SimBiology 2
User's Guide

MATLAB®

The MathWorks

Accelerating the pace of engineering and science
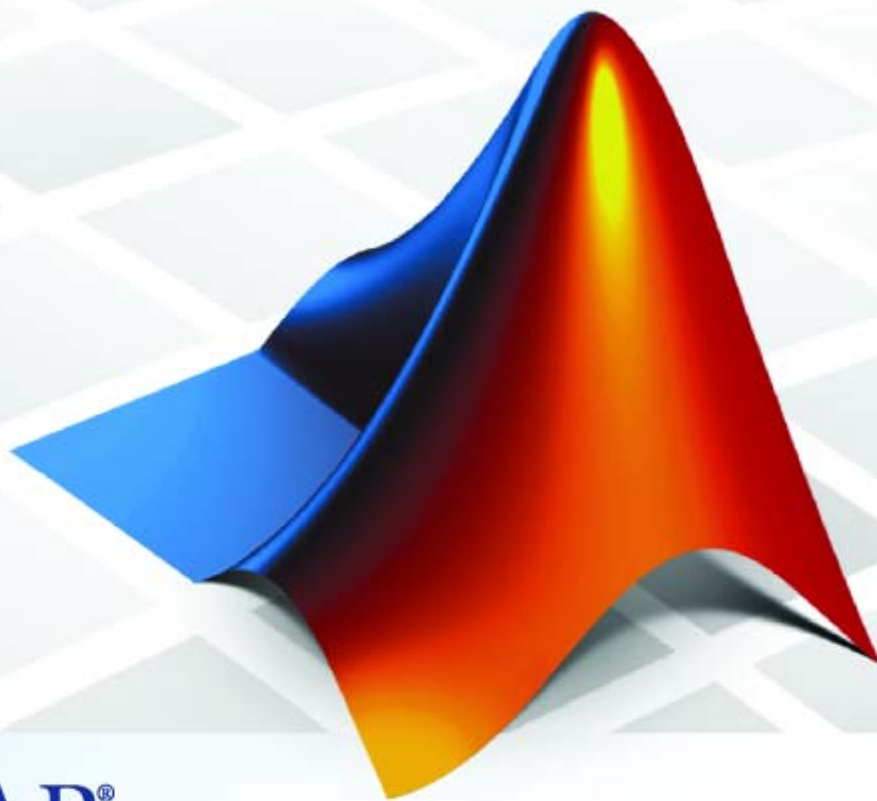
How to Contact The MathWorks

	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, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, and xPC TargetBox are registered trademarks, and SimBiology, SimEvents, and SimHydraulics are trademarks of The MathWorks, Inc.

Other product or brand names are 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)

Contents

Simulation

2

Analysis

3

Index

Modeling

This chapter describes how you can use SimBiology to model biological processes. It begins with the familiar concepts of mass action and enzyme kinetics.

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 (p. 1-3)	Reaction rate does not depend on the concentration of the reactants.
First-Order Reactions (p. 1-4)	Reaction rate is proportional to the concentration of a single reactant.
Second-Order Reactions (p. 1-5)	Reaction rate is proportional to the concentration of two reactants or the square of a single reactant.
Reversible Mass Action (p. 1-7)	Total reaction rate is the difference between the forward and reverse reaction rates.

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: R = 10 mole
                P =  0 mole
    parameters: k =  1 mole/(liter*second)
```

Entering the reaction above into SimBiology and simulating produces the following result:
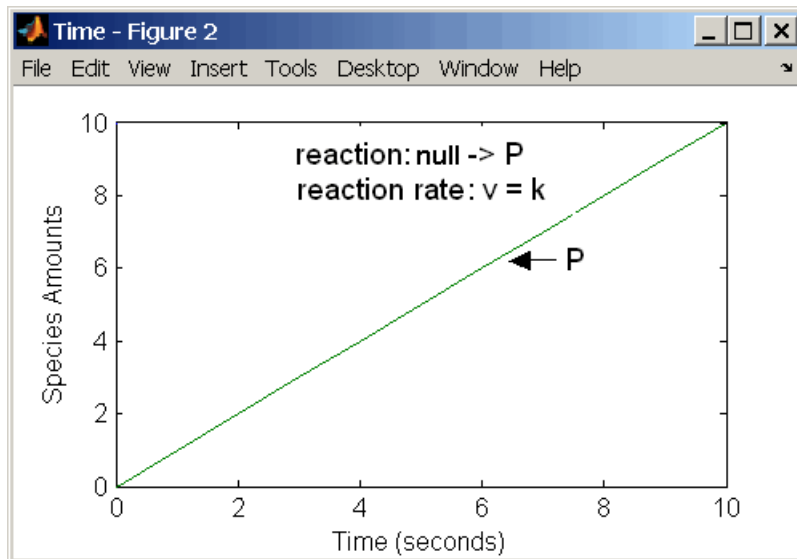
Zero-Order Mass Action Kinetics

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

First-Order Reactions

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

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

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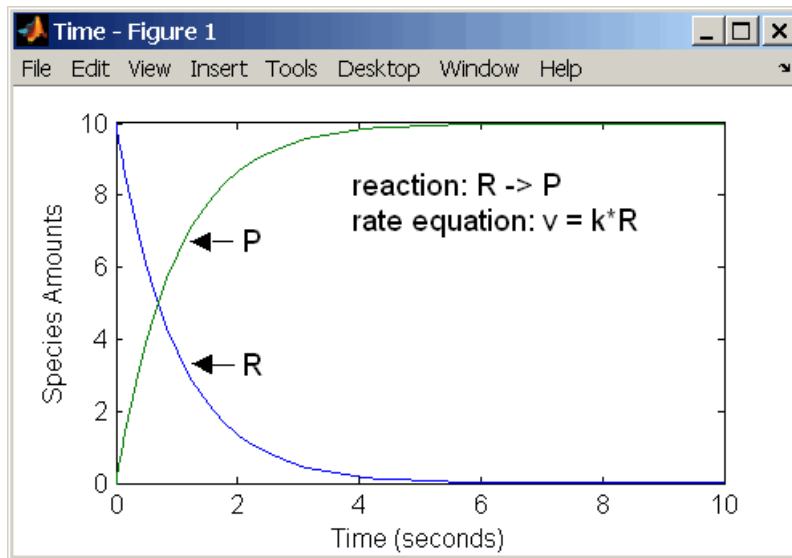
First-Order Mass Action Kinetics

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 SimBiology and simulating produces the following results:
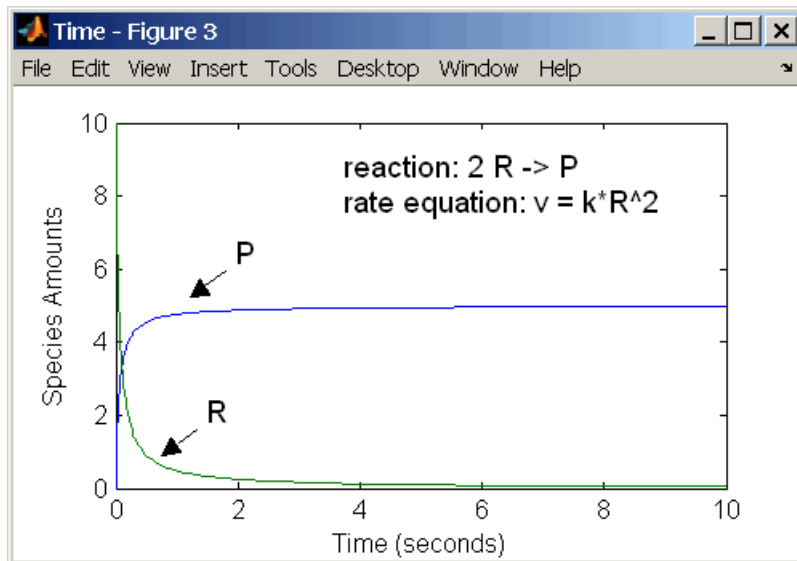
Second-Order Kinetics with Single Reactant

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 SimBiology 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.
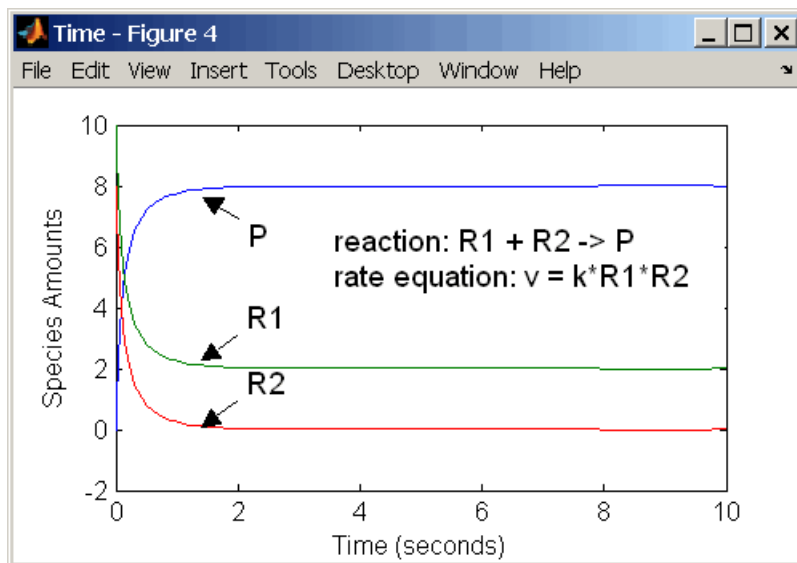
Second-Order Kinetics with Two Reactants

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 SimBiology 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`.
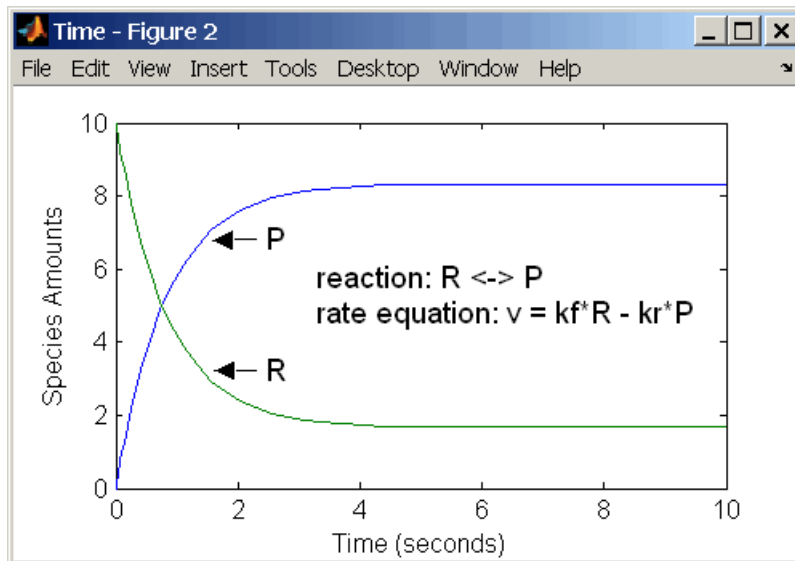
Enzyme Kinetics

Enzymes can increase the rate of a reaction by using a reaction mechanism or pathway with a lower activation energy. This section describes a common single substrate enzyme model using a mass action mechanism or rate equations derived from mass action mechanisms.

Simple Model for Single Substrate Catalyzed Reactions (p. 1-8)	Model for a single substrate reaction catalyzed irreversibly by an enzyme
Enzyme Reactions with Differential Rate Equations (p. 1-9)	Model reactions with differential rate equations derived from the reactions and reaction rates
Enzyme Reactions with Mass Action Kinetics (p. 1-11)	Model reactions directly with their reaction rate equations
Enzyme Reactions with Irreversible Henri-Michaelis-Menten Kinetics (p. 1-12)	Model a mass action mechanism with a derived kinetic equation

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.

$$S + E \xrightleftharpoons[k1r]{k1} ES \xrightarrow{k2} E + P$$
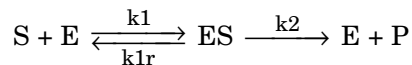
$$v1 = k1[S][E], \quad v1r = k1r[ES], \quad v2 = k2[ES]$$

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 1-8) can be described with differential rate equations. You can enter the differential rate equations into SimBiology 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).
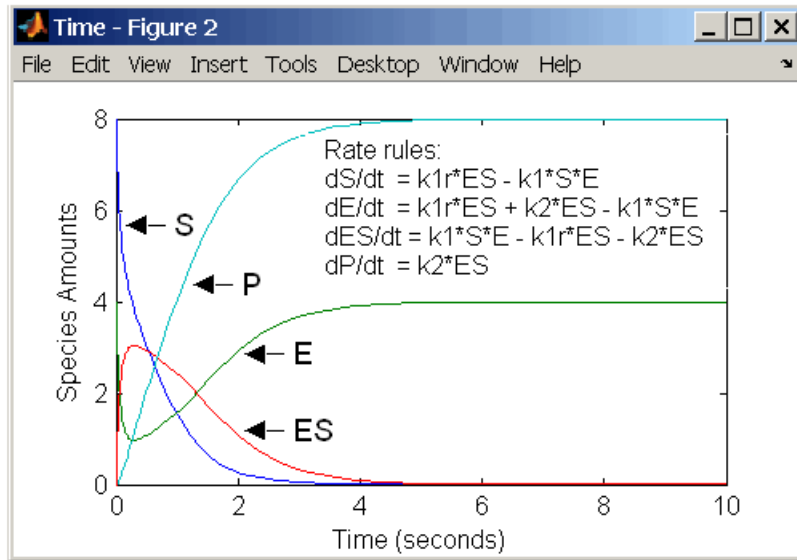
Alternatively, you could remove the rate rule for ES, add a new species Etotal for the total amount of enzyme, and add an algebraic rule 0 = Etotal - E - ES, where the initial amounts for Etotal and E are equal.

```
       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 SimBiology. 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 1-8.

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

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

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

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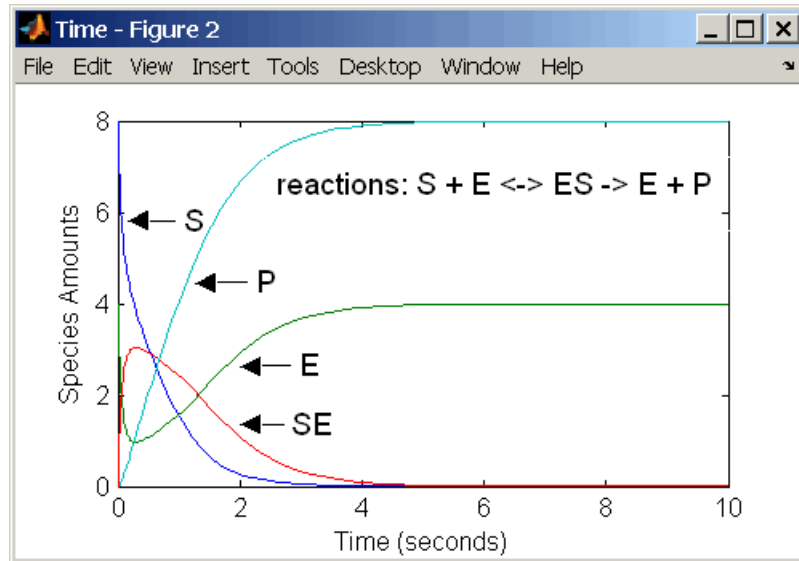
Enzyme Reactions with Irreversible Henri-Michaelis-Menten Kinetics

Representing an enzyme-catalyzed reaction with mass action kinetics requires you to know the rate constants k1, k1r, and k2. However, these rate constants are rarely reported in the literature. It is more common to give the rate constants for Henri-Michaelis-Menten kinetics with the maximum velocity Vm=k2*E and the constant Km = (k1r + k2)/k1. The reaction rate for a single substrate enzyme reaction using Henri-Michaelis-Menten kinetics is given below. For information about the model, see "Simple Model for Single Substrate Catalyzed Reactions" on page 1-8.

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

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 SimBiology 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
Henri-Michaelis-Menten rate equation.
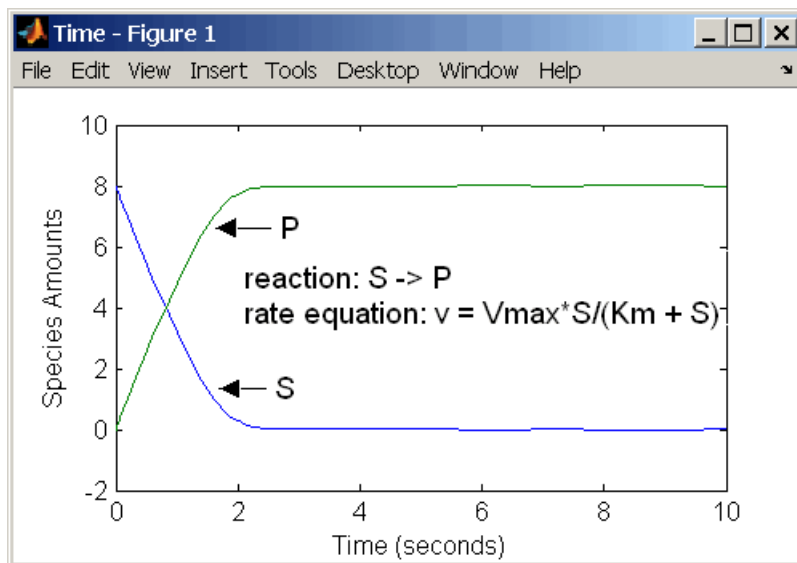
Constant Amount and Boundary Condition

SimBiology has two properties (constant amount, boundary condition) to specify how the amount of a species changes or does not change during a simulation.

Definition of Constant and Boundary 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.

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
```

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 can 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 in SimBiology 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 Henri-Michaelis-Menten rate equation with Vmax specified as a parameter assumes that the amount of enzyme catalyzing the reaction remains constant.

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

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,

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

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.

Parameter and Scope

A *parameter* is a quantity that can change or can be constant. In SimBiology, parameters are generally used to define rate constants.

Definition of Parameter Scope

A 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 rule, any submodel, 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.

For reaction rate, SimBiology hierarchically uses the value of the parameter at the kinetic law level first. If no such parameter is at the kinetic law level, SimBiology 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, SimBiology uses the value of the parameter at the kinetic

law level for the reaction rate. SimBiology uses the value of the parameter at the model level for any rules or submodels 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 Rules

When you want to refer to a parameter in a rule expression, in submodels, 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, add the new parameter, and then configure the reaction to use this new parameter. For help with this procedure, see "Changing the Scope of a Parameter" on page 1-20.

Note To vary a parameter with a **Rule**, 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. SimBiology hierarchically uses the value of the parameter at the kinetic law level first. If no such parameter is at the kinetic law level, SimBiology 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, you can create a new parameter with scope at the model level and simultaneously configure the reaction to use the new parameter.

Changing the Scope of a Parameter Without Changing the Name

The following procedure assumes that you want to replace the parameter at the kinetic law level with another parameter with the identical name. Because SimBiology references a parameter by its name, this procedure enables you to make a direct replacement.

1 In the **Project Explorer**, click **Parameters**.

2 In the **Project Settings-Parameters** pane, click the **Enter name** box and enter the name of the parameter exactly as previously specified, and then click the **Add** button.

```
K_LactoseBinding
```

SimBiology adds the parameter to the model level by default.

3 Double-click the **Value** cell and enter a value for the parameter.

```
1.0
```

4 Type a valid unit in the **ValueUnits** box.

```
1/mole*second
1/molecule*second
```

Valid Unit and Units with Prefix Examples

```
second
mole
molecule
1/(mole*second)
1/(molecule*second)

microsecond
millimole
molecule
1/(micromole*second)
1/(molecule*second)
```

5 Select the parameter previously specified at the kinetic law level (denoted by the reaction in the **Scope** column), and click **Delete**.

Changing the Scope and Name of a Parameter

1 In the **Project Explorer**, click **Reactions**.

2 In the **Project Settings-Reactions** pane, select the reaction that uses the parameter you want to change.

3 Click the **Kinetic Law** tab.

4 In the **Specify Corresponding Parameter Names** box, locate the parameter you want to replace and click **New**. The New Parameter dialog box opens.

   a In the **Name** box, specify the name of the new parameter. You can use the same name; however, you must then follow step 7 and delete the parameter that is at the kinetic law level.

   b From the **Scope** list, select

```
Model
```

   c Click **Create**.

5 Click the **Rate Parameters** tab, then double-click the **Value** cell and enter a value for the parameter.

```
1.0
```

6 Type a valid unit in the **ValueUnits** box.

```
1/moles*second

1/molecule*second
```

Valid Unit and Units with Prefix Examples

```
second
mole
molecule
1/(mole*second)
1/(molecule*second)

microsecond
```

```
millimole
molecule
1/(micromole*second)
1/(molecule*second)
```

7 (Optional) Select the parameter previously specified at the kinetic law level (denoted by the reaction in the **Scope** column), and click **Delete**.

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

Algebraic Rules

An algebraic rule is a model component that defines the value for a nonconstant parameter or the amount of a species that is determined through a algebraic equation instead of a differential relationship.

What Is an Algebraic Rule? (p. 1-24)	Define changes in species amounts and parameters values without using a reaction.
Mass Balance Equations (p. 1-24)	Use mass balance equations with reactions to define species amounts.

What Is an Algebraic Rule?

An algebraic rule is an equation that defines the value of a variable that you may not be able to define with a reaction. Use algebraic rules for defining equity constraints that are not rates of change.

There are two types of rules that are evaluated at each time step during a simulation. The first is a rate rule (see "Rate Rules" on page 1-26) while the second is an algebraic rule. An algebraic rule is defined by the equation

```
O = f(W) - x
```

The variable x can be a species amount or parameter value. The function `f(W)` is an expression that can include other species and parameters. In SimBiology, you enter an algebraic rule using the form

```
f(W) - x
```

Mass Balance Equations

There are some models in the literature that are defined with differential rate equations and algebraic mass balance 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 active form to an inactive form and back. The amount of inactive enzyme Ei is defined by the differential rate equation

`dEi/dt = Vm*Ei/Km + Ei`. If the total amount of the enzyme is known or remains constant, the total amount of enzyme `Ea` can be defined with the algebraic equation `Ea = Et - Ei` instead of a differential equation.

With SimBiology, models are defined by reactions, and the corresponding differential rate equations are calculated for all species. Adding a mass balance equation as an algebraic rule, and setting `Et` to be constant, would overdefine the model and cause a simulation error (the number of equations cannot be greater then the number of independent variables). If want to use a mass balance equation, you have to let `Et` vary, then `Et` is an independent variable that is not defined by a reaction and the simulation works.

Rate Rules

A rule is an model component that defines the value for a parameter or the amount of a species. Use algebraic rules for equations that are not rates of change. Use rate rules for equations that determine the rate of change for a parameter value or species amount.

What Is a Rate Rule?

A rule is an equation that defines the value for a variable. For species, use rate rules as an alternative to the differential rate expression generated from reactions.

There are two types of rules that are evaluated at each time step during a simulation. The first is an algebraic rule (see "Algebraic Rules" on page 1-24) while the second is a rate rule. A rate rule is defined by the equation

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

The variable x can be a species amount, parameter value, or compartment dimension (volume or area). The function f(W) is an expression that can include other species and parameters. In SimBiology, you enter a rate rule using the form

```
x = f(W)
```

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 rule. For example, the species c increases by a constant rate k. You could also include species and parameters that have their ConstantAmount or ConstantValue properties selected.

```
      reaction: none
rate equation: none
    rate rule: dc/dt = k
      species: c = O mole
   parameters: k = 1 mole/second
```

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

Enter the rule described above as c = k. From the **RuleType** list, select rate, enter the values for c and k, and then simulate.
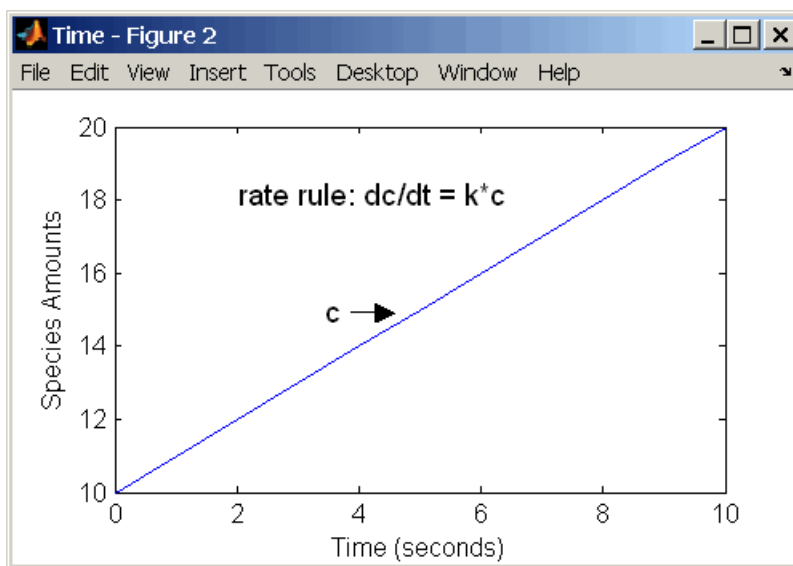
Alternatively, you could model a constant increase in a species with the reaction null -> C.

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, the species c decays exponentially. You could also include a parameter with its ConstantValue property cleared or set to false.

```
      reaction: none
rate equation: none
     rate rule: dc/dt = -k*c
       species: c = 10 mole
    parameters: k =  1 1/second
```

The solution for the rate rule dc/dt = -k*c is $c = c_0e^{-kt}$.

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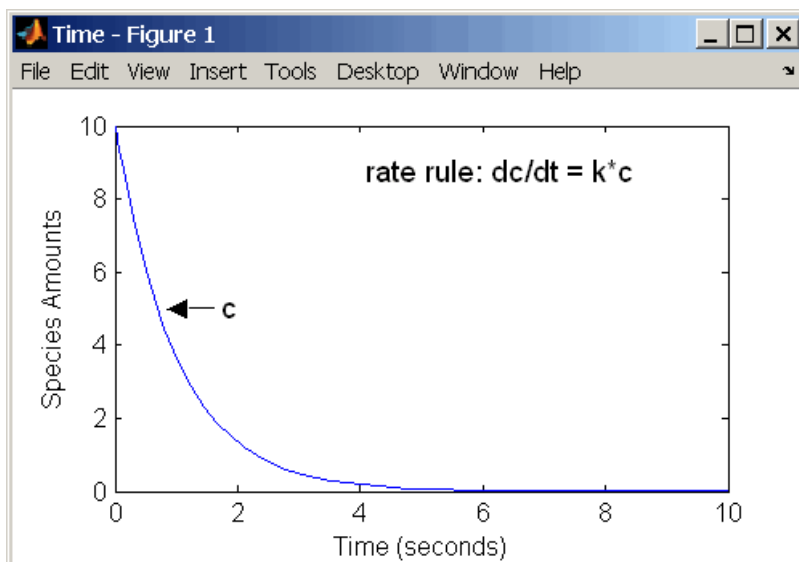
Notice that if the amount of a species c is determined by a rate rule and c is also in a reaction, c must have its property for BoundaryCondition selected. For example, with a reaction a -> c and a rate rule dc/dt = k*c, select the BoundaryCondtion for c so that a differential rate term is not created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

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. In a similar way, a species from a reaction can determine the rate of another species if it is in the rate rule that defines that other species.

```
      reaction: a -> b
rate equation: v = -k1*a
     rate rule: dc/dt = k2*a
       species: a = 10 mole
                b =  0 mole
                c =  5 mole
    parameters: k1 = 1 1/second
                k2 = 1 1/second
```

The solution for the species in the reaction are

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

With the rate rule $dc/dt = k_2 * a$ dependent on the reaction, $dc/dt = k_2(a_0 e^{-k1t})$, and the solution is

```
c = c₀ + k₂a₀/k₁(1 - e⁻ᵏ¹ᵗ)
```
$$c = c_0 + k_2 a_0 / k_1 (1 - e^{-k_1 t})$$

Enter the reaction and rule described above and simulate.
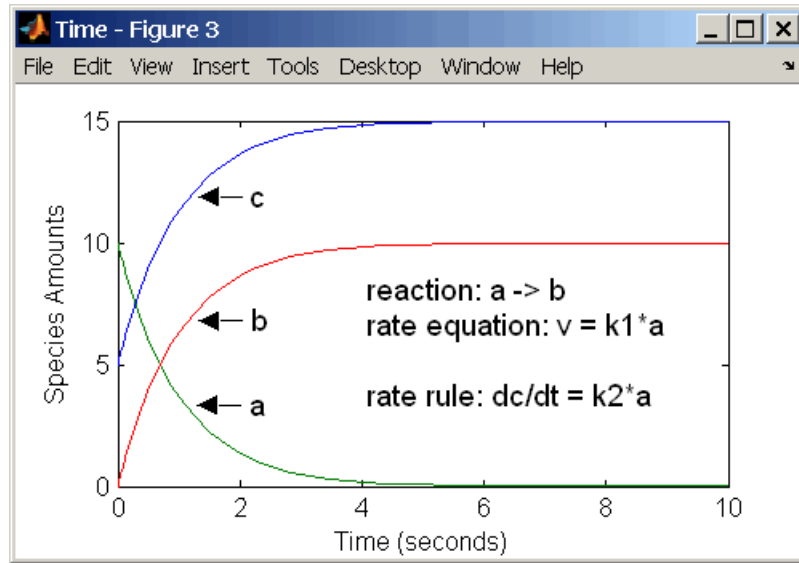
Differential Rate Equations as Rules

Many mathematical models in the literature are described with differential rate equations for the species. You could manually convert the equations to reactions, or you could enter the equations as rate rules. For example, you could enter the following differential rate equation for a species C,

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

as a rate rule in SimBiology:

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

2

Simulation

Simulation Overview

Simulation Settings

The SimBiology integrated desktop environment provides convenient access to the configuration sets for simulations. In the **Project Explorer** select the **Simulation Settings** node where you can set, change, and save simulation parameters. You can run simulations, plot simulation results, configure data logging, and export simulation data.

SimBiology

File Edit Simulate Data Plot types D

Current Model: untitled Si

SolverType: ode15s ▼ Stop Time: 1

Project Explorer ⤢ ✕

My Project
  Model Session - untitled
    SimBiology Model
      Species
      Reactions
      Parameters
      Rules
      Submodels
    Simulation And Analysis
      Simulation Settings: default (a
      Data
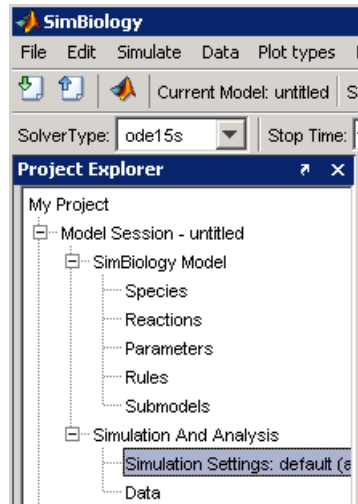
When you save simulation settings a new item containing the saved settings appears in the **Project Explorer** under **Simulation Settings**. Double-click the saved simulation setting to open the pane.

Where to Find Simulation Settings Controls

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

- Use the **Solver** tab to set the simulation solver and timing parameters for the currently selected model. The common simulation properties are also accessible in the simulation toolbar.

- Use the **Data Logging** tab to choose which species to log and how often.

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

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 2-7.

- ode45 — Best first guess.
- 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 2-8.

- ode15s — Try first if you suspect that a problem is stiff, or if ode45 failed or was very inefficient.
- 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 2-10.

- 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, SimBiology ignores any rate, assignment, or algebraic rules if present in the model.

Nonstiff Deterministic Solvers

Models with either all fast or all slow changing variables may not be numerically stiff. Nonstiff deterministic solvers are appropriate to try.

ode45 (Dormand-Prince) (p. 2-7)	One-step solver based on an explicit Runge-Kutta (4,5) formula
ode23 (Bogacki-Shampine) (p. 2-7)	One-step solver based on an explicit Runge-Kutta (2,3) formula
ode113 (Adams) (p. 2-7)	Variable order Adams-Bashforth-Moulton PECE solver

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.

Stiff Deterministic Solvers

Models with fast and slow changing variables are numerically stiff. Stiff deterministic solvers are the best choice.

ode15s (stiff/NDF) (p. 2-8)	Variable order solver based on the numerical differentiation formulas
ode23s (stiff/Mod. Rosenbrock) (p. 2-8)	One-step solver based on a modified Rosenbrock formula of order 2
ode23t (Mode. stiff/Trapezoidal) (p. 2-8)	An implementation of the trapezoidal rule
ode23tb (stiff/TR-BDF2) (p. 2-9)	An implementation of an implicit Runge-Kutta formula

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.

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, SimBiology ignores any rate, assignment, or algebraic rules if present in the model.

Stochastic Simulation Algorithm (SSA) (p. 2-10)	Solver uses an exact stochastic simulation algorithm that simulates one reaction at a time based on the propensity function for each reaction
Explicit Tau-Leaping Algorithm (p. 2-11)	Solver is based on an approximation algorithm that speeds up simulation at the cost of accuracy; the degree of accuracy depends on a user-specified error tolerance
Implicit Tau-Leaping Algorithm (p. 2-12)	Similar to Explicit Tau-Leaping Algorithm with better stability for numerically stiff problems
Ensemble Runs of Stochastic Simulations (p. 2-12)	Functions for performing ensemble runs for stochastic simulations
References (p. 2-13)	Stochastic solver references

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

You can perform ensemble simulations using the stochastic solvers to gather data from multiple stochastic runs of the model. The following functions let you perform ensemble runs:

- `sbioensemblerun` – Performs a stochastic ensemble run of the SimBiology 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`.

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.

3

Analysis

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

Sensitivity Analysis (p. 3-2)

Calculating the sensitivities of species with respect to species initial conditions and parameter values in the model

Parameter Estimation (p. 3-12)

Estimating missing parameters or optimizing existing parameters

Moiety Conservation (p. 3-28)

Analyzing conservation relationships in a model

Importing and Exporting Model Component Data (p. 3-38)

Importing and exporting lists of species, reactions, parameters, and rules to and from the SimBiology desktop

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. Sensitivity analysis is supported only by the ODE solvers.

Sensitivity Analysis in SimBiology

In SimBiology 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 `StatesToLog` property with respect to the initial conditions of the species specified in `SpeciesInputFactors` and the values of the parameters specified in `ParameterInputFactors` that you specify in the `SensitivityAnalysisOptions` property of the configuration set object.

- `SensitivityAnalysisOptions` – An object that holds the sensitivity analysis options in the configuration set object. Properties of `SensitivityAnalysisOptions` are summarized below:

- ■ `SpeciesInputFactors` – Specify the species with respect to which you want to compute the sensitivities of the species states in your model. Sensitivities are calculated with respect to the initial conditions of the specified species.

- ■ `ParameterInputFactors` – Specify the parameters with respect to which you want to compute the sensitivities of the species states in your model. Sensitivities are calculated with respect to the values of the specified parameters.

- ■ `Normalization` – Specify the normalization for the calculated sensitivities.

  - • `'None'` specifies no normalization.

  - • `'Half'` specifies normalization relative to the numerator (species quantity) only.

  - • `'Full'` specifies full dedimensionalization.

Sensitivity Analysis Example Using a G Protein Model

This example uses a G protein model built shown in the "Model of the Yeast Heterotrimeric G Protein Cycle " example to illustrate sensitivity analysis options in SimBiology.

You can also access two demos that show you sensitivity analysis of this model by typing the following at the command line:

```
gprotein_paramestim
```

(for sensitivity analysis followed by parameter estimation)

```
gprotein_paramscan
```

(for sensitivity analysis followed by parameter scanning)

Loading and Exploring the Model

1 Load the G protein model for the wild-type strain.

```
sbioloadproject gprotein_norules m1
```

The project gprotein_norules.sbproj contains two models, one for the wild-type strain, and one for the mutant strain. If you previously know that the model you want to load from the project is the first one, you can specify m1 as shown above. If you do not have any knowledge of the project contents, you can either load it into the SimBiology desktop and explore the project on the GUI or, at the command line type,

```
sbioloadproject ProjectName
```

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

2 Type the object name.

```
m1
```

MATLAB returns model information, for example:

```
SimBiology Model - Yeast_G_Protein_wt

   Model Components:
     Models:           0
     Parameters:       8
     Reactions:        6
     Rules:            0
     Species:          7
```

3 Display the species information.

```
m1.Species

Species Object Array

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

4 Display reaction information.

```
m1.Reactions

Reaction Object 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
```

5 By convention the G protein example uses the object name `wtmodelObj` to refer to the model object for the wild-type strain. To use this convention, type the following:

```
wtModelObj = m1;
```

Note `m1` and `wtModelObj` are equivalent; they point to the same object. If you change one, the other is changed.

You can see that the two objects are equivalent in "Setting the Configuration Set Object for Sensitivity Analysis" on page 3-5 where you set the states that SimBiology should return the data for, during simulation.

Setting the Configuration Set Object for Sensitivity Analysis

The configuration set object holds the options for simulations in SimBiology. In the configuration set object, you can specify the following:

- The type of solver to use for the simulation

- Stop time of the simulation

- The solver options

- States whose data is logged for you during the simulation

- Whether SimBiology should perform unit conversion and dimensional analysis

- The input factors for sensitivity analysis, and the type of normalization for the sensitivity data

This example shows you how to calculate and visualize the sensitivity data for one species in the model, active G protein (Ga):

1 Retrieve the configuration set object from the model, and change the StopTime for the simulation.

```
csObj = getconfigset(wtModelObj);
set(csObj, 'StopTime', 600);

csObj

  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:
    ParameterInputFactors:      0
    SpeciesInputFactors:        0
```

2 As an exercise to see that m1 and wtmodelObj are equivalent in SimBiology, retrieve the configuration set object for m1

```
m1csObj = getconfigset(m1);
```

```
% display the StatesToLog for m1csobj
m1csObj.RunTimeOptions.StatesToLog

Species Object Array

   Index:    Name:      InitialAmount:    InitialAmountUnits:
   1         R          10000
   2         G          7000
   3         Gd         3000
   4         freeGbg    3000
   5         Ga         0
   6         RL         0


% Display the StatesToLog for csObj
csObj.RunTimeOptions.StatesToLog

Species Object Array

   Index:    Name:      InitialAmount:    InitialAmountUnits:
   1         R          10000
   2         G          7000
   3         Gd         3000
   4         freeGbg    3000
   5         Ga         0
   6         RL         0
```

3 Change the StatesToLog settings to log only Ga in wtModelObj.

```
csObj.RunTimeOptions.StatesToLog = sbioselect...
    (wtModelObj, 'Type', 'species', 'Where', 'Name', '==', 'Ga');
```

4 Display the logged states for the configuration set in wtModelObj and in m1.

```
csObj.RunTimeOptions.StatesToLog

Species Object Array

   Index:   Name:    InitialAmount:    InitialAmountUnits:
   1        Ga       0

m1csObj.RunTimeOptions.StatesToLog
```

```
Species Object Array

    Index:      Name:      InitialAmount:      InitialAmountUnits:
    1           Ga         0
```

As shown above, the objects m1 and wtModelObj are equivalent, and the objects m1csObj and csObj are equivalent.

Enabling and Setting Sensitivity Analysis Options

To calculate the sensitivity of a species, first enable sensitivity analysis in the configuration set object (csObj) by setting the SensitivityAnalysis option to true.

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

In this example, there is only one configuration set object (csObj). You can, however, have multiple configuration set objects in a model, but only one configuration set can be active at a time. You could have more than one configuration set object, each of which holds a different configuration for simulation; for example, different solver options, different options for sensitivity, and so on.

Setting Sensitivity Analysis Options

The SensitivityAnalysisOptions property holds the input factors that you want to specify, and the type of normalization in use for sensitivity calculations. This example uses all the parameters in the G protein model as input factors for sensitivity analysis. Further, the data is fully normalized and therefore made dimensionless to facilitate the comparison.

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

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

2 Set the ParameterInputFactors property of the SensitivityAnalysisOptions object.

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

3 Set the Normalization property of the SensitivityAnalysisOptions object to perform 'Full' normalization. Often sensitivity numbers are

so wide ranging that it is hard to compare the data. Full normalization enables more meaningful comparisons.

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

Simulating with Sensitivity Analysis Enabled

1 Since the initial amount of Ga is set to 0.0 and the data is fully normalized, there is an expected divide by zero warning. You can turn this off before simulating, by typing the following command:

```
warning('off','MATLAB:divideByZero');
```

2 Simulate the model and return the data to a time series object (tsObj).

```
tsObj = sbiosimulate(wtModelObj);
```

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 you 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] = sbiogetsensmatrix(tsObj);
```

2 Reshape R to facilitate visualization and plotting.

a Note the size of R.

```
size(R)

342     1     8
```

MATLAB indicates that R is a 342X1X8 matrix, where the time data = size(R,1) = 342, the StatesToLog = size (R,2) = 1, and the number of input factors is size(R,3) = 8.

    b Reshape the matrix such that the data is organized into 8 columns (for the 8 parameter input factors).

```
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
legend(ifacs);
```
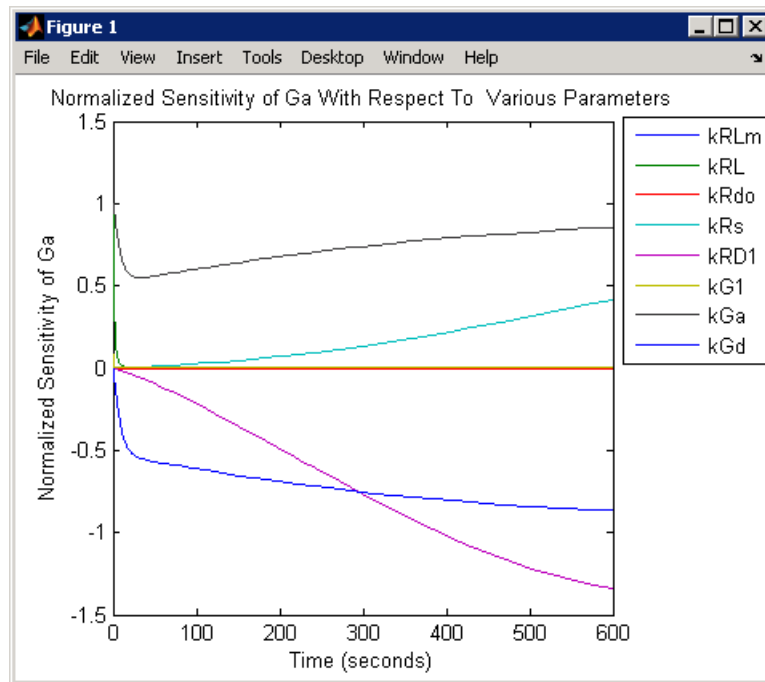
From the previous plot you can see that Ga is sensitive to parameters kGd, kRs, kRD1, and kGa. The example for parameter estimation uses this data to illustrate how you can estimate parameters in your model.

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 .

Parameter Estimation in SimBiology

You can estimate a single parameter or all parameters in your model using the `sbioparamestim` function in SimBiology. SimBiology uses the optimization functions in MATLAB, Optimization Toolbox, and Genetic Algorithm and Direct Search Toolbox to enable parameter 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. See sbioparamestim in the SimBiology Reference for more information.

Parameter Estimation Example Using a G Protein Model

This example uses a G protein model built in the "Model of the Yeast Heterotrimeric G Protein Cycle " tutorial to illustrate parameter estimation in SimBiology. 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 "Sensitivity Analysis Example Using a G Protein Model" on page 3-3, 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_paramestim
```

Loading and Exploring the Model

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

```
sbioloadproject gprotein_norules m1
```

2 Type the object name that you see in the workspace.

```
m1
```

MATLAB returns model information, for example:

```
SimBiology Model - Yeast_G_Protein_wt

   Model Components:
     Models:          0
     Parameters:      8
     Reactions:       6
     Rules:           0
     Species:         7
```

3 Display the species information.

```
m1.Species

Species Object Array

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

4 Display reaction information.

```
m1.Reactions

Reaction Object 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
```

5 By convention the G protein example uses the object name `wtmodelObj` to refer to the model object for the wild-type strain. To use this convention, type the following:

```
wtModelObj = m1;
```

Note `m1` and `wtModelObj` are equivalent; they point to the same object. If you change one, the other is changed.

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 "Importing Data to MATLAB", 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.

```
wtModelObj.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:
                  ParameterInputFactors:      0
                  SpeciesInputFactors:        0
```

The model configuration set has StopTime set to 600 seconds.

2 Simulate the model and return the results to a time series object.

```
   ts_orig = sbiosimulate(wtModelObj);
```

3 Retrieve the time and state data.

```
   [t_orig, Ga_orig] = sbiogetnamedstate(ts_orig,'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 "Evaluating the Goodness of Fit" 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');
```
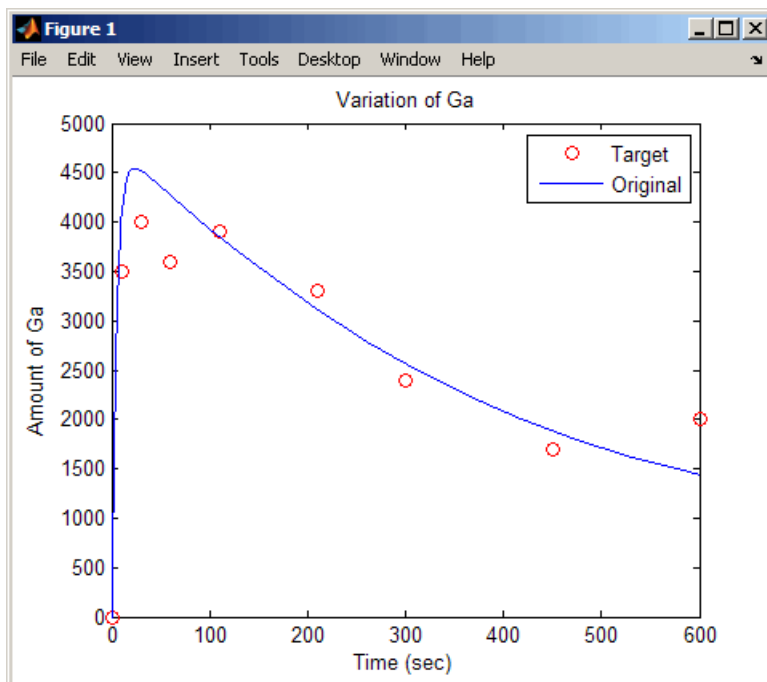
Leave this figure window open so that you can use it to plot and compare results of using the estimated parameters later in this example.

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(wtModelObj,'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(wtModelObj, t_span, ...
   Ga_target, Ga, param_to_tune, {}, {'lsqcurvefit',opt1});
```

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(wtModelObj, ...
   t_span, Ga_target, Ga, param_to_tune, {}, {'fminsearch',opt1});
```

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

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.

```
ts1 = sbiosimulate(m1);
[t1, Ga1] = sbiogetnamedstate(ts1,'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');
```
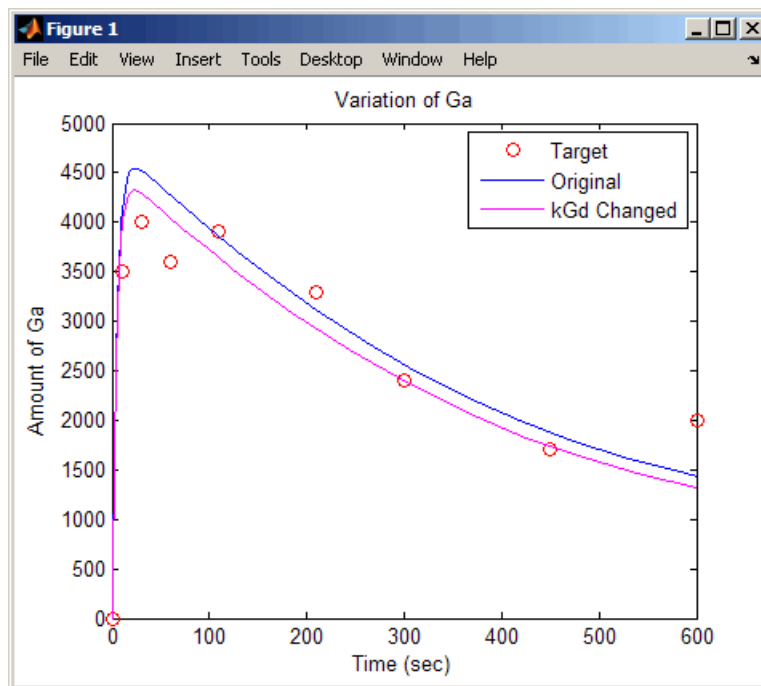

The figure shows the best fit achieved by changing the parameter `kGd`.

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 Other Parameters in the G Protein Model

The example illustrating sensitivity analysis ("Sensitivity Analysis Example Using a G Protein Model" on page 3-3) showed that Ga is sensitive to parameters kGd, kRs, kRD1, and kGa. Based on this data, this tutorial shows you how to estimate these parameters. The sensitivity data is presented in "Extracting and Plotting Sensitivity Data" on page 3-9.

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')
```

Parameter Object 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(wtModelObj, t_span,...
   Ga_target, Ga, param_to_tune, {}, {'lsqcurvefit',opt2});
```

Iteration	Func-count	f(x)	Norm of step	First-order 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(wtModelObj, 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

   Parameter Object 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 =

    8.8253
    0.0041
    0.0000
    0.1229
```

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

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


rsquare2 =

    0.9583
```

Simulating and Plotting Results Using Estimated Parameter Values

Now that the estimated parameters values are known, set the parameter values in the model to the new values, simulate, and compare the results with the original values.

1 Display the model's current parameter values.

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

  Parameter Object 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       0.120862
```

2 Set the parameters in the model to the estimated values.

```
for i=1:length(k_new2)
    set(param_to_tune(i), 'Value', k_new2(i));
end
```

3 Verify that the model now has new values for the parameters.

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

  Parameter Object Array

  Index:    Name:     Value:         ValueUnits:
  1         kRLm      0.01
  2         kRL       3.32e-018
  3         kRdo      0.0004
  4         kRs       8.8253
  5         kRD1      0.00413679
  6         kG1       1
  7         kGa       9.14712e-006
  8         kGd       0.122862
```
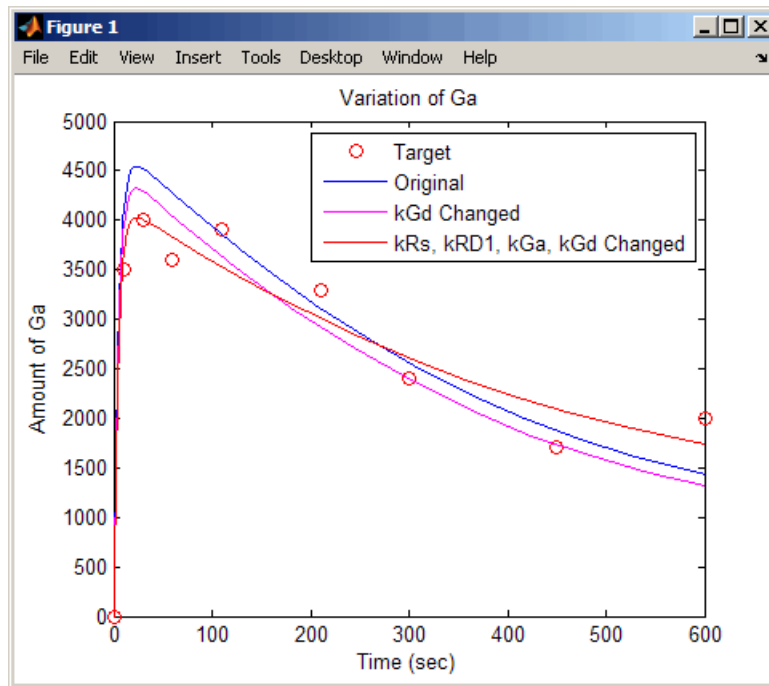
4 Simulate the model with the new values.

```
ts3 = sbiosimulate(m1);
```

```
[t3, Ga3] = sbiogetnamedstate(ts3,'Ga');
```

5 Compare the simulation results with the target experimental data.

```
plot(t3, Ga3, 'r-');
legend('Target', 'Original', 'kGd Changed', ...
       'kRs, kRD1, kGa, kGd Changed');
```


6 Evaluate the fit. Compare the improvement in R-square values.

```
disp('Original R-square value:');
fprintf('%g\n', rsquare_orig);
fprintf('\n');
disp('R-square value with one estimated parameter value:');
fprintf('%g\n', rsquare1);
fprintf('\n');
disp('R-square value with four estimated parameter values:');
fprintf('%g\n', rsquare2);
```

```
Original R-square value:
0.896737

R-square value with one estimated parameter value:
0.919485

R-square value with four estimated parameter values:
0.958255
```

In summary, this example showed you how to estimate parameters in a model, when given experimental target data, and how to compare the simulation results after each estimation, and how to evaluate the fit with R-square values.

Moiety Conservation

You can analyze conservation relationships in a model using the `sbiomoiety` function in SimBiology.

Introduction

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.

Finding Conserved Moieties with SimBiology

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" on page 3-28 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.

Examples of Determining Conserved Moieties

G Protein Example

1 Load the G protein model for the wild-type strain.

    sbioloadproject gprotein_norules

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

2 Type an object name that you see in the workspace.

    m1

MATLAB returns model information, for example:

    SimBiology Model - Yeast_G_Protein_wt

       Model Components:
         Models: 0
         Parameters: 8
         Reactions: 6

```
        Rules:              0
        Species:            7
```

3 Display the species information.

```
    m1.Species

    Species Object Array

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

4 Display reaction information.

```
    m1.Reactions

    Reaction Object 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
```

5 By convention, the G protein example uses the object name `wtmodelObj` to refer to the model object for the wild-type strain. To use this convention, type the following:

```
    wtModelObj = m1;
```

Note `m1` and `wtModelObj` are equivalent; they point to the same object. If you change one, the other is changed.

---

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

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

g =

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


sp =

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

7 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(wtModelObj,'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.

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

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

9 Show the list of independent species.

```
SI
```

```
SI =

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

10 Show the list of dependent species.

```
SD

SD =

    'freeGbg'
    'Ga'
```

11 Show the link matrix relating SD and SI.

```
L0


L0 =
(1,2)        -1
(2,2)        -1
(2,4)        -1
```

12 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
```

13 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 Explore the model.

```
m1
```

MATLAB returns model information, for example:

```
SimBiology Model - Goldbeter Mitotic Oscillator with reactions

   Model Components:
     Models:          0
     Parameters:      13
     Reactions:       7
     Rules:           4
     Species:         10
```

3 Display the species information.

```
m1.Species
```

```
Species Object Array

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

4 Display reaction information.

```
m1.Reactions

Reaction Object 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
```

5 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'
```

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

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

cons_rel =

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

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

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

8 Show the list of independent species.

```
SI

SI =

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

9 Show the list of dependent species.

```
SD

SD =

    'Mplus'
```

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

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

```
     L0

     L0 =

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

11 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
```

12 Show the dependent stoichiometry matrix, N_D.

```
     ND

     ND =

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

Importing and Exporting Model Component Data

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

Importing Model Component Data

You can import 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, select point to **Load Data from File** and select the component type, for example, **Species**. The Load Species from File dialog box opens.

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 or browse to select a file name.

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, the **Name** of the species 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, SimBiology 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 the Excel or text file. During importation, SimBiology will select the **ConstantAmount** check box for that species. The default is unchecked.

Exporting Model Component Data

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, **Species**. The Export Species to File dialog box opens.

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 or browse to select a file name.

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, the **Name** of the species 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.

Index