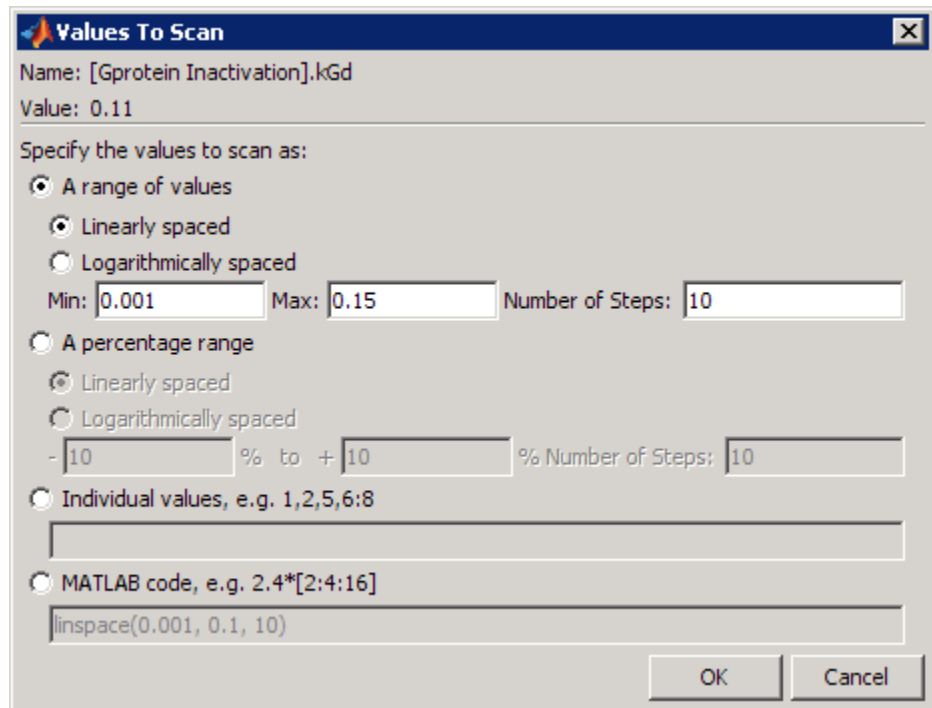
7 Click **OK**.
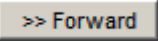
8 Select **File > Save Project**.

Results of Scanning with One Parameter

1 Click ![Run]. The Figures window opens with two plots.
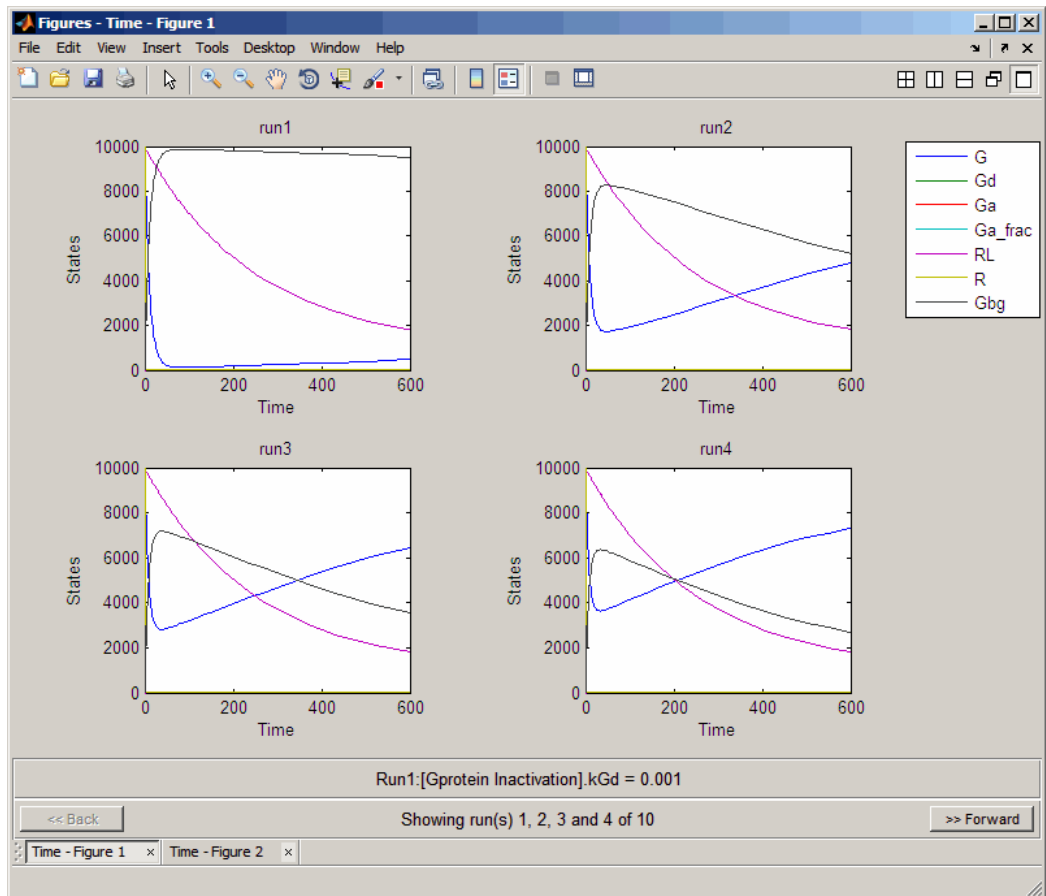
Tip

- Click the **Time - Figure 1** tab at the bottom of the Figures window to display each run plotted in a separate subplot.

- Click the **Time - Figure 2** tab at the bottom of the Figures window to display a consolidated plot showing the results of all the runs.

- View the range of plots by clicking   >> Forward   .

- Hover the mouse cursor over a subplot to display the value(s) of scanned species, parameters, or compartments associated with the subplot.

2 Close the Figures 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**, select the saved data node, for example, scan_ex1, to open the **Data** pane for the saved data.

6 Click the **Plots** tab.

7 In the table, select the first row containing the Time plot. This row specifies a subplot.

Notice that in the **Arguments** section, the **plotStyle** list option displays 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 unnamed.Ga.

	State Name	Type
☐	unnamed.G	species
☐	unnamed.Gd	species
☑	unnamed.Ga	species
☐	unnamed.Ga_frac	species
☐	unnamed.Gt	species
☐	unnamed.RL	species
☐	unnamed.L	species
☐	unnamed.R	species
☐	unnamed.Gbg	species

Select Values for y — Select the states for the y argument: | Select All | Clear All | OK | Cancel
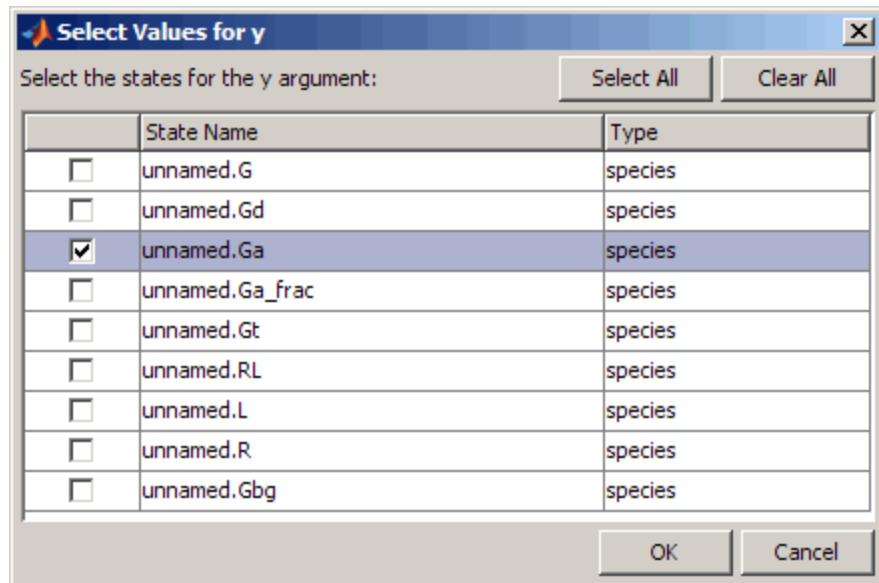
10 Click **OK**.

11 In the 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.
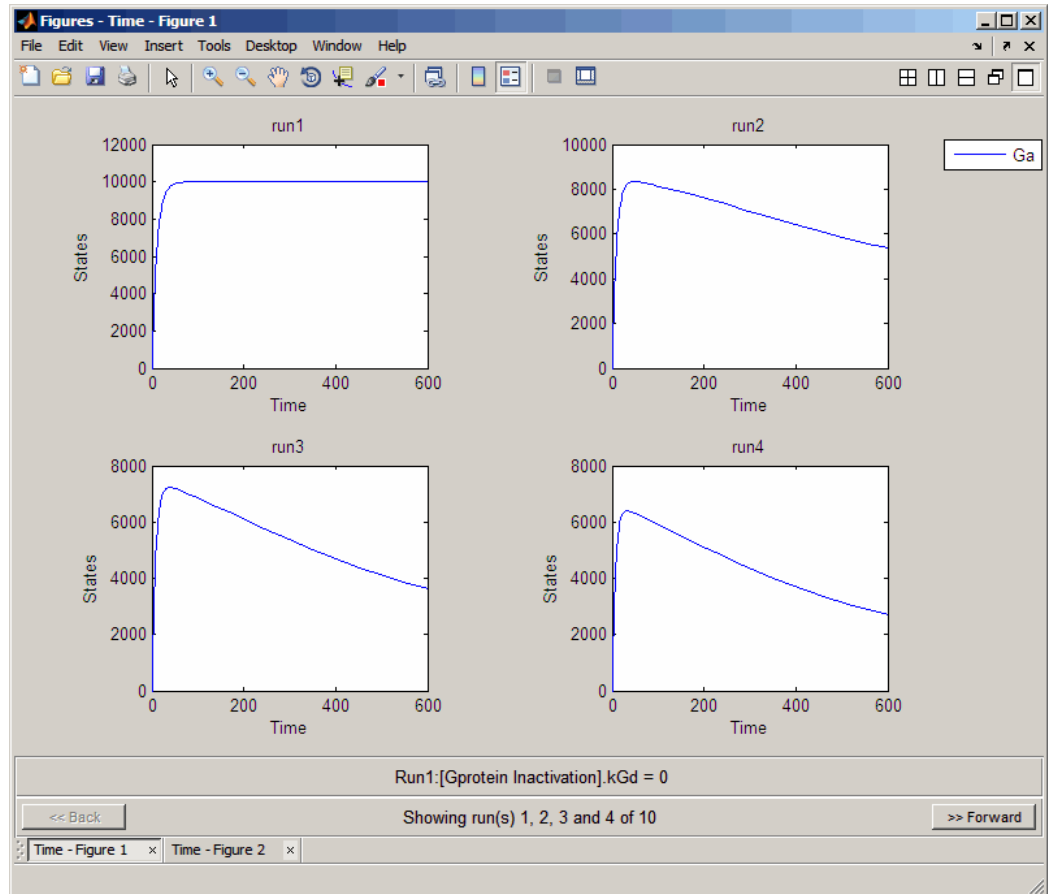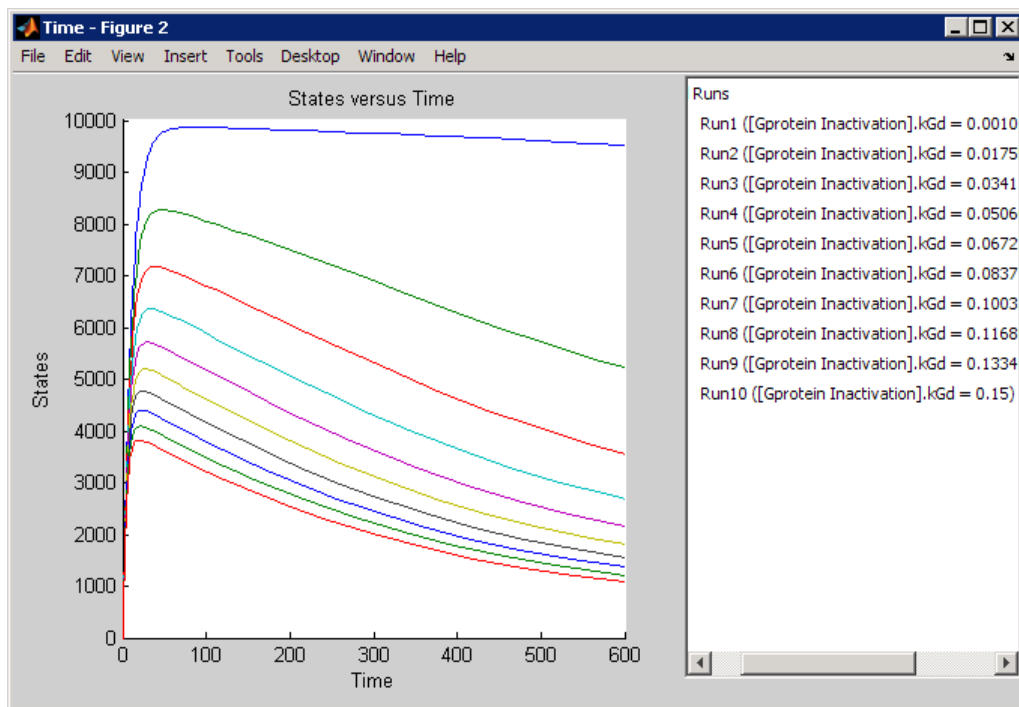
Scanning with Multiple Parameters

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 (Ga) is also sensitive to the parameter kRL. This result follows from the model because parameter kRL 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 which each other on the levels of active G protein. Perform a scan that iterates through each value of parameter kGd with each value of parameter kRL.

To scan using multiple parameters:

1 In the **Project Explorer**, expand **Model Tasks** and select **Scan**.

2 In the **Scan Settings** tab, add more components to scan.

a Double-click the **Component Name** cell and type the name of the component.

```
[Receptor ligand interaction].kRL
```

Tip Alternately, double-click the **Component Name** cell, press the down arrow, select `[Receptor ligand interaction].kRL` from the list, and press **Enter**.

| Use In ...	Component Name	Values To Scan
☑	[Gprotein Inactivation].kGd	linspace(0.001, 0.15, 10)
☑	[Receptor ligand interaction].kRL	
☑	*Double click to enter name or drag from the Component Palette*	

b 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.

c Select **Individual values** 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.

d Click **OK**.

3 Select **File > Save Project**.

Notice that you specified five values for kRL in step 2. 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 the **Plots** tab.

2 In the **Project Explorer**, click the saved data for the previous scan, scan_ex.

3 Press and hold the **Shift** key to select both rows in the table. Copy the two rows by pressing **Ctrl+C** or by right-clicking and selecting **Copy**.

4 In the **Project Explorer**, click the **Scan** node to open the **Scan** pane. Paste the rows into the **Plots** tab by pressing **Ctrl+V** or by right-clicking and selecting **Paste**.

5 Clear the **Create Plot** check boxes for the default plots in the first two rows that specify y = <all> in the **Arguments** column. Select the **Create Plot** check boxes for the two plots you pasted in the table

	Create Plot	Plot Behavior	Plot Type	Arguments
1	☐	New figure ▾	Time	tobj = Simulation Results; y = <all>; plotStyle = subplot
2	☐	New figure ▾	Time	tobj = Simulation Results; y = <all>; plotStyle = one axes
3	☑	New figure ▾	Time	tobj = Simulation Results; y = unnamed.Ga; plotStyle = subplot
4	☑	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.
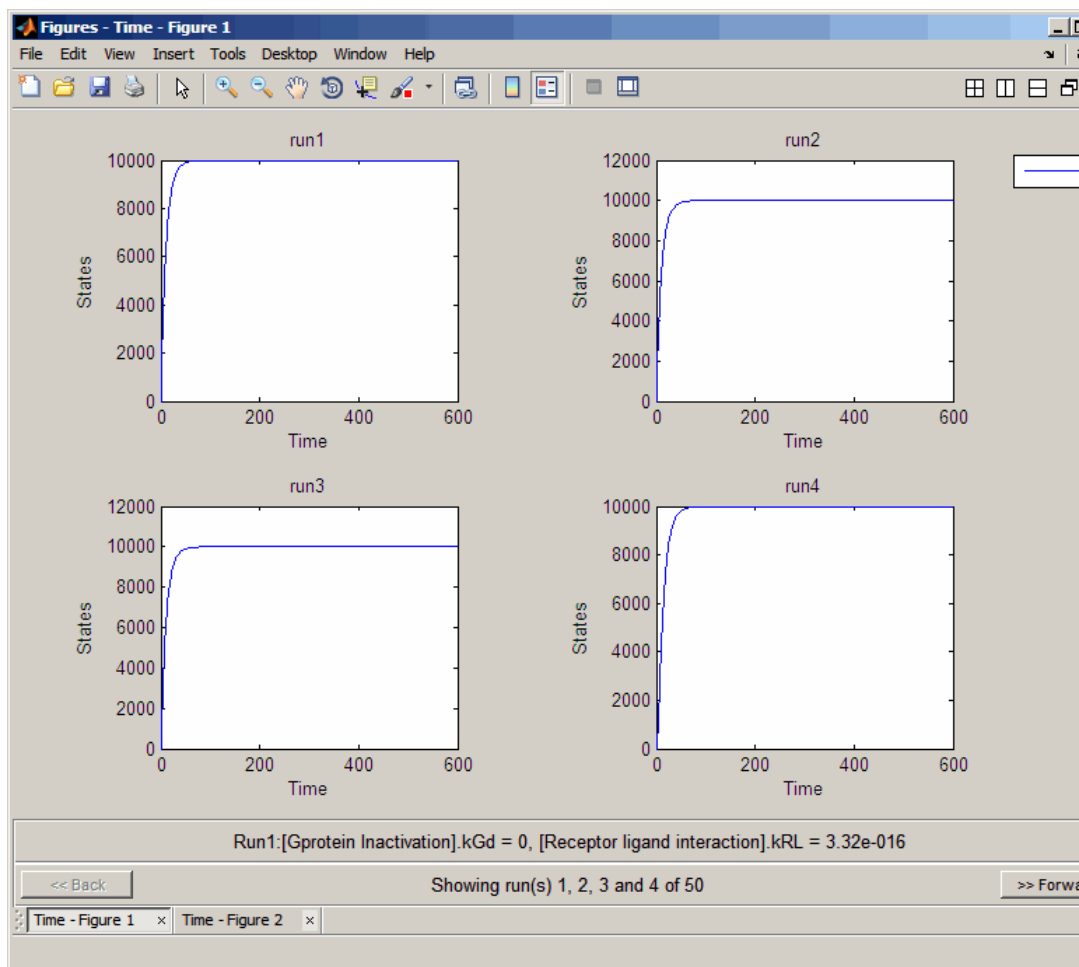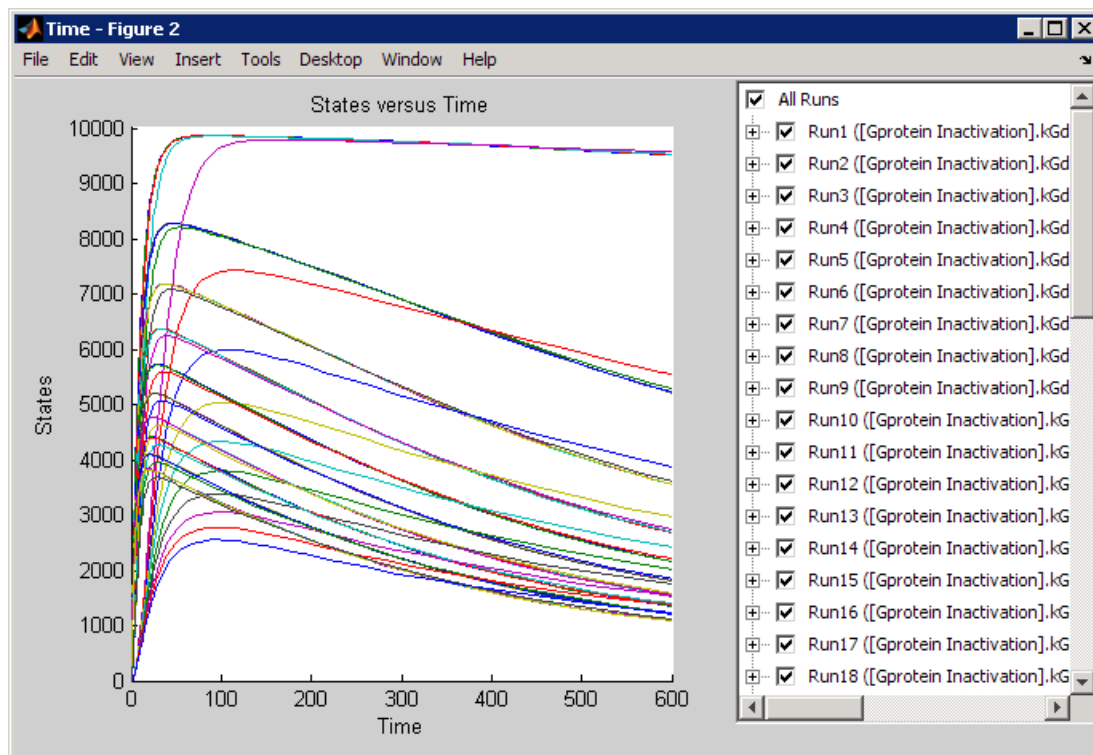
Tip In the Time plot (second figure above), select or clear the **Run#** check boxes to view each run individually or in combination with other runs.

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

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 the following active components do not support sensitivity analysis:

- Events

- Doses

- Rate rules

- Algebraic rules

- Repeated assignment rules

For more information on the calculations performed, see "References" 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 of the SolverOptions property of your configset object:

- 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 3-37

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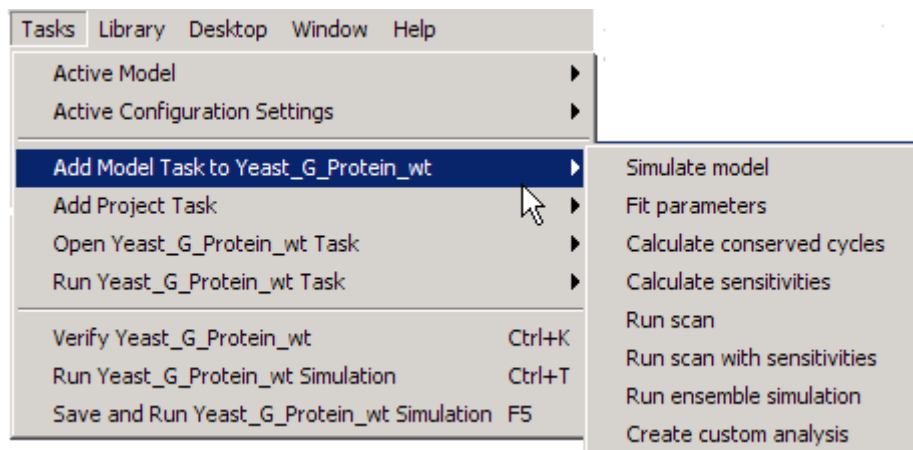
The desktop adds **Sensitivity Analysis** in the **Project Explorer** and opens the **Sensitivity Analysis** pane.

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

References

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

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

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

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 3-35
"References" on page 3-36

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 3-37.

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 <-> RL`	`kRL,kRLm`
2	Heterotrimeric G protein formation	`Gd + Gbg -> G`	`kG1`
3	G protein activation	`RL + G -> Ga + Gbg + RL`	`kGa`
4	Receptor synthesis and degradation	`R <-> null`	`kRdo, kRs`

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 (`kGd = 0.004`) relative to the wild-type strain (`kGd = 0.11`), which explains the higher levels of activated G protein (`Ga`) over time as shown in the following figure.
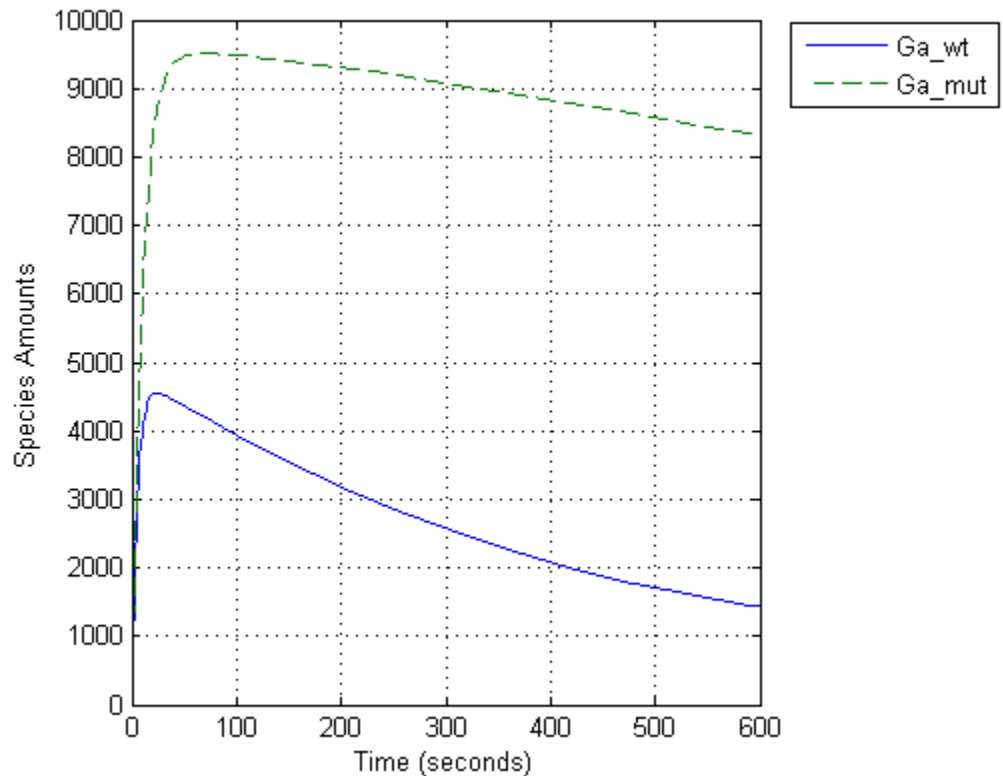
Thus, the active G protein, Ga, is sensitive to the value of the parameter, kGd. Other species or parameters in the model can also affect levels of active G protein. To study the sensitivity of a species to other species or parameters in a model, you can perform sensitivity analysis. Sensitivity analysis 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:

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

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 Save the project.

    a Select **File > Save Project As**. The Save SimBiology Project dialog box opens.

    b 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 Add species to the **Sensitivity Settings** tab:

   a Select the **Component Palette** (left pane by default).

   b Press and hold the **Shift** key to select all components of the Type species.

   c Click-drag the selected species components to the table in the **Sensitivity Settings** tab.

2 In the table, mark all the *species* components as outputs by selecting their **Output** check boxes.

3 Add parameters to the **Sensitivity Settings** tab:

   a Select the **Component Palette** (left pane by default).

   b Press and hold the **Shift** key to select all components of the Type parameter.

   c Click-drag the selected parameter components to the table in the **Sensitivity Settings** tab.

4 In the table, mark all the *parameter* components as inputs by selecting their **Input** check boxes.

5 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.

	Output	Input	Component Name
Sensitivity Settings	**Simulation Settings**	**Plots**	**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	Component Name
1	☑	☐	unnamed.L
2	☑	☐	unnamed.R
3	☑	☐	unnamed.G
4	☑	☐	unnamed.Gd
5	☑	☐	unnamed.freeGbg
6	☑	☐	unnamed.Ga
7	☑	☐	unnamed.RL
8	☐	☑	[Receptor-ligand].kRLm
9	☐	☑	[Receptor-ligand].kRL
10	☐	☑	[Receptor-synthesis-degradation].kRdo
11	☐	☑	[Receptor-synthesis-degradation].kRs
12	☐	☑	[Receptor-ligand_degradation].kRD1
13	☐	☑	[RL-Gprotein_association].kG1
14	☐	☑	Gprotein_activation.kGa
15	☐	☑	Gprotein_Inactivation.kGd
16	☐	☐	*Double click to enter name or drag from the Component Palette*

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

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

Getting Results for Sensitivity Analysis

Click ⏵ Run . The **Sensitivity Matrix Subplot**, which is the default plot for
sensitivity analysis, opens.
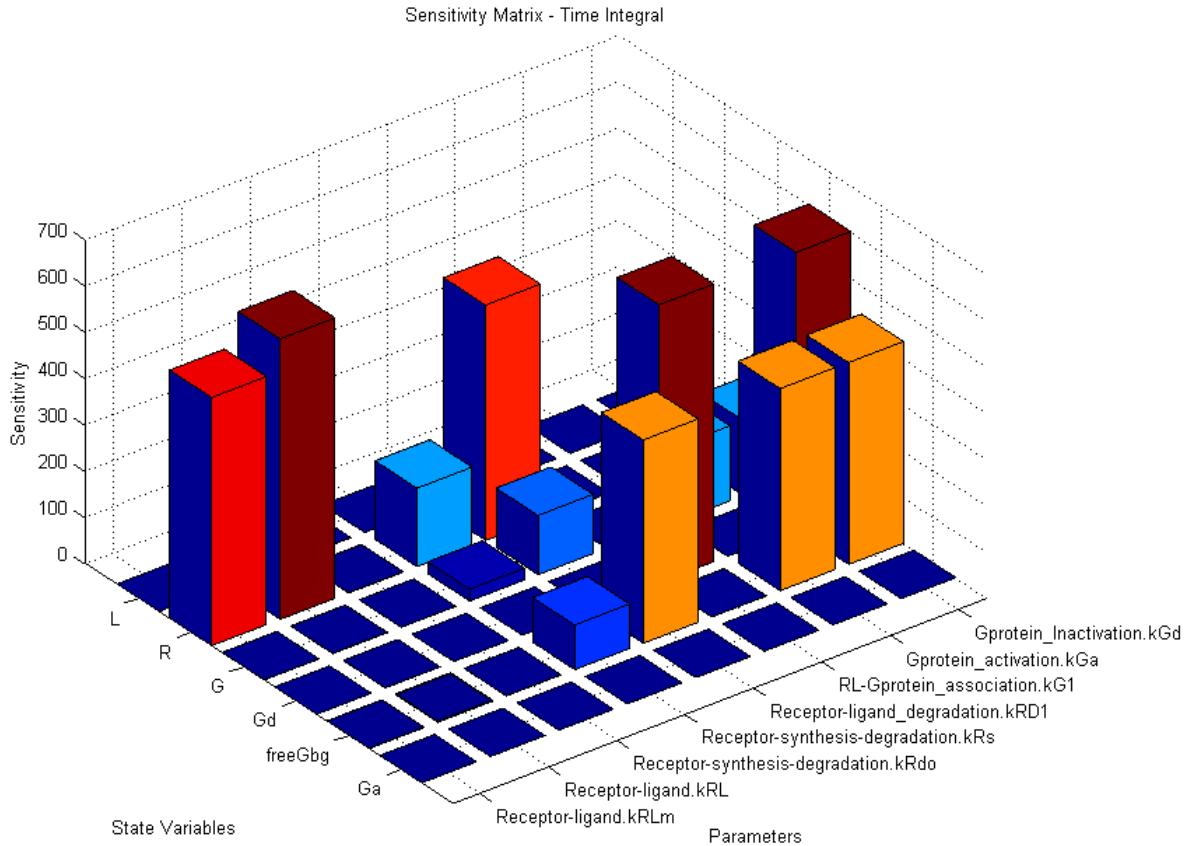
The plot results show for example that as expected, receptor R is sensitive to
values of parameters involved in the receptor-ligand complex formation and
receptor degradation.

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 3-37
"Loading and Configuring the Model for Sensitivity Analysis" on page 3-38
"Performing Sensitivity Analysis" on page 3-39
"Extracting and Plotting Sensitivity Data" on page 3-39
"See Also" on page 3-42

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 <-> RL`	`kRL, kRLm`
2	Heterotrimeric G protein formation	`Gd + Gbg -> G`	`kG1`
3	G protein activation	`RL + G -> Ga + Gbg + RL`	`kGa`
4	Receptor synthesis and degradation	`R <-> null`	`kRdo, kRs`
5	Receptor-ligand degradation	`RL -> null`	`kRD1`
6	G protein inactivation	`Ga -> Gd`	`kGd`

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:

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

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.

See also the following demo: Parameter Scanning, Parameter Estimation, and Sensitivity Analysis in the Yeast Heterotrimeric G Protein Cycle

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`. You can display the model species using

m1.Compartments(1).Species, which shows that species Ga is listed as the 6th species in the model.

```
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 the getsensmatrix method. 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');
```

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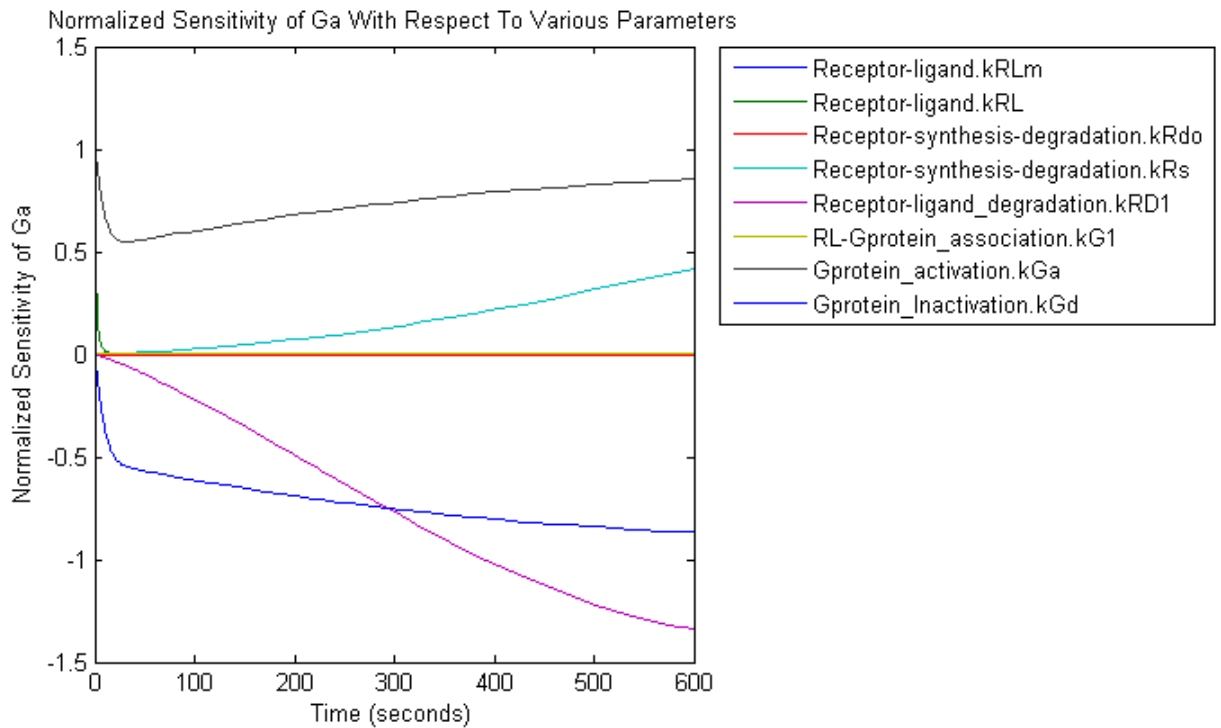
From the previous plot you can see that `Ga` is most sensitive to parameters `kGd`, `kRs`, `kRD1`, and `kGa`. This suggests that the amounts of active G protein in the cell depends on the rate of:

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

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 3-38	`sbioselect`

Parameter Estimation

In this section...
"About Parameter Estimation" on page 3-43
"SimBiology Parameter Estimation" on page 3-43

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 Global Optimization Toolbox to enable estimation.

Optimization Toolbox, and Global Optimization 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 3-44

Command-Line Example — Parameter Estimation

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:                 sundials
      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. (Your results might be slightly different.)

```
rsquare_orig = 1-sse/sst


rsquare_orig =

    0.8968
```

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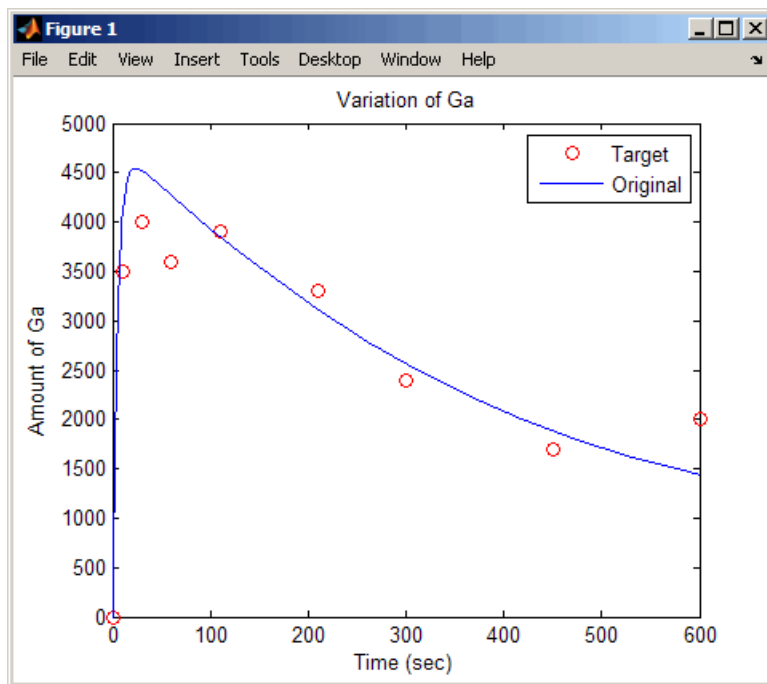
Leave this figure window open so that you can use it to plot and compare results of using the estimated parameters later in this example.

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','MaxIter',15);
```

3 Use the current values of parameters in the model as the starting values for optimization.

```
[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          1194.18
    1            2          1091.14          initial simplex
    2            4          1055.32          reflect
    3            6          1055.32          contract outside
    4            8          1053.53          contract inside
    5           11          1053.53          shrink
    6           13          1052.5           contract inside
    7           16          1052.5           shrink
    8           19          1052.5           shrink
    9           22          1052.5           shrink
   10           25          1052.5           shrink
   11           27          1052.5           contract outside
   12           30          1052.5           shrink
   13           33          1052.5           shrink
   14           36          1052.5           shrink
   15           39          1052.5           shrink

Exiting: Maximum number of iterations has been exceeded
         - increase MaxIter option.
         Current function value: 1052.502212
```

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

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


rsquare1 =
```

```
0.9999
```

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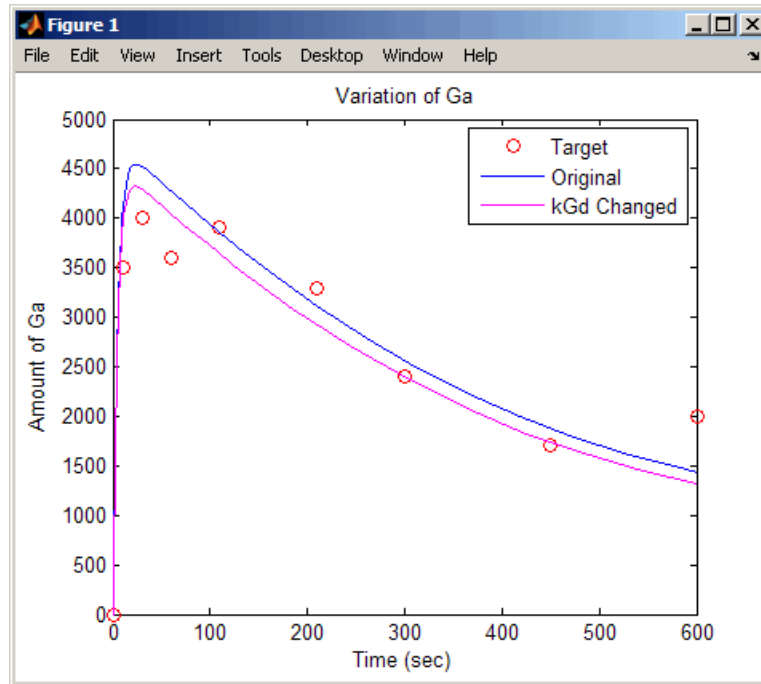
The figure shows the best fit achieved by changing the parameter kGd.

Estimating Other Parameters in the G Protein Model

The example illustrating sensitivity analysis ("Command-Line Example — Calculating Sensitivities" on page 3-37) showed that Ga is sensitive to parameters kGd, kRs, kRD1, and kGa. 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 3-39.

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','MaxIter',15);
```

4 Estimate the parameters. Use the current values of parameters in the model as the starting values for optimization. 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, {}, {'fminsearch',opt2});
```

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

```
     Iteration   Func-count      min f(x)         Procedure
         0            1           1194.18
         1            5           1091.14         initial simplex
         2            7           1066.4          reflect
         3            9           933.698         expand
         4           10           933.698         reflect
         5           11           933.698         reflect
         6           13           907.395         reflect
```

```
 7              15           758.069          expand
 8              16           758.069          reflect
 9              18           739.428          reflect
10              19           739.428          reflect
11              21           739.428          contract outside
12              23           739.428          contract inside
13              25           739.428          contract inside
14              27           739.428          contract inside
15              28           739.428          reflect
```

```
Exiting: Maximum number of iterations has been exceeded
         - increase MaxIter option.
         Current function value: 739.427605
```

5 Compare original parameter values and the estimated parameter values obtained with `'fminsearch'`.

```
% 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

k_new2 =

    4.5490
    0.0031
    0.0000
    0.1238
```

6 Calculate the R-Square value with the new estimates obtained with `'fminsearch'`.

3-53

```
sse = result2.fval;
rsquare2 = 1-sse/sst


rsquare2 =

    0.9999
```

Moiety Conservation

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 3-55 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 `L0` is defined so that N_D = `L0`$*N_R$. In other words, the link matrix gives the dependent rows N_D of the stoichiometry matrix, in terms of the independent rows N_R. Because each row in the stoichiometry matrix corresponds to a species in the model, each row of the link matrix encodes how one dependent species is determined by the `k` independent species.

Command-Line Example — Determining Conserved Moieties

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. The default call to sbioconsmoiety, in which no algorithm is specified, uses an algorithm based on row reduction.

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

g =

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


sp =

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

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

5 Call sbioconsmoiety again, this time specifying 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.

```
LO

LO =

   (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 3-62
"Open the Example Project" on page 3-62
"Setting Up a Custom Task" on page 3-63
"Parameter Estimation Using Custom Task" on page 3-63

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 Save the project.

  a Select **File > Save Project As**. The Save SimBiology Project dialog box opens.

  b 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 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.
%
kGdO = 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(lgndO, lgnd_orig, lgnd_kGd);

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

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

2 Click ▶ Run to run the task. The plot should look similar to this:
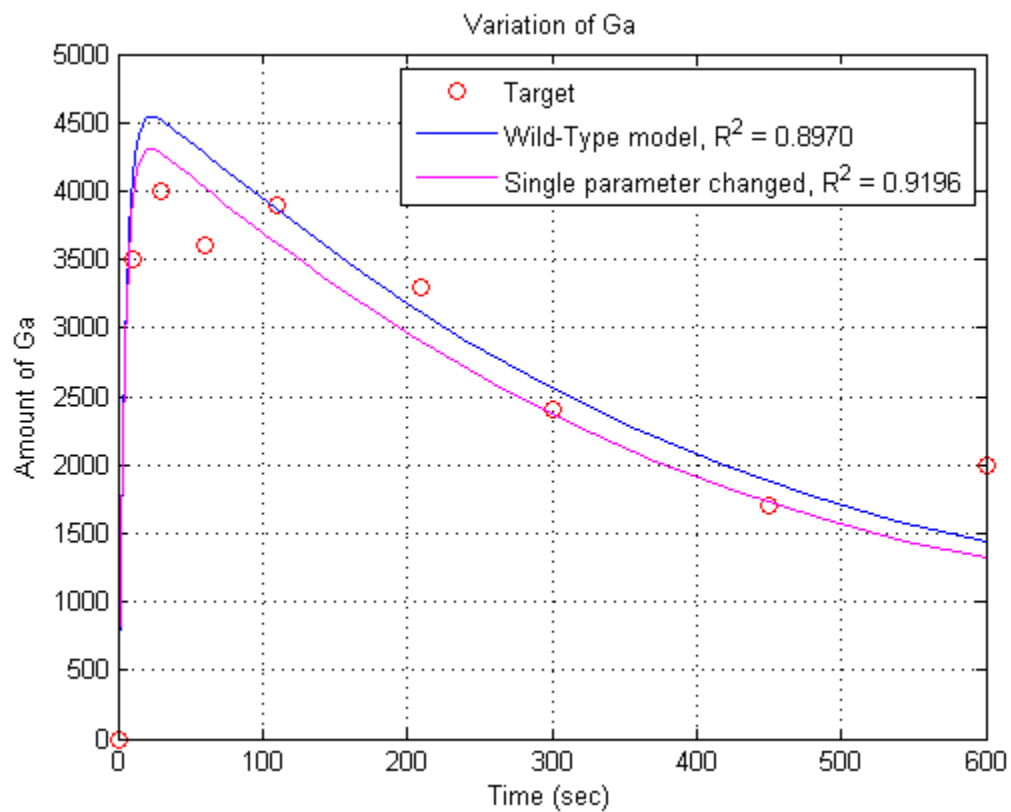
3 If you choose to call the `custom` function from the command line rather than running it as a task in the desktop, pass in a model object.

Visualizing Results Using Plot Types

| In this section... |
| --- |
| "About Plot Types" on page 3-67 |
| "How Plot Types Work" on page 3-69 |
| "Summary of Built-In Plot Types" on page 3-73 |
| "When to Use User-Defined Plot-Types" on page 3-74 |

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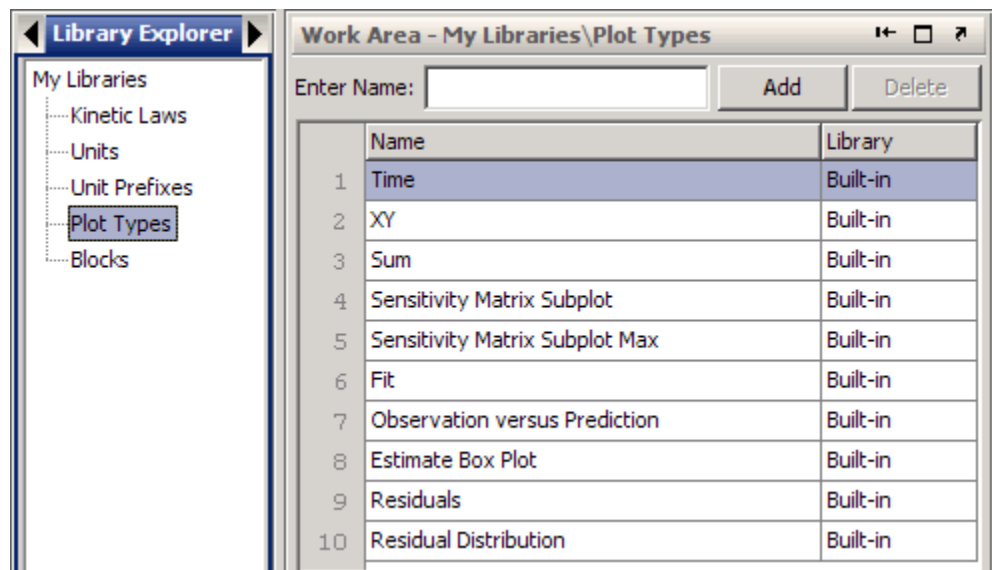
Visualize analysis task results with any plot type in the library. The desktop displays various default plots in the context of each task. For example:

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 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 files and feeds the relevant data output to the plot code. The default plots appear after task completion. You can see this configuration in the **Code** tab on any 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 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 function line states that `Time` takes the input arguments `tobj`, `y`, and `plotStyle`.

- The remaining 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 3-75 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.
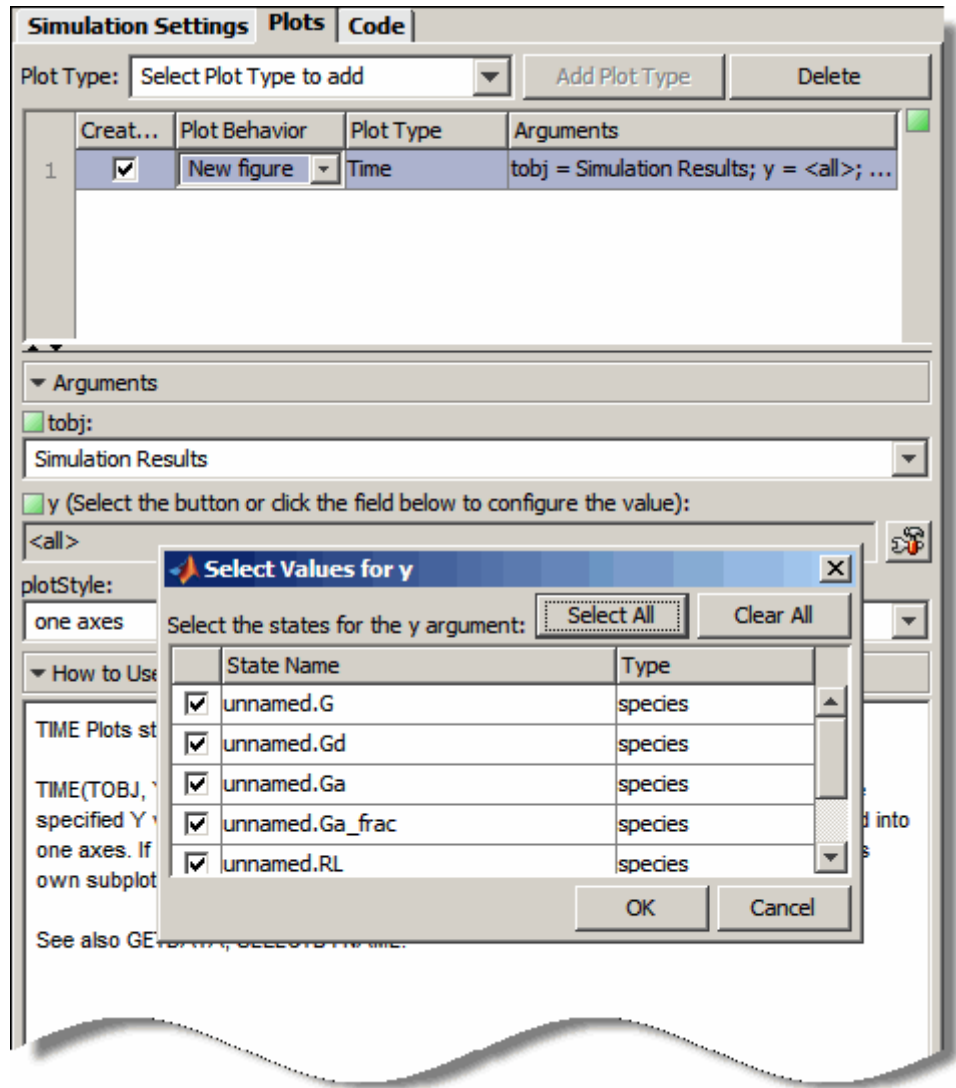
Click 🔧, to open the Select Values for y dialog box.

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), the code for the analysis task stores the results in a *taskResults* 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	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>'. 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.
XY	Specify two states or column names to plot against each other. 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.
Sum	Plot the sum of the simulation results of one or more states against time. 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.
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 3-75.

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 (y2) and prey (y1) in the Lotka-Volterra model.

unnamed.y1 versus unnamed.y2 over time
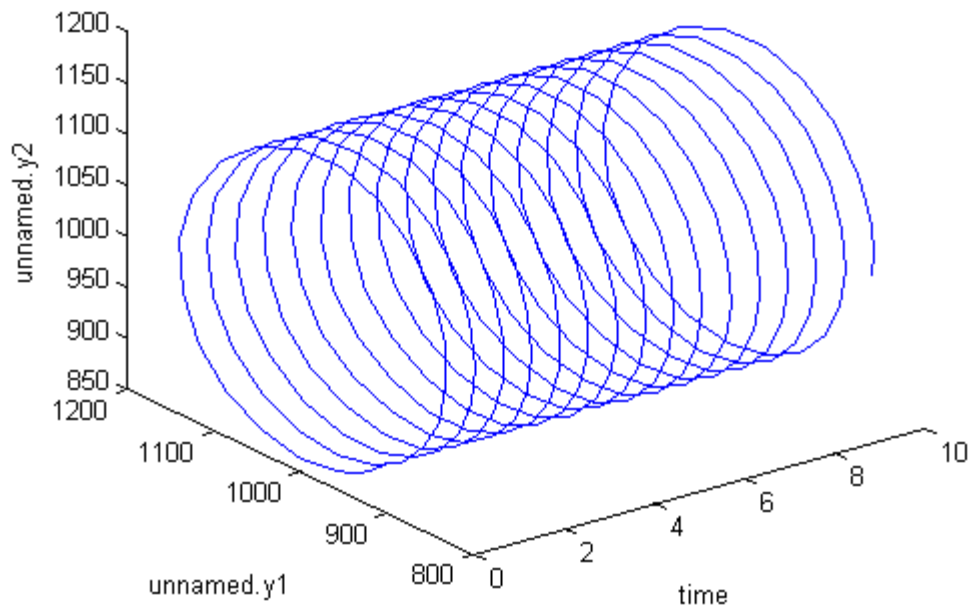
Workflow

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 (y2) and prey (y1)
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)
```
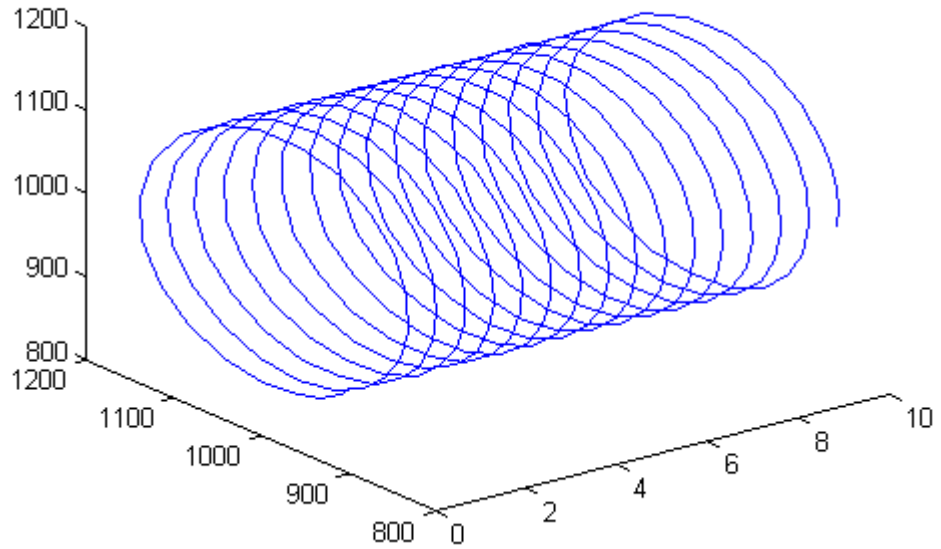
To use this plot for any model, create a function, simplot3, that takes the results returned by the simulation (simDataObj) and creates a 3-D plot showing the relationship between two species over time. x and y are strings representing the species names for function simplot3(simDataObj, x, y).

```
function simplot3(simDataObj, x, y)
% Takes the results returned by the simulation (simDataObj)
% and creates a 3-D plot showing the relationship between
% two species over time.

% 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);
```

3-77

Now to create the above figure, save the function definition in a file on the path (or change directories to the location of the function 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 MATLAB Command Window, type

```
simbiology
```

The SimBiology desktop opens.

2 In the SimBiology desktop, select **Desktop > Library Explorer**. The **Library Explorer** opens.

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

4 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.

5 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 in "Why Use User-Defined Plot Types?" on page 3-76.

```
function simplot3(simDataObj, x, y)
% Takes the results returned by the simulation (simDataObj)
% and creates a 3-D plot showing the relationship between
% two species over time.

% 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.

6 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, under **Define the values that the plot type argument can be configured to**, select Data Source and click **OK**.

7 Double-click x. The Define Argument x dialog box opens

8 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.

9 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.

10 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.
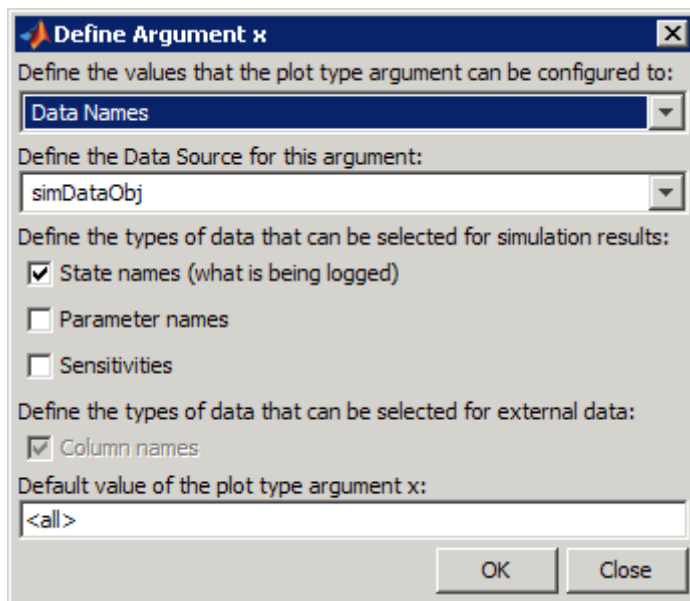
11 Click **OK**.

12 Repeat steps 7 to 9 for the y argument to produce the following.

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>

List of Arguments Passed to Plot Type Code:

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 3-72

13 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

Simulating the Model

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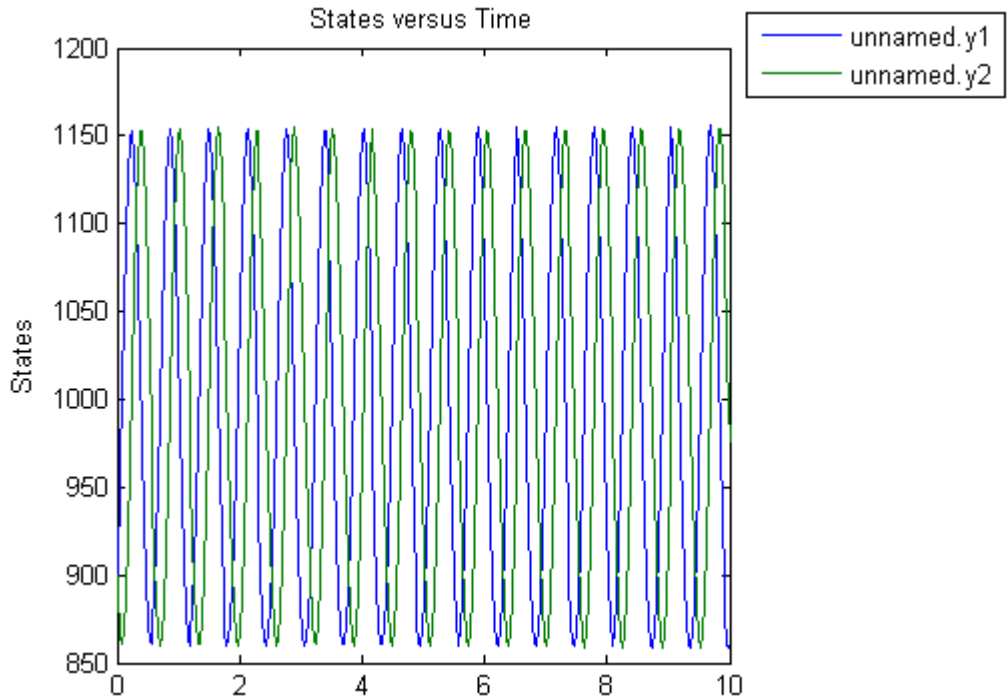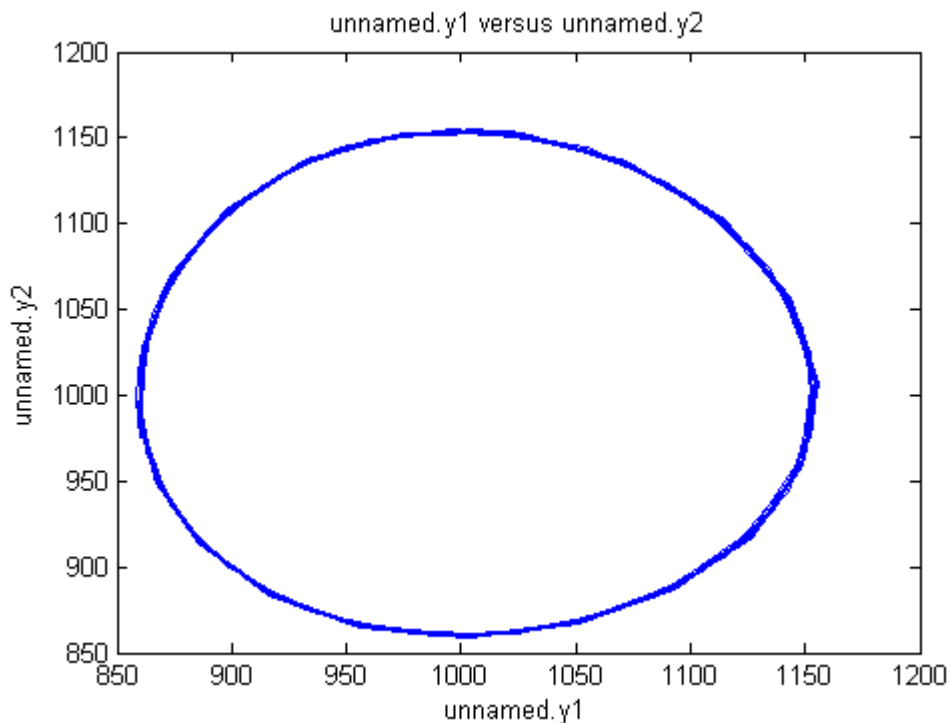
Plotting the Results with the Plot Type

1 In the **Project Explorer** under **Model Tasks**, expand **Simulation** and select **Data**.
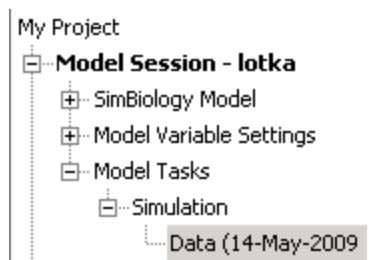
2 In the **Data** pane, click **Plots**.

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 .
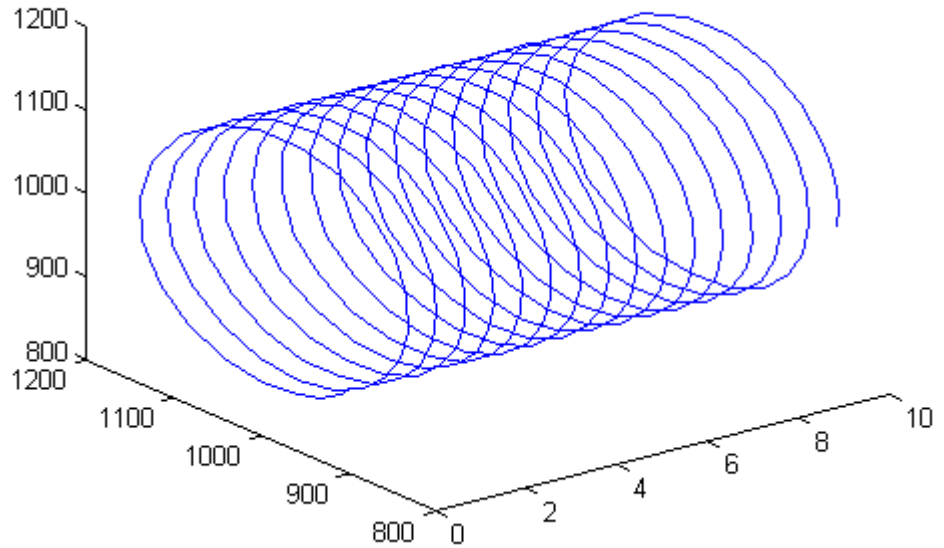
8 Add a legend to the plot.

   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 simplot3.

   d After the simplot3 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]

4

Pharmacokinetic Modeling

Pharmacokinetic Modeling Functionality

In this section...
"Overview" on page 4-2
"Required and Recommended Software for Pharmacokinetic Modeling" on page 4-3
"How This Product Supports Pharmacokinetic Modeling" on page 4-4
"Using the Command Line Versus the SimBiology Desktop" on page 4-6
"Accessing a Pharmacokinetic Modeling Demo" on page 4-6
"Acknowledgements: Tobramycin Data Set" on page 4-6

Overview

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

- Create models — Use a model construction wizard. Alternatively, extend any model with pharmacodynamic (PD) model components, or build higher fidelity models. See "Model" on page 4-4 for more information.

- Fit data — Fit nonlinear, mixed-effects models to data, and estimate the fixed and random effects, or fit the data using nonlinear least squares. For more information, see "Analyze Data Using Models" on page 4-4.

- Generate diagnostic plots — For more information, see "Analyze Data Using Models" on page 4-4.

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

Required and Recommended Software for Pharmacokinetic Modeling

Required Software

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

Recommended Software

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

uses `fminsearch` provided by MATLAB
for likelihood maximization.

How This Product Supports Pharmacokinetic Modeling

Import and Work with Data

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

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

Model

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

- Create a PK model using a model construction wizard to specify the number
  of compartments, the route of administration, and the type of elimination.

- Extend any model with pharmacodynamic (PD) model components, or build
  higher fidelity models.

- Build or load your own SimBiology, or SBML model.

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

Analyze Data Using Models

Perform both individual and population fits to grouped longitudinal data:

- Individual fit — Fit data using nonlinear least-squares method, specify parameter transformations, estimate parameters, and calculate residuals and the estimated coefficient covariance matrix.

- Population fit — Fit data, specify parameter transformations, and estimate the fixed effects and the random sources of variation on parameters using nonlinear mixed-effects models.

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

  - `LME` — Linear mixed-effects approximation

  - `RELME` — Restricted LME approximation

  - `FO` — First-order estimate

  - `FOCE` — First-order conditional estimate

  For more information about each of these methods, see `nlmefit` in the Statistics Toolbox documentation.

- Population fit using a stochastic algorithm — Fit data, specify parameter transformations, and estimate the fixed effects and the random sources of variation on parameters, using the Stochastic Approximation Expectation-Maximization (SAEM) algorithm. SAEM is more robust with respect to starting values. This functionality relaxes assumption of constant error variance.

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

In addition, you can generate diagnostic plots that show:

- The predicted time courses and observations for an individual or the population

- Observed versus predicted values

- Residuals versus time, group, or predictions

- Distribution of the residuals

- A box-plot for random effects or parameter estimates from individual fitting

Using the Command Line Versus the SimBiology Desktop

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

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

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

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

Accessing a Pharmacokinetic Modeling Demo

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

Acknowledgements: Tobramycin Data Set

The original data set was modified as follows:

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

Importing Data — Supported Files and Data Types

Supported Files and Data Types

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

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

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

Unit Conversion

Regardless of whether unit conversion functionality is on or off, dosing in the data file must be expressed in amounts (or as `amount/time` for infusion rate). By default **Unit Conversion** is off, so you must ensure that units for the data are consistent with each other. If you want to turn on unit conversion, see "Unit Conversion for Imported Data and the Model" on page 4-54 .

Support for Importing NONMEM Formatted Files

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

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

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

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

Unsupported NONMEM Definitions

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

• EVID values 3 and 4

• SS column for specifying steady state doses

- `PCMT` column to define whether to compute a prediction for the row

- `CALL` column for calling the ERROR or the PK subroutine

- If rate is specified as being less than zero, it is assumed to be zero

Creating a Data File with SimBiology Definitions

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

- Group column — Specify text or numeric values. The rows in the file that have the same Group column value are for the same individual.

- Time column — Specify monotonically increasing positive values within each group that define the time of the dose, observation and/or covariate measurements.

- Zero or more dosing columns — Create one dosing column for each compartment being dosed. In each column, specify positive values representing doses in amount that are added to a species. Use `0` or `NaN` to specify that no dose was applied at the specified time. This is useful for times when an observation was recorded but no dose was applied.

- Zero, or more rate columns — Specify positive values. Zero specifies an infinite rate and `NaN` specifies that no rate applies. The rate column is associated with a dosing column and defines the rate at which the dose is administered.

- Zero or more observation columns — Specify numeric values or `NaN`s. You can only specify one observation value at a particular time for each group. `NaN` values define that no observation was recorded at the specified time. This is useful for times when a dose was applied but no observation was recorded.

- Zero or more covariate columns — Specify numeric values or `NaN`s. Each value defines the covariate value at the specified time. `NaN` values define that no covariate observation was recorded at the specified time.

Importing Data at the Command Line

In this section...

Importing Data From NONMEM Formatted Files

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

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

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

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

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

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

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

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

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

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

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

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

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

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

For information about creating a model to fit the data, see "Creating PK Models at the Command Line" on page 4-44.

Importing Data Using the dataset Function

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

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

Examples:

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

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

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

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

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

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

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

Other Resources for Importing Data

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

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

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

Importing Data into the SimBiology Desktop

Data Import Using the SimBiology Desktop

From theSimBiology desktop, you can import any one of the following:

- Delimited text files, including NONMEM formatted files

- Microsoft® Excel® spreadsheets, including NONMEM formatted files

- A MAT-file containing variables that are of `double` data type; or a `dataset` object, time series object, or a `SimData object`

- Variables that are in the MATLAB Workspace (for example, `double`, `dataset` object, time series objects, or a `SimData object`)

- A SimBiology project (`.sbproj` file) containing data

During import, you can specify whether the first row contains header information, and whether the import process should interpret any characters in the file 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. After you import the data, 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. Preparing the data helps the import process to construct the `dataset` array on the SimBiology desktop. If you have data in a NONMEM formatted file, for additional details see "Support for Importing NONMEM Formatted Files" on page 4-8 .

- 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 a File to the SimBiology Desktop

This section shows how to import data from a text, Excel, or `.mat` file, including data from NONMEM formatted files.

1 Open the SimBiology desktop if it 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 Import your file.

Import a Text File

a Select a `.txt` file and click **Open**. The Text File Import dialog box opens. This dialog box lets you preview the data. For example, browse to *matlabroot*/toolbox/simbio/simbiodemos/, where *matlabroot* is the folder where MATLAB is installed. Select tobramycin.txt [1].
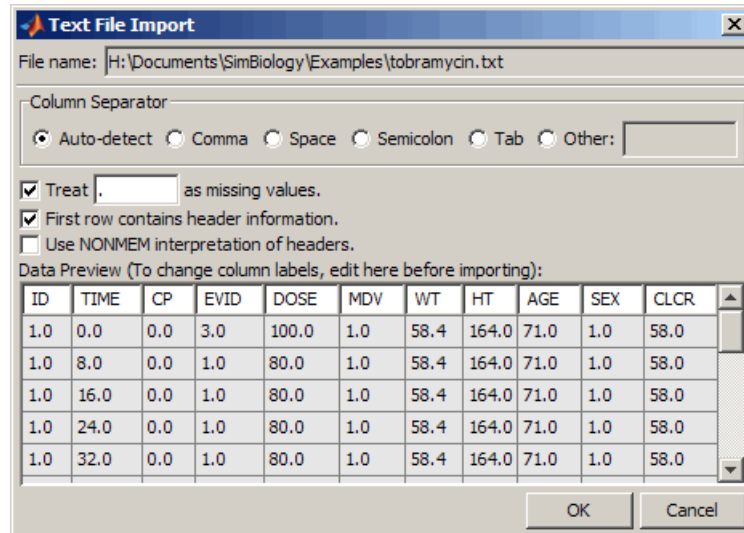
b Under **Column Separator**, select the type of separator in the text file:

- 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.

c 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.

Import an Excel File

- Select a .xls file and click **Open**. The Excel File Import dialog box opens. This dialog box lets you preview the data.
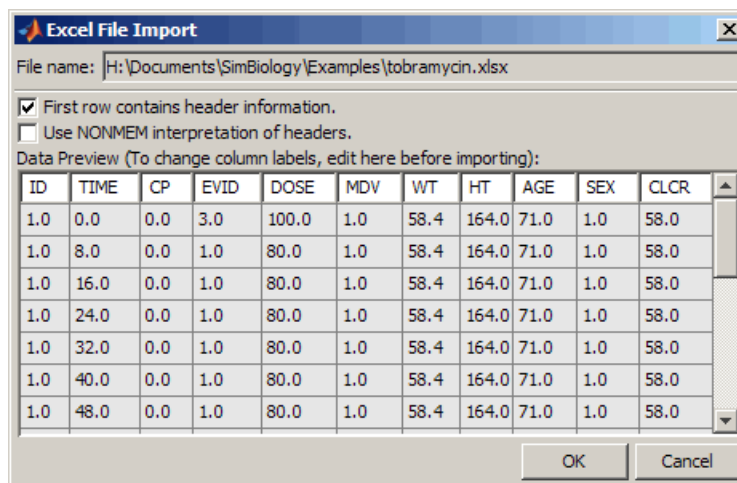
If your data file contains missing values, the **Data Preview** table shows missing values as NaN.

Import a MAT-File

a 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.
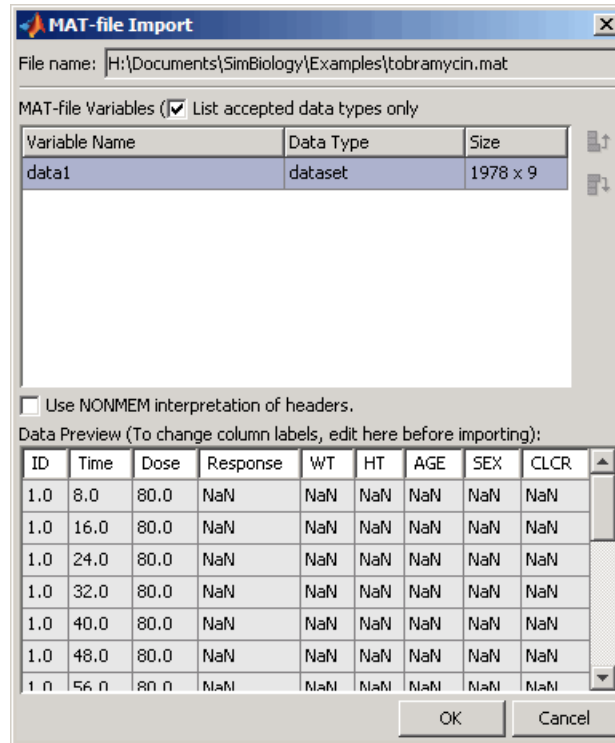
b In the MAT-file Variables table, select the variables to import.

- If the **Data Type** of the variable you want 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 want 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.

  - The import process concatenates selected variables column-wise.

  - The import process concatenates the variables in the order that they appear in the table, from top to bottom, resulting in columns ordered from left to right.

  - If necessary, the import process transposes a matrix 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.

4 If your data file does not contain header information in the first row, clear
the **First row contains header information** check box. If using the
tobramycin data set [1], select this check box.

5 If the data is in a NONMEM formatted file, select the **Use NONMEM
interpretation of headers** check box. If not, skip this step. If using the
tobramycin data set [1], select this check box.

a In the **Column Heading Classification** section, verify or select the
classification that applies to each column header, and then click **Update
Preview**. If you are using the tobramycin data set [1], for the row
containing the **Column Heading** CP, from the **Classification** list select
dependent variable.

For a complete list of available classifications and their description,
see "Column Heading Classification for NONMEM Interpretation of
Headers" on page 4-27.

The **Preview** tab updates with the data.
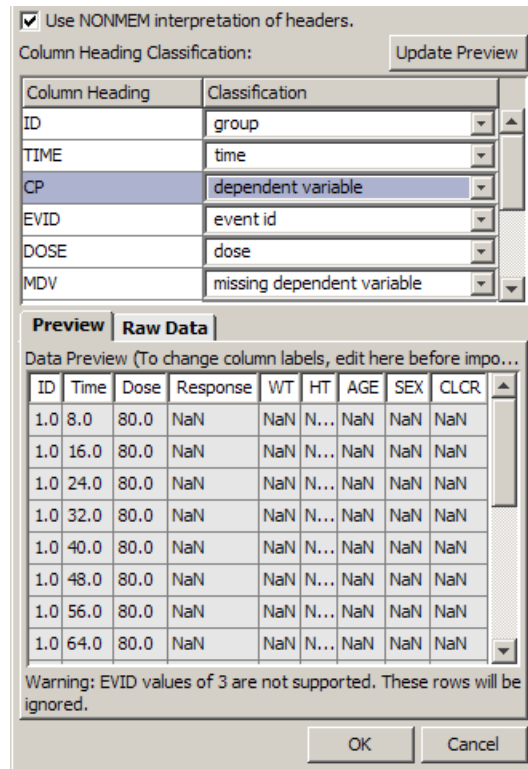
The covariate columns, such as weight, height, and age, are listed as
NaNs; this is because the EVID values for these rows is not 2. The data
shows the covariate values for the row with value 2 in the EVID column.

The representation of the data in SimBiology differs from that of the
imported data file because SimBiology uses a separate column for each
dose and observation, for each given compartment. For example, data
from a NONMEM formatted file, in the first image below, is represented
in SimBiology, as shown in the second image. Click the **Preview** and
Raw Data tabs to alternate between the two views.

6 (Optional) To edit the names of the column headers, double-click the cells in the first row of the **Data Preview** table.

Note After importing the data to 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

7 Click **OK**. The SimBiology desktop adds the data to the **Project Explorer** under **External Data**, and opens the pane containing the data.

8 (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 the MATLAB Workspace to the SimBiology Desktop

If you already have 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 4-6.

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.
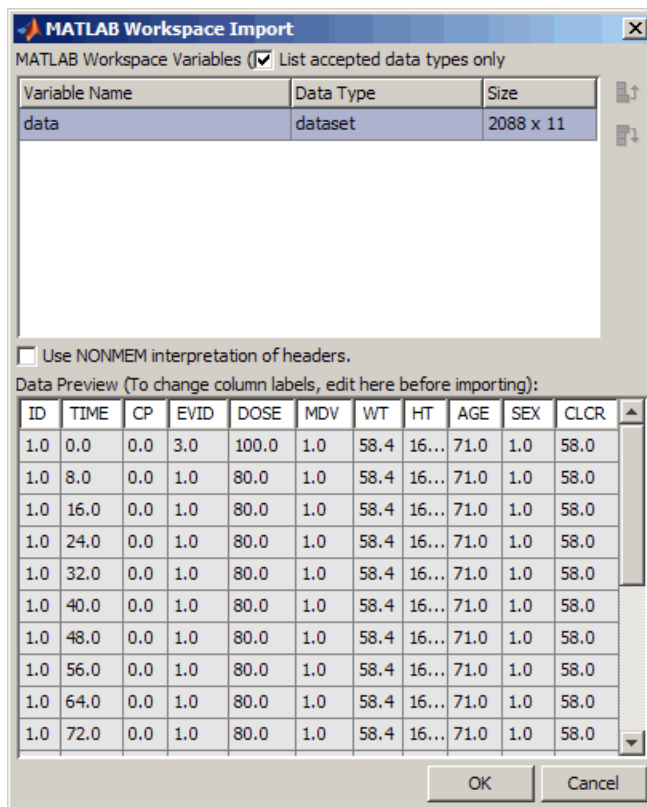
3 In the **MATLAB Workspace Variables** table, select the variables to import.

- If the **Data Type** of the variable you want 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 want 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.

  - The import process concatenates selected variables column-wise.

- The import process concatenates the variables in the order that they appear in the table, from top to bottom, resulting in columns ordered from left to right.

- If necessary, the import process transposes a matrix to create a matrix with compatible size.

The **Size** column shows *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 Follow steps 5 – 8 in "Importing Data from a File to the SimBiology Desktop" on page 4-18.

Column Heading Classification for NONMEM Interpretation of Headers

Column Heading Classification

Classification	Description
compartment	Indicates which compartment is used for observation value or for dose received. The interpretation also depends on `EVID`: • Observation event (`EVID = 0`) — `CMT` column indicates which compartment was used for observation value. • Dose Event (`EVID = 1`) — `CMT` column indicates which compartment received the dose.
continuous covariate	Covariates such as age.

Classification	Description
date	Defines the day of the observation or the dose. The data can contain the month as a number (9) or a string (Sep). The data file can specify date in the following formats: • month/day/year or month-day-year. If the column is titled DATE the import process assumes this format. If you specify two numbers, the import process assumes they are month and day. • day/month/year or day-month-year. If the column is titled DAT1 the import process assumes this format. If you specify two numbers, the import process assumes they are day and month. • year/month/day or year-month-day. If the column is titled DAT2 the import process assumes this format. If you specify two numbers, the import process assumes they are month and day. • year/day/month or year-day-month. If the column is titled DAT3 the import process assumes this format. If you specify two numbers, the import process assumes they are day and month. Note the following additional assumptions: • If you specify only one number the import process assumes it to be the day • You can omit the year or specify 1, 2, 3 or 4 digits. If you specify two digit years the import process assumes it be in the 1900s.
dependent variable	Numeric value of an observation. Column cannot contain Inf or Inf.
dose	Positive number indicating dose. 0 or NaN specifies no dose administered. The column cannot contain Inf.

Classification	Description
dose interval	Positive number defining the time between doses.
dose repeat	When the data specifies a number of identical serial doses at specific intervals (defined by dose interval), dose repeat specifies the number of doses in the series excluding the initial dose. If you specify dose interval but not dose repeat, then SimBiology assumes that the dosing occurs for the duration of that data record.
event id	Defines the type of event that is described for that row in the record: • 0 — Observation event; row contains an observed value. • 1 — Dose event; row describes a dose. • 2 — Other event; row describes some other event such as measurement of a covariate. If a column contains values for dose, but event id is 0, the import process ignores the value. You will see a warning and the value is ignored. The import process does not support values 3 and 4 (which are used to reset the compartment or reset the compartment and administer a dose). You will see a warning and the value is ignored.
group	Text or numeric values that identify the record. The import process assumes that contiguous data with the same value contains data from one individual. If the data contains non-contiguous references to the same value, during import the second time the value is encountered is assigned an indexed valued derived from the value first encountered. For example if the group column contains [1 1 1 2 2 2 1 1 1] the IDs assigned are 1, 2, 1_1

Classification	Description
missing dependent variable	Defines whether a row describes an observation: • 0 — Observation event • 1 — Not an observation event
rate	Positive number indicating rate of infusion. 0 specifies an infinite rate (equivalent to a bolus dose), and NaN specifies no rate. The column cannot contain Inf.
time	Monotonically increasing positive values within each group, indicating time of observation or dose. The data file can specify clock (2:30) or decimal values (6.25). The import process assigns a value of 0 to the first time value in the data file . The import process assigns subsequent values relative to the first value. For example [10:05 10:30 11 12:30 21.3] is interpreted as: [0 0.25 0.95 2.25 14.2].If the data file also contains a date column, it is used with the time column in calculating the relative time values. The column cannot contain Inf.

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	"Specifying Units and Identifying Group and Independent Variables in the Data" on page 4-33
Plotting data	"Plotting Data" on page 4-34
Calculating statistics	"Calculating Statistics for Imported Data" on page 4-39

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	• "Creating PK Models at the Command Line" on page 4-44
	• "Creating PK Models in the SimBiology Desktop Using a Wizard" on page 4-46
Fitting data	• "Fitting Pharmacokinetic Model Parameters at the Command Line" on page 4-61
	• "Fitting Pharmacokinetic Model Parameters in the SimBiology Desktop" on page 4-80

Working with Imported Data and the SimBiology Desktop

In this section...

Accessing Imported Data

1 Open SimBiology desktop. At the MATLAB command line type:

```
simbiology
```

2 Open the data set you want to use. In the **Project Explorer**, under **External Data**, click the name of the data set.

3 View information about the imported data. Click the **Description** tab to see warnings generated during import.

Specifying Units and Identifying Group and Independent Variables in the Data

After importing data to 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. You also can specify units for your data. SimBiology lets you perform unit conversion.

In addition, SimBiology also populates the independent and group variable settings for the parameter fitting task.

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.

- Under the column header in the **Units** row, select a unit from the list.

Plotting Data

Open the pane containing the data as shown in "Accessing Imported Data" on page 4-32.

1 If you want to plot the data by group, do the following. If not, go to step 2.

   a On the **Plot** tab, select the **Group data by column** check box.

   b From the list of columns, select the column that contains the groups.
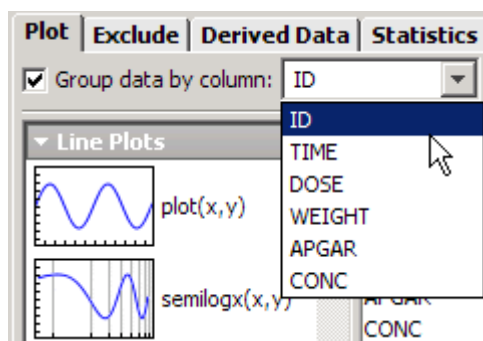
2 From the list of plots, select the type of plot.
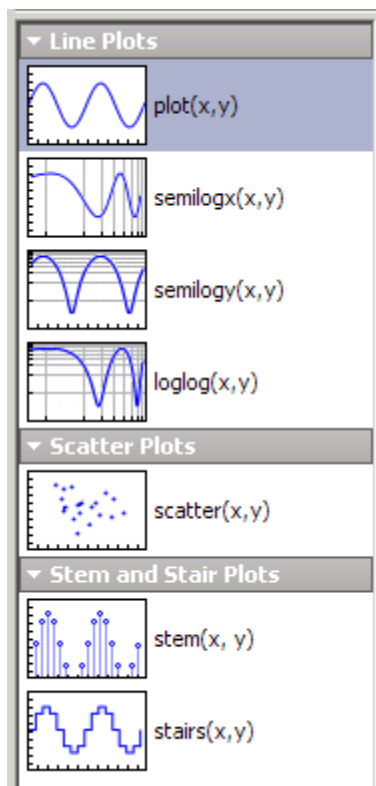
3 In the **x** table, select the column to plot in the x-axis.

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**.
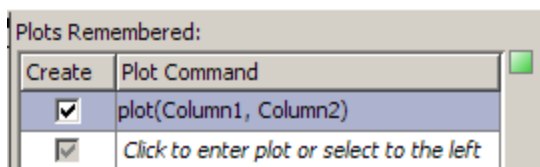
6 (Optional) In the **Plots Remembered** table, enter any plot command. If the command is invalid and you select the **Create** check box, you will see a red indicator. If the **Create** check box is clear, you see a yellow indicator for invalid commands.

7 (Optional) By default, the **Plot when selection changes** check box appears selected. If you do not want plotting to occur automatically, clear this check box. 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 4-32.

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 appear 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.
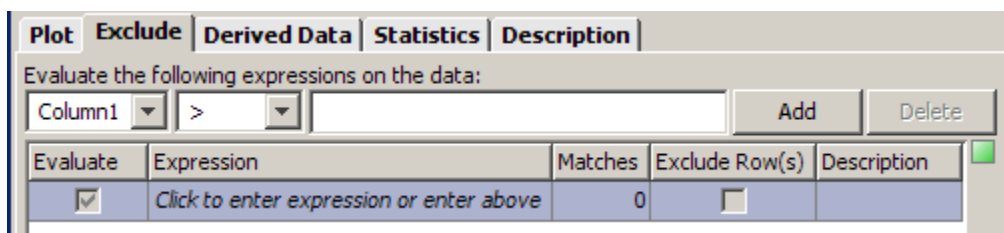
2 Enter an expression in the **Expression** cell. For example:

```
WEIGHT < (1.4)
```

Summary of Matches to the right of the table shows the all the matches from the evaluated expressions.

3 Exclude the rows that matched the expression by selecting the **Exclude Row(s)** check box. The excluded rows appear 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 4-32.

2 Click the **Exclude** tab.
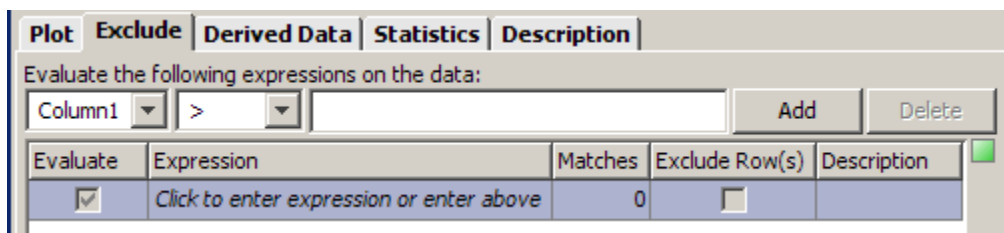
3 From the list under **Evaluate the following expressions on the data**, select a column heading. For example in the previous figure, the column heading is Column1.

4 Select a relational operator from the list.
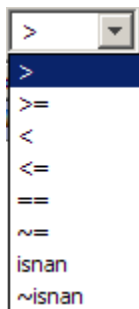
5 In the text box, enter a value to compare against and click **Add** or press **Enter**. The table in the **Exclude** tab updates with the **Expression** and the number of **Matches** in the data. Notice that the table containing the data shows black markers for the rows that matched the expression.

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. Use this feature when you want to derive data—for example, body surface area or body mass index.

To create a new column of derived data:

1 Click the **Derived Data** tab.

2 In the **Column Label** column, double-click to enter a name for the new column. For example:

    DOSE_RATE

3 Press **Enter**.

4 In the **Expression** column, double-click to enter a MATLAB expression, using valid column names from the data set. For example, assume that the

data set contains a DOSE column, specifying drug dose, and a DOSE_TIME column, specifying dose time. To calculate dose rate, type:

```
DOSE/DOSE_TIME
```

5 Press **Enter**.

The new column of data is calculated and added to the table.

6 To remove the column, clear the **Add Column** check box.

Calculating Statistics for Imported Data

You can calculate statistics for the grouped data in the **Statistics** tab:

1 Open the pane containing the data set as shown in "Accessing Imported Data" on page 4-32.

2 Click the **Statistics** tab.

3 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).

4 To calculate auc, aumc, and mrt, right-click the table containing the statistics, and select **Statistics to Calculate**.

5 Click **OK**.

6 In the Statistics to Calculate dialog box, you can specify the additional statistics to view.

7 (Optional) 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
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

Creating Dose Schedules Using Imported Data

1 Open the pane containing the data set as shown in "Accessing Imported Data" on page 4-32.

2 From the context menu, select **Create Dose**. The Create Dose dialog box opens.

3 From the **Select model to add dose to** list, select a model to which you want to apply the dose schedule.

4 From the **Select time heading for dose** list, select the name of the column header in the data set that contains the time of dose administration.

5 From the **Select groups to create dose for** list, select the name of the column header in the data set that contains the group labels.

6 Under **Group Name**, select the check boxes for the group data to use in creating a dose schedule.

7 Under **Select amount headings to create dose for**, select the names of the column headers in the data set that contain the dose amount (**Amount Column** list) and the rate (**Rate Column** list) if applicable.

8 Click **OK**.

SimBiology creates the dose and adds it to the **Dose** pane. Select **Project Explorer > Doses**, to access the dose.

To apply the dose, in a task pane (for example, **Simulation**) select the **Use in Task** check box for the dose.

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 format uses the file extension `.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 on the SimBiology desktop, the model and the data are saved together in the project.

1 To open the Save SimBiology Project dialog box, select **File > Save Project as**.

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 4-42
"How SimBiology Models Represent Pharmacokinetic Models" on page 4-42
"Creating PK Models at the Command Line" on page 4-44
"Creating PK Models in the SimBiology Desktop Using a Wizard" on page 4-46
"About Dosing Types" on page 4-48
"About Elimination Types" on page 4-51
"About Intercompartmental Clearance" on page 4-53
"Unit Conversion for Imported Data and the Model" on page 4-54
"Prerequisites for Using Custom SimBiology Models in Parameter Fitting" on page 4-55

Overview

To start modeling, you can:

- Create a PK model using a model construction wizard that lets you specify the number of compartments, the route of administration, and the type of elimination.

- Extend any model to build higher fidelity models.

- Build or load your own model. Load a SimBiology project or SBML model.

How SimBiology Models Represent Pharmacokinetic Models

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

Comparison of views in PK
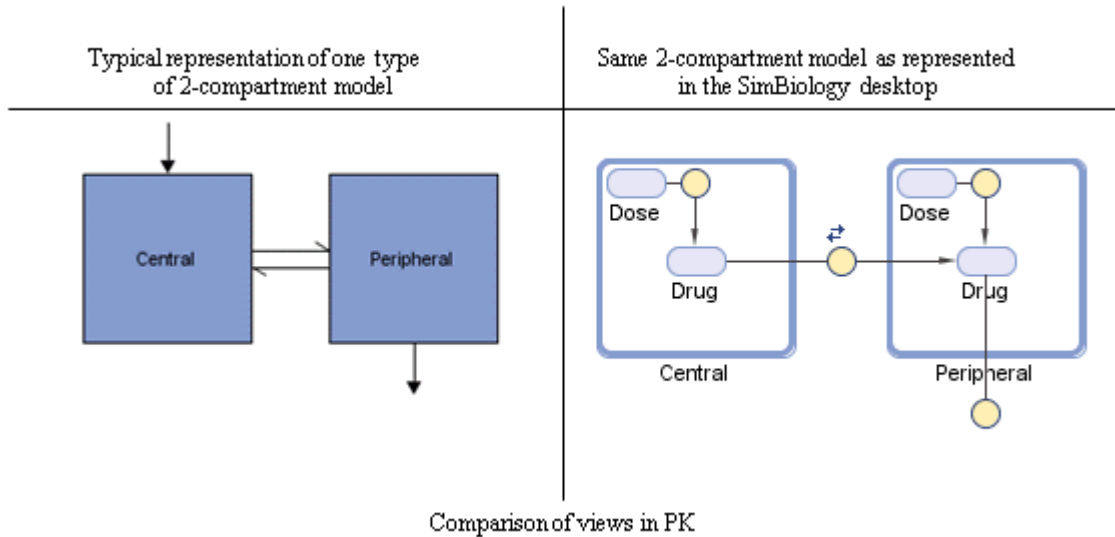
Note the following:

- SimBiology represents the concentration or amount of a drug in a given compartment or volume by a species *object* contained within the compartment.

- SimBiology represents the exchange or flow of the drug between compartments and the elimination of the drug by reactions.

- SimBiology represents intercompartmental clearance by a parameter (`Q`) which specifies the clearance between the compartments.

- SimBiology drives the dosing schedule with a combination of species (`Drug` and/or `Dose`) and reactions (`Dose -> Drug`), depending on whether the administration into the compartment follows bolus, zero-order, infusion, or first-order dosing kinetics. For more information on the components added and parameters estimated, see "About Dosing Types" on page 4-48.

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

Creating PK Models 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, and rules.

```
pkm = PKModelDesign;
```

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

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

For a description of other `DosingType` and `EliminationType` property values, see "About Dosing Types" on page 4-48 and "About Elimination Types" on page 4-51.

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

> **Note** To add a compartment that has a time lag associated with any dose that targets it, set the `HasLag` property to `true`:
>
> ```
> pkc_lag = addCompartment(pkm, 'Central', 'DosingType', 'Bolus', ...
>                                 'EliminationType', 'linear-clearance', ...
>                                 'HasResponseVariable', true, 'HasLag', true);
> ```
>
> Or after adding a compartment, set its `HasLag` property to `true`:
>
> ```
> pkc1.HasLag = true;
> ```

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

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

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

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

4 Construct a SimBiology model object.

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

The `construct` method returns a SimBiology model object (`modelObj`) and a `PKModelMap` object (`PKModelMapObj`) that contains the mapping of the model components to the elements of the regression function. For more information about the `PKModelMap` object, see "Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters" on page 4-56.

> **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 4-61.

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

For information on ...	See ...
Dosing types	"About Dosing Types" on page 4-48
Elimination types	"About Elimination Types" on page 4-51
Parameter fitting	"Fitting Pharmacokinetic Model Parameters at the Command Line" on page 4-61
Simulating the model and a description of configuration sets	• "About Simulation Settings and Specifying Alternate Values for Initial Estimates" on page 4-79 • "Performing Simulations" on page 2-2

Creating PK Models in the SimBiology Desktop Using a Wizard

To create a PK model, in the Add Model wizard, specify the number of compartments, route of administration, and method of clearance.

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. If you are using the tobramycin data set [1], select 1 compartment. 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. If you are using the tobramycin data set, select `bolus`.

```
None
bolus
infusion
zero-order
first-order
```

See "About Dosing Types" on page 4-48 for more information on each of the above options.

7 From the **Elimination** list, select one of the following. If you are using the tobramycin data set, select `Linear{Clearance, Volume}`.

```
None
Linear{Clearance, Volume}
Linear{Elimination Rate, Volume}
Enzymatic(Michaelis-Menten)
```

See "About Elimination Types" on page 4-51 for more information on the above options.

8 In the Response column, select or clear the **Response** check box to indicate if there is a measured response for this compartment. At least one compartment in a model must have a response. Although SimBiology supports multiple responses per compartment, when using the Add Model

wizard, you are limited to one response per compartment. If you are using the tobramycin data set, select this check box.

9 In the Time Lag column, select or clear the **Time Lag** check box to indicate that any dose targeting this compartment has a time lag associated with it. If you are using the tobramycin data set, do not select this check box.

10 (Optional) Click the **SimBiology View** tab to see how SimBiology represents the model. For more information, see "How SimBiology Models Represent Pharmacokinetic Models" on page 4-42.

11 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

Whether creating models at the Command Line or from the SimBiology Desktop, SimBiology creates the following model components for each compartment in the model, regardless of the dosing type:

- Two species (Drug_CompartmentName and Dose_CompartmentName) for each compartment.

- A reaction (`Dose_CompartmentName -> Drug_CompartmentName`) for each compartment, governed by mass action kinetics.

- A parameter (`ka_CompartmentName`) for each compartment, representing the absorption rate of the drug when absorption follows first-order kinetics. This is the forward rate parameter for the `Dose_CompartmentName -> Drug_CompartmentName` reaction.

- A parameter (`TkO_CompartmentName`) for each compartment, representing the duration of drug absorption when absorption follows zero-order kinetics.

- A parameter (`TLag_CompartmentName`) for each compartment, representing the time lag for any dose that targets that compartment and also that is specified as having a time lag.

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

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

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

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

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

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

For information on ...	See ...
Preparing the model before simulating	"Prerequisites for Using Custom SimBiology Models in Parameter Fitting" on page 4-55

About Elimination Types

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

Elimination Type	Description	SimBiology Model Components Created	Default Parameters to Estimate
	elimination rate constant specified by the forward rate parameter (`ke`).	specific to the compartement	there is more than one compartment. See "About Intercompartmental Clearance" on page 4-53.
SimBiology desktop — `Linear {Clearance, Volume}` Command line — `'linear-clearance'`	Assumes simple mass-action kinetics in the elimination of the drug. In the SimBiology model, similar to `Linear {Elimination Rate, Volume}`. But, in addition, this option lets you specify the model in terms of clearance (Cl) where, Cl = ke * volume).	• A parameter representing the clearance (`Cl_CompartmentName`) • A parameter representing the elimination rate constant (`ke_CompartmentName`) • An `InitialAssignment` rule that initializes `ke_CompartmentName` based on the initial values for `Cl_CompartmentName` and compartment volume • A MassAction reaction (drug > null) with forward rate parameter (`ke_CompartmentName`)	• Compartment volume (`Capacity` property) • Clearance (`Cl_CompartmentName`) • Inter-compartmental clearance (`Q`) when there is more than one compartment. See "About Intercompartmental Clearance" on page 4-53.
SimBiology desktop — `Enzymatic (Michaelis-Menten)`	Assumes that elimination is governed by	• Parameter representing the	• Compartment volume (`Capacity` property)

Elimination Type	Description	SimBiology Model Components Created	Default Parameters to Estimate
Command line — `'enzymatic'`	Michaelis-Menten kinetics.	Michaelis constant, (Km_CompartmentName) • A parameter for maximum velocity (Vm_CompartmentName • A reaction with Michaelis-Menten kinetics (drug -> null), with kinetic law parameters Vm_CompartmentName and Km_CompartmentName	• Parameter (Km_CompartmentName) • Parameter (Vm_CompartmentName) • Inter-compartmental clearance (Q) when there is more than one compartment. See "About Intercompartmental Clearance" on page 4-53

About Intercompartmental Clearance

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

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

• A parameter Q_{mn} representing the compartmental clearance between those two compartments. This parameter is added to the list of parameters to be estimated (PKModelMapObj.Estimated property).

• A parameter (kmn) representing the rate of transfer of the drug from Cm to Cn, where $k_{mn} = Q_{mn}/V_m$.

• A parameter (knm) representing the rate of Cn to Cm, where $k_{nm} = Q_{mn}/V_n$.

- A reversible mass-action reaction between the two compartments, Cm.Drug_Cm <-> Cn.Drug_Cn, with forward rate parameter kmn, and reverse rate parameter knm.

- An initial assignment rule that initializes the value of the parameter kmn, based on the initial values for Cm and Qmn.

- An initial assignment rule that initializes the value of the parameter knm, based on the initial values for Cn and Qmn.

Unit Conversion for Imported Data and the Model

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

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

If **Unit Conversion** is on, you must specify units in the **External Data** pane (SimBiology desktop) or in the PKData object (command line)

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

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

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

Use the configuration settings options to turn unit conversion on or off. For more information see, "Performing Simulations" on page 2-2 in the SimBiology documentation.

See "Evaluation of Reaction Rate" on page 1-25 in the SimBiology documentation for more information on dimensional analysis for reaction rates.

Prerequisites for Using Custom SimBiology Models in Parameter Fitting

Overview

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

When using a custom model, you must provide information about whether dosing is applicable and define which components of the SimBiology model represent the observed response, the dose, and the estimated parameters. Use the PKModelMap object to define these settings as shown in "Defining Model Components for Observed Response, Dose, Dosing Type, and Estimated Parameters" on page 4-56.

Another point to consider is the solver you use in performing simulations. During fitting the solver type must be sundials to support any events in the model. See "About Simulation Settings and Specifying Alternate Values for Initial Estimates" on page 4-79, for more information.

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

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

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

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

1 Import an SBML model:

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

2 Create a PKModelMap object:

```
PKModelMapObj = PKModelMap;
```

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

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

For example:

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

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

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

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

For example:

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

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

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

For information on ...	See ...
PKModelMap object properties and allowed values	PKModelMap object Dosed, DosingType, Estimated, and Observed, ZeroOrderDurationParameter
Allowed dosing types	"About Dosing Types" on page 4-48

For information on ...	See ...
Parameter scoping	"Using a Parameter in Events and Rules" on page 1-33
Parameter fitting	"Fitting Pharmacokinetic Model Parameters at the Command Line" on page 4-61

Dosing Multiple Compartments in a Model

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

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

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

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

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

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

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

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

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

Parameter Fitting in Pharmacokinetic Models

In this section...
"Parameter Fitting Functionality " on page 4-59
"Prerequisites for Parameter Fitting" on page 4-60

Parameter Fitting Functionality

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

- Individual fit — Fit data separately for each individual using the nonlinear least squares method, estimate parameters, and calculate residuals and the estimated coefficient covariance matrix.

- Population fit — Estimate the fixed effects and the random sources of variation on parameters, using nonlinear mixed-effects models.

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

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

The following results are returned for population fitting:

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

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 — Supported Files and Data Types" on page 4-8 for more information.)
- A model to fit (See "Creating Pharmacokinetic Models" on page 4-42 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 4-61
- "Fitting Pharmacokinetic Model Parameters in the SimBiology Desktop" on page 4-80

Fitting Pharmacokinetic Model Parameters at the Command Line

In this section...
"Fitting Parameters" on page 4-61
"Specifying and Classifying the Data to Fit" on page 4-62
"Setting Initial Estimates" on page 4-65
"Specifying a Nonlinear, Mixed-Effects Model" on page 4-66
"Specifying a Covariate Model at the Command Line" on page 4-67
"Specifying the Covariance Pattern of Random Effects" on page 4-68
"Specifying an Error Model" on page 4-70
"Specifying Parameter Transformations" on page 4-71
"Performing Population Fitting Using sbionlmefit or sbionlmefitsa" on page 4-72
"Performing Individual Fitting Using sbionlinfit" on page 4-77
"About Simulation Settings and Specifying Alternate Values for Initial Estimates" on page 4-79

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 4-13.

2 Specify the structural model by creating a PK model as shown in "Creating PK Models at the Command Line" on page 4-44. Alternatively, if you have a SimBiology model that you want to use in fitting, see "Prerequisites for Using Custom SimBiology Models in Parameter Fitting" on page 4-55.

3 Classify the data set to use in fitting. See "Specifying and Classifying the Data to Fit" on page 4-62.

4 Specify the initial guesses for the parameters to be estimated, as shown in "Setting Initial Estimates" on page 4-65.

5 Perform individual or population fits:

- For individual fits:
    - (Optional) Set tolerances and specify maximum iterations as shown in "Setting Tolerance and Maximum Iteration for Individual Fits" on page 4-96.

- For population fits:
    - Specify the statistical model:

        Specify the covariate model and the covariance matrix. See "Specifying a Covariate Model at the Command Line" on page 4-67 and "Specifying the Covariance Pattern of Random Effects" on page 4-68.

        Specify the error model. See "Specifying an Error Model" on page 4-70.

    - (Optional) Set tolerances and specify maximum iterations as shown in "Setting Tolerance and Maximum Iteration for Population Fits" on page 4-90.

6 Obtain and visualize results as shown in "Visualizing Parameter Fitting Results and Generating Diagnostic Plots" on page 4-97.

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 4-13.

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

To create the `PKData` object:

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

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

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

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

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

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

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

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

- These data points are not averaged, but fitted individually.

- Different number of observations at different times cause some time points to be weighted more.

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

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

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

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

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

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

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

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

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

Setting Initial Estimates

To set the initial estimates (or initial guesses) for the parameters with fixed effects to estimate, first identify the sequence of the parameters in the model by querying the PKModelMap object. Next, construct a vector, betaO, containing the initial conditions. For information about PKModelMap objects, see step 4 in "Creating PK Models at the Command Line" on page 4-44.

1 Query the Estimated property of the PKModelMap object:

```
PKModelMapObj.Estimated
```

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

```
ans =

    'Central'
    'Cl_Central'
```

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

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

For information on ...	See ...
The parameters added to the model	• "About Dosing Types" on page 4-48 • "About Elimination Types" on page 4-51
Default units for the above parameters	"Unit Conversion for Imported Data and the Model" on page 4-54

Specifying a Nonlinear, Mixed-Effects Model

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

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

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

The model is:

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

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

φ_i	A vector of group-specific model parameters
β	A vector of fixed effects, modeling population parameters
b_i	A vector of multivariate, normally distributed, group-specific, random effects
A_i	A group-specific design matrix for combining fixed effects
B_i	A group-specific design matrix for combining random effects
X_i	A data matrix of group-specific covariate values
y_i	A data vector of group-specific response values
f	A general, real-valued function of φ_i and X_i
ε_i	• For `sbionlmefit`, you can specify different error models as shown in "Specifying an Error Model" on page 4-70. • For `sbionlmefitsa`, you can specify different error models as shown in "Specifying an Error Model" on page 4-70.

Ψ   A covariance matrix for the random effects

σ^2 The error variance, assumed to be constant across observations

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

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

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

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

Specifying a Covariate Model at the Command Line

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

$$\begin{pmatrix} Cl \\ V \\ ka \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & w_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \beta_{cl} \\ \beta_{v} \\ \beta_{ka} \\ \beta_{w} \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} b_{cl} \\ b_{v} \\ b_{ka} \end{pmatrix}$$

Thus, the parameter for clearance (*Cl*) for an individual is $Cl_i = \beta_{Cl} + \beta_{Cl/w} * w_i + b_{Cl_i}$.

Assuming that 1 is the initial estimate for β_{Cl} and 0 is the initial estimate for $\beta_{Cl/w}$, use the `PKCovariateModel.Expression` property to define the covariate model.

```
covmodel = PKCovariateModel;
covmodel.Expression = ({'Cl = 1*exp(0*weight)'});
```

Use the `PKCovariateModel.getInitialEstimate` method to determine the order of elements:

```
[initialEstimates elementNames] = ...
    covmodel.getInitialEstimate(PKModelMapObj, PKDataObj)];
disp('Element Names:');
disp(elementNames);
```

Your output appears as follows:

```
Element Names:
     'Central'
     'Cl_Central'
     'Central/WEIGHT'
     'Cl_Central/WEIGHT'
```

Update the initial estimates using the values estimated from fitting the base model (model without covariate effects):

```
covmodel.setInitialEstimate(PKModelMapObj, PKDataObj, ...
    [nlmeResults.estimate' 0 0 ]);
disp('Covariate model expression with updated initial estimates:');
disp(covmodel.Expression)
```

Your output appears as follows:

```
Covariate model expression with updated initial estimates:
     'Central = 1.40855891462797 * exp(0*(WEIGHT))'
     'Cl_Central = 0.00613734942978561 * exp(0*(WEIGHT))'
```

Specifying the Covariance Pattern of Random Effects

By default, the function you use to perform population fits (`nlmefit` or `nlmefitsa`) assumes a diagonal covariance matrix (no covariance among the random effects). To specify a different covariance pattern of random effects,

use the `'CovPattern'` option. In the previous example, assuming that each of the parameters has random effects and that *Cl* and *V* exhibit covariance, the covariance pattern of random effects would be a logical array:

$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

1 Create an options `struct` with the specified covariance pattern:

```
options.CovPattern = [1, 1, 0; 1, 1, 0; 0, 0, 1];
```

2 Specify the arguments for `sbionlmefit` or `sbionlmefitsa`:

```
[results, simdataI, simdataP] = sbionlmefit(modelObj,...
    PKModelMapObj, PKDataObj, betaO, options)
```

If you are using the tobramycin data set [1], do the following:

1 Create an options `struct` with the specified covariance pattern:

```
options.CovPattern = [1, 0; 0, 1];
```

2 Specify the arguments for `sbionlmefit`:

```
[results, simdataI, simdataP] = sbionlmefit(modelObj,...
    PKModelMapObj, PKDataObj, betaO, options)

results =

    estimate: [2x1 double]
        phiI: [2x97 double]
        phiP: [2x97 double]
        beta: [2x1 double]
         psi: [2x2 double]
       stats: [1x1 struct]
           b: [2x97 double]

results.estimate
```

4-69

```
ans =

    21.6535
     3.7172

results.beta

ans =

     3.0752
     1.3130
```

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

Fitting the model and estimating the covariance matrix Ψ often leads to further refinements. A relatively small estimate for the variance of a random effect suggests that it can be removed from the model. Similarly, relatively small estimates for covariances among certain random effects suggest that a full covariance matrix is unnecessary. Since random effects are unobserved, Ψ must be estimated indirectly. Specifying a diagonal or block-diagonal covariance pattern for Ψ can improve convergence and efficiency of the fitting algorithm.

Specifying an Error Model

You can specify error models with the `sbionlmefitsa` or `sbionlmefit` function.

To define an error model, use the `ErrorModel` option. Each model defines the error using a standard normal (Gaussian) variable *e*, the function value *f*, and one or two parameters, *a* and *b*. You can specify the following error models:

- constant: `y = f + a*e`
- proportional: `y = f + b*f*e`
- combined: `y = f + (a+b*f)*e`
- exponential: `y = f*exp(a*e)` or `log(y) = log(f) + a*e`

1 Use `ErrorModel` in `nlmefitsa` or `nlmefit`. Create a `struct` with the specified error model. For example:

```
options.ErrorModel = 'proportional';
```

2 (Optional) Specify starting values for parameters of the error model using `ErrorParameters`. For example:

```
options.ErrorParameters = 1;
```

Tip To specify a and b for `combined`, enter:

```
options.ErrorParameters = [1 1];
```

---

3 Specify the arguments for `sbionlmefitsa` or `sbionlmefit`, as shown in "Performing Population Fitting Using sbionlmefit or sbionlmefitsa" on page 4-72.

See also `nlmefitsa` or `nlmefit` in the Statistics Toolbox documentation.

Specifying Parameter Transformations

To specify parameter transformations, use the `ParamTransform` option in `sbionlinfit`, `sbionlmefit` and `sbionlmefitsa`. The `ParamTransform` option lets you specify either no transformation, or the `log`, `probit`, or `logit` transformation.

The underlying algorithm in `nlmefit` assumes that parameters follow a normal distribution. This assumption may not hold for biological parameters that are constrained to be positive, such as volume and clearance. You may specify a transformation function for the estimated parameters, so that the transformed parameters follow a normal distribution.

By default, the SimBiology fitting functions choose a log transform for all estimated parameters. Parameters that are constrained between the values `0` and `1`, like absorption fraction, can be transformed by the `probit` or `logit` transformations described below.

The `probit` function is the inverse cumulative distribution function (CDF) associated with the standard normal distribution. To apply the `probit` transform to a variable x in MATLAB, use the Statistics Toolbox function `norminv`: `t = norminv(x)`. To untransform a variable `t`, use the function `normcdf`: `x = normcdf(t)`.

The `logit` function is the inverse of the sigmoid function. To apply the `logit` transform to a variable x in MATLAB, use the following expression: `t = log(x) - log(1-x)`. To untransform the variable `t`, use `x = 1/(1+exp(-t))`.

1 For the `ParamTransform` option, specify a vector of values equal to the number of parameters to be estimated. The values must be one of the integer codes listed in `nlmefitsa` or `nlmefit` specifying the transformation for the corresponding value of the parameters to be estimated. For example

```
options.ParamTransform = [0 1 2];
```

See `nlmefit` and `nlmefitsa` for more information.

2 Specify the arguments for `sbionlmefit` or `sbionlmefitsa`, as shown in "Performing Population Fitting Using sbionlmefit or sbionlmefitsa" on page 4-72.

For individual fitting, see "Performing Individual Fitting Using sbionlinfit" on page 4-77.

Performing Population Fitting Using sbionlmefit or sbionlmefitsa

The `sbionlmefit` and `sbionlmefitsa` functions let you specify a SimBiology model that you want to use in fitting. These functions use the `nlmefit` and `nlmefitsa` functions from the Statistics Toolbox to fit data with both fixed and random sources of variation using nonlinear mixed-effects and return the estimates. `nlmefit` fits the model by maximizing an approximation to the marginal likelihood with random effects integrated out assuming the following:

• Random effects are multivariate, normally distributed, and independent between groups.

- Observation errors are independent, identically normally distributed, and independent of random effects.

1 (Optional) Set the tolerance and maximum iteration options. Use an options structure that is an argument for sbionlmefit:

```
options.Options.TolX = 1.0E-4;
options.Options.TolFun = 1.0E-4;
options.Options.MaxIter = 200;
```

2 Specify the model object, the PKModelMap object, the PKData object, the PKCovariateModel object, a vector containing the initial estimates for the fixed effects, and the options:

```
[results, simdataI, simdataP] = sbionlmefit(modelObj,...
 PKModelMapObj, pkDataObject, PKCovariateModelObject, beta0, options);
```

or, if options includes an error model, you must use sbionlmefitsa:

```
[results, simdataI, simdataP] = sbionlmefitsa(modelObj,...
 PKModelMapObj, pkDataObject, PKCovariateModelObject, beta0, options);
```

Note If your population fit uses multiple doses, make sure each element in the Dosed property of the PKModelMap object is unique.

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

- These data points are not averaged, but fitted individually.

- Different number of observations at different times cause some time points to be weighted more.

sbionlmefit and sbionlmefitsa return the following:

- A results structure containing:

- - `estimate` — Estimates for the fixed effects.

  - `phiP` — Estimated parameter values for the population

  - `phiI` — Estimated parameter values for individuals

  - `beta` — Estimates for the fixed effects in transformed space as specified in the `ParamTransform` option.

  - `psi` — Estimated covariance matrix of random effects

  - `stats` — Structure with statistics, such as `AIC` and `BIC` (For a complete list, see `nlmefit` and `nlmefitsa` in the Statistics Toolbox documentation.)

  - `b` — Estimated random effects for each group in *observedData*

- *simdataI* contains the data from simulating the model using the estimated parameter values for individuals, which includes both the fixed and random effects.

- *simdataP* contains the data from simulating the model using the estimated parameter values for the population, which includes only the fixed effects.

3 Plot the data from the data set. For example, in the imported data set used for fitting, `ds`, `ID`, `Time`, and `Response` are the column headers for the columns containing group IDs, time, and the response variable, respectively.

```
p = sbiotrellis(ds, 'ID', 'Time', 'Response')
```

Note If your data set has multiple responses, with column headers `Response1` and `Response2` containing the response variables, you plot the data as follows:

```
Response = {'Response1', 'Response2'}
p = sbiotrellis(ds, 'ID', 'Time', Response)
```

4 Use the `plot` method on the trellis plot object `p`, returned by `sbiotrellis` to overlay data, using default values for the second and third input arguments.

```
p.plot(simdataP, [], '', PKModelMapObj.Observed);
```

For a description of the results, see `sbionlmefit` in the SimBiology documentation.

For more information, see the following topics in the Statistics Toolbox documentation:

- "Nonlinear Regression Models"
- "Mixed-Effects Models"
- `nlmefit`

Obtaining the Status of Fitting

The `sbiofitstatusplot` function dynamically plots the progress of the fitting task. During the task, the function plots the fixed effects (ß), the estimates for the diagonal elements of the covariance matrix for the random effects (Ψ), and the log-likelihood. This functionality is useful for large and complex models when you expect the time to return the results to be longer than a few minutes. Use the options structure that is an argument for the `sbionlmefit` function:

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

The following figure shows the type of plots obtained.
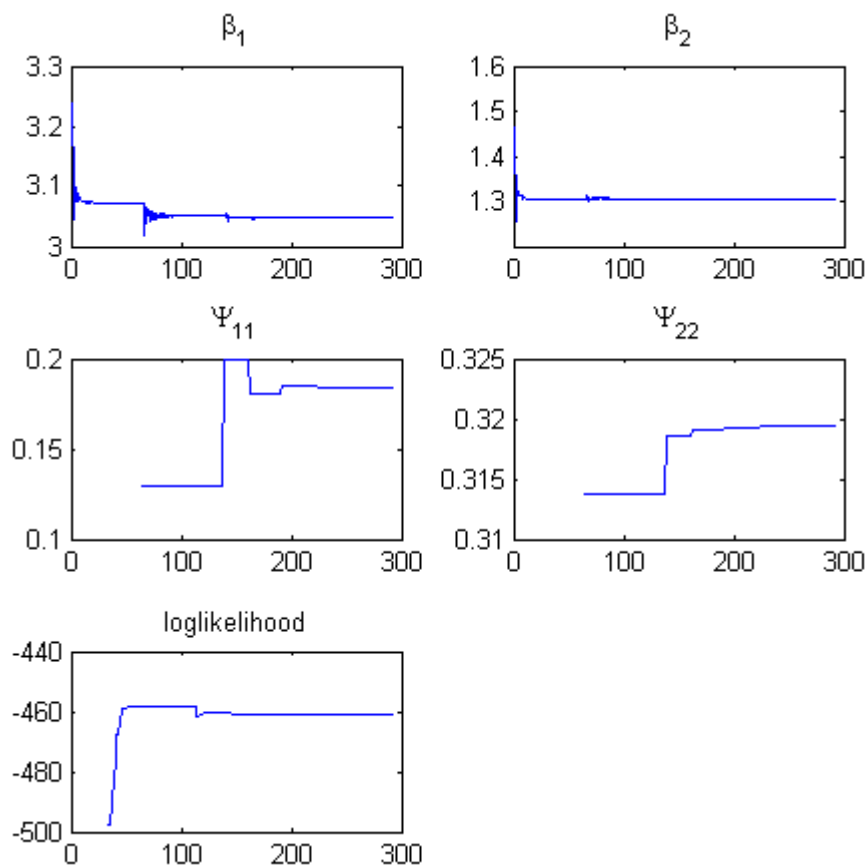
Tips for interpreting status plots:

- The fitting function tries to maximize the log-likelihood. When the plot begins to display a flat line, this might indicate that maximization is complete. Try setting the maximum iterations to a lower number to reduce the number of iterations you need and improve performance. For information on how to set iteration options, see "Performing Population Fitting Using sbionlmefit or sbionlmefitsa" on page 4-72.

- Plots for the fixed effects (ß) and the estimates for the diagonal elements of the covariance matrix for the random effects (Ψ), should show convergence.

If you see oscillations, or jumps without accompanying improvements in the log-likelihood, the model may be over-parameterized. Try the following:

- Reduce the number of fixed effects.

- Reduce the number of random effects.

- Simplify the covariance matrix pattern of random effects.

See also `sbiofitstatusplot` in the SimBiology documentation.

Performing Individual Fitting Using sbionlinfit

The `sbionlinfit` function lets you specify a SimBiology model to fit using the `nlinfit` function (individual fit). The `nlinfit` function fits using nonlinear least squares and returns parameter estimates, residuals, and the estimated coefficient covariance matrix.

1 (Optional) Set the tolerance and maximum iteration options:

```
options.TolX = 1.0E-8;
options.TolFun = 1.0E-8;
options.MaxIter = 100;
```

2 Specify the model object, the `PKModelMap` object, the `PKData` object, a vector containing the initial estimates for the fixed effects, and the options:

```
[results, simdataI] = sbionlinfit(modelobj,...
 PKModelMapObj, PKDataObj, beta0, options);
```

Note If your individual fit uses multiple doses, make sure each element in the `Dosed` property of the `PKModelMap object` is unique.

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

- These data points are not averaged, but fitted individually.

- Different number of observations at different times cause some time points to be weighted more.

`sbionlinfit` returns the following:

- A `results` array of structures, each containing the following for one group:

  - `beta` — Fitted coefficients

  - `R` — Residuals

  - `J` — Jacobian of `modelObject`

  - `COVB` — Estimated covariance matrix for the fitted coefficients

  - `mse` — Estimate of the error of the variance term

- *simdataI* contains the data from simulating the model using the estimated parameter values, for individuals.

3 Plot the data from the data set. For example, in the imported data set (`ds`), `ID`, `Time` and `Response` are the column headers for the columns containing group IDs, time, and the response variable respectively.

```
p = sbiotrellis(ds, 'ID', 'Time', 'Response')
```

Note If your data set has multiple responses, with column headers `Response1` and `Response2` containing the response variables, you plot the data as follows:

```
Response = {'Response1', 'Response2'}
p = sbiotrellis(ds, 'ID', 'Time', Response)
```

---

4 Use the plot method on the trellis plot object p, returned by sbiotrellis to overlay data, using default values for the second and third input arguments.

```
p.plot(simdataI, [], '', PKModelMapObj.Observed);
```

For more information, see "Nonlinear Regression Models" and nlinfit in the Statistics Toolbox documentation.

About Simulation Settings and Specifying Alternate Values for Initial Estimates

Use the Variant object to store and apply alternate values for model components during a simulation. For more information about variants and how to add variants see "Storing and Applying Alternate Model Values Using Variants" on page 1-59.

Note The fitting functions, population fit (NLMEFIT) and individual fit (NLINFIT), use the initial estimate values as the initial parameter estimates when fitting parameters (overriding the values in the variant). Specify initial estimate values, as shown in "Setting Initial Estimates" on page 4-65.

Use the Configset object to change settings for simulations. For more information about performing simulations, see sbiosimulate. The model object created using the PKModelDesign object's construct method contains a default configuration set that uses the sundials solver.

If you change the solver type in the configuration set during fitting, SimBiology temporarily changes the solver to sundials to support events in the model. SimBiology reverses the change after returning the results. If you want to change tolerance options for simulations select a deterministic solver and use the tolerance options provided in the deterministic solver. The fitting functions (sbionlmefit or sbionlinfit) will use the tolerance options specified in the deterministic solver.

Fitting Pharmacokinetic Model Parameters in the SimBiology Desktop

In this section...
"Fitting Parameters" on page 4-80
"Opening a SimBiology Project" on page 4-81
"Adding a Model Task to Fit Parameters" on page 4-81
"Performing Population Fitting" on page 4-81
"Performing Individual Fitting" on page 4-92
"About Simulation Settings and Specifying Alternate Values for Initial Estimates" on page 4-96
"Visualizing Parameter Fitting Results and Generating Diagnostic Plots" on page 4-97

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 4-81. To learn more about how to add data to a project and how to create a PK model, see "Importing Data into the SimBiology Desktop" on page 4-17 and "Creating PK Models in the SimBiology Desktop Using a Wizard" on page 4-46.

2 Add a task that lets you fit parameters, as shown in "Adding a Model Task to Fit Parameters" on page 4-81.

3 Choose the type of fitting:

- To perform population fitting, see "Performing Population Fitting" on page 4-81.

- To perform individual fitting, see "Performing Individual Fitting" on page 4-92.

4 (Optional) Specify alternate values for initial guesses of the parameter values and apply the values during parameter fitting. For more information on applying alternate values and to learn about constraints on simulation settings, see "About Simulation Settings and Specifying Alternate Values for Initial Estimates" on page 4-96.

5 Run the task and visualize results as shown in "Visualizing Parameter Fitting Results and Generating Diagnostic Plots" on page 4-97.

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 Model Task to *ModelName* > Fit parameters**. The **Project Explorer** updates with the **Parameter Fit** task and the **Parameter Fit** pane opens.

Performing Population Fitting

Specifying the Estimation Method, Parameters to Estimate, and Data to Fit

View the **Parameter Fit** pane as shown.
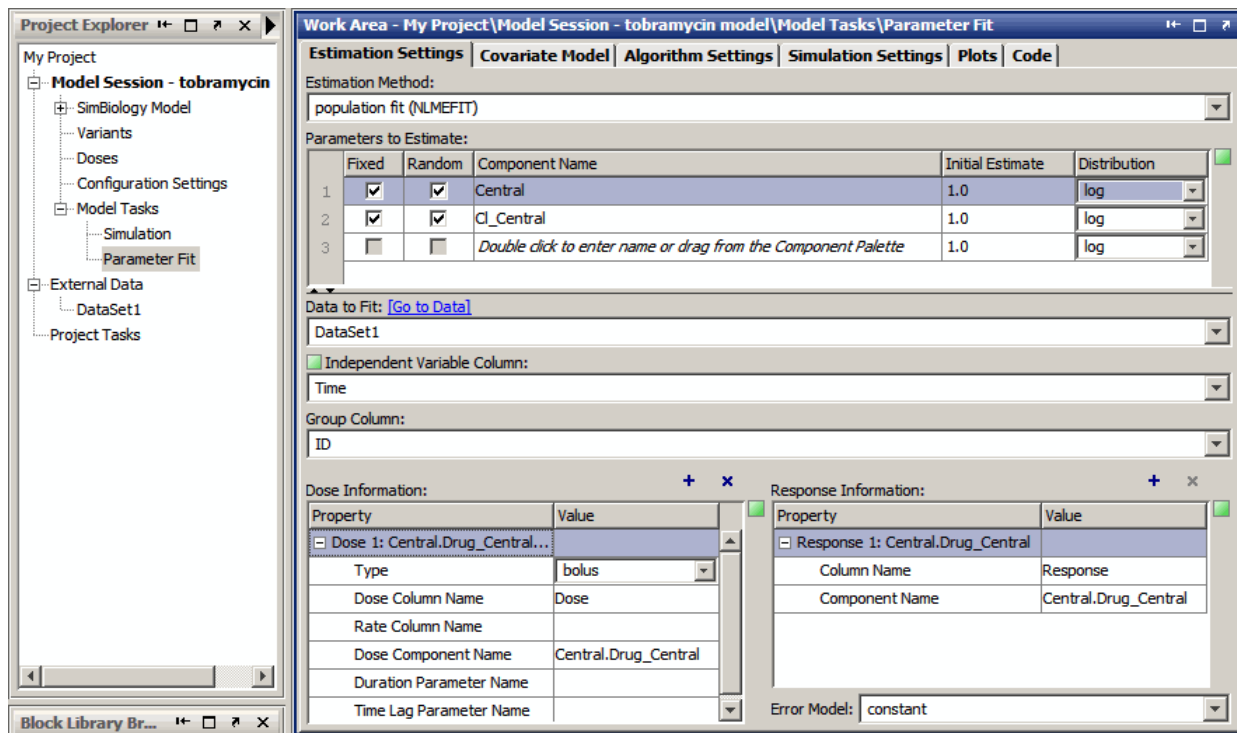
1. Click the **Estimation Settings** tab if needed.

2. From the **Estimation Method** list, select `population fit (NLMEFIT)` or `Stochastic EM fit (NLMEFITSA)`. These methods use the Statistics Toolbox functions, `nlmefit` and `nlmefitsa`, respectively. You can 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.

   If you are using the tobramycin data set [1], keep the default settings shown in the previous figure.

3. The **Parameters to Estimate** table shows the SimBiology components (parameters, species, and compartments) to estimate. To add components to this table, double-click a cell in the **Component Name** column and type the name of the component.

Tip Alternatively, double-click the **Component Name** cell, press the down arrow to display a list of available components, select a component from the list, and press **Enter**.

If the name entered in the table is invalid, a red indicator appears to the right of 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 Under **Distribution**, select the distribution for each parameter. The available options are:

- none
- log
- probit
- logit

7 From the **Data To Fit** list, select the data set to use in fitting.

8 From the **Independent Variable Column** list, select the column header in the data set that represents the independent variable (for example, time).

9 From the **Group Column** list, select the column header in the data set that represents the group column (for example, ID).

Note For the subset of data belonging to a single group (as defined in the data column you select from the **Group Column** list), the software allows multiple observations made at the same time. If this is true for your data, be aware that:

- These data points are not averaged, but fitted individually.

- Different number of observations at different times cause some time points to be weighted more.

10 In the **Dose Information** section, verify the information. To edit a dose, select a type from the **Type** drop-down list. For the remaining properties, double-click the cells in the **Value** column to edit them.

- **Dose Column Name** — Column header in the data set containing dosing information.

- **Rate Column Name** — Column header in the data set containing dose rate information. This information applies only to infusion dosing types, and is required by infusion dosing types.

- **Dose Component Name** — Species in the model that receives the dose. If your population fit uses multiple doses, make sure each dose has a unique **Dose Component Name**.

- **Duration Parameter Name** — Parameter in the model representing the time it takes to administer the dose. This information applies only to zero-order dosing types, and is required by zero-order dosing types.

- **Time Lag Parameter Name** — Parameter in the model representing the time lag associated with the dose (optional).

Tip After double-clicking a cell, press the down arrow key to display a list of possible values, select a name from the list, and then press **Enter**.

To add a dose, click ✚. To delete a dose, select the Dose row, and then click ✖.

11 In the **Response Information** section, verify the information. To edit a response, double-click the cells in the **Value** column to edit them.

- **Column Name** — Column header in the data set containing the observed response.

- **Component Name** — Component in the model representing the observed response.

Tip After double-clicking a cell, press the down arrow key to display a list of possible values to choose from, select a name from the list, and then press **Enter**.

To add a response, click ✚. To delete a response, select the Response row, and then click ✖.

12 From the **Error Model** list, select an error model and (optionally) specify a starting value. The software uses the error model you select for all responses in the model.

Each model defines the error for the response, *y*, using a standard normal (Gaussian) variable *e*, the function value *f*, and one or two parameters *a* and *b*. You can specify the following error models:

- constant: y = f + a*e

- proportional: y = f + b*f*e

- combined: y = f + (a+b*f)*e

- exponential: y = f*exp(a*e) or log(y) = log(f) + a*e

13 Run the parameter fitting task with the default settings by clicking ⏵ Run .
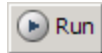
Specifying a Covariate Model in the SimBiology Desktop

A covariate model defines the dependencies between the estimated parameters and covariates. In defining your covariate model, you must make two basic choices:

- The type of model (linear, exponential, or power)

- The type of transformation

For more information about covariate analysis, see "Specifying a Covariate Model at the Command Line" on page 4-67.

1 Click the **Covariate Model** tab.

2 Under the heading **Covariate Column Name**, specify the names of your covariates. Double-click in the cell and type in the covariate name. Or, once you have begun typing, use auto-complete to select a covariate name from the drop-down list. If you specify a name that does not appear in the data set or that appears already in the list of covariate names, a red error indicator appears.

3 In the menus to the right of the **Covariate Column Name** heading, select appropriate covariate transformations. For **Center By**, you can choose none, mean, or median. For **Scale By**, you can choose none, max,

mean, median, or std. You can also type numeric values for either of these transformations.

Your covariate transformations appear in the section **Define the Covariate Model** unless you choose to hide them. The variable name for each transformation is the covariate name prefixed with "t". Covariates with warnings, unlike covariates with errors, also appear in this section.

4 Enter a MATLAB expression to the right of each parameter in the **Covariate Model** section. The expression describes the relationship between the parameter and the covariates. Enter any linear, exponential, or power MATLAB expression. Use either a covariate or a transformed covariate.

Tip If you prefer to enter your expression with the Covariate Model Editor, select **Editor** to the right of the expression box.

Avoiding Errors and Warning Messages. The following situations result in errors or warning messages. The indicator boxes in the SimBiology Desktop, which are usually green, change to yellow for warnings or red for errors. Move your cursor over the indicator boxes to read the error or warning messages.

Error or Warning	Problem Description	Result or Recovery Action
Error	You enter an invalid MATLAB expression, reference something other than a covariate or transformed covariate, or enter an expression that is not a supported model expression.	Enter a valid expression.
Warning	You enter an initial estimate in the covariate model that does not match the initial	The initial estimate you provide in the covariate model overrides the value from the

Error or Warning	Problem Description	Result or Recovery Action
	estimate in the **Parameters to Estimate** table.	**Parameters to Estimate** table.
Warning	You enter a covariate in the covariate model although you previously created a transformation of this covariate in Step 3.	The covariate you enter in the covariate model is used.
Warning	You specify a covariate model expression that is exponential or power and do not select **log** in the Distribution column of the **Parameters to Estimate** table.	A **log** value is used.

Specifying the Covariance Matrix and Error Models for Population Fits

- For both population fit (NLMEFIT) and Stochastic EM fit (NLMEFITSA):

    - Optionally specify the covariance matrix pattern of random effect parameters, and parameter transformations.

    - Optionally specify an error model.

- For population fit (NLMEFIT), select the method used to approximate the NLME model likelihood.

1 Click the **Algorithm Settings** tab.

2 In the **Covariance Matrix Pattern of Random Effect Parameters** section, do 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 following figure indicates covariance between the parameters ka_Central and Cl_Central.
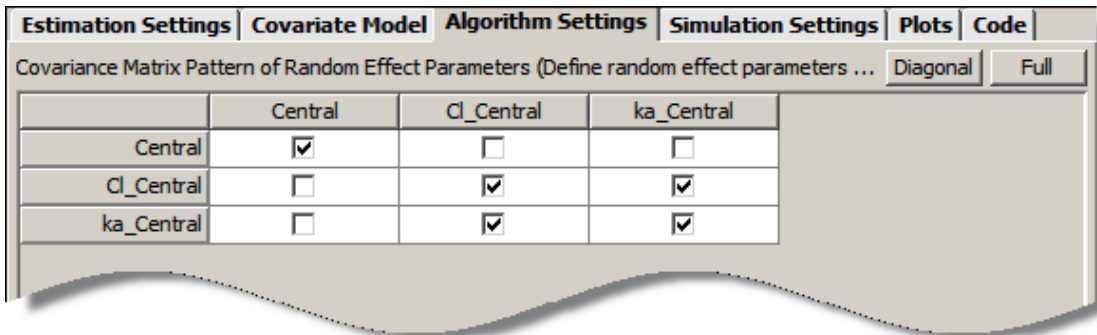
3 If you selected population fit (NLMEFIT) as your **Estimation Method**, in the **Estimation Settings** tab, select one of the following from the **Method to Approximate the Non-linear Mixed Effects Model Likelihood** list in the **Algorithm Settings** tab:

- '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.

- 'FOCE' — First-order Laplacian approximation at the conditional estimates of the random effects.

- 'FO' — First-order Laplacian approximation without 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 4-97.

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`.

Setting Iteration Options for Stochastic EM Fit (SAEM)

1 Click the **Algorithm Settings** tab.

2 Expand **Advanced Settings**.

3 From the **Optimization Function** list, select `fminsearch` (default) or `fminunc` (available if Optimization Toolbox is installed).

4 To configure the other options provided, see the `NBurnIn`, `NIterations`, `NMCMCIterations`, and `LogLikMethod` options for the `nlmefitsa` function, in the Statistics Toolbox documentation.

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 (ß), the estimates for the diagonal elements of the covariance matrix for the random effects (Ψ), and the log-likelihood. This functionality is useful for large and complex models when 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.
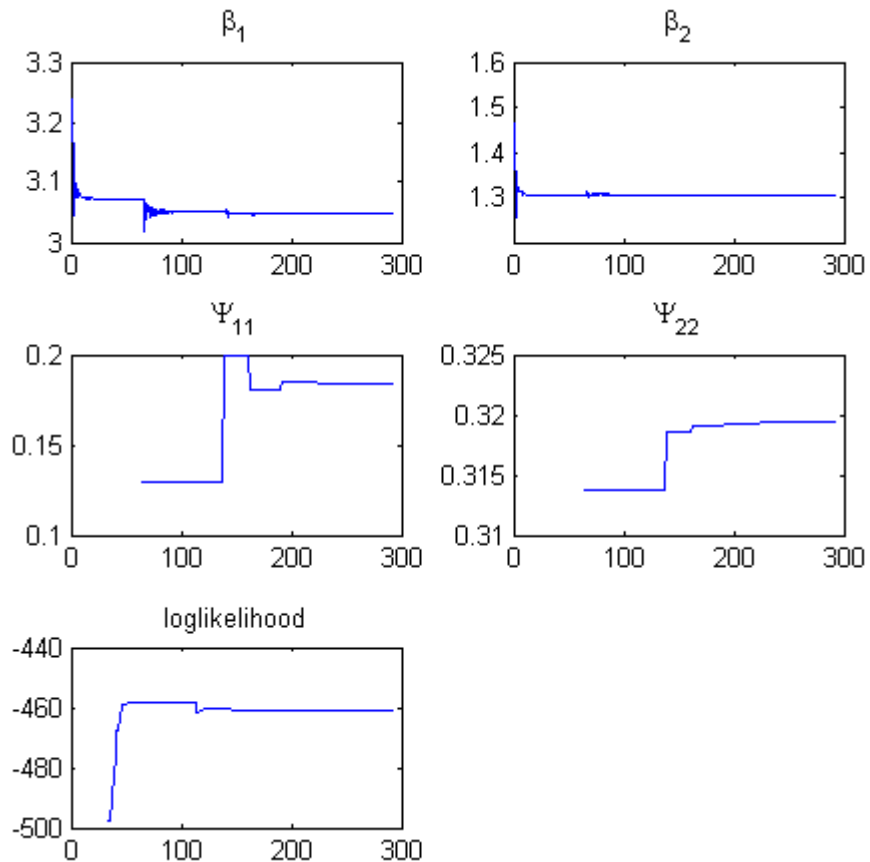
To turn the status reporting functionality on or off:

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 "Setting Tolerance and Maximum Iteration for Population Fits" on page 4-90, 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** pane as shown.

1 Click the **Estimation Settings** tab in the **Parameter Fit** pane if needed.

2 From the **Estimation Method** list, select `individual fit (NLINFIT)`, which uses the `nlinfit` function from the Statistics Toolbox.

3 The **Parameters to Estimate** table shows the SimBiology components (parameters, species, and compartments) to estimate. To add components to this table, double-click a cell in the **Component Name** column and type the name of the component.

Tip Alternatively, double-click the **Component Name** cell and press the down arrow to list available components. Select a component from the list and press **Enter**.

If the name entered in the table is invalid, a red indicator appears to the right of 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 Under **Distribution**, select the distribution for each parameter. The available options are:

- none
- log
- probit
- logit

7 From the **Data To Fit** list, select the data set to use in fitting.

8 From the **Independent Variable Column** list, select the column header in the data set that represents the independent variable, for example, time.

9 From the **Group Column** list, select the column header in the data set that represents the group column, for example, ID.

Note For the subset of data belonging to a single group (as defined in the data column you select from the **Group Column** list), the software allows multiple observations made at the same time. If this is true for your data, be aware that:

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

---

10 In the **Dose Information** section, verify the information. To edit a dose, select a type from the **Type** drop-down list. For the remaining properties, double-click the cells in the **Value** column to edit them.

- **Dose Column Name** — Column header in the data set containing dosing information.

- **Rate Column Name** — Column header in the data set containing dose rate information. This information applies only to infusion dosing types, and is required by infusion dosing types.

- **Dose Component Name** — Species in the model that receives the dose. If your individual fit uses multiple doses, make sure each dose has a unique **Dose Component Name**.

- **Duration Parameter Name** — Parameter in the model representing the time it takes to administer the dose. This information applies only to zero-order dosing types, and is required by zero-order dosing types.

- **Time Lag Parameter Name** — Parameter in the model representing the time lag associated with the dose (optional).

Tip After double-clicking a cell, press the down arrow key to display a list of possible values, select a name from the list, and then press **Enter**.

To add a dose, click ➕. To delete a dose, select the Dose row, and then click ✖.

11 In the **Response Information** section, verify the information. To edit a response, double-click the cells in the **Value** column to edit them.

- **Column Name** — Column header in the data set containing the observed response.

- **Component Name** — Component in the model representing the observed response.

Tip After double-clicking a cell, press the down arrow key to display a list of possible values to choose from, select a name from the list, and then press **Enter**.

To add a response, click ✚. To delete a response, select the Response row, and then click ✖.

12 Run the parameter fitting task with the default settings by clicking

▶ Run
.
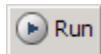
Setting Tolerance and Maximum Iteration for Individual Fits

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 1-59.

• 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), stochastic EM fit (NLMEFITSA), 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 (`sbionlmefit`, `sbionlmefitsa` 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
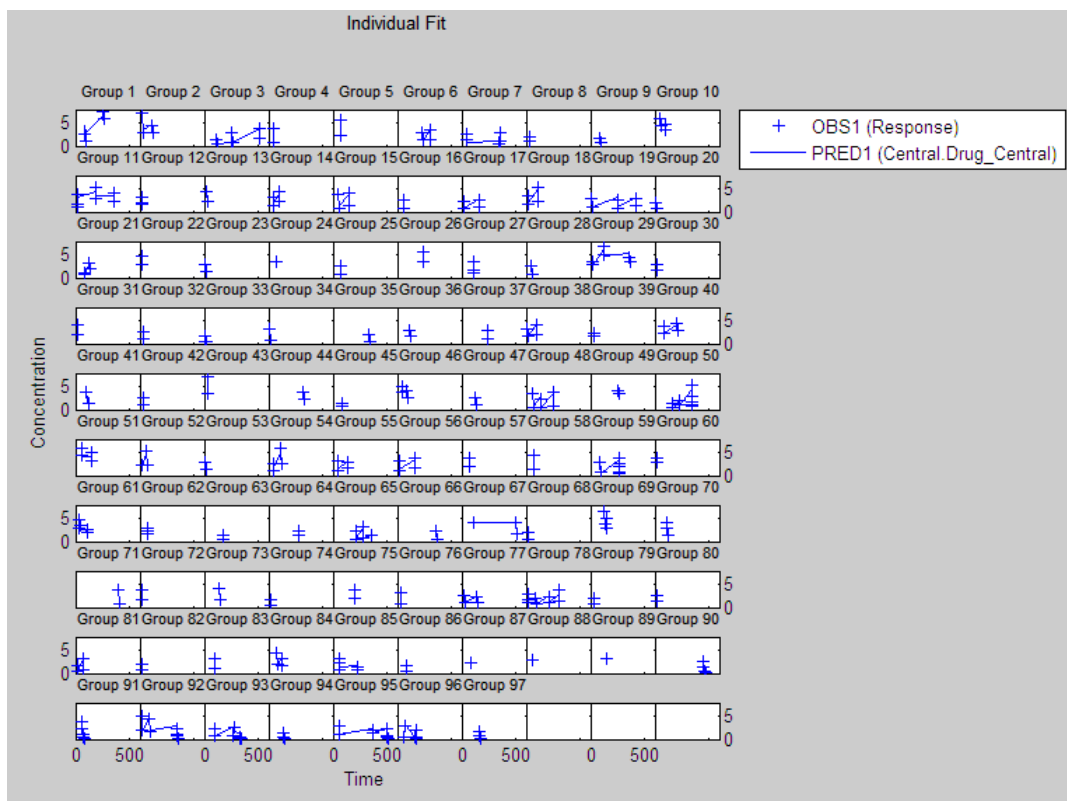
- Observed versus predicted values

Observation versus Prediction (Response1: Response)

- Residuals versus time, group or predictions

- Distribution of the Residuals

Population Residuals(Response1: Response)
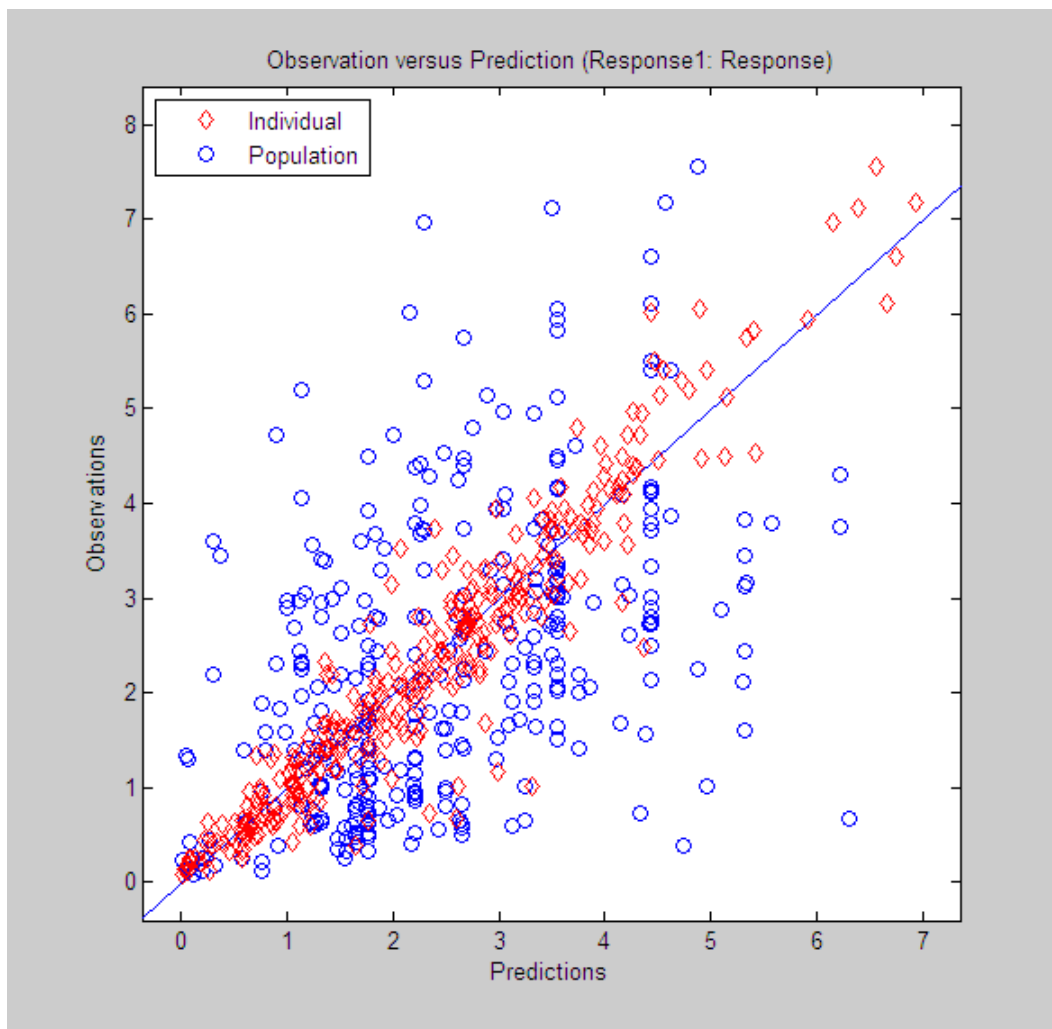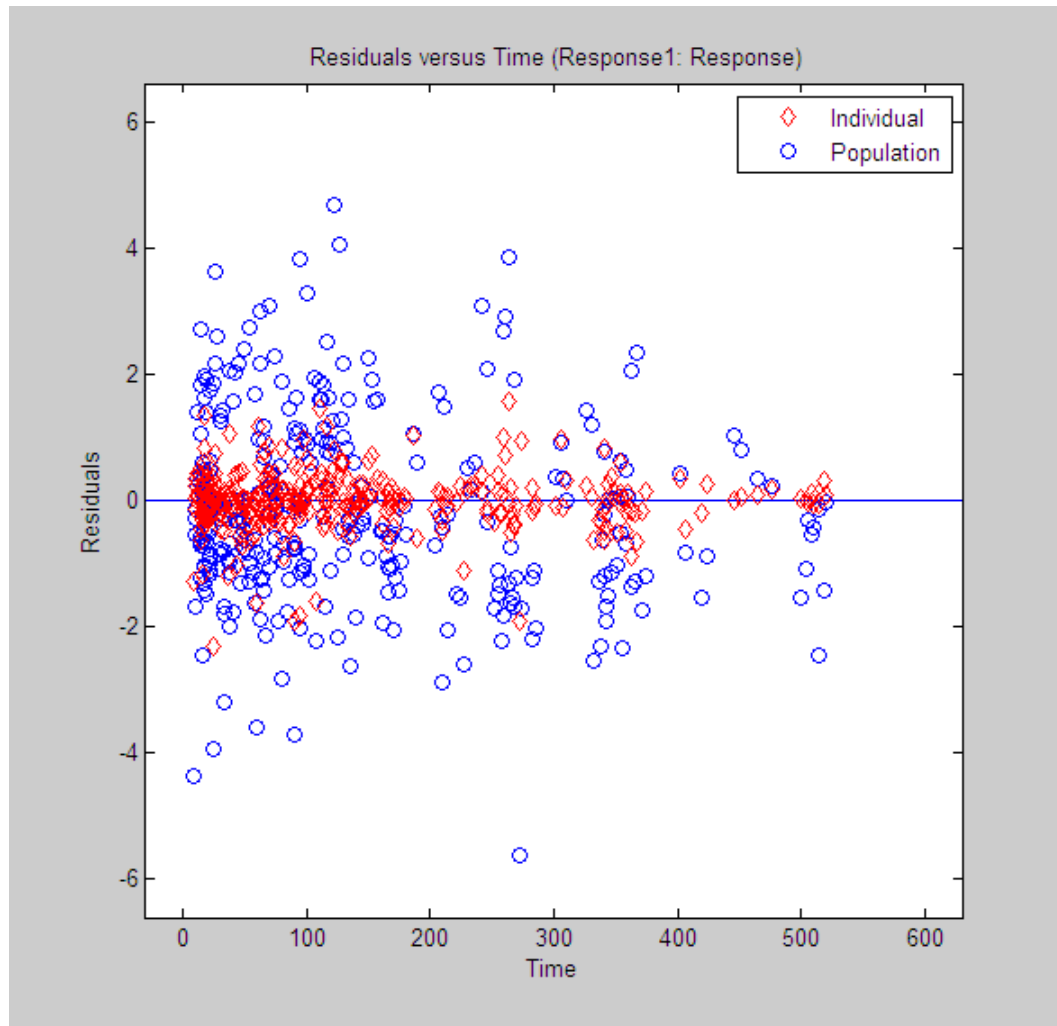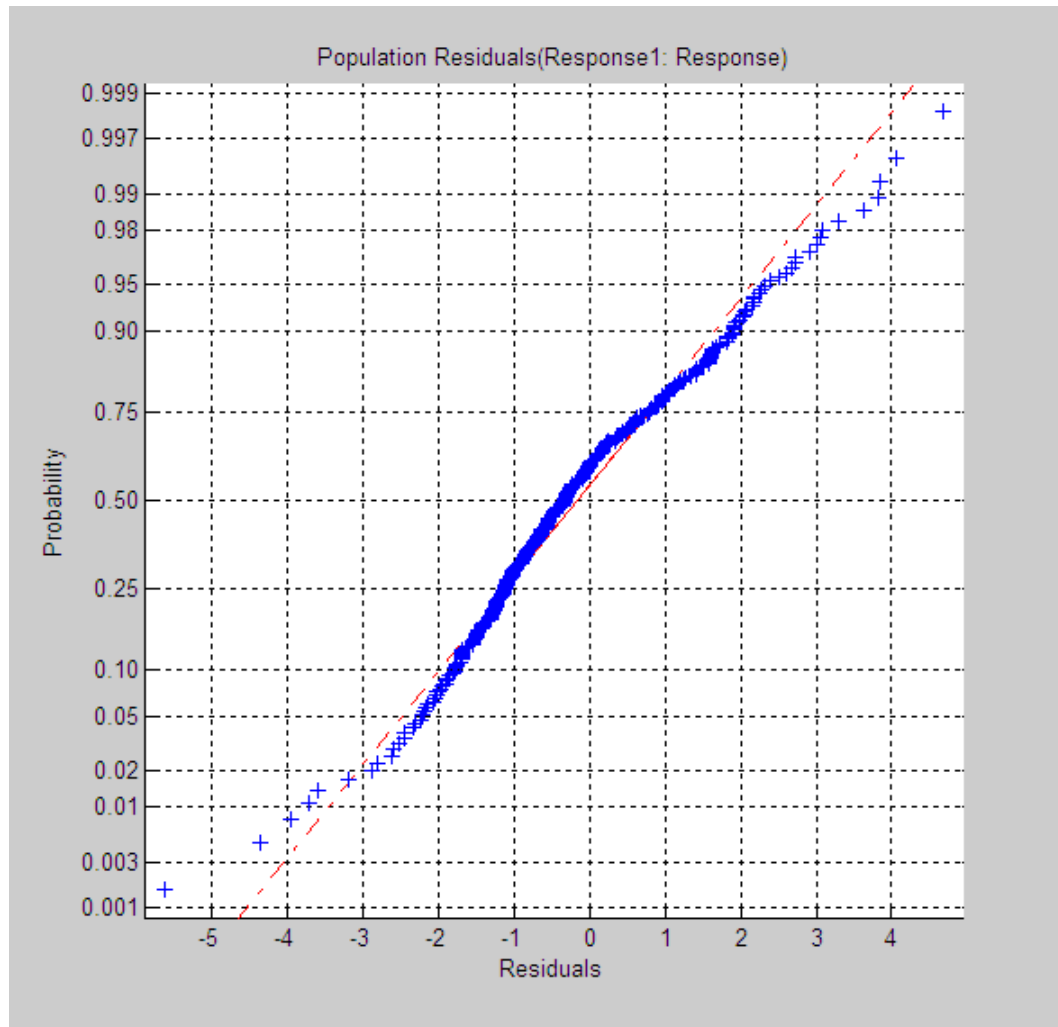
- A box-plot for random effects or parameter estimates from individual fitting
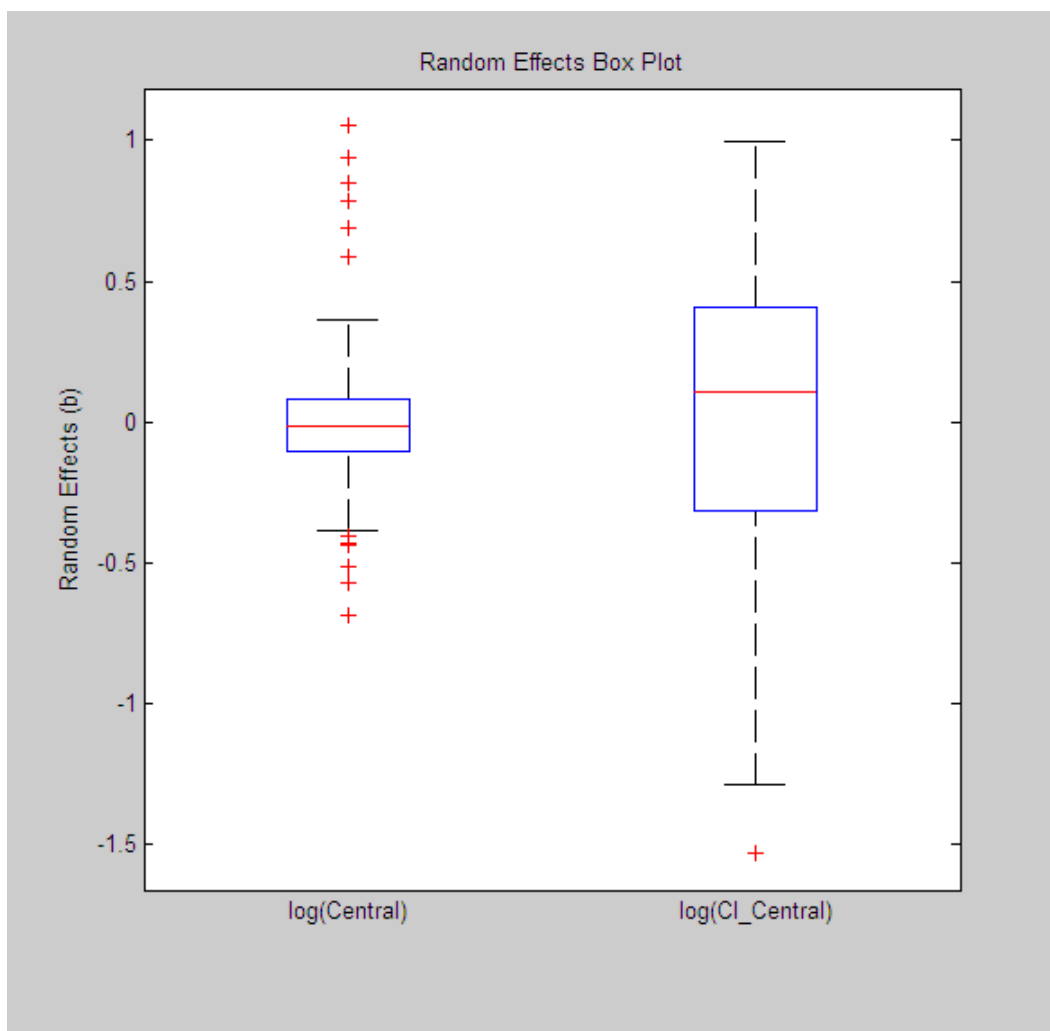
For help on working with plots and a description of each plot type, see the SimBiology context-sensitive help for the **Parameter Fit** pane.

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 3-67

Index