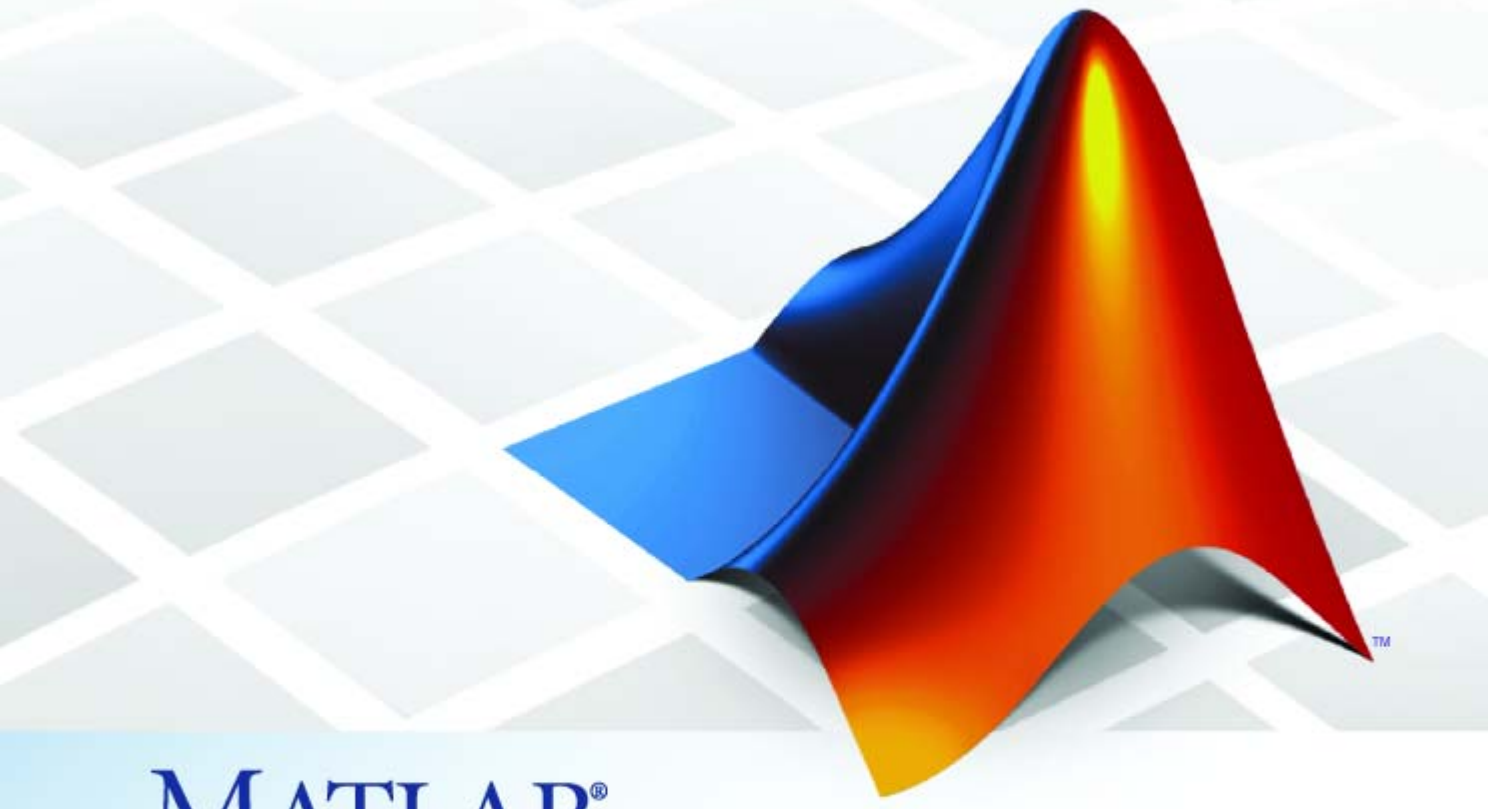
SimBiology® 2
User's Guide



MATLAB®

The MathWorks™
Accelerating the pace of engineering and science

How to Contact The MathWorks

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

`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

SimBiology® User's Guide

Trademarks

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

Patents

The MathWorks products are protected by one or more U.S. patents. Please see `www.mathworks.com/patents` for more information.

Revision History

September 2005	Online only	New for Version 1.0 (Release 14SP3+)
March 2006	Online only	Updated for Version 1.0.1 (Release 2006a)
May 2006	Online only	Updated for Version 2.0 (Release 2006a+)
September 2006	Online only	Updated for Version 2.0.1 (Release 2006b)
March 2007	Online only	Rereleased for Version 2.1.1 (Release 2007a)
September 2007	Online only	Rereleased for Version 2.1.2 (Release 2007b)
October 2007	Online only	Updated for Version 2.2 (Release 2007b+)
March 2008	Online only	Updated for Version 2.3 (Release 2008a)
October 2008	Online only	Updated for Version 2.4 (Release 2008b)

Contents

1 Modeling

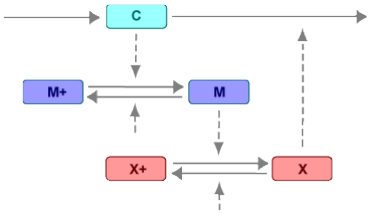
2 Simulation

3 Analysis

Index

Modeling

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



Defining Reaction Rates with Mass Action Kinetics

In this section...

Definition of Mass Action Kinetics

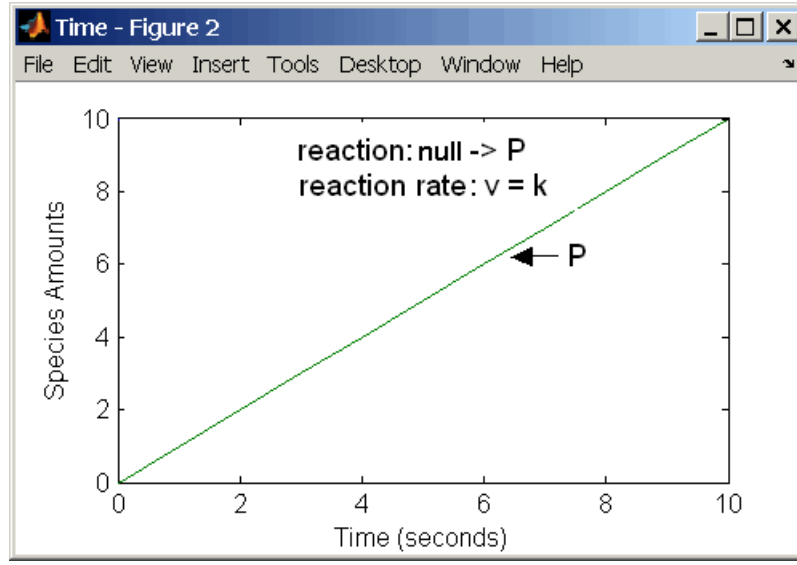
Mass action describes the behavior of reactants and products in an elementary chemical reaction. Mass action kinetics describes this behavior as an equation where the velocity or rate of a chemical reaction is directly proportional to the concentration of the reactants.

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

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



Zero-Order Mass Action Kinetics

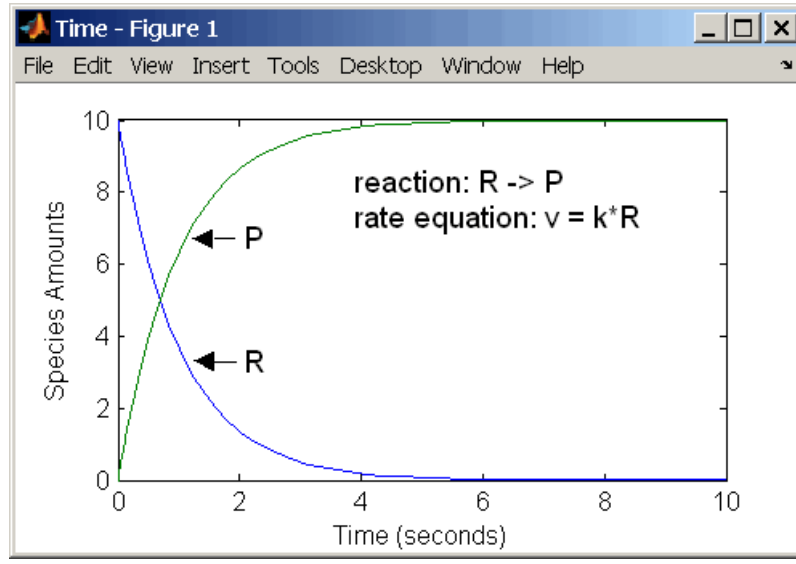
Note If the amount of a reactant with zero-order kinetics reaches zero before the end of a simulation, then the amount of reactant can go below zero regardless of the solver or tolerances you set.

First-Order Reactions

With a first-order reaction, the reaction rate is proportional to the concentration of a single reactant. An example of a first-order reaction is radioactive decay.

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

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



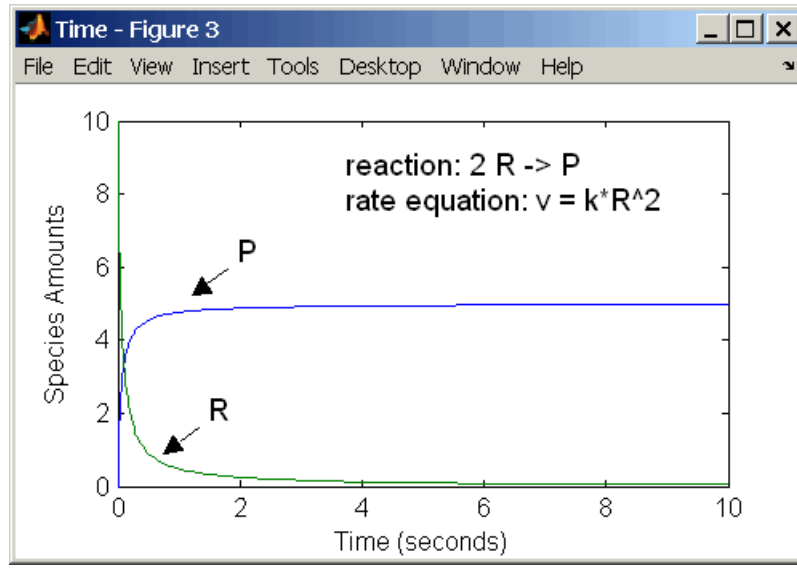
First-Order Mass Action Kinetics

Second-Order Reactions

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

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

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



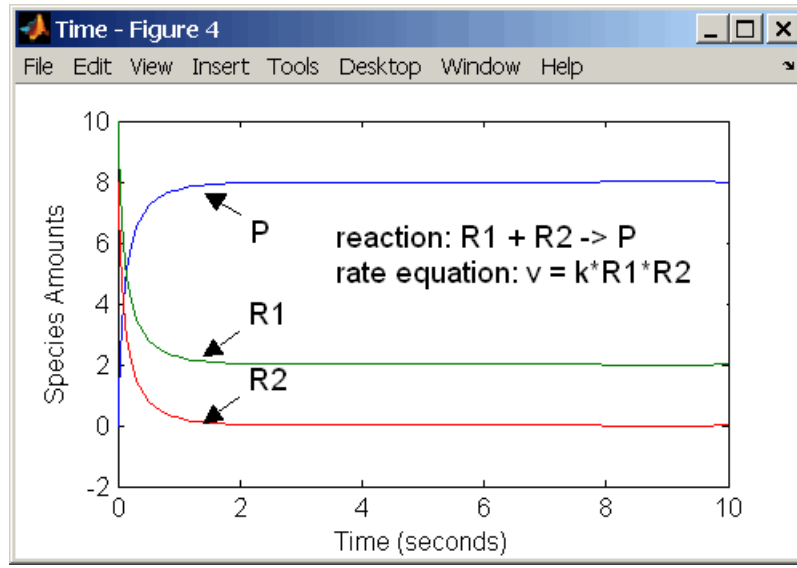
Second-Order Kinetics with Single Reactant

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

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

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

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



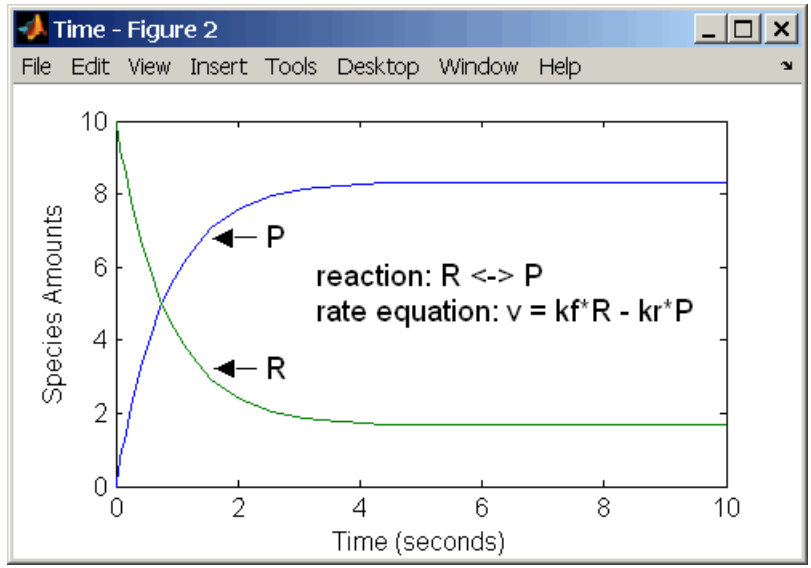
Second-Order Kinetics with Two Reactants

Reversible Mass Action

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

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

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



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

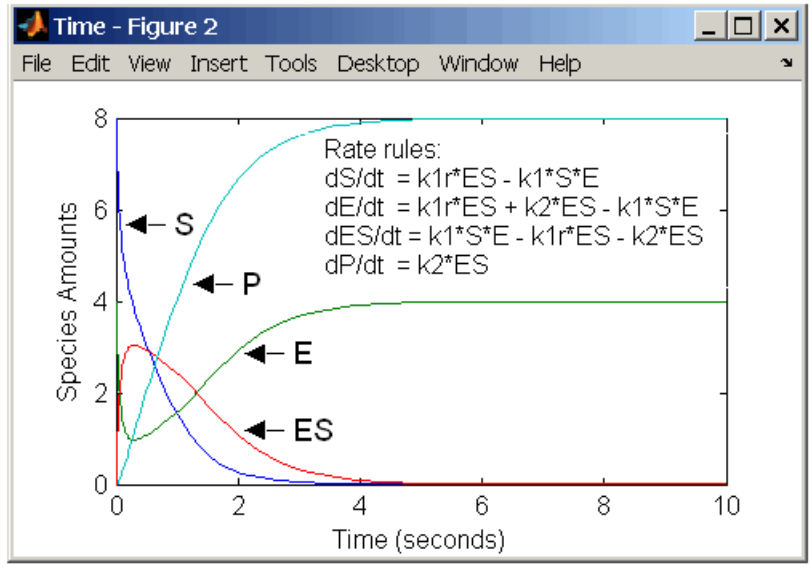
- Differential rate equations. See "Enzyme Reactions with Differential Rate Equations" on page 1-9.
- Reactions with mass action kinetics. See "Enzyme Reactions with Mass Action Kinetics" on page 1-11.
- Reactions with Henri-Michaelis-Menten kinetics. See "Enzyme Reactions with Irreversible Henri-Michaelis-Menten Kinetics" on page 1-12.

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 1-9) 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 to enter rate rules using the form `dS/dt = f(x)` as `S = f(x)`.



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
```

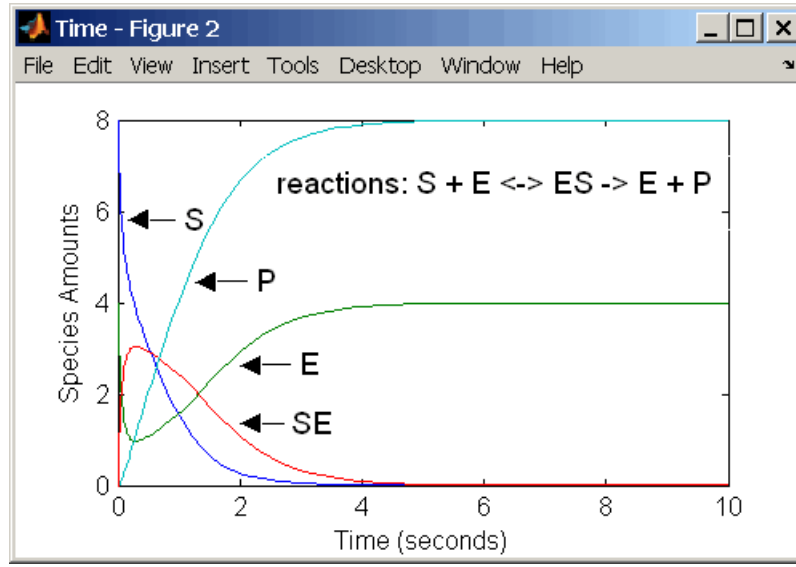
```
 reaction rate: none
    rate rules: dS/dt = k1r*ES - k1*S*E
                dE/dt = k1r*ES + k2*ES - k1*S*E
                dP/dt = k2*ES
algebraic rule: 0 = Etotal - E - ES
       species: S =  8   mole
                E =  4   mole
               ES =  0   mole
                P =  0   mole
           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 1-9.

```
     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 1-9.

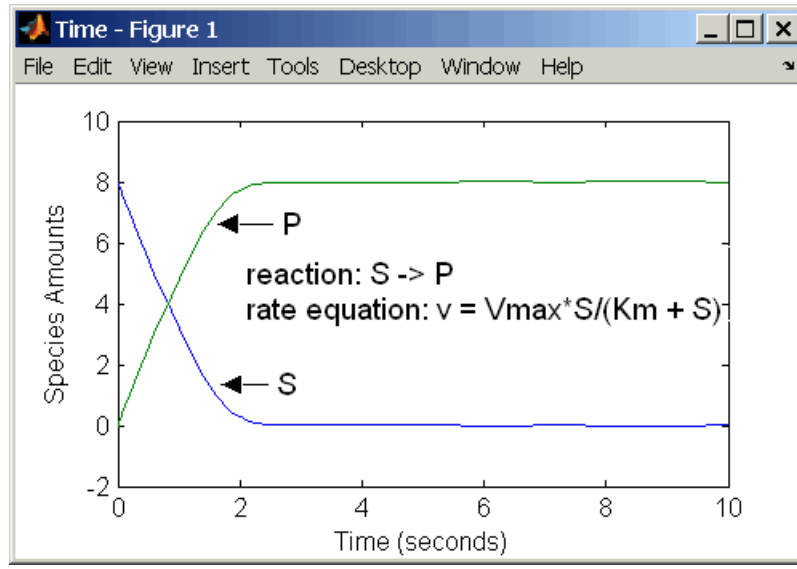
$$v = \frac{Vmax[S]}{Km + [S]}$$

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.



Using Constant Amount and Boundary Condition for Species

In this section...

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

In this section...

Definition of Parameter Scope

A *parameter* is a quantity that can change or can be constant. 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.

Therefore, 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 1-19 for more information.

To change the scope from kinetic law level to model level,

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. 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 searches for the parameter at the model level.

If you have already configured a reaction to use a parameter that is at the kinetic law level, change the scope to the model level by doing the following:

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

Changing Model Component Values Using Rules

In this section...

What Is a Rule?

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, you could use the rule to set the initial amount of `species1` to be proportional to `species2`.

```
species1 = k*species2
```

What Is a repeatedAssignment Rule?

`repeatedAssignment` rules are evaluated at every time step during a simulation. `repeatedAssignment` rules are expressed as `Variable = Expression`. For example, you could use the rule to specify the amount of `species1` to always be proportional to `species2` at all times during a simulation.

```
species1 = k*species2
```

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.

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.

Considerations When Imposing Constraints

Consider the mathematical constraint `y = mx - c`. In the software this rule is written as `mx - 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.

Mass Balance Equations

There are some models in the literature that are defined with differential rate equations and algebraic mass balance equations (system of differential algebraic equations).

A mass balance equation can define the amount of a species and reduce the number of differential rate equations that need to be solved. For example, a common signal transduction pathway can include a reaction `Ei -> Ea` where an enzyme transforms from an inactive form (`Ei`) to an active form (Ea). The amount of inactive enzyme `Ei` is defined by the differential rate equation `dEi/dt = Vm*Ei/(Km + Ei)`. If the total amount of enzyme is known, the amount of active enzyme `Ea` can be defined with the algebraic equation `Ea = Et - Ei` instead of a differential equation.

What Is a Rate Rule?

A rate rule is defined by the equation:

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

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

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

Typical Uses of Rate Rules

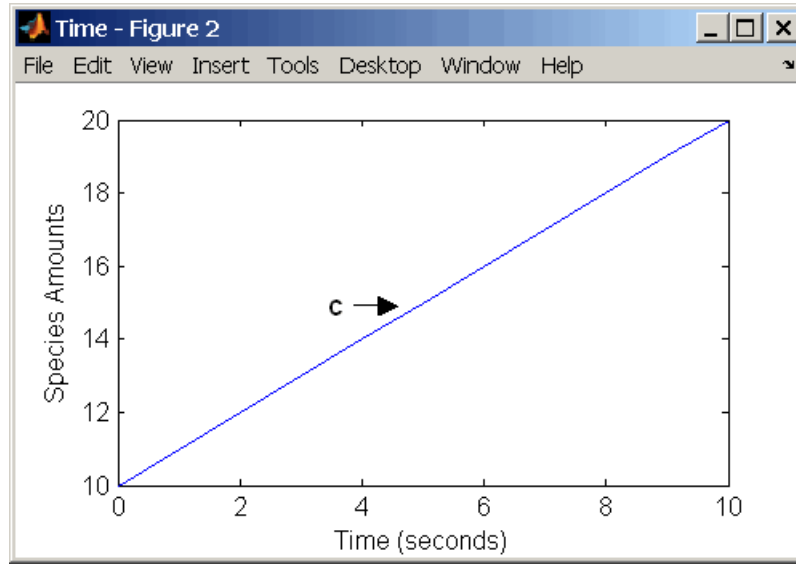
When the Rate of Change Is Constant

You can increase or decrease the amount or concentration of a species by a constant value using a zero order rate rule. For example, suppose species `c` increases by a constant rate `k`.

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

The analytical solution is $c = kt + c_o$, 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.



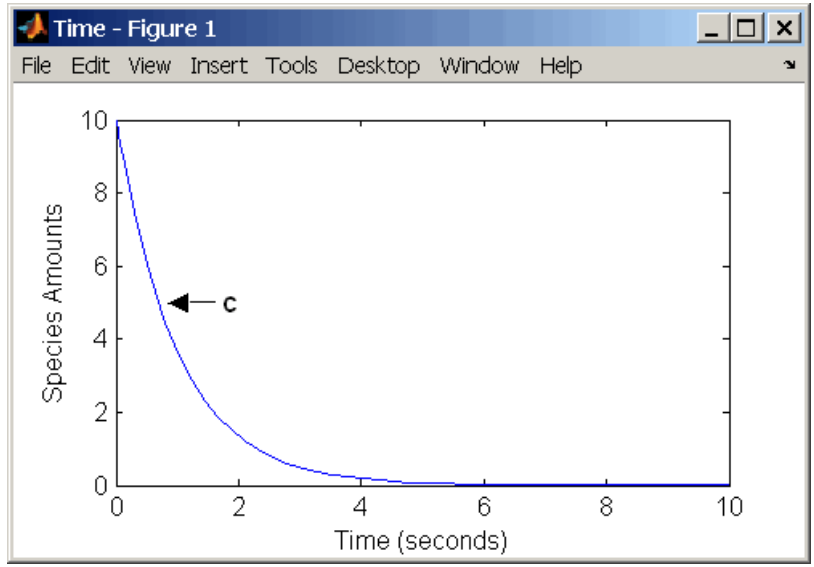
Alternatively, you could model a constant increase in a species using Mass Action reaction `null -> C`.

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
```

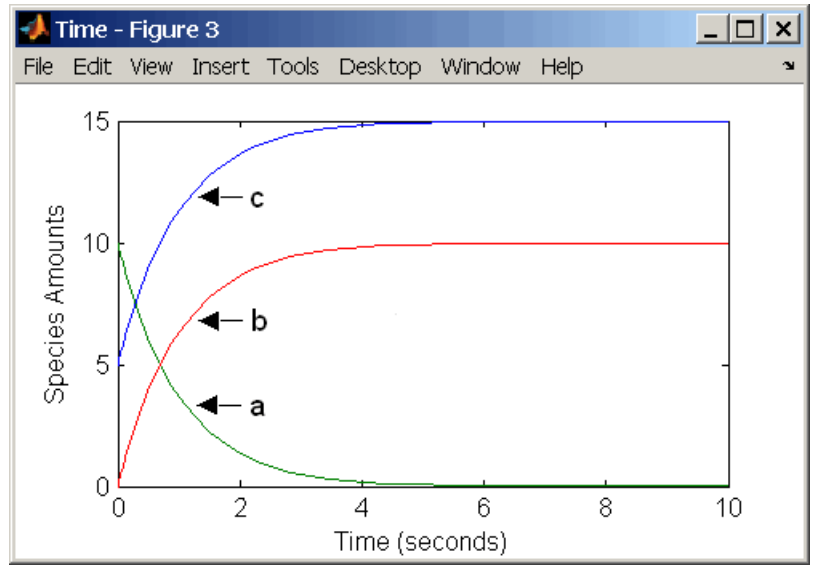
The solution for the species in the reaction are:

$a=a_0e^{-k1t}$ and $b=a_0(1-e^{-k1t})$

With the rate rule `dc/dt = k2*a` dependent on the reaction, `dc/dt = k2(a0e-k1t)`, and the solution is:

$c = c_o + k_2a_o/k_1(1 - e^{-k_1t})$

Enter the reaction and rule described above and simulate.



Expressing Differential Rate Equations as Rules

Many mathematical models in the literature are described with differential rate equations for the species. You could manually convert the equations to

reactions, or you could enter the equations as rate rules. For example, you could enter the following differential rate equation for a species `C`,

$$\frac{dC}{dt} = v_i - v_d X \frac{C}{K_c + C} - k_d C$$

as a rate rule in SimBiology:

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

Changing Model Component Values Using Events

In this section...

What Is an Event?

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

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

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

- Cause an event to occur at a specific time during simulation — Specify that the event must change the amounts or values of species or parameters. For example, at time = 5 s, increase the amount of an inhibitor species above the threshold to inhibit a given reaction.
- Cause an event to occur in response to state or changes in the system — Change amounts/values of certain species/parameters in response to events that are not tied to any specific time. For example, when species `A` reaches

an amount of 30 molecules, double the value of reaction rate constant `k`; or when temperature reaches 42 `C`, inhibit a particular reaction by setting its reaction rate to zero.

The event that is executed when the `Trigger` becomes true is called an event function (`EventFcn`). Event functions could range from simple to complex, for example, an event function might:

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

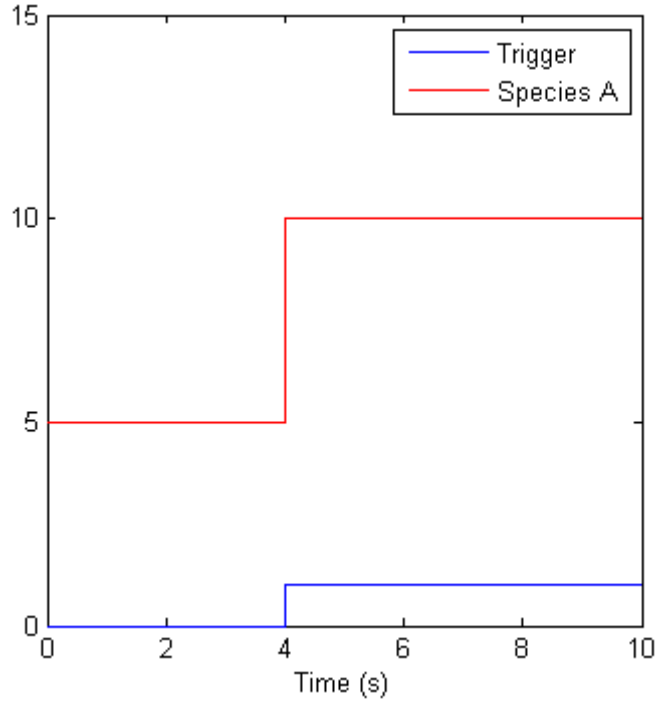
To simulate SimBiology models containing events, use the deterministic `sundials` solver or the stochastic `ssa` solver; other solvers do not support events. See "Sundials Solvers" on page 2-17 and "Stochastic Solvers" on page 2-12 for more information.

Procedures

- For an example using events in the SimBiology desktop, see "Desktop Example — Changing Species Amounts Using an Event" on page 1-38 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.

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.

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

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

Specifying Event Functions

The event that is executed when a `Trigger` condition has a rising edge is called an event function (`EventFcn`). You can use an event function to change

the value of a species or a parameter, or you can specify complex tasks by calling an M-file containing a user-defined function or script.

An event function is either a single valid MATLAB expression (without ';' in the expression) or a cell-array of single valid MATLAB expressions. For more information see also `EventFcns` in the SimBiology Reference Guide. Some examples of event functions include:

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

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'}

Desktop Example — Changing Species Amounts Using an Event

In this section...

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 1-29.

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*.

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`	`kRLm`, `kRL`
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" on page 3-62
- "Preparing to Modify the Example Model" on page 3-63

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:

   ```
   sbiodesktop(m1)
   ```

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

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

4. Specify a name (for example, `gprotein_ex`) and location for your project, and click **Save**.

Preparing to Modify the Example Model

Copy a model in the project and use the copy to work on this example.

1 In the **Project Explorer**, right-click **Model Session-Heterotrimeric_G_Protein_wt** and select **Copy Model and Add to Project**. The desktop adds the copied model to the **Project Explorer**.

2 Rename the copied model.

   a In the **Project Explorer**, right-click **Model Session** for the copied model and select **Rename Model**. The Rename Model dialog box opens.

   b Modify the name, for example, `gprotein`.

   c Click **Save**.

Adding an Event to the Example Model

1 In the **Project Explorer** expand **SimBiology Model** for the `gprotein` model and click **Events** to open the **Events** pane.

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

```
time >= 100
```

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

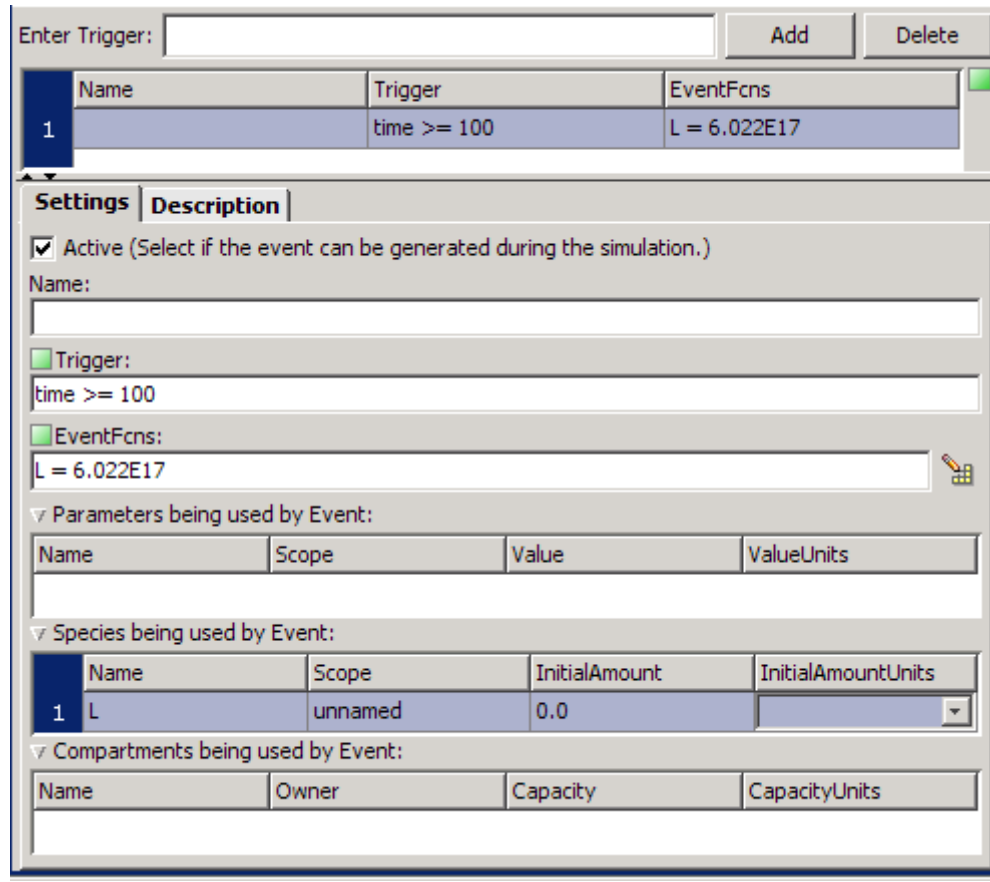
```
L = 6.022E17
```

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

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

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

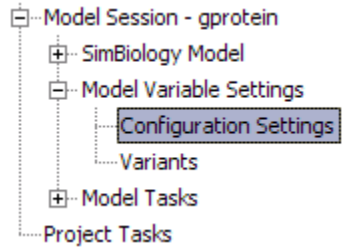
   The Events pane should now resemble the following:



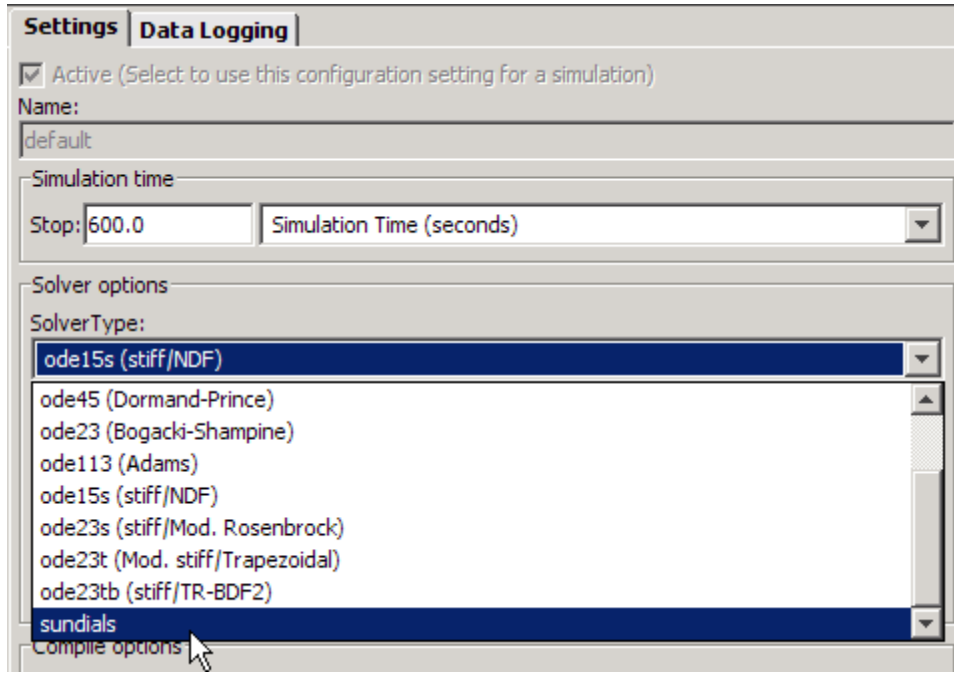
5 Save the project by selecting **File > Save Project**.

Simulating the Modified Model

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

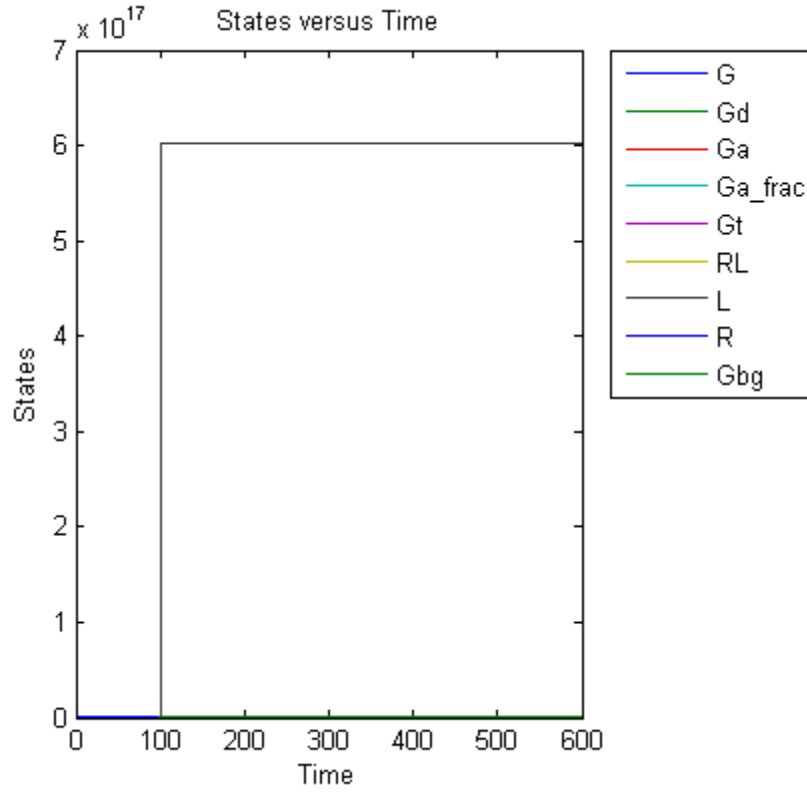


2 In the **Settings** tab, from the **SolverType** list, select `sundials`. This solver lets you simulate models with events.



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

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



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

4 In the **Project Explorer**, for the `gprotein` model, right-click **Data** and select **Save Data**. The Save Data dialog box opens.

5 Specify a name for the saved data, for example, `event_ex`, and click **Save**. The **Project Explorer** shows a new item with the saved data name under **Simulation**.

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

7 Click the **Plots** tab.

8 In the **Plot Type** box, select `Time` and click **Add Plot Type**.

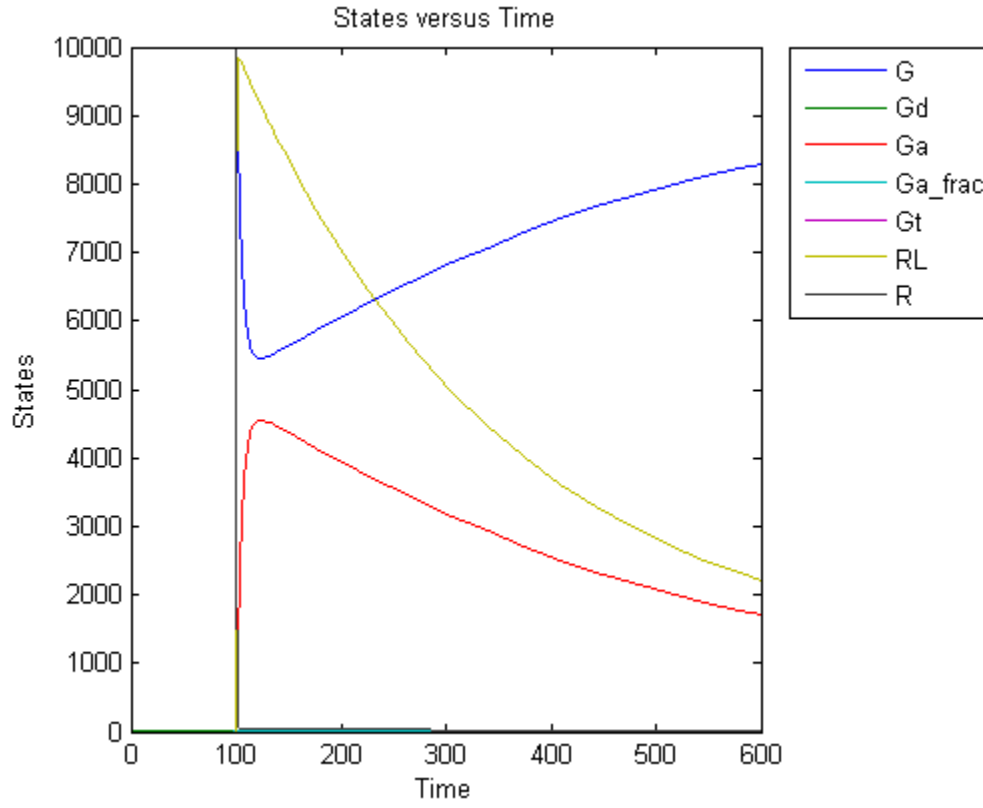
9 Select the new plot (second row), and in the **Arguments** section, click  . The Select Values for y dialog box opens.

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

11 Click **OK**.

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

13 Click **Plot**. Your plot should resemble the one below. Notice the increase in activation of G protein (species `Ga`, shown in red) after ligand (L) is added at `time = 100` (simulation time).


States versus Time
States
Time
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0
100
200
300
400
500
600
G
Gd
Ga
Ga_frac
Gt
RL
R

Storing and Applying Alternate Model Values Using Variants

In this section...

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** expand **Model Variable Settings** and select **Variants**.

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

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

- "Desktop Example — Applying Changes to Parameter Value Using a Variant" on page 1-49

Applying Multiple Variants in a Model

When multiple variants are used during a simulation, and there are duplicate specifications for a property's value, the last occurrence for the property value in the array of variants is used during simulation. You can find out which variant is applied last by looking at the indices of the variant objects stored on the model (at the command line) or the last variant created.

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

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

Desktop Example — Applying Changes to Parameter Value Using a Variant

In this section...

Overview

For a description of variants, see "Storing and Applying Alternate Model Values Using Variants" on page 1-46.

About the Example Model

This example uses the model from "Modeling a G Protein Cycle" in the *SimBiology Model Reference*.

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`	`kRLm`, `kRL`
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

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

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

Prerequisites

Opening and Saving the Example Model

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

   ```
   sbioloadproject gprotein
   ```

   The model is stored in a variable called m1.

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

   ```
   sbiodesktop(m1)
   ```

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

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

4 Specify a name (for example, `gprotein_ex`) and location for your project, and click **Save**.

Preparing to Modify the Example Model

Copy a model in the project and use the copy to work on this example.

1 In the **Project Explorer**, right-click **Model Session-Heterotrimeric_G_Protein_wt** and select **Copy Model and Add to Project**. The desktop adds the copied model to the **Project Explorer**.

2 Rename the copied model.

   a In the **Project Explorer**, right-click **Model Session** for the copied model and select **Rename Model**. The Rename Model dialog box opens.

   b Modify the name, for example, `gprotein`.

   c Click **Save**.

Applying Alternate Values Using Variants

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

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

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

```
mut_value_ex
```

3 Add content to the variant:

   a In the **Settings** tab, click  (Add table row). The table updates with a row containing information about a model component.

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

   c In the **Name** cell, type the name of the component.

```
Gprotein Inactivation.kGd
```

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

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

```
0.004
```

See Also

"Storing and Applying Alternate Model Values Using Variants" on page 1-46 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 — gprotein**, under **Model Tasks**, select **Simulation** to open the pane.

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

2. Click the **Plot Results** tab.

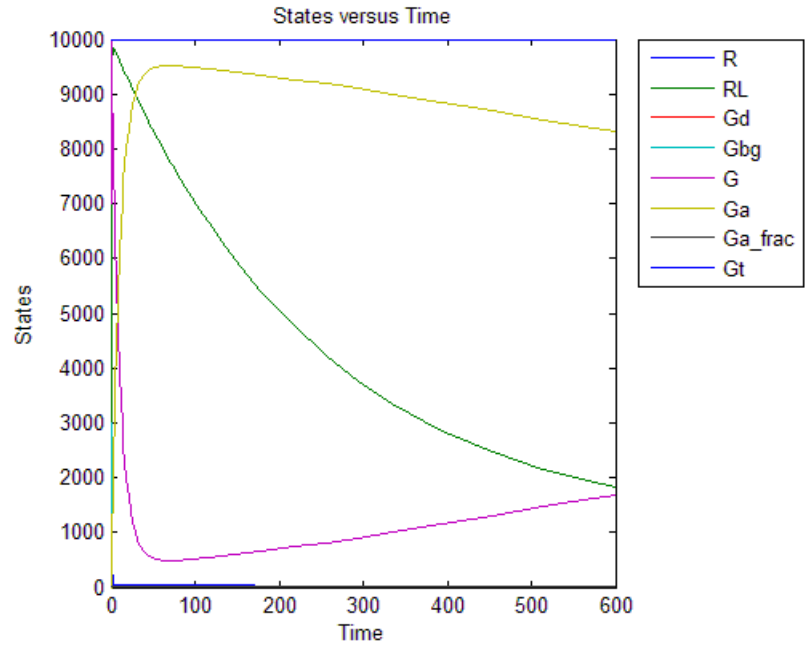
3. Under **Arguments**, click  . The Select Values for y dialog box opens.

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

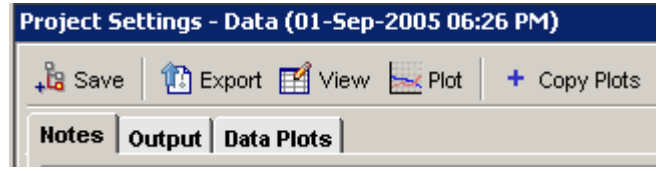
5. Click the **Simulation Settings** tab.

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

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



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



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

```
mut_model_run1
```

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

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

Verifying that the Model Has No Warnings or Errors

In this section...

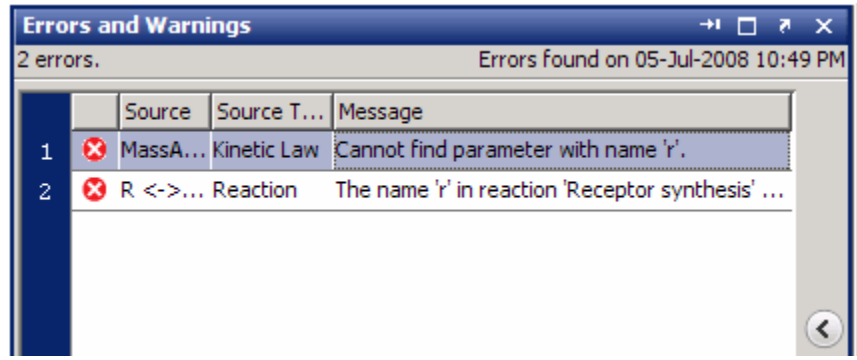
The SimBiology desktop and MATLAB command line have functionality that helps you find and fix warnings that you might need to be aware of, and errors that would prevent you from simulating and analyzing your model.

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.

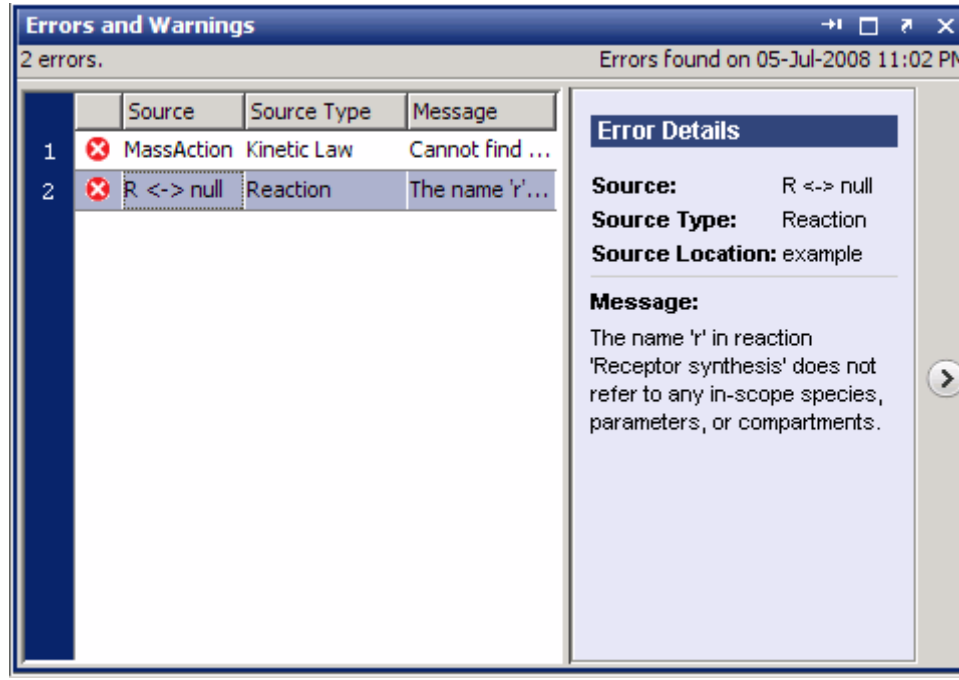
Verifying the Model in the SimBiology Desktop

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

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



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



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

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

Troubleshooting Using Colored Indicators

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

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

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

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

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

Verifying the Model at the Command Line

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

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

Desktop Example — Using User-Defined Functions in Expressions

In this section...

Prerequisites

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

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

Overview

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

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

Requirements For Using User-Defined Functions in SimBiology Expressions

- Create an M-file function. To find out more about M-file functions, see `function` in *MATLAB Function Reference*. For an example of a function to be used in a SimBiology model, see "Creating an M-File Function" on page 1-59 in this topic.

- Change the working directory to the directory containing your M-file using the `cd` command or using the Desktop as shown in in *MATLAB Desktop Tools and Development Environment*. Alternatively, add the path to the directory containing your M-file using `addpath` or using the GUI as shown in "Overview of Viewing and Changing the Search Path" in *MATLAB Desktop Tools and Development Environment*.
- 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*.

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`	`kRLm`, `kRL`
2	Heterotrimeric G protein formation	`Gd + Gbg -> G`	`kG1`

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

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 an M-File Function

1. In the MATLAB desktop, select **File > New > M-File**, to open a new M-file in the MATLAB Editor.
2. Copy and paste the following function declaration:

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

% This function takes the input arguments t, Cpo, and kel
% and returns the value of the inhibitor Cp.
% You can define the input arguments in a
% SimBiology rule expression.
% For example in the rule, define:
% t as time (a keyword recognized as simulation time),
% Cpo as a parameter that represents the initial amount of inhibitor a
% kel as a parameter that governs the amount of inhibitor.
```

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

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

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

Calling the Function in a Rule Expression

Opening and Saving the Example Model

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:

```
sbiodesktop(m1)
```

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

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

4 Specify a name (for example, `gprotein_ex`) and location for your project, and click **Save**.

Preparing to Modify the Example Model

Copy a model in the project and use the copy to work on this example.

1 In the **Project Explorer**, right-click **Model Session-Heterotrimeric_G_Protein_wt** and select **Copy Model and Add to Project**. The desktop adds the copied model to the **Project Explorer**.

2 Rename the copied model.

   a In the **Project Explorer**, right-click **Model Session** for the copied model and select **Rename Model**. The Rename Model dialog box opens.

   b Modify the name, for example, `gprotein`.

   c Click **Save**.

Adding User-Defined Functions to the Example Model

The previously defined function `inhibvalex` in "Creating an M-File Function" on page 1-59 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** for the `gprotein` model and click **Rules** to open the **Rules** pane.

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

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

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

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

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

Leave `inhibvalex` undefined in the Rule Variables dialog box and click **OK**. In the MATLAB desktop, change the **Current Directory** field to the location of the M-file and you can now safely ignore the error indicator for the rule in the SimBiology desktop.

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 3-35 , 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** for the `gprotein` 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** for the `gprotein` model 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 the `gprotein` 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**, for the `gprotein` model, 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'.

---

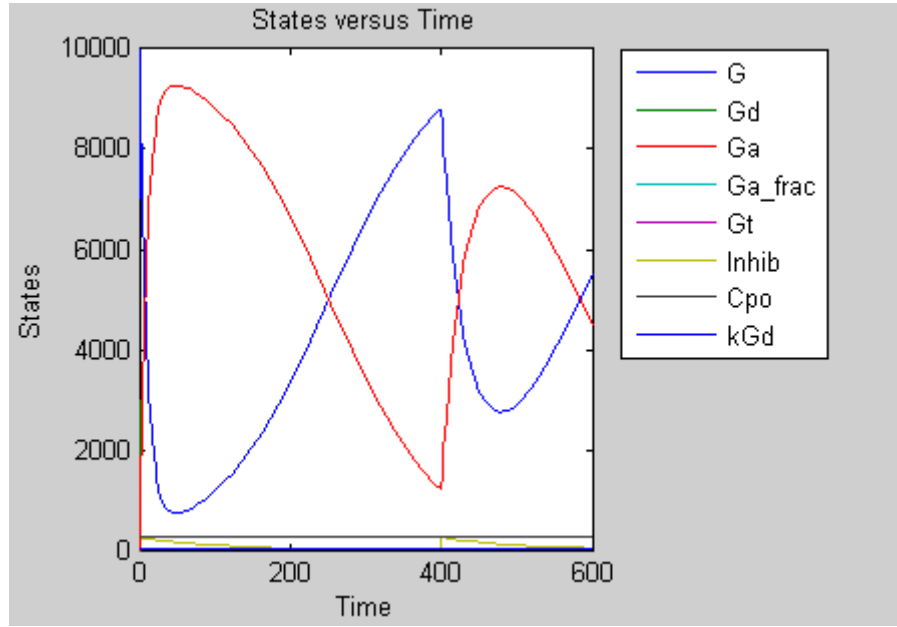
6 In the **Plot Type** box, select `Time` and click **Add Plot Type**.

7 Select the row containing the new plot, and then in the **Arguments** section, click  . The Select Values for y dialog box opens.

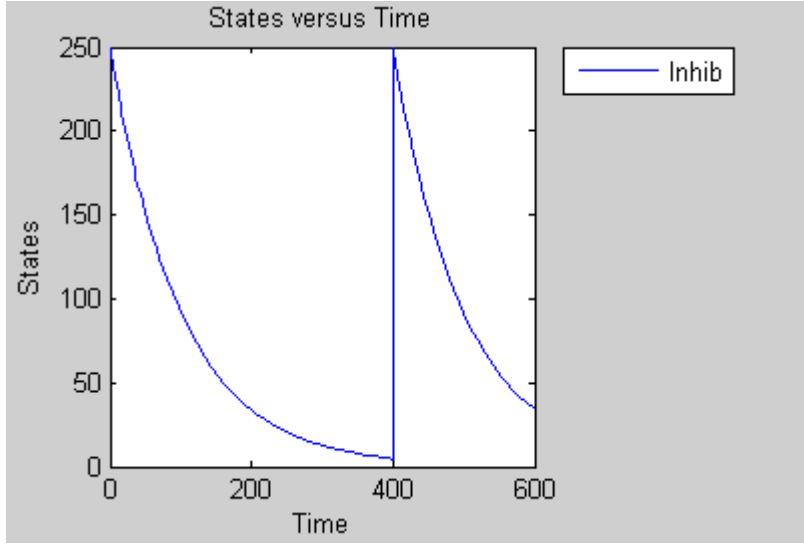
8 Click **Clear All**, and then select the check box for the species `Inhib`.

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*.
M-file functions	`function` in the *MATLAB Function Reference*.
Changing the working directory to the directory containing your M-file	`cd` command in the *MATLAB Function Reference* or the in *MATLAB Desktop Tools and Development Environment*.
Adding the directory containing your M-files to the MATLAB search path	`addpath` in the *MATLAB Function Reference* or "Overview of Viewing and Changing 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 Data from File** menu item. The Excel® option is only supported on the Windows® platform.

1. From the **File** menu, point to **Load Data 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 Data to File** menu item. The Excel option is only supported on the Windows platform.

1 From the **File** menu, point to **Export Data 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

In this section...

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 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

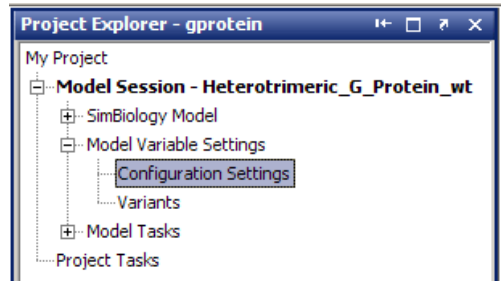
A `Configset object` (configuration set object) contains a configuration set. You can manipulate the object's properties to change simulation settings. You can add many configuration sets to a model, each containing a different combination of simulation settings. To use a configuration set during a simulation you can either set the object's `Active` property to true, or specify the object as an argument to `sbiosimulate`.

Performing Simulations in the SimBiology Desktop

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

To access your configuration settings:

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



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

Where to Find Configuration Settings Controls

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

- Use the **Settings** tab to set the simulation solver and timing options for the currently selected model, and compile options for unit checking.

  The toolbar specifies the active configuration set, which is used when simulating from the **Diagram** or **Analysis** menu.

- 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 the **Simulation** task pane:

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

Note Use the **Run** button at the top of the **Simulation** task pane to run the simulation with your currently selected Configuration Settings.

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

- Use the **Export Results** tab to export simulation data to the MATLAB Workspace and/or to file every time you run a simulation.
- Use the **Plot Results** tab to configure what plots to generate when you run a simulation.

About Simulation Solvers

In this section...

How Solvers Work

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

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

The solver functions implement numerical integration methods for solving initial value problems for ordinary differential equations (ODEs). Beginning at the initial time with initial conditions, they step through the time interval, computing a solution at each time step. If the solution for a time step satisfies the solver's error tolerance criteria, it is a successful step. Otherwise, it is a failed attempt; the solver shrinks the step size and tries again.

Stiff Versus Nonstiff Models

An ordinary differential equation problem is stiff if the solution being sought is varying slowly, but there are nearby solutions that vary rapidly, so the numerical method must take small steps to obtain satisfactory results. The ODE solvers in MATLAB whose name ends in "s" are for "stiff" problems. 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. If you don't care how much time a computation takes, you need not be concerned about stiffness. Nonstiff methods can solve stiff problems; they just take a long time to do it.

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

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

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

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

Selecting a Solver

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

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

   Use “Nonstiff Deterministic Solvers” on page 2-8.

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

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

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

Use "Stiff Deterministic Solvers" on page 2-10.

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

3 Models with a small number of molecules:

Use "Stochastic Solvers" on page 2-12.

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

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

Nonstiff Deterministic Solvers

In this section...

When to Use Nonstiff Deterministic Solvers

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

ode45 (Dormand-Prince)

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

ode23 (Bogacki-Shampine)

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

ode113 (Adams)

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

See Also

"ODEs" in *MATLAB Mathematics*.

Stiff Deterministic Solvers

In this section...

When to Use Stiff Deterministic Solvers

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

ode15s (stiff/NDF)

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

ode23s (stiff/Mod. Rosenbrock)

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

ode23t (Mode. stiff/Trapezoidal)

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

ode23tb (stiff/TR-BDF2)

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

See Also

"ODEs" in *MATLAB Mathematics*.

Stochastic Solvers

In this section...

When to Use Stochastic Solvers

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.

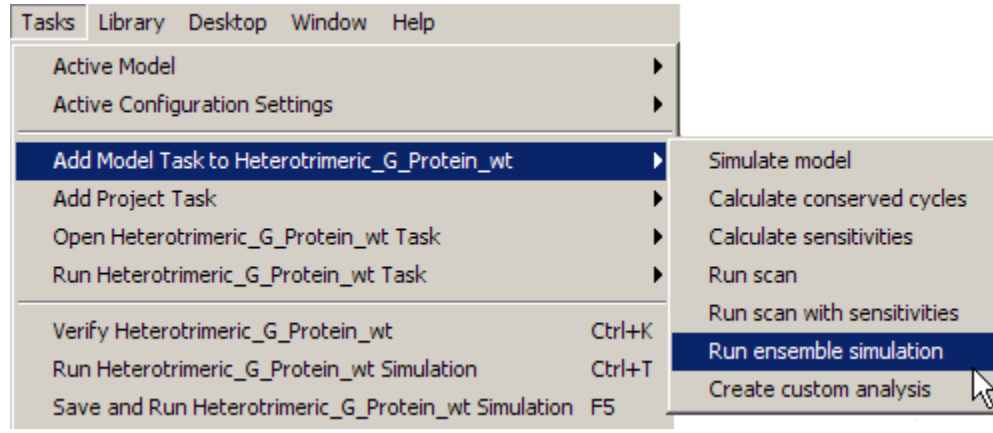
Running Ensemble Simulations at the Command Line

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.

3

Analysis

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

Sensitivity Analysis

In this section...

About Sensitivity Analysis

Sensitivity analysis lets you calculate the time-dependent sensitivities of all the species states with respect to species initial conditions and parameter values in the model.

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

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

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

Sensitivity analysis is supported only by the ordinary differential equation (ODE) solvers. The software calculates local sensitivities by combining the original ODE system for a model with the auxiliary differential equations for the sensitivities. The additional equations are derivatives of the original equations with respect to parameters. This method is sometimes called "forward sensitivity analysis" or "direct sensitivity analysis". This larger system of ODEs is solved simultaneously by the solver.

SimBiology sensitivity analysis uses the "complex-step approximation" to calculate derivatives of reaction rates. This technique yields accurate results for the vast majority of typical reaction kinetics, which involve only simple mathematical operations and functions. When a reaction rate involves a non-analytic function, this technique can lead to inaccurate results; in this case, either sensitivity analysis is disabled, or sensitivity analysis warns

you that the computed sensitivities may be inaccurate. An example of such a non-analytic function is the MATLAB function `abs`. If sensitivity analysis gives questionable results on a model whose reaction rates contain unusual functions, you may be running into limitations of the complex-step method. Contact the MathWorks Technical Support group for additional information.

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

For more information on the calculations performed, see "Reference" on page 3-4

Performing Sensitivity Analysis Using the Command Line

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

- `SensitivityAnalysis` – Lets you calculate the time-dependent sensitivities of all the species states defined by the `SpeciesOutputs` property with respect to the initial conditions of the species specified in `SpeciesInputFactors` and the values of the parameters specified in `ParameterInputFactors`.
- `SensitivityAnalysisOptions` – An object that holds the sensitivity analysis options in the configuration set object. Properties of `SensitivityAnalysisOptions` are summarized below:
  - `SpeciesOutputs` – Specify the species for which you want to compute the sensitivities. Sensitivities are calculated with respect to the initial conditions of the specified species. This is the numerator as described in "About Sensitivity Analysis" on page 3-2.
  - `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 3-2.
  - `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 3-2.

- `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 3-16

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 **Analysis** menu select **Add Analysis 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 3-5

Reference

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

Desktop Example — Calculating Sensitivities

In this section...

Overview

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 3-16.

About the Example Model

This example uses the model from "Modeling a G Protein Cycle" in the *SimBiology Model Reference*.

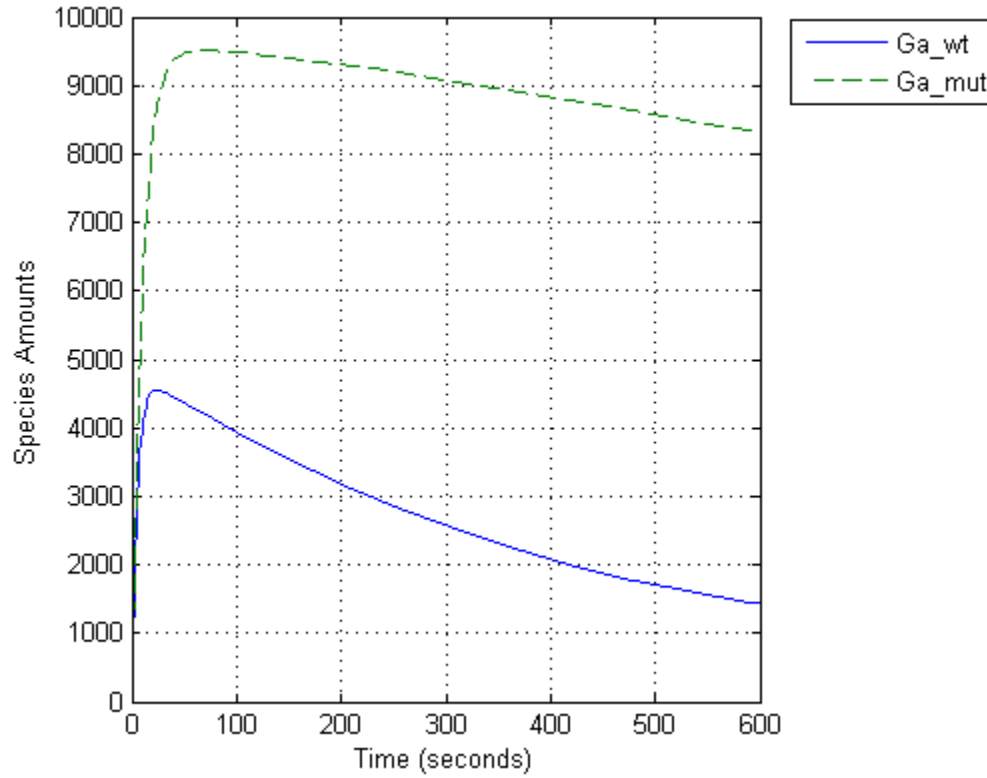
This table shows the reactions used to model the G protein cycle and the corresponding rate parameters (rate constants) for each reaction. For reversible reactions, the forward rate parameter is listed first.

No.	Name	Reaction	Rate Parameters
1	Receptor-ligand interaction	`L + R <-> RL`	`kRLm`, `kRL`
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:

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

Prerequisites

- "Opening and Saving the Example Model" on page 3-8
- "Preparing to Modify the Example Model" on page 3-8
- "Adding a Task to Calculate Sensitivities" on page 3-9

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:

   ```
   sbiodesktop(m1)
   ```

   The SimBiology desktop opens with **Yeast_G_Protein_wt**.

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

4. Specify a name (for example, `gprotein_ex`) and location for your project and click **Save**.

Preparing to Modify the Example Model

Copy a model in the project and use the copy to work on this example.

1. In the **Project Explorer**, right-click **Yeast_G_Protein_wt** and select **Copy Model and Add to Project**. The desktop adds the copied model to the **Project Explorer**.

2. Rename the copied model.

a In the **Project Explorer**, right-click **Model Session** for the copied model and select **Rename Model**. The Rename Model dialog box opens.

b Modify the name, for example, `gprotein`.

c Click **Save**.

Adding a Task to Calculate Sensitivities

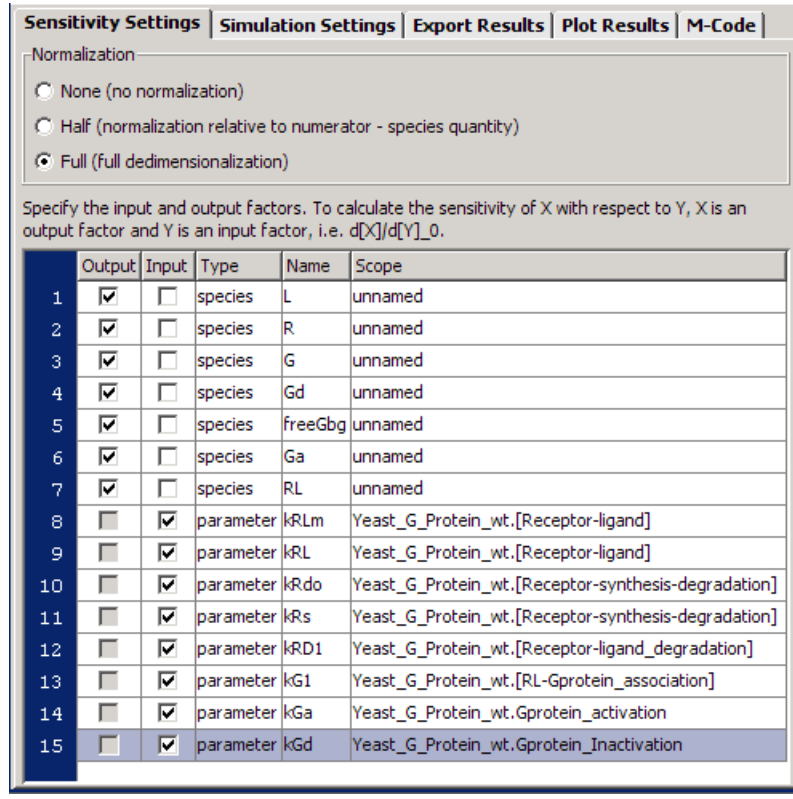
To perform sensitivity analysis in a model, first add a model task to calculate sensitivities:

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

  The **Sensitivity Analysis** pane opens.

Setting Options for Sensitivity Analysis

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



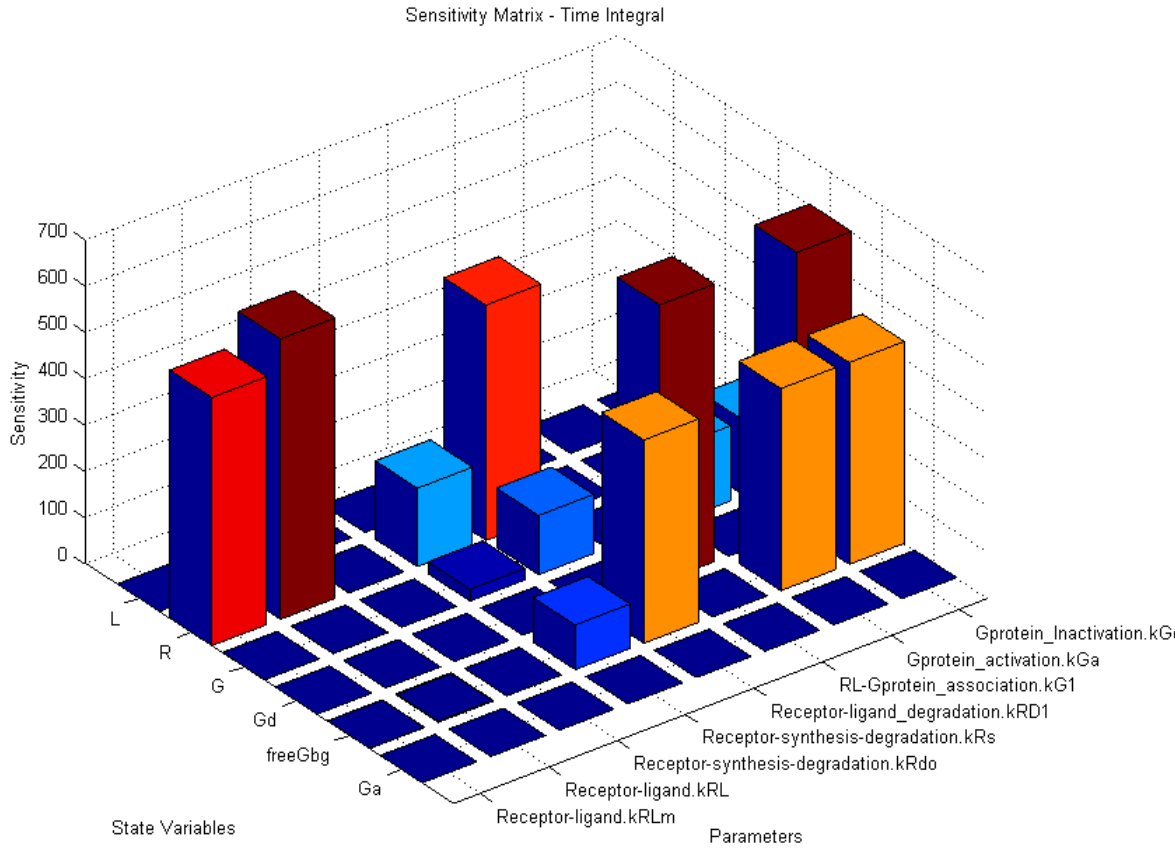
	Output	Input	Type	Name	Scope
1	☑	☐	species	L	unnamed
2	☑	☐	species	R	unnamed
3	☑	☐	species	G	unnamed
4	☑	☐	species	Gd	unnamed
5	☑	☐	species	freeGbg	unnamed
6	☑	☐	species	Ga	unnamed
7	☑	☐	species	RL	unnamed
8	☐	☑	parameter	kRLm	Yeast_G_Protein_wt.[Receptor-ligand]
9	☐	☑	parameter	kRL	Yeast_G_Protein_wt.[Receptor-ligand]
10	☐	☑	parameter	kRdo	Yeast_G_Protein_wt.[Receptor-synthesis-degradation]
11	☐	☑	parameter	kRs	Yeast_G_Protein_wt.[Receptor-synthesis-degradation]
12	☐	☑	parameter	kRD1	Yeast_G_Protein_wt.[Receptor-ligand_degradation]
13	☐	☑	parameter	kG1	Yeast_G_Protein_wt.[RL-Gprotein_association]
14	☐	☑	parameter	kGa	Yeast_G_Protein_wt.Gprotein_activation
15	☐	☑	parameter	kGd	Yeast_G_Protein_wt.Gprotein_Inactivation

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

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

Getting Results for Sensitivity Analysis

1 In the **Sensitivity Analysis** pane, click **Run**. The **Sensitivity Matrix Subplot**, which is the default plot for sensitivity analysis, opens.



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

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

2 Close the figure window. Notice that the **Data** pane showing the most recent data is now open in the SimBiology desktop . If the **Data** pane is not open:

- In the **Project Explorer**, for the `gprotein` model, under **Sensitivity Analysis**, click **Data** ***(date)***.

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

3 In the **Project Explorer**, for the `gprotein` model, under **Sensitivity Analysis**, right-click **Data** and select **Save Data**. The Save Data dialog box opens.

4 Specify a name for the saved data, for example, `sensitivity_ex1`, and click **Save**. The **Project Explorer** shows a new item with the saved data name under **Sensitivity Analysis**.

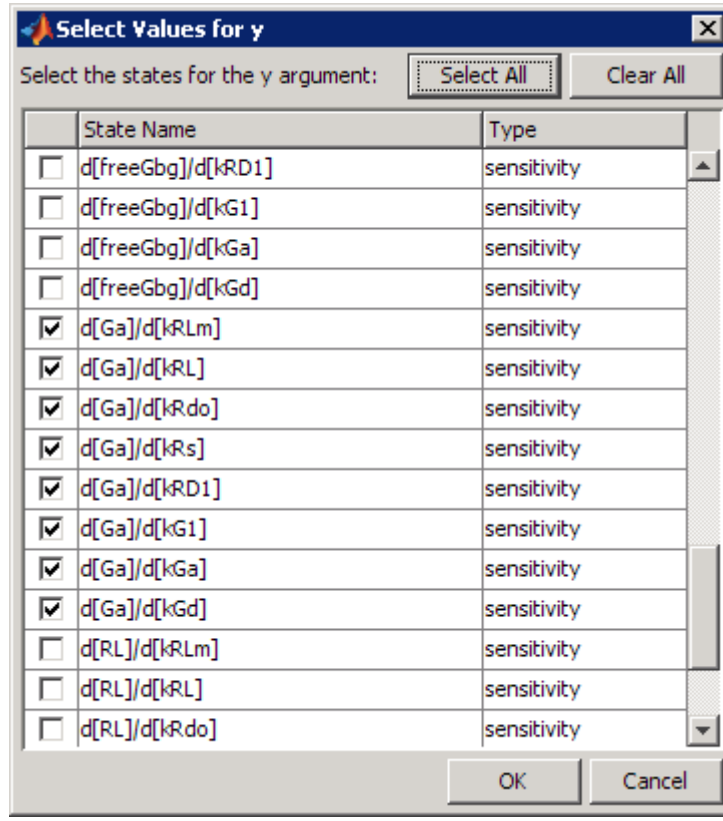
5 In the **Project Explorer**, for the `gprotein` model, click the saved data, for example, `sensitivity_ex1`, to open the **Data** pane for the saved data.

6 Click the **Plots** tab. Here, you can plot the results using the default `Sensitivity Matrix Subplot`, the other relevant plots for these results might be a `Time plot`, or the `Sensitivity Matrix Subplot Max` which plots the sensitivity matrix using maximum values of the sensitivities for the specified input and output factors.

7 In the **Plot Type** box, select `Time` and click **Add Plot Type**.

8 Select the new plot (second row), and in the **Arguments** section click  . The Select Values for y dialog box opens.

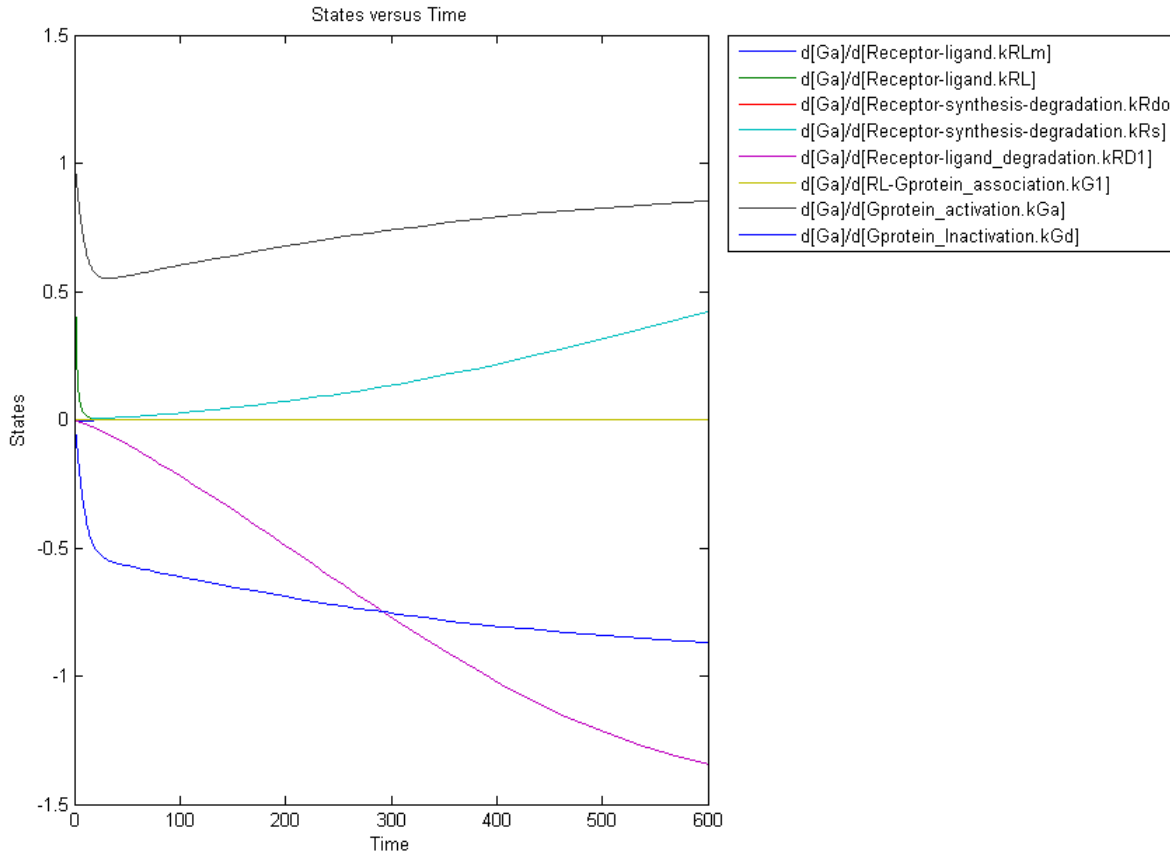
9 Click **Clear All** and then select the check box for the rows containing the time-dependent derivatives of `Ga` with respect to each parameter.



10 Click **OK**.

11 Clear the **Create Plot** check box for the `Sensitivity Matrix Subplot` plot.

12 Click **Plot**. Your plot should following one.



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

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

References

[1] Tau-Mu Yi, Hiroaki Kitano, and Melvin I. Simon. A quantitative characterization of the yeast heterotrimeric G protein cycle. PNAS (2003) vol. 100, 10764–10769.

Command-Line Example — Calculating Sensitivities

In this section...

Overview

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`	`kRLm`, `kRL`
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:

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

To calculate these sensitivities:

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

The following sections explain the details of the procedure.

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

```
edit gprotein
```

Loading and Configuring the Model for Sensitivity Analysis

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

   ```
   sbioloadproject gprotein_norules m1
   ```

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

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

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

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

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

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

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

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

Performing Sensitivity Analysis

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

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

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

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

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

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

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

Extracting and Plotting Sensitivity Data

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

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

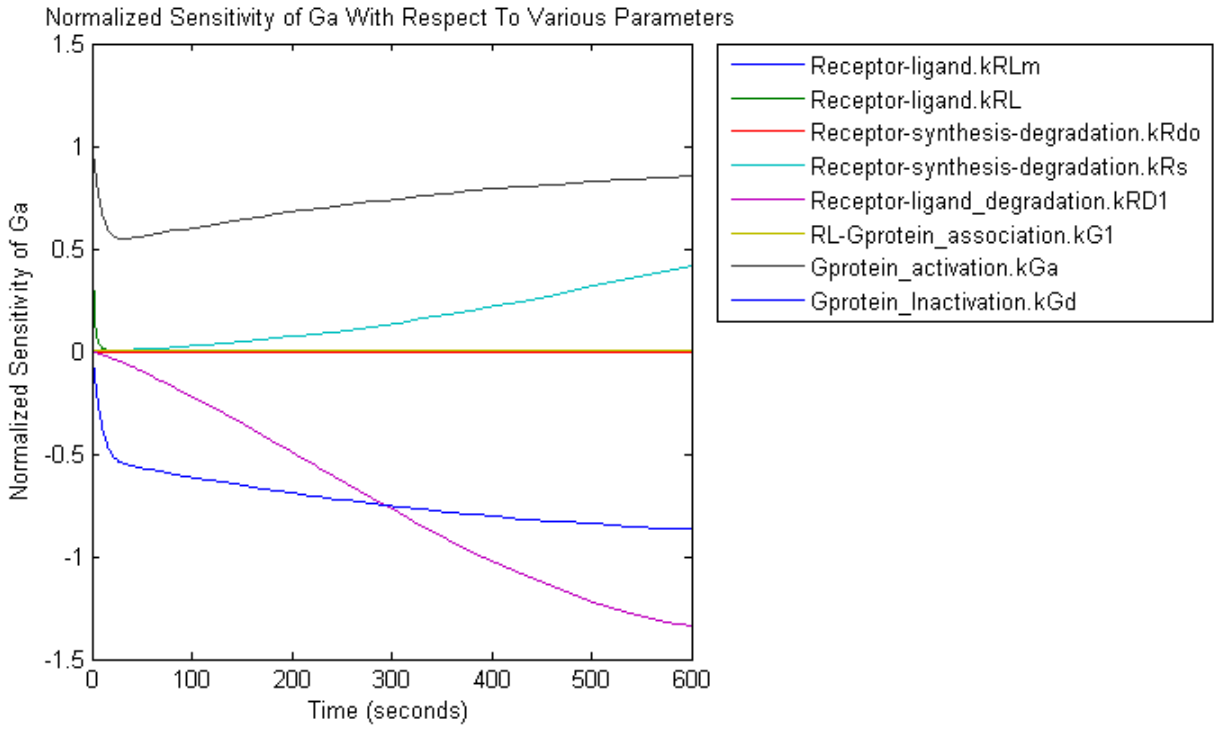
```
[T, R, snames, ifacs] = getsensmatrix(simDataObj);
```

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

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

3. After extracting the data and reshaping the matrix, you can now plot the data.

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



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

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

See Also

For information about...	See...
Configuring simulation settings	"Performing Simulations at the Command Line" on page 2-2
Normalizing the data	`Normalization` in the SimBiology Reference
Selecting model component objects by querying the model as shown in Step 4 in "Loading and Configuring the Model for Sensitivity Analysis" on page 3-17	`sbioselect`

Parameter Estimation

In this section...

About Parameter Estimation

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

SimBiology Parameter Estimation

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

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

For more information, see `sbioparamestim` in the SimBiology Reference. For an example, see "Command-Line Example — Parameter Estimation" on page 3-23

Command-Line Example — Parameter Estimation

In this section...

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 , 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. If you need to import data into MATLAB see Upgrade to Random Number Generator, in the MATLAB documentation for more information.

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_span`.

```
t_span  = [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.

```
m1.configset

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

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

   RuntimeOptions:
     StatesToLog:                 6

   CompileOptions:
     UnitConversion:              false
     DimensionalAnalysis:         false

   SensitivityAnalysisOptions:
     InputFactors:                0
     Outputs:                     0
```

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

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

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

3 Retrieve the time and state data.

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

Calculating R-Square for the G Protein Model

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

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

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

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

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

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

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

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

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

```
rsquare_orig =

    0.8967
```

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

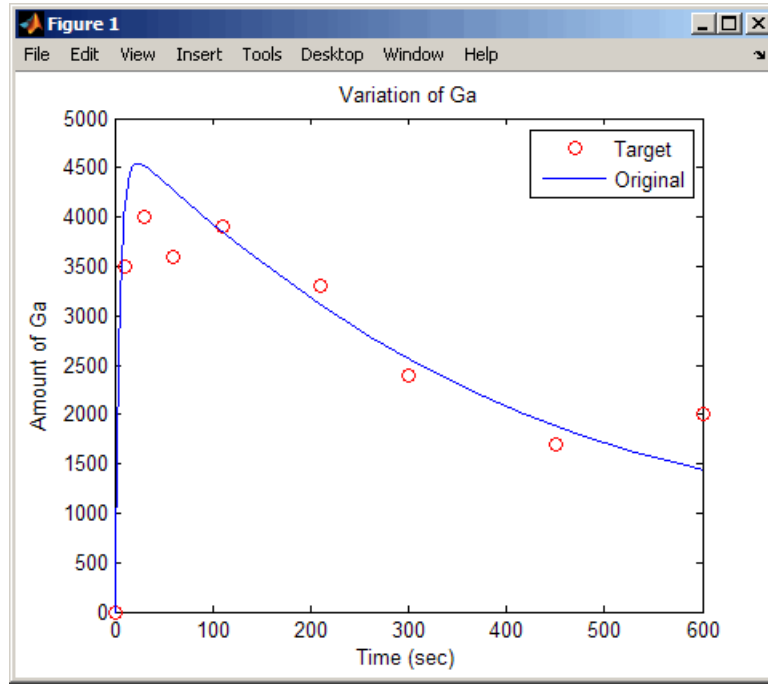
Plotting the Experimental Results and Simulation Data

1. Plot the experimental data for `Ga`.

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

2 Plot the simulation data in the same plot.

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



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

Estimating a Parameter (kGd) in the G Protein Model

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

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

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

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

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

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

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

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

```
Iteration  Func-count     f(x)          step          optimality   CG-iterations
     0          2     1.4264e+006                     2.84e+007
     1          4    1.11306e+006      0.0105776      8.23e+006            1
     2          6    1.11306e+006      0.0045504      8.23e+006            1
     3          8    1.11306e+006      0.0011376      8.23e+006            0
     4         10    1.11183e+006      0.0002844      6.93e+005            0
     5         12    1.11183e+006      0.0002844      6.93e+005            1
     6         14    1.11183e+006   7.10999e-005      6.93e+005            0
     7         16    1.11183e+006    1.7775e-005      6.93e+005            0
     8         18    1.11183e+006   4.44375e-006      6.93e+005            0
     9         20    1.11183e+006   1.11094e-006      6.93e+005            0
    10         22    1.11183e+006   2.77734e-007      6.93e+005            0
Optimization terminated: norm of the current step is less
 than OPTIONS.TolX.
```

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

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

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

```
 Iteration   Func-count     min f(x)         Procedure
     0           1          1194.32
     1           2          1091.86         initial simplex
     2           4           1054.2         reflect
     3           6           1054.2         contract outside
     4           8           1054.2         contract inside
     5          11           1054.2         shrink
     6          13           1054.2         contract outside
     7          15          1053.95         contract outside
     8          17          1053.95         contract inside
     9          19          1053.86         reflect
    10          21          1053.86         contract inside
    11          23          1053.86         contract inside
    12          25          1053.84         reflect
    13          27          1053.84         contract inside
    14          29          1053.82         reflect
    15          31          1053.82         contract inside
    16          33          1053.34         reflect
    17          36          1053.34         shrink
    18          38          1053.32         reflect
    19          40          1053.32         contract inside
    20          42          1053.32         contract inside
    21          44          1053.32         contract inside
    22          46          1053.32         contract outside
    23          48          1053.32         contract inside
    24          51          1053.32         shrink
    25          53          1053.32         contract outside

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

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

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

```
rsquare1 =

    0.9195
```

Simulating and Plotting Results Using the Estimated Parameter

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

1 Before changing the value, save the old value in case you need it later.

```
kGd0 = get(param_to_tune, 'Value');
```

2 Set the parameter to the new value. The `param_to_tune` variable was previously defined as the parameter `kGd` in this exercise.

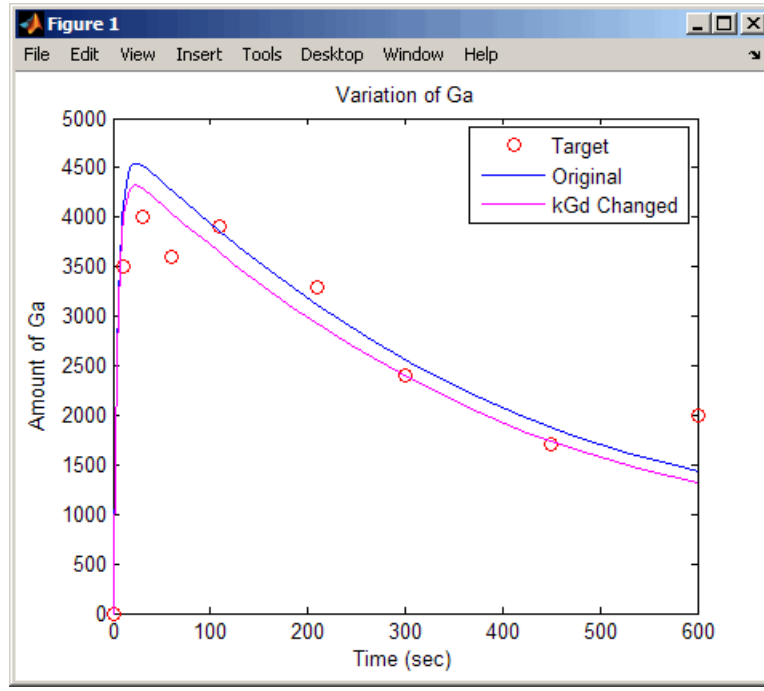
```
set(param_to_tune, 'Value', k_new1);
```

3 Simulate the model and get the results.

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

4 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 () 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 .

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. Reset the value of the parameter `kGd` to the original value.

```
set(param_to_tune, 'Value', kGd0);
```

2. 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`.

3 Set the parameter array for estimation.

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

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

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

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

---

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

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

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

```
                                        Norm of      First-order
 Iteration  Func-count     f(x)          step          optimality   CG-iterations
     0          5     1.4264e+006                      2.84e+007
     1         10          611055        3.63737      2.58e+006            1
     2         15          576458          1.188      6.76e+005            1
     3         20          576458      0.0540078      6.76e+005            1
     4         25          576458      0.0135019      6.76e+005            0
     5         30          576458     0.00337549      6.76e+005            0
     6         35          576458    0.000843872      6.76e+005            0
     7         40          576458    0.000210968      6.76e+005            0
     8         45          576458    5.2742e-005      6.76e+005            0
     9         50          576458   1.31855e-005      6.76e+005            0
    10         55          576458   3.29637e-006      6.76e+005            0
    11         60          576458   8.24093e-007      6.76e+005            0
Optimization terminated: norm of the current step is less
 than OPTIONS.TolX.
```

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

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

6 Compare original parameter values and the estimated parameter values obtained with `'lsqcurvefit'`.

```
% Original parameter values.
param_to_tune

SimBiology Parameter Array

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

% Estimated parameter values.
k_new2 =
```

```
    5.9330
    0.0034
    0.0000
    0.1346
```

7 Calculate the R-Square value with the new estimates obtained with `'lsqcurvefit'`.

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


rsquare2 =

    0.9585
```

Desktop Example — Performing Scans

In this section...

Overview

This example shows you how to set up and run scans using 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*.

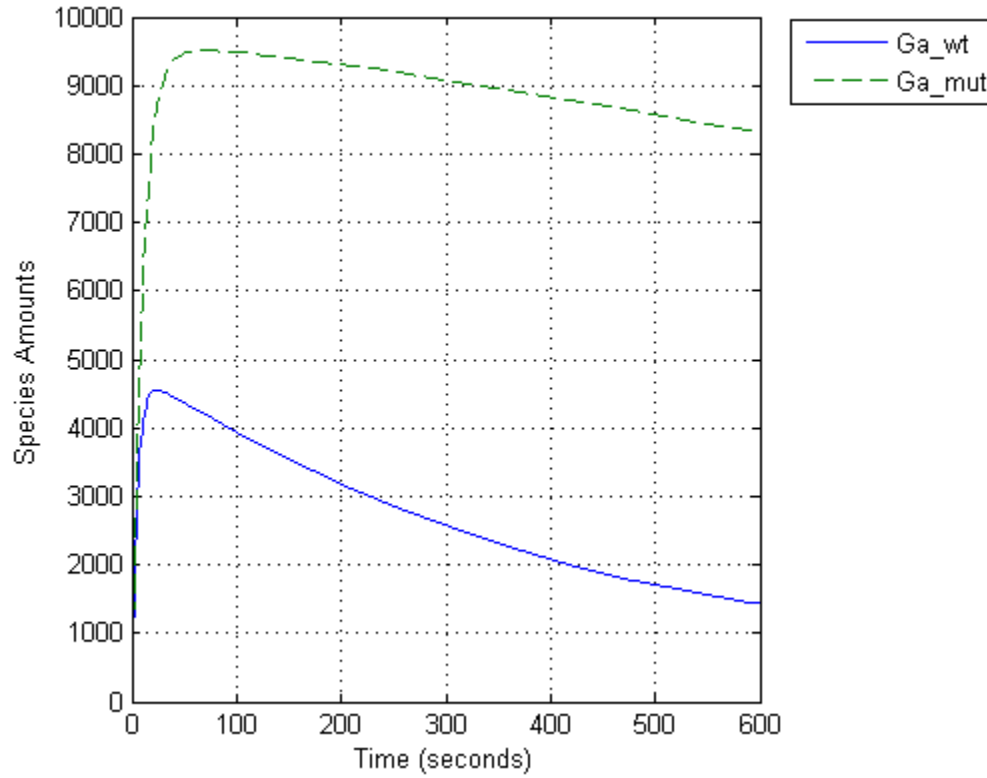
This table shows the reactions used to model the G protein cycle and the corresponding rate parameters (rate constants) for each reaction. For reversible reactions, the forward rate parameter is listed first.

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

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

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 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` is varied over a range of values. This example illustrates a scan over 10 values of the parameter `kGd`.

Prerequisites

- “Opening and Saving the Example Model” on page 3-62
- “Preparing to Modify the Example Model” on page 3-63
- “Adding a Scanning Task” on page 3-38

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:

   ```
   sbiodesktop(m1)
   ```

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

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

4. Specify a name (for example, `gprotein_ex`) and location for your project, and click **Save**.

Preparing to Modify the Example Model

Copy a model in the project and use the copy to work on this example.

1. In the **Project Explorer**, right-click **Model Session-Heterotrimeric_G_Protein_wt** and select **Copy Model and Add to Project**. The desktop adds the copied model to the **Project Explorer**.

2. Rename the copied model.

   a. In the **Project Explorer**, right-click **Model Session** for the copied model and select **Rename Model**. The Rename Model dialog box opens.

   b. Modify the name, for example, `gprotein`.

   c. Click **Save**.

Adding a Scanning Task

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

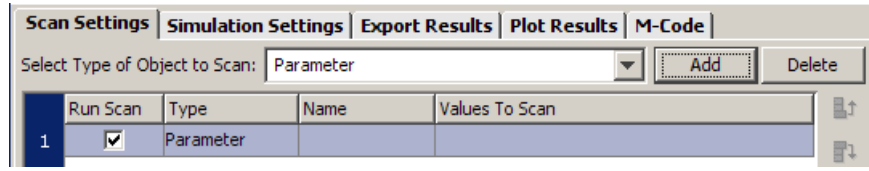
1 In the **Project Explorer**, under **Model Session-gprotein**, right-click **Model Tasks**.

2 Select **Add Task > Run scan**. The **Scan** pane opens.

Setting Options to Scan with One Parameter

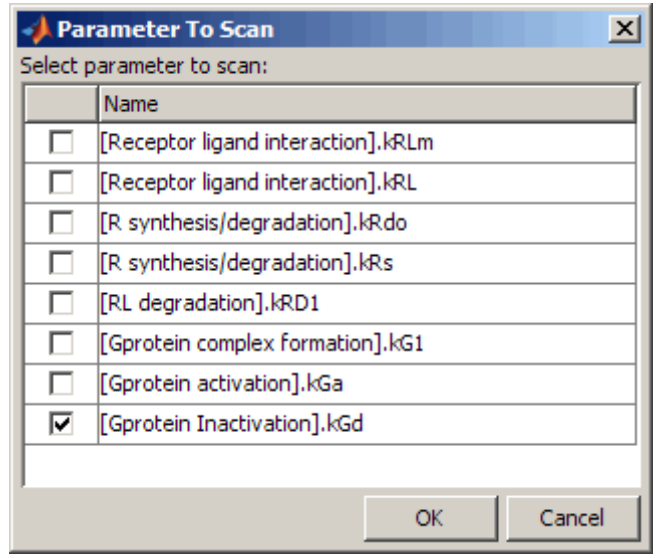
1 In the **Scan** pane, from the **Select Type of Object to Scan** list select the model component you want to scan. For this example, leave the selection as the default `Parameter`.

2 Click **Add**. A row containing the type of scan to run appears in the table.



3 Double-click the **Name** cell. The Parameter To Scan dialog box opens.

4 Select a parameter and click **OK**. For this example, select `kGd`.



5 Double-click the **Values To Scan** cell. The Values To Scan dialog box opens. This dialog box shows you the current value of the parameter, and lets you select a range of values to scan.

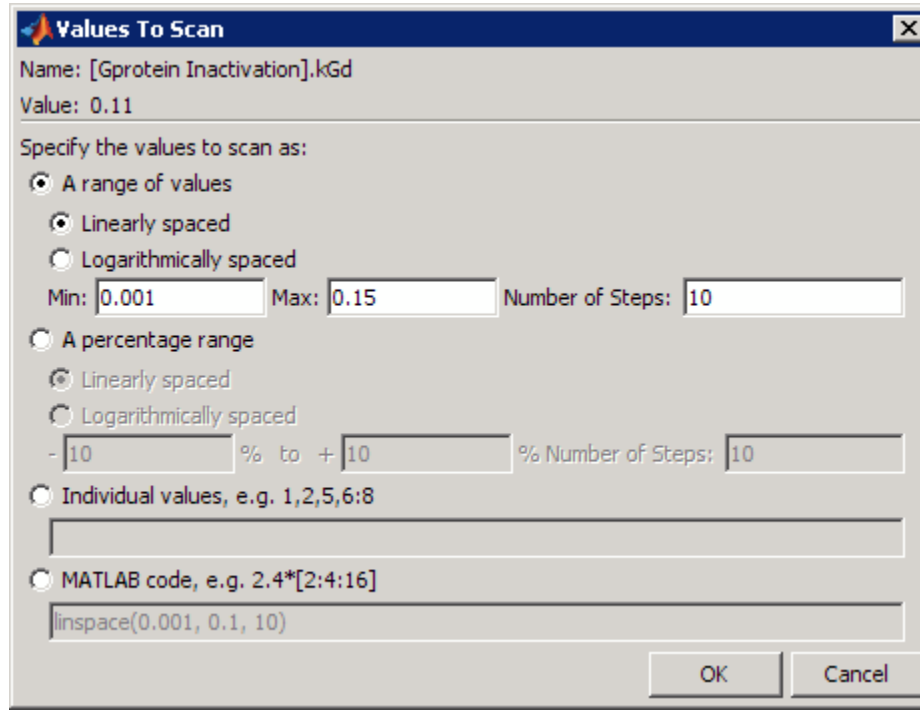
   Leave the following options selected (default):

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

6 In the **Min** box enter `0.001`.

7 In the **Max** box enter `0.15`.

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

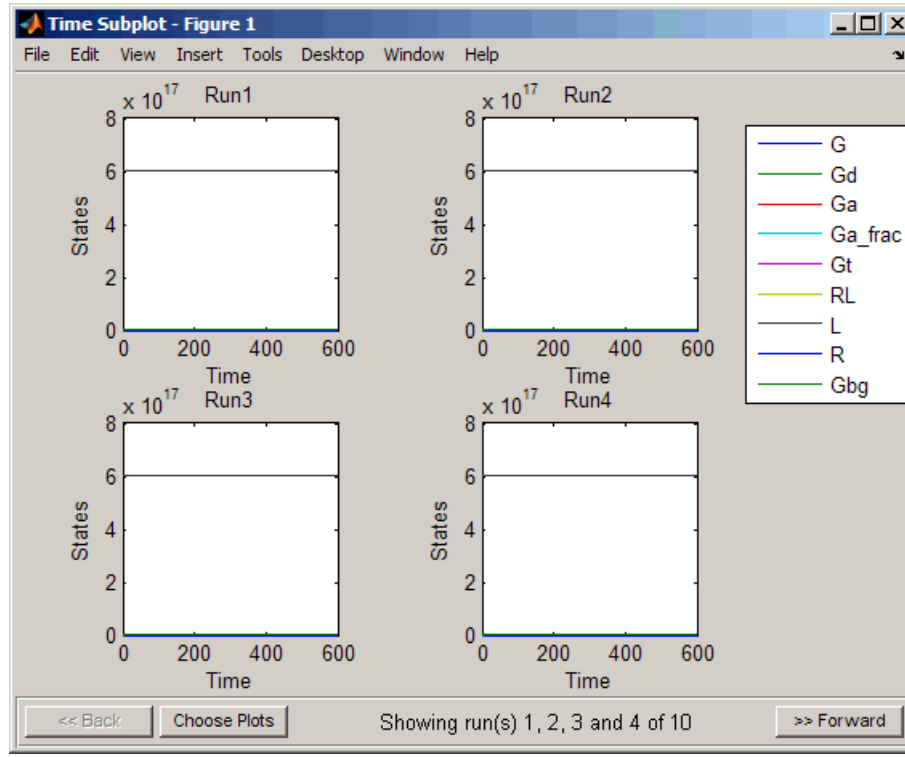


9 Click **OK**.

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

Results of Scanning with One Parameter

1 In the **Scan** pane, click **Run**. The Time Subplot window opens with each run plotted in a separate subplot.



Tip Notice that you can select the plots you want to see by clicking **Choose Plots**. You can also view the range of plots by clicking **Forward**.

The results in these plots are not very useful as shown. This could be due to axis scaling not showing the interesting scan results. Another reason could be that there is a negligible effect of changes to the value of parameter `kGd` on the model. But, sensitivity analysis of this model shows that several species are sensitive to the value of `kGd`. To see the more interesting scans, you must view the subset of species that could be affected by the parameter scanned. One possibility is the active G protein (`Ga`) as discussed in "About This Example" on page 3-36.

2. Close the figure window. Notice that the **Data** pane showing the most recent scan data is now open in the SimBiology desktop. If the **Data** pane is not open:

   - In the **Project Explorer**, for the `gprotein` model, under **Scan**, click **Data (*`Date`*)**.

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

3. In the **Project Explorer**, for the `gprotein` model, 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**.

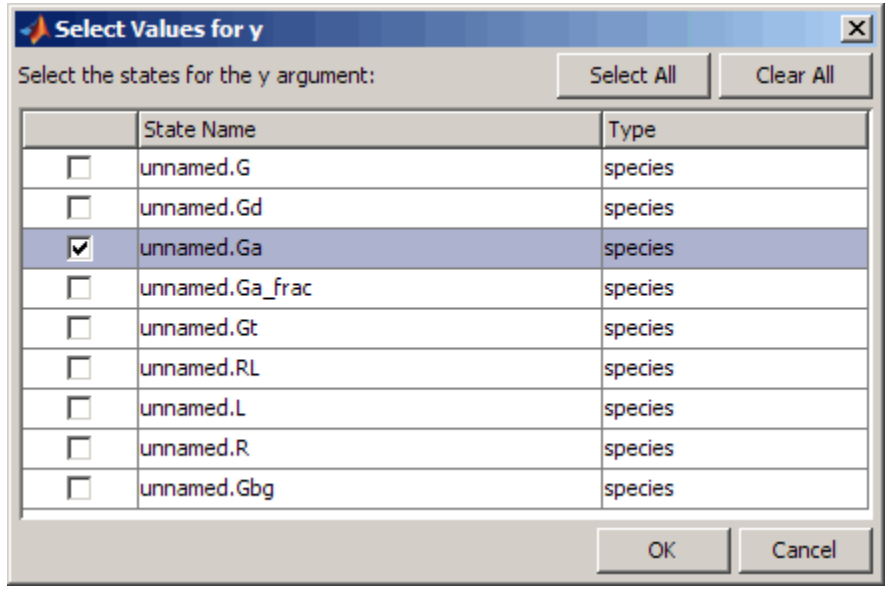
5. In the **Project Explorer**, for the `gprotein` model, click the saved data, for example, `scan_ex1`, to open the **Data** pane for the saved data.

6. Click the **Plots** tab. Here, you select a plot to visualize the results. For example, you can plot the results in separate subplots (`Time Subplot`) or in a single overlay plot (`Time`).

7. In the **Plot Type** box, select `Time` and click **Add Plot Type**.

8. Select the new plot (second row), and in the **Arguments** section, click  . The Select Values for y dialog box opens.

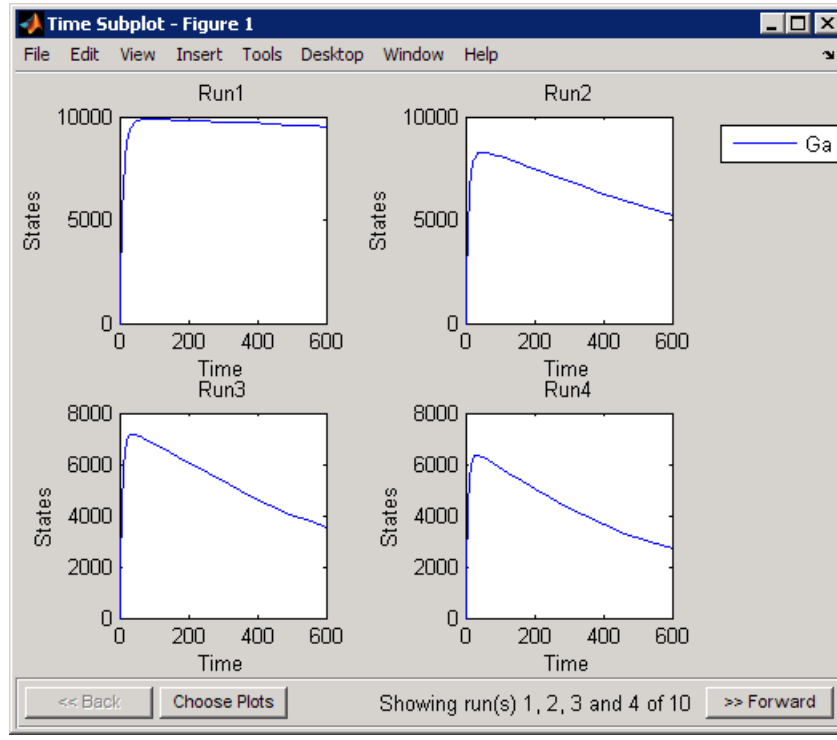
9. Click **Clear All** and then select the check box for the species `Ga`.

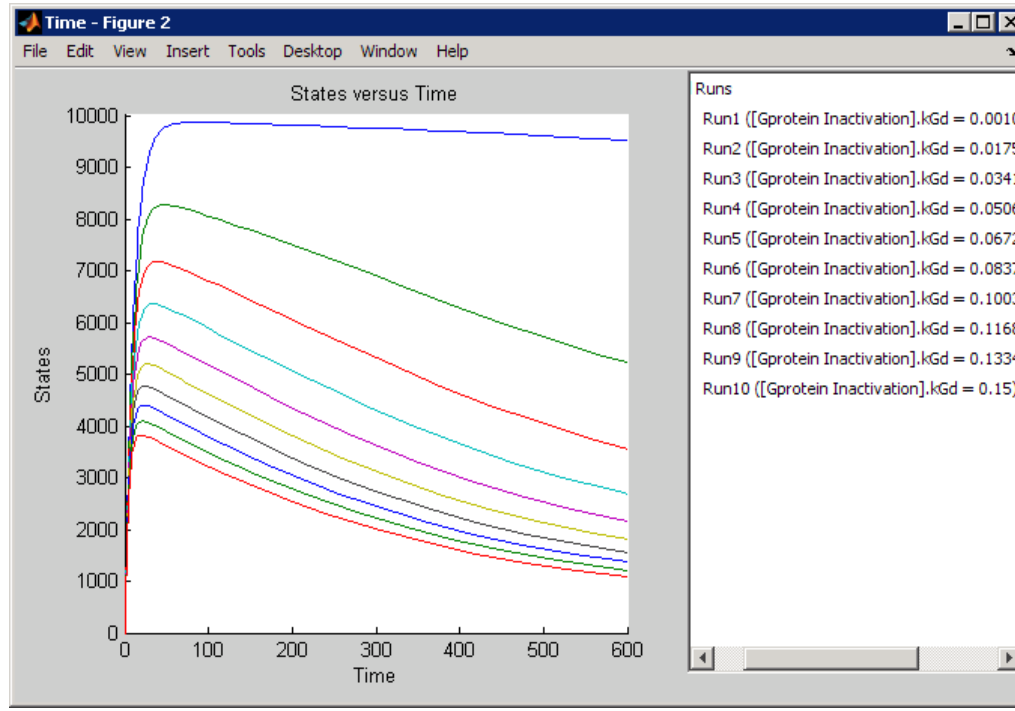


10 Click **OK**.

11 Select the `Time Subplot` and 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 plots below.


Time Subplot - Figure 1
File Edit View Insert Tools Desktop Window Help
Run1
Run2
Run3
Run4
States
Time
10000
5000
0
8000
6000
4000
2000
0 200 400 600
Ga
<< Back
Choose Plots
Showing run(s) 1, 2, 3 and 4 of 10
>> Forward



Scanning with Multiple Parameters

If there is evidence to show that more than one parameter has an effect on the species of interest, it might be useful to scan over a range of values for two or more parameters. Similarly, you can also scan using two or more species.

For example, in this model, previous sensitivity analysis shows that the amount of active G protein (`Ga`) is also sensitive to the parameter `kRL`. Indeed, 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 more than one parameter:

1. In the **Project Explorer**, under **Model Session – gprotein**, expand **Model Tasks** and click **Scan**.

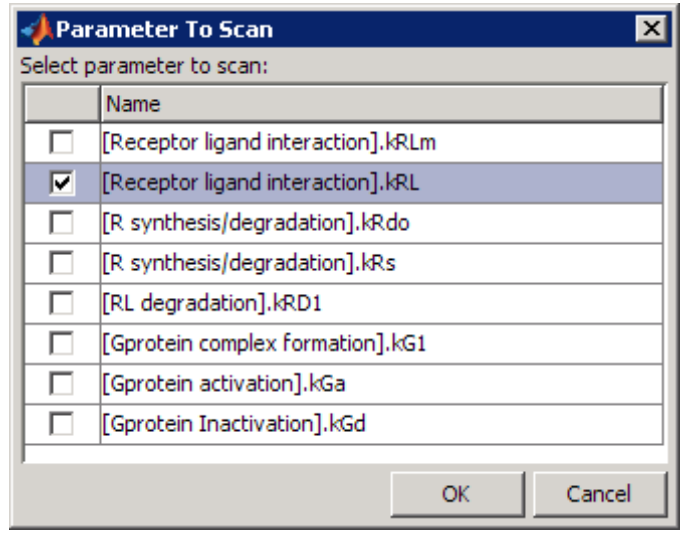
2. In the **Scan** pane, from the **Select Type of Object to Scan** list leave the selection as the default (`Parameter`) and click **Add**.

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

	Run Scan	Type	Name	Values To Scan
1	☑	Parameter	[Gprotein Inactivation].kGd	linspace(0.001, 0.15, 10)
2	☑	Parameter		

3. Double-click the **Name** cell. The Parameter To Scan dialog box opens.

4. Select `kRL` and click **OK**.



5. Double-click the **Values To Scan** cell. The Values To Scan dialog box opens. This dialog box shows you the current value of the parameter, and lets you select a range of values to scan.

6 Select the **Individual values** option and enter the following values:

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

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

7 Click **OK**.

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

Notice that five values have been specified for `kRL` in step 6. Thus, there will be 50 iterative simulations when the task is run because the simulations must loop through each value of `kGd` (there are 10 values), against each value of `kRL`.

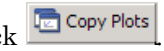
To minimize the number of simulations and maximize the performance, you can specify that the scan must occur along the diagonal, that is you can specify that the first value of `kGd` should be used with the first specified value of `kRL` and so on. To do this, you must specify the same number of values for each species, parameter, or compartment being scanned and select the **Scan along the diagonal** check box.

Results of Scanning with Multiple Parameters

Previously, in “Results of Scanning with One Parameter ” on page 3-41 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 **Plot Results** 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 **Plot Results**.

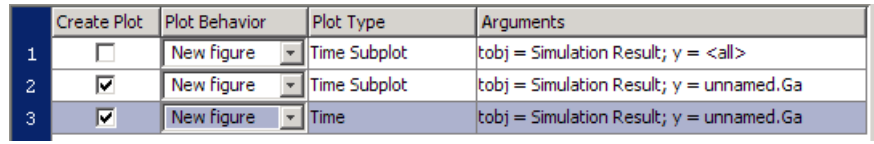
2 Click 

. The Append Plot Settings dialog box opens. This dialog box lets you add plots that you might have specified in another task, or in the **Data** pane.

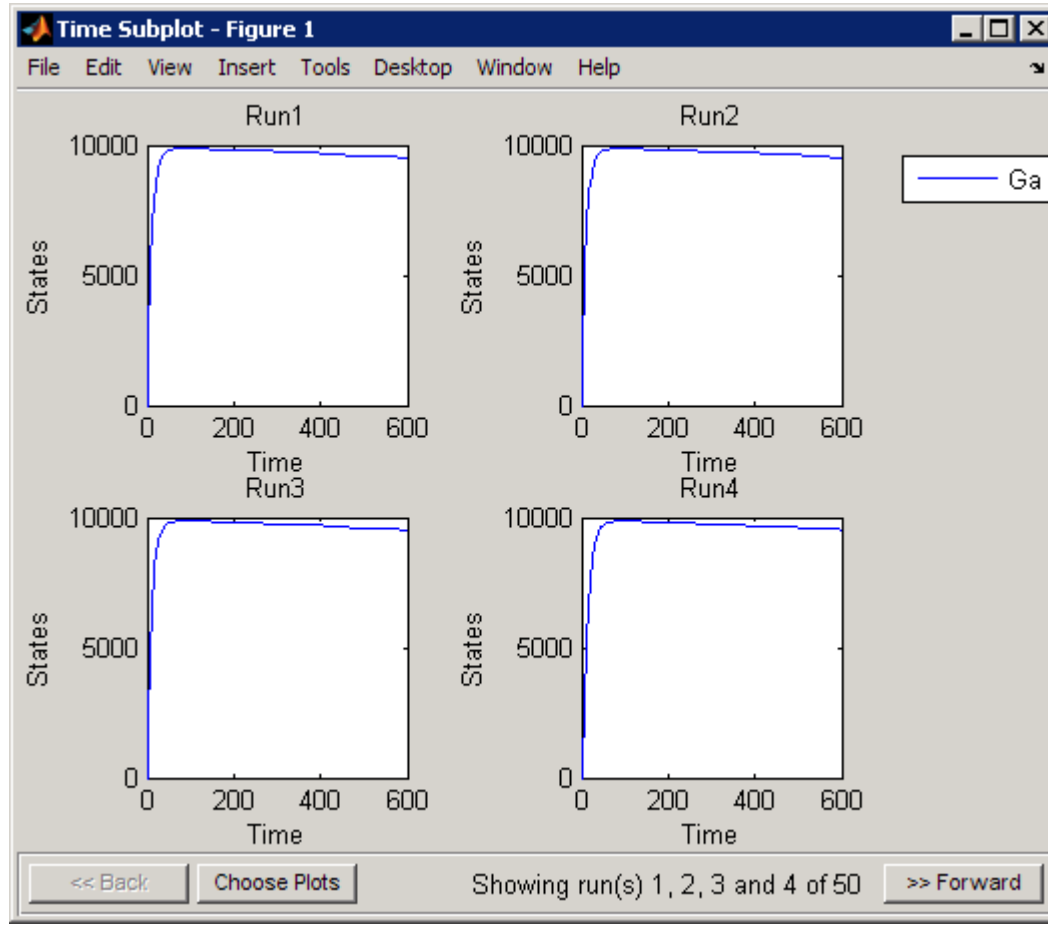
3 Select the name of the saved data for the previous scan, `scan_ex`, and click **OK**.

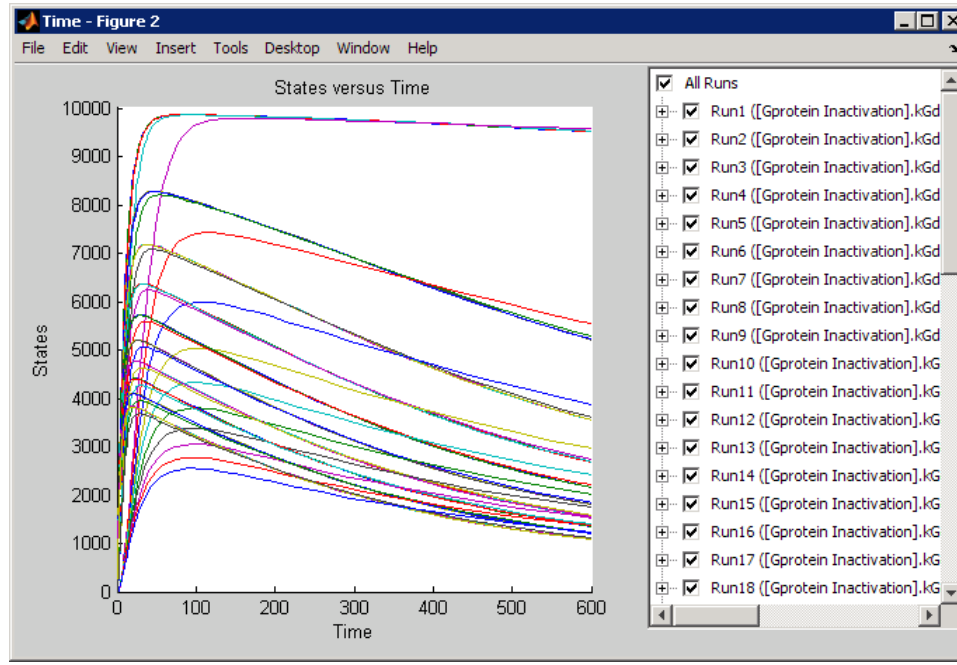
The plots previously specified while scanning with one parameter are added to the table.

4 Clear the **Create Plot** check box for the default plot in the first row, that specifies`y = <all>` for in the **Arguments** column.



5 In the **Scan** pane, click **Run**. Your plots should resemble the following. Depending on the values selected automatically for scanning, your results may not exactly match the plots below.


Time Subplot - Figure 1
File Edit View Insert Tools Desktop Window Help
Run1
Run2
Run3
Run4
States
Time
10000
5000
0
0 200 400 600
Ga
<< Back
Choose Plots
Showing run(s) 1, 2, 3 and 4 of 50
>> Forward



Tip In the `Time` plot (second figure above), notice that you can 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.

Moiety Conservation

In this section...

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 3-52 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.

Desktop Examples — Determining Conserved Moieties

In this section...

G Protein Example

1 Load the project `gprotein_norules.sbproj`

```
sbioloadproject gprotein_norules
```

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

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

2 Display the species information.

```
m1.Compartments.Species

SimBiology Species Array

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

3 Display reaction information.

```
m1.Reactions

SimBiology Reaction Array
```

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

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

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

g =

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


sp =

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

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

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

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

ans =

    'G + freeGbg'
```

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

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

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

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

7 Show the list of independent species.

```
SI

SI =

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

- Show the list of dependent species.

```
SD

SD =

    'freeGbg'
    'Ga'
```

- Show the link matrix relating `SD` and `SI`.

```
LO


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

- Show the independent stoichiometry matrix, N_R.

```
NR


NR =

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

- Show the dependent stoichiometry matrix, N_D.

```
ND

ND =

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

Mitotic Oscillator Example

1 Load the Goldbeter Mitotic Oscillator model.

```
sbioloadproject Goldbeter_Mitotic_Oscillator_with_reactions
```

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

2 Display the species information.

```
m1.Compartments.Species

SimBiology Species Array
```

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

3 Display reaction information.

```
m1.Reactions

SimBiology Reaction Array

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

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

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

g =

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


sp =
```

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

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

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

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

cons_rel =

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

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

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

7 Show the list of independent species.

```
SI

SI =

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

8 Show the list of dependent species.

```
SD

SD =
```

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

9 Show the link matrix relating `SD` and `SI`.

```
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

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:

   ```
   sbiodesktop(m1)
   ```

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

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

4. Specify a name (for example, `gprotein_ex`) and location for your project, and click **Save**.

Preparing to Modify the Example Model

Copy a model in the project and use the copy to work on this example.

1. In the **Project Explorer**, right-click **Model Session-Heterotrimeric_G_Protein_wt** and select **Copy Model and Add to Project**. The desktop adds the copied model to the **Project Explorer**.

2. Rename the copied model.

   a. In the **Project Explorer**, right-click **Model Session** for the copied model and select **Rename Model**. The Rename Model dialog box opens.

   b. Modify the name, for example, `gprotein`.

   c. Click **Save**.

Setting Up a Custom Task

1. From the **Tasks** menu select **Add Model Task to Heterotrimeric_G_Protein_wt > Create custom analysis**. The SimBiology desktop adds the custom task to the **Project Explorer**, and opens the task pane.

2. In the **Custom Settings** tab you can define your own script that you can run in the desktop.

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

Parameter Estimation Using Custom Task

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

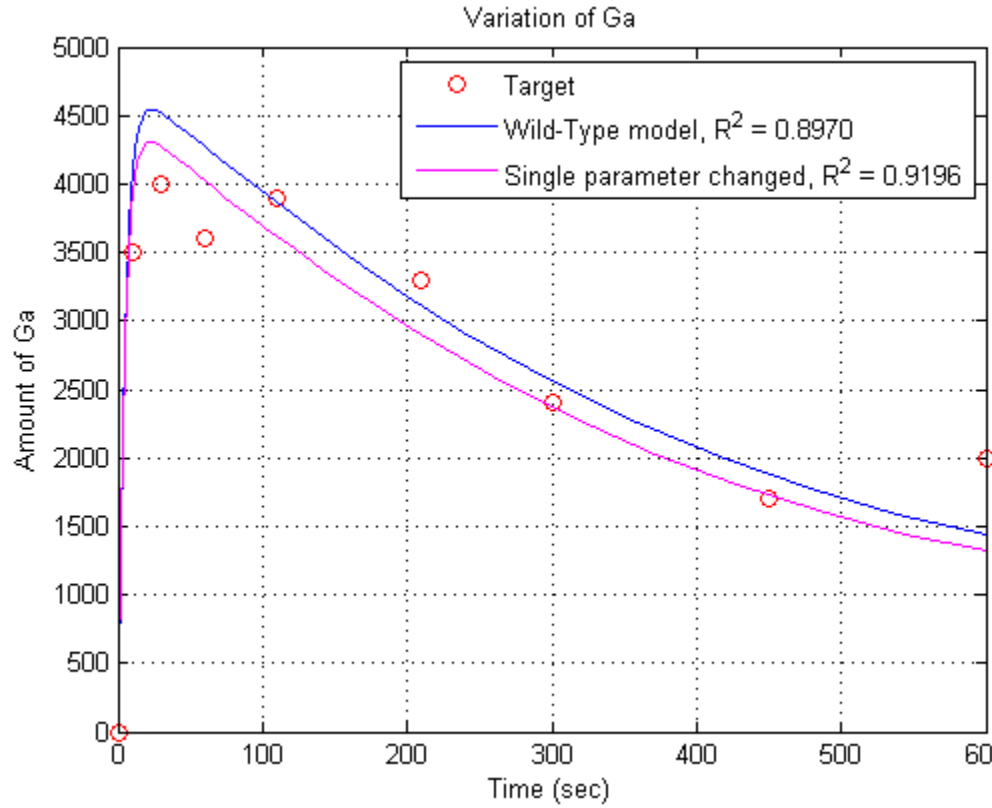
```
function data = custom(modelobj)
%Parameter Estimation of a G Protein Model

%Target Data

%Preprocess the experimental data

% The estimated amount of total G protein (Gt) is 10000
Gt = 10000;
t_span  = [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_span, 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_span, '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_span,...
 Ga_target, Ga, param_to_tune);

% Simulation Results of Using the Estimated Parameter Value
%
% Use the estimated value of kGd and see how it affects simulation
% results. Before changing the value, save it to reset it later. We
% will also compute the new R-square value.
%
kGd0 = get(param_to_tune, 'Value');
set(param_to_tune, 'Value', k_new1);
simdata1 = sbiosimulate(modelobj);
[t1, Ga1] = selectbyname(simdata1,'Ga');
sse = result1.fval;
rsquare1 = 1-sse/sst;

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

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

% Reference
```

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

2 Click 

to run the task. The plot should look similar to this:



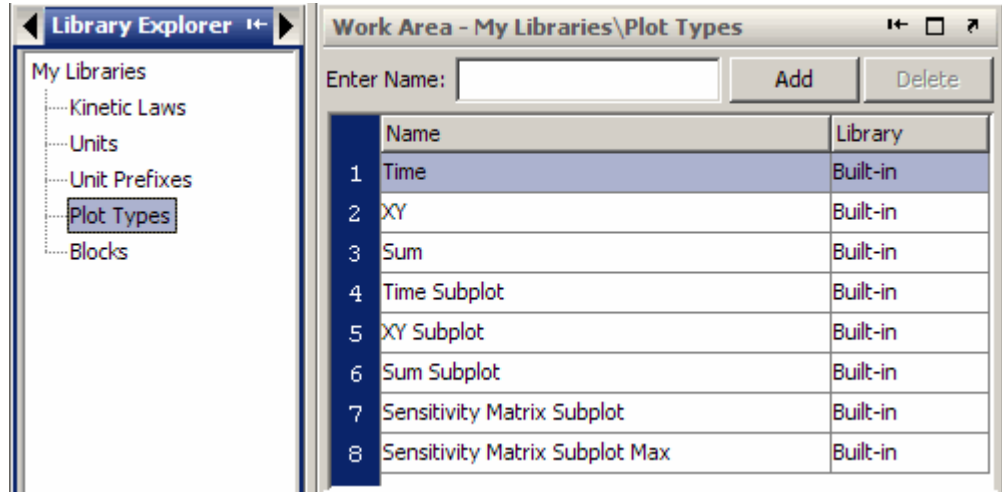
Visualizing Results Using Custom Plot Types

In this section...

About Plot Types

In the SimBiology desktop the **Plot Types** pane contains all the built-in and user-defined plot types. You can use the built-in plot types to visualize the data from any task. If you need custom visualization, you can either extend the built-in plot types or write your own M-code to create new plot types.

You can also specify additional inputs for plot types and define their types, default values, and ranges where applicable. For example, you can use this feature to extend the `Time` plot with an additional input to specify markers.



Summary of Built-In Plot Types

- **Time** — Specify one or more states to plot against time. By default all states are plotted against time, specified by the string `'<all>'`.

Data from all runs is plotted into a single set of axes. Use the Time Subplot type to plot data from each run into its own subplot.

- **XY** — Specify two states to plot against each other.

  Data from all runs is plotted into a single set of axes. Use the XY Subplot type to plot data from each run into its own subplot.

- **Sum** — Plot the sum of the simulation results of one or more states against time.

  Data from all runs is plotted into a single set of axes. Use the Sum Subplot type to plot data from each run into its own subplot.

- **Time Subplot** — Specify one or more states to plot against time. By default all states are plotted against time, specified by the string `'<all>'`.

  For each run the data is plotted into its own subplot.

- **XY Subplot** — Specify two states to plot against each other.

  For each run the data is plotted into its own subplot. Use the XY plot type to plot the data into one axes.

- **Sum Subplot** — Plot the sum of the simulation results of one or more states against time.

  For each run the data is plotted into its own subplot. Use the Sum plot type to plot the data into one axes.

- **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.

Creating and Using Custom Plot Types

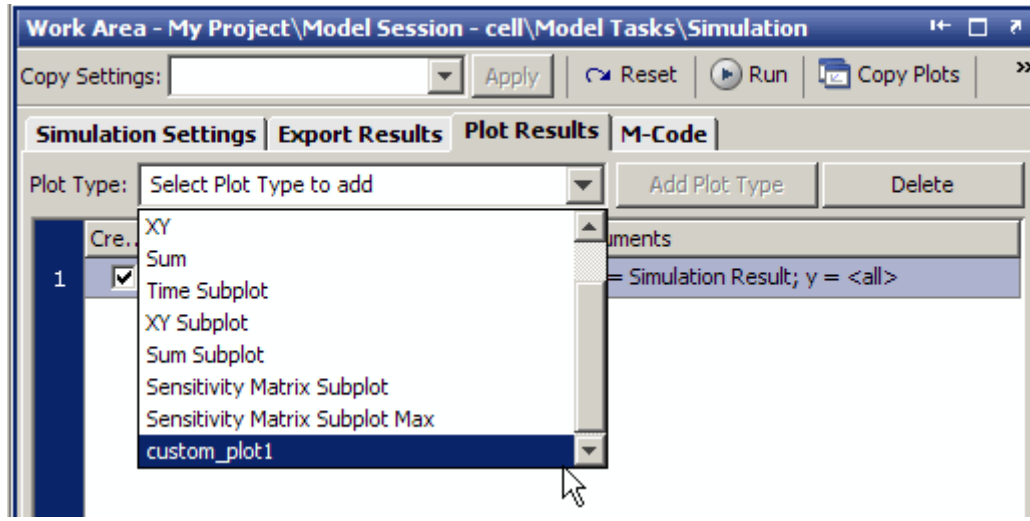
To use a plot type to visualize task results you have must create the custom plot in the **Library Explorer**.

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

2. Select **Plot Types**.

3 Select **Help > SimBiology Desktop Help** to see the context-sensitive help for information on how to create plots.

After you have created a plot, it is available for use in any **Plot Results** tab. For example, the following figure shows how a custom plot that was created in the **Library Explorer** is added to the **Simulation** pane in the **Plot Results** tab.



For more examples, see the following topics in the *SimBiology Model Reference*:

- For an example showing how to create a plot types, see “Creating a Custom Plot-Type to View Simulation Results”.
- For an example showing how to use a plot type in a task see “Visualizing Results for the Mutant Strain Using a Custom Plot”.

Index

P

R

S

Y