SimBiology®

User's Guide

MATLAB®

R2020a

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

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

SimBiology® User's Guide

Trademarks

Patents

Revision History

September 2005	Online only	New for Version 1.0 (Release 14SP3+)
March 2006	Online only	Updated for Version 1.0.1 (Release 2006a)
May 2006	Online only	Updated for Version 2.0 (Release 2006a+)
September 2006	Online only	Updated for Version 2.0.1 (Release 2006b)
March 2007	Online only	Rereleased for Version 2.1.1 (Release 2007a)
September 2007	Online only	Rereleased for Version 2.1.2 (Release 2007b)
October 2007	Online only	Updated for Version 2.2 (Release 2007b+)
March 2008	Online only	Updated for Version 2.3 (Release 2008a)
October 2008	Online only	Updated for Version 2.4 (Release 2008b)
March 2009	Online only	Updated for Version 3.0 (Release 2009a)
September 2009	Online only	Updated for Version 3.1 (Release 2009b)
March 2010	Online only	Updated for Version 3.2 (Release 2010a)
September 2010	Online only	Updated for Version 3.3 (Release 2010b)
April 2011	Online only	Updated for Version 3.4 (Release 2011a)
September 2011	Online only	Updated for Version 4.0 (Release 2011b)
March 2012	Online only	Updated for Version 4.1 (Release 2012a)
September 2012	Online only	Updated for Version 4.2 (Release 2012b)
March 2013	Online only	Updated for Version 4.3 (Release 2013a)
September 2013	Online only	Updated for Version 4.3.1 (Release 2013b)
March 2014	Online only	Updated for Version 5.0 (Release 2014a)
October 2014	Online only	Updated for Version 5.1 (Release 2014b)
March 2015	Online only	Updated for Version 5.2 (Release 2015a)
September 2015	Online only	Updated for Version 5.3 (Release 2015b)
March 2016	Online only	Updated for Version 5.4 (Release 2016a)
September 2016	Online only	Updated for Version 5.5 (Release 2016b)
March 2017	Online only	Updated for Version 5.6 (Release 2017a)
September 2017	Online only	Updated for Version 5.7 (Release 2017b)
March 2018	Online only	Updated for Version 5.8 (Release 2018a)
September 2018	Online only	Updated for Version 5.8.1 (Release 2018b)
March 2019	Online only	Updated for Version 5.8.2 (Release 2019a)
September 2019	Online only	Updated for Version 5.9 (Release 2019b)
March 2020	Online only	Updated for Version 5.10 (Release 2020a)

Contents

1 SimBiology Apps

2 Modeling

3 Structural Analysis

4 Simulation and Analysis

5 Pharmacokinetic Modeling

A Creating Reaction Rates

B Models Used in Examples

SimBiology Apps

Keyboard Shortcuts for SimBiology App

On Macintosh platforms, use the **command** key instead of **Ctrl**.

Shortcuts for the Diagram View

Action	Shortcut
Select all blocks	**Ctrl** + **A**
Highlight block in the Table view on page 1-20 or in the Browser on page 1-8 if open	**Ctrl** + **B**
Copy block	**Ctrl** + **C**
Duplicate block	**Ctrl** + **D**
Paste block	**Ctrl** + **V**
Print the diagram on page 1-4	**Ctrl** + **P**
Delete block	**Delete**
Move block up by 5 pixels	Up arrow
Move block down by 5 pixels	Down arrow
Move block left by 5 pixels	Left arrow
Move block right by 5 pixels	Right arrow
Move block by 1 pixel	**Ctrl** + any arrow key
Move block by 10 pixels	**Ctrl** + **Shift** + any arrow key
Pan the diagram	**Shift** + any arrow key
Zoom in	**Ctrl** + **+**
Zoom out	**Ctrl** + **-**
Toggle selection	**Ctrl** + **I**
Select the next cloned block	**N**
Select the previous cloned block	**P**
Move one or more blocks in a clockwise fashion	**Shift** + **U**
Move one or more blocks in an anti-clockwise fashion	**Ctrl** + **Shift** + **U**
Expand one or more blocks by increasing the distance between them	**Shift** + **E**
Collapse one or more blocks by decreasing the distance between them	**Ctrl** + **E**
Show usages of a quantity	**Ctrl** + **Shift** + **F**
Show search bar	**Ctrl** + **F**

See Also

More About

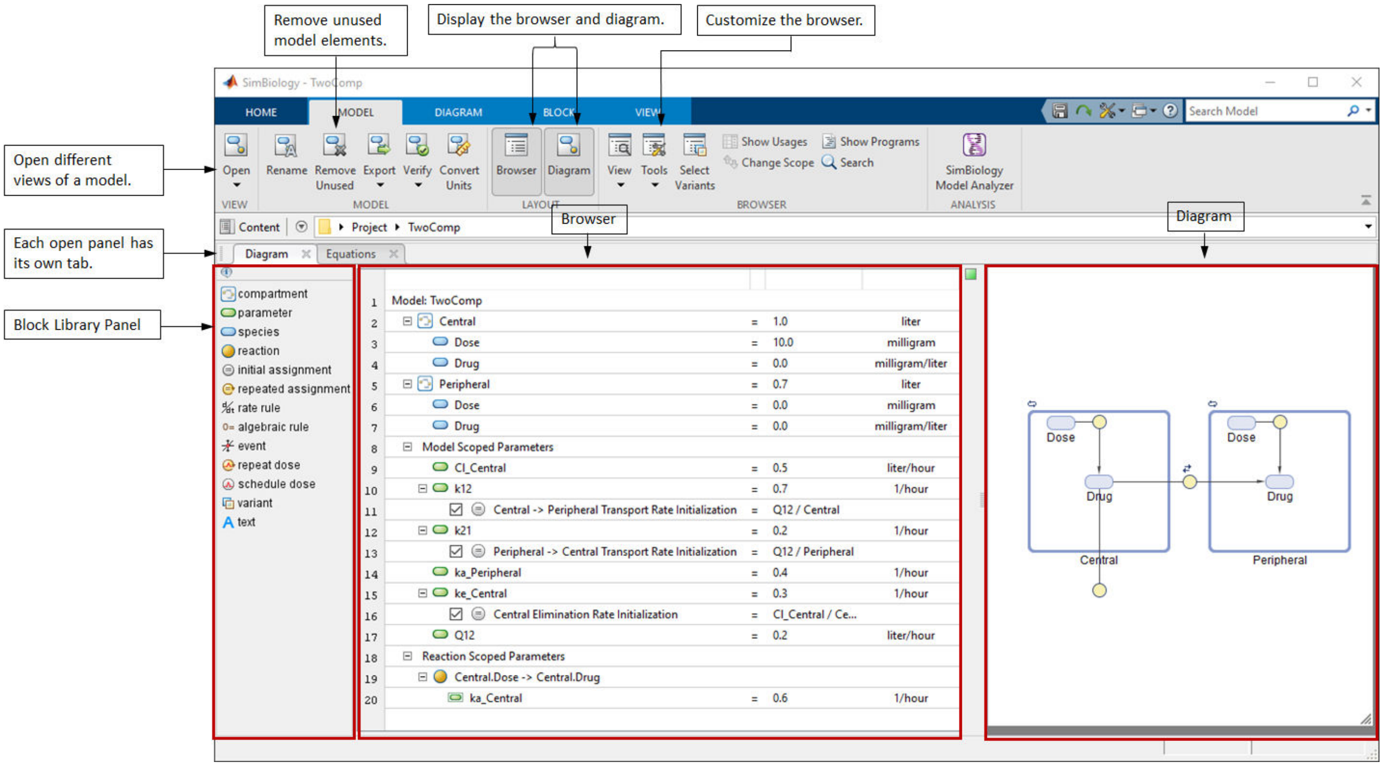
- “Diagram View” on page 1-4
- “Table View” on page 1-20

Model Views in SimBiology App

The SimBiology app provides three different views of a model on page 2-2: the Diagram view, Table view, and Equations view. The Diagram view shows a graphical representation of a model. It describes the model using a set of connected blocks. Blocks represent different modeling elements such as quantities and expressions, and the view shows the relationships between model elements graphically. The Table view displays modeling elements and their properties in a tabular form. You can use either view to build a model. The Equations view shows the differential equations and other expressions such as assignment rules of a model. This view does not let you edit the model. If you make a change to the model using the Diagram or Table view, the change is reflected in all views. To open any view, select **Open** from the **Model** tab.

Diagram View

The Diagram view shows the structure of a model graphically. It provides information about how model elements, such as species, interact with one another using a block diagram. To supplement the diagram on page 1-4, this view also contains the browser on page 1-8 that shows the relationships between quantities and expressions. You can build a model by dragging and dropping blocks on page 1-17 from the block library panel to the browser or diagram. The browser and diagram are synchronized, and any update or change is reflected in both. The next figure shows a model open in the Diagram view.



Remove unused model elements.

Display the browser and diagram.

Customize the browser.

Open different views of a model.

Each open panel has its own tab.

Block Library Panel

Browser

Diagram

1	Model: TwoComp			
2	Central	=	1.0	liter
3	Dose	=	10.0	milligram
4	Drug	=	0.0	milligram/liter
5	Peripheral	=	0.7	liter
6	Dose	=	0.0	milligram
7	Drug	=	0.0	milligram/liter
8	Model Scoped Parameters			
9	Cl_Central	=	0.5	liter/hour
10	k12	=	0.7	1/hour
11	☑ Central -> Peripheral Transport Rate Initialization	=	Q12 / Central	
12	k21	=	0.2	1/hour
13	☑ Peripheral -> Central Transport Rate Initialization	=	Q12 / Peripheral	
14	ka_Peripheral	=	0.4	1/hour
15	ke_Central	=	0.3	1/hour
16	☑ Central Elimination Rate Initialization	=	Cl_Central / Ce...	
17	Q12	=	0.2	liter/hour
18	Reaction Scoped Parameters			
19	Central.Dose -> Central.Drug			
20	ka_Central	=	0.6	1/hour

Diagram

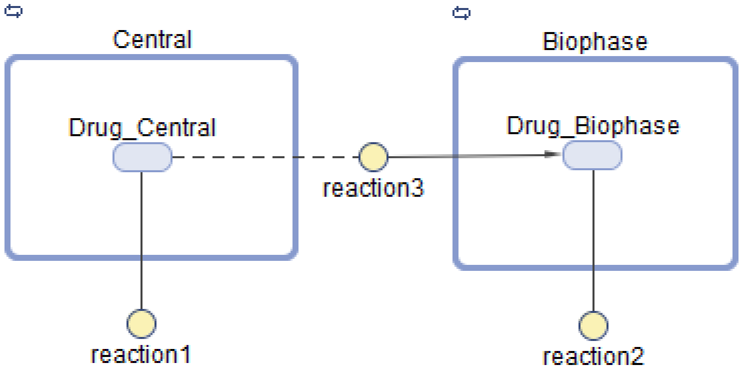
The diagram section contains a set of connected blocks that shows the structure of a model and the relationships between model quantities and expressions graphically.

You can add new quantities and expressions by dragging and dropping blocks from the block library panel to the block diagram. To connect two blocks, **Ctrl** + click (**Option** + click for a Mac) the first block and drag to the second block. Double-click a block to edit its properties. Right-click a block to see the context menu with more options, such as hiding the block to avoid clutter. Double-click the name of a block to rename.

The app uses contextual icons to provide more information about a block. The icons appear above each block. For instance, if a species is being dosed, a dose icon appears above the species block. Hover the mouse over the icon for more information.

Adding and configuring reactions

Suppose you want to model the pharmacokinetics of antibacterial drugs in a time-kill curve experiment [1]. Such an experiment involves exposing an in vitro bacterial inoculum to a fixed antibiotic dose and monitoring bacterial activity over time. You can model the in vitro drug kinetics using a one-compartment model with linear elimination to account for drug degradation due to compound instability. This is represented by the first reaction *`Drug_Central`* `-> null`. Include a second compartment (`Biophase`) to incorporate potential pharmacologic delay, which correspond to the second and third reactions: *`Drug_Central`* `->` *`Drug_Central`* `+` *`Drug_Biophase`*, *`Drug_Biophase`* `-> null`.

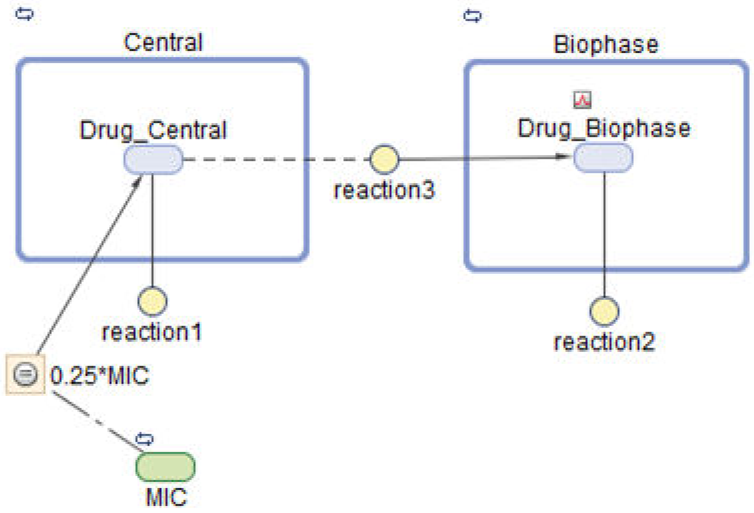


Drag and drop two compartment blocks onto the diagram. Then drop a species block inside each compartment. To build the first reaction, *reaction1*, drag and drop a reaction block onto the diagram. Then draw a line (**Ctrl**+Click or **Options**+Click on a Mac) from the *Drug_Central* species to *reaction1*. Double-click the reaction block and the **Block Property Editor** opens. In the **Quantities Used by Reaction** table, enter *kdeg* in the **Name** column as the **Forward Rate Parameter**. This automatically adds a parameter named *kdeg*, which is used as the forward rate constant for the reaction rate, that is, *`kdeg*Drug_Central`*. Similarly, configure the second reaction *reaction2*. Enter *ke* as the forward rate parameter for the reaction. To build the third reaction, *reaction3*, draw a line from *Drug_Central* to *Drug_Biophase*. A reaction block is automatically added between two species blocks. The input and output of the *Biophase* compartment are modeled as first-order kinetic processes. *Drug_Central* is both a reactant and product in the input process, assuming that the presence of the *Biophase* compartment does not affect the mass balance. SimBiology indicates such species using a dashed line. The dashed line can be achieved by drawing another line from *reaction2* to *Drug_Central*. You can also update a reaction by dragging a reaction line or arrow (**Ctrl**+Click or **Option**+Click in a Mac) to another species.

Defining or updating a quantity value using mathematical equations

The equations can take the form of initial assignments, assignments during the course of a simulation (repeated assignments), algebraic relationships, or differential equations (rate rules). Each equation is represented by a unique block. For details about rule blocks, see the block library on page 1-17. For more information about rules, see “Definitions and Evaluations of Rules in SimBiology Models” on page 2-13.

Suppose, in the above example, you want to set the initial concentration of *Drug_Central* to a factor of minimal inhibitory concentration (MIC). You can do so by using an assignment rule that initializes the concentration of *Drug_Central* at simulation time = 0, that is, $Drug\_Central_0$ = 0.25**MIC*, where *MIC* is a parameter. The following figure shows the graphical representation of the assignment rule.



Drag and drop the initial assignment rule block from the block library onto the diagram. To define the left-hand-side (LHS) of the equation, draw a line from the rule block to *Drug_Central*. An arrow appears pointing at *Drug_Central*. To display the right-hand-side (RHS) of the equation, right-click the rule block and select **Show Only Expressions**. By default, the RHS is set to 1. Double-click it and enter the equation: *Drug_Central* = 0.25**MIC*. Drag and drop a parameter block and rename it as *MIC*. A dash–dot line automatically appears connecting *MIC* to the rule block indicating *MIC* is referenced by the RHS. You can also change the LHS by dragging the arrow (**Ctrl**+Click or **Option** +Click in a Mac) to another quantity.

Incorporating sudden changes in model behavior

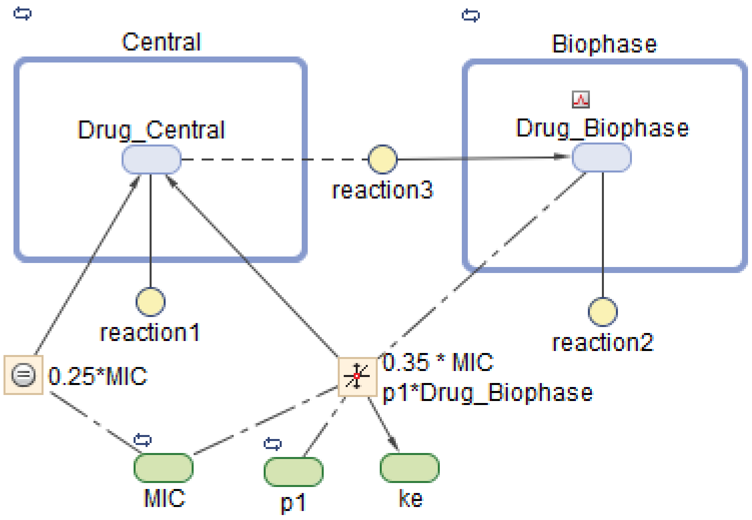
You can model sudden changes in model behavior based on a specified condition. For example, you can reset a parameter value at a certain time point or when a certain concentration threshold is crossed. In SimBiology, you can model such changes using a modeling element called event. An event lets you specify discrete transitions in quantity values that occur when a custom condition becomes true. Such condition is called an event trigger. Once the condition becomes true, one or more event functions are executed. For details, see events on page 2-22.

Suppose you want to set the concentration of *Drug_Central* at time = 5 to another factor of *MIC* as well as the parameter value of the elimination rate constant *ke*. Specifically,

```
if (time >= 5)
    Drug_Central  = 0.35*MIC;
    ke = p1*Drug_Biophase;
```

where *p1* is another parameter.

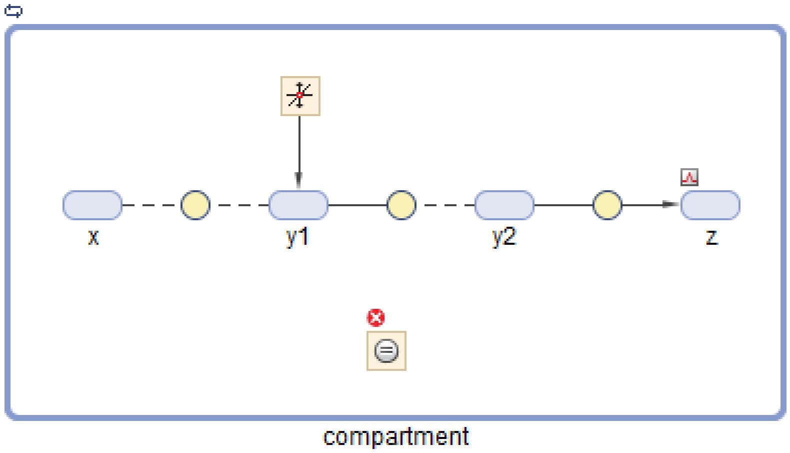
The next figure shows the graphical representation of the event.



By default, parameter blocks are hidden from the diagram. To see hidden parameter blocks such as *ke*, right-click the diagram, and select **Show All Hidden Blocks and Lines**. Drag and drop an event block onto the diagram. To define the quantities that are being updated by the event, draw a line from the event block to *Drug_Central* and another line to *ke*. Two arrows appear pointing at *Drug_Central* and *ke*, which are the LHS of event functions. To show the RHS of event functions, right-click the event block and select **Show Only Expressions**. Double-click the expressions to edit the event functions as shown. To see the dash–dot lines indicating quantities referenced in the RHS of an event function, right-click the event block and select **Show Expressions and Lines (Alt+Click)**. In this example, *MIC*, *p1*, and *Drug_Biophase* are the referenced quantities as indicated by dash–dot lines. You can also change the LHS by dragging the arrow (**Ctrl**+Click or **Option**+Click in a Mac) to another species.

Interpreting a model from its diagram

The next figure shows an example of a block diagram of another model and interpretations from looking at the graphical semantics.



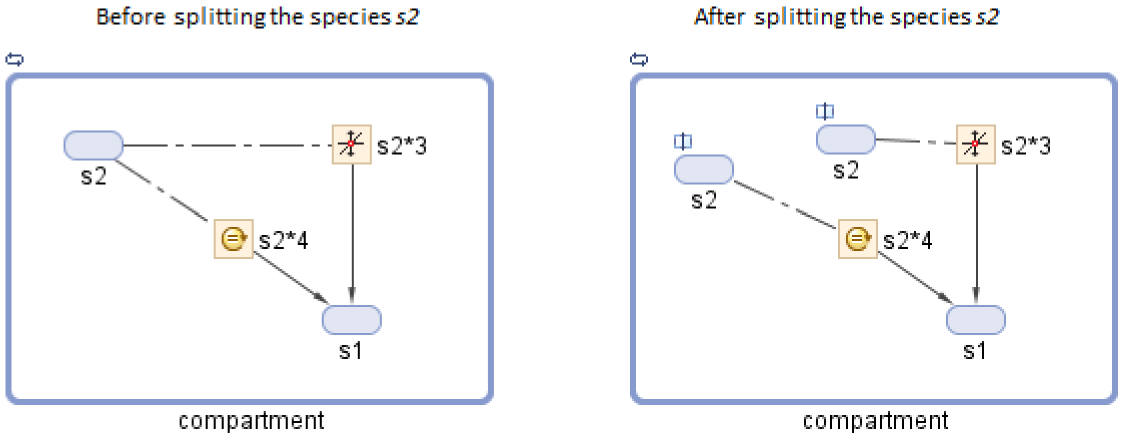
There are three reactions in this model. Species *x*, *y1*, and *y2* are catalysts, that is, both reactants and products, and *z* is a product. The amount of species *y1* is being modified by an event. Different contextual icons above some blocks indicate:

- The amount of species *z* is being increased by a dose.
- There is an error with an initial assignment rule.
- The *compartment* has constant volume.

Making the diagram clearer

When there are multiple references to the same quantity, multiple lines are connected to the quantity block. This can cause the diagram to look cluttered. To make the diagram clearer, you can split the block, that is, create copies of the same block, so that each reference is connected to a different copy of the block. You can also clone a block to add another use for it. For instance, you can first clone a species block that you know will be referenced in multiple expressions. Then use each clone in each expression as you build the model.

SimBiology lets you clone or split a species block. For event blocks, you can only split them if there are multiple event functions, but you cannot clone an event. To clone or split a species block, first select the block. Then select **Split** or **Clone** from the drop-down menu of the **Split** button on the **Block** tab. The next figure shows an example where a species (*s2*) is referenced by a repeated assignment rule and event, and splitting the block creates a copy for each reference. Copied blocks are then marked by a contextual icon to indicate that they have been cloned.


Before splitting the species *s2*
s2
s2*3
s2*4
s1
compartment
After splitting the species *s2*
s2
s2
s2*3
s2*4
s1
compartment

You can also hide blocks to avoid clutter. To hide a block, right-click it, and select **Hide Block** from the context menu. By default, blocks that represent constant parameters are hidden. Hidden blocks can be shown by selecting **Hidden Blocks** on the **Block** tab.

Browser

The browser supplements the diagram by providing a table of model elements that shows the relationships between quantities and expressions. You can edit these elements in the browser or add new quantities or expressions by dragging and dropping blocks from the block library panel onto the browser. If you make a change in the browser, the diagram is automatically updated.

You can view the model elements shown in the browser by quantities or expressions. In the quantities view, the browser shows all quantities of the model. If a quantity is being modified by any expressions and doses, the view also lists them. Alternatively, the expressions view lists all expressions of the model. It also shows quantities that are referenced by each expression. To switch between the two views, select **View > By Quantities** or **View > By Expressions** on the **Model** tab.

View By Quantities

The quantities view shows a table of quantities and expressions that are modifying them. Use this view to check the values of quantities and if they are being modified by expressions such as events or assignment rules. You can also see the right-hand-side, of each modifying expression. To display quantity units, select **Tools > Show Quantity Units** on the **Model** tab. The check box next to each expression indicates whether the expression is active and used during simulation.

You can model biological variability using a modeling element called variant. A variant is a collection of quantities with alternative values. For instance, in the above example, you can have one set of parameter values such as the elimination rate (*ke*) and degradation rate (*kdeg*) for each antibacterial drug.

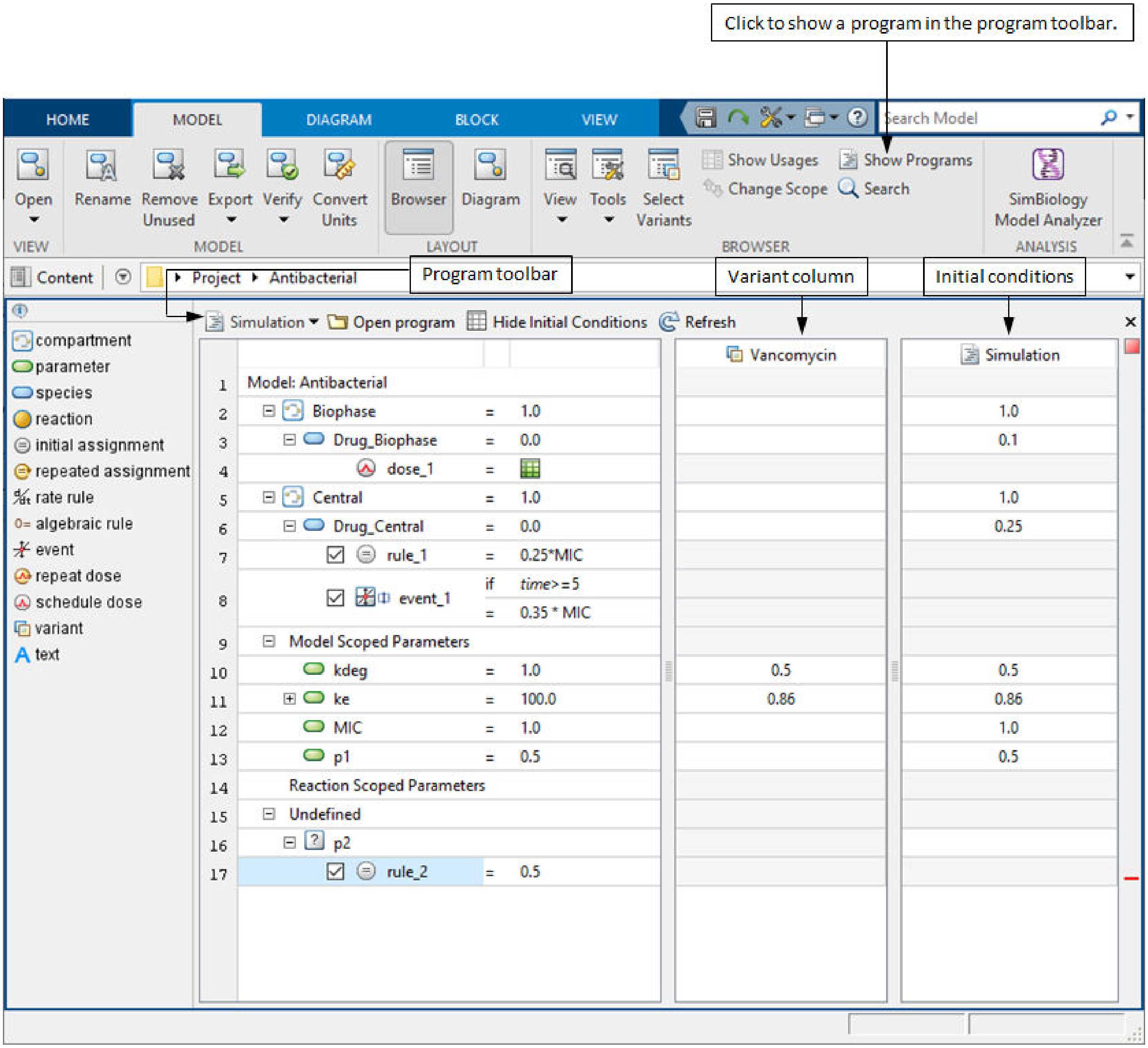
The browser shows a variant column for each variant of a model. For instance, a variant named *Vancomycin* is shown in the next figure. You can edit each quantity value by double-clicking it. When there are multiple variants, you can display a subset by clicking **Select Variants** on the **Model** tab. To add a variant, drag and drop a variant block from the block panel onto the browser.

If there are expressions for which the left-hand-sides are not defined, these expressions are listed under the section named **Undefined**. For example, in the next figure, the initial assignment *rule_2* is modifying a quantity *p2* that is not defined yet in the model. You can right-click the undefined quantity and define it as a species, parameter, or compartment.

Before running any analysis program, SimBiology prepares a model for simulation and updates the quantity values according to the variant values, assignment rules, and doses. For details, see “Model Simulation” on page 4-2. You can check if the quantity values are initialized as you expect by checking the initial conditions of the program.

To see the initial conditions, first, select **Show Programs** on the **Model** tab. Then select a program in the program toolbar. For instance, in the following figure, the initial conditions of a simulation program is shown. In this program, a variant called *Vancomycin* and a dose called *dose_1* have been selected to apply to the model during simulation. The concentration of *Drug_Central* at the beginning of the simulation is 0.25, compared to the model value 0.0. The value is updated because, at simulation time = 0, SimBiology evaluates the initial assignment *rule_1* that initializes *Drug_Central* to `0.25*MIC`. Since the variant *Vancomycin* has been selected, the alternate values stored in it, namely `0.5` and `0.86` for *kdeg* and *ke*, are used instead of model values.

Since a dose is applied to species *Drug_Biophase* at time = 0, the browser displays the total amount of the species, that is, the initial condition value of the species plus the dose amount. For instance, in the next figure, the total amount of species *Drug_Biophase* after applying the dose is 0.1. Alternatively, you can view the initial value and dose amount separately by selecting **Tools > Program Initial Conditions Options > Show the dose amount separate from the initial condition value** from the **Model** tab.



Click to show a program in the program toolbar.

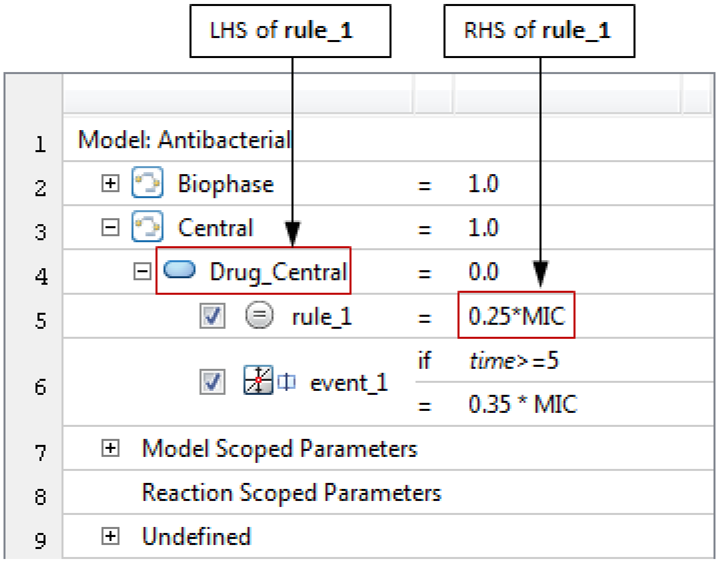
Program toolbar

Variant column

Initial conditions

				Vancomycin	Simulation
1	Model: Antibacterial				
2	⊟ Biophase	=	1.0		1.0
3	⊟ Drug_Biophase	=	0.0		0.1
4	dose_1	=	▦		
5	⊟ Central	=	1.0		1.0
6	⊟ Drug_Central	=	0.0		0.25
7	☑ rule_1	=	0.25*MIC		
8	☑ event_1	if =	*time*>=5 0.35 * MIC		
9	⊟ Model Scoped Parameters				
10	kdeg	=	1.0	0.5	0.5
11	⊞ ke	=	100.0	0.86	0.86
12	MIC	=	1.0		1.0
13	p1	=	0.5		0.5
14	Reaction Scoped Parameters				
15	⊟ Undefined				
16	⊟ p2				
17	☑ rule_2	=	0.5		

Defining or updating a quantity value using mathematical equations. The equations can take the form of initial assignments, assignments during the course of a simulation (repeated assignments), algebraic relationships, or differential equations (rate rules). For details, see rules on page 2-13. In the following example, the concentration of *Drug_Central* at time = 0 is defined by an initial assignment equation, that is, $Drug\_Central_0 = 0.25*MIC$. The next figure shows how the browser displays the assignment rule in relation to the species *Drug_Central*.



LHS of **rule_1**

RHS of **rule_1**

1	Model: Antibacterial		
2	⊞ Biophase	=	1.0
3	⊟ Central	=	1.0
4	⊟ Drug_Central	=	0.0
5	☑ rule_1	=	0.25*MIC
6	☑ event_1	if =	*time*>=5 0.35 * MIC
7	⊞ Model Scoped Parameters		
8	Reaction Scoped Parameters		
9	⊞ Undefined		

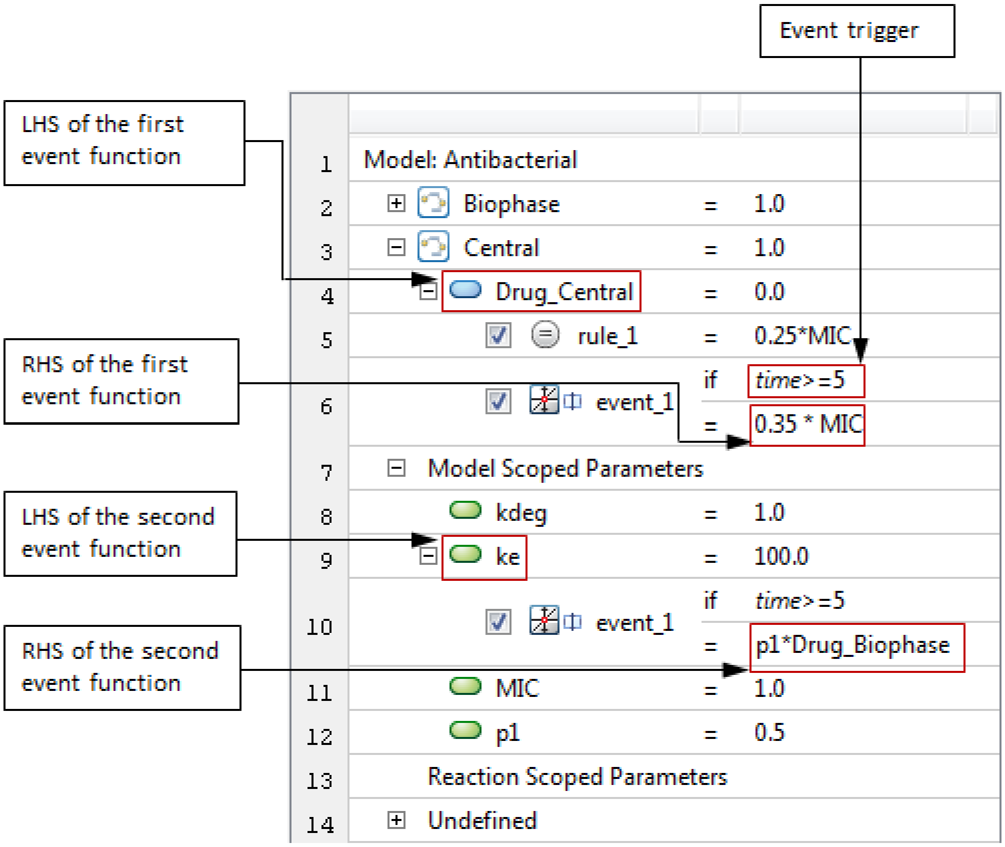
Drag and drop the initial assignment rule block onto the species *Drug_Central*. The rule block appears as a separate row below the species. Double-click the RHS of the equation to edit it.

Incorporating sudden changes in model behavior. You can model sudden changes in model behavior based on a specified condition by using a modeling element called event. In the following example, you want to set the concentration of *Drug_Central* at time = 5 to another factor of *MIC* as well as the parameter value of the elimination rate constant *ke*. Specifically,

```
if (time >= 5)
    Drug_Central  = 0.35*MIC;
    ke = p1*Drug_Biophase;
```

where *p1* is another parameter.

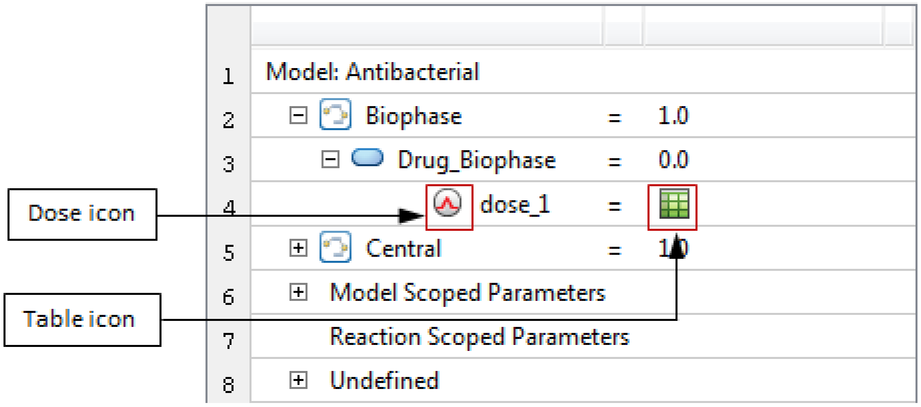
The next figure shows how the browser displays the event.


Event trigger
LHS of the first event function
RHS of the first event function
LHS of the second event function
RHS of the second event function

1	Model: Antibacterial		
2	⊞ Biophase	=	1.0
3	⊟ Central	=	1.0
4	⊟ Drug_Central	=	0.0
5	☑ rule_1	=	0.25*MIC
6	☑ event_1	if =	*time*>=5 0.35 * MIC
7	⊟ Model Scoped Parameters		
8	kdeg	=	1.0
9	⊟ ke	=	100.0
10	☑ event_1	if =	*time*>=5 p1*Drug_Biophase
11	MIC	=	1.0
12	p1	=	0.5
13	Reaction Scoped Parameters		
14	⊞ Undefined		

Drag and drop the event block onto the species *Drug_Central*. An event appears as a separate row below the species. To add the second event function, right-click the event and select **Add EventFcn**. Double-click the event trigger or RHS of an event function to edit.

Increasing a species amount using doses. You can model the increase in the amount of a species due to a stimulus such as an oral or intravenous administration of a drug. To model such an increase in a species amount, use the modeling element called dose on page 2-30. In the following example, the concentration of species *Drug_Biophase* is being increased by a dose.



1	Model: Antibacterial		
2	⊟ Biophase	=	1.0
3	⊟ Drug_Biophase	=	0.0
4	dose_1	=	▦
5	⊞ Central	=	1.0
6	⊞ Model Scoped Parameters		
7	Reaction Scoped Parameters		
8	⊞ Undefined		

Drag and drop a dose block onto the target species. A dose appears as a separate row below the species. To edit the dose amount, double-click the table icon. In this example, the concentration of species *Drug_Biophase* is being increased by a schedule dose *dose_1*. For more details about dose objects, see “Doses in SimBiology Models” on page 2-30.

View by Expressions

In the expressions view, you can see all expressions of a model grouped by their types such as reactions, assignment rules, and events. It shows how each expression is defined and the relationship between each expression and quantities. The check box next to each expression indicates whether the expression is active and used during simulation.

You can expand each reaction in the browser to configure its properties, such as reaction rate, kinetic law, quantities referenced by the reactions, and quantity values. You can define your own reaction rate or use a predefined rate that follows some particular reaction kinetics such as mass action.

A built-in kinetic law in SimBiology specifies a rate law that defines the reaction rate. Specifically each law has a predefined reaction rate containing parameters and species that need to be mapped to the corresponding model quantities to determine the final reaction rate. The browser displays the mapping information in parentheses next to a quantity. For instance, if a parameter `kf` is used as the forward rate parameter in the reaction rate, the browser displays `kf(Forward)`. In case you map incorrectly, you can reset the mapping by right-clicking the quantity and selecting **Remove Kinetic Law Mapping**.

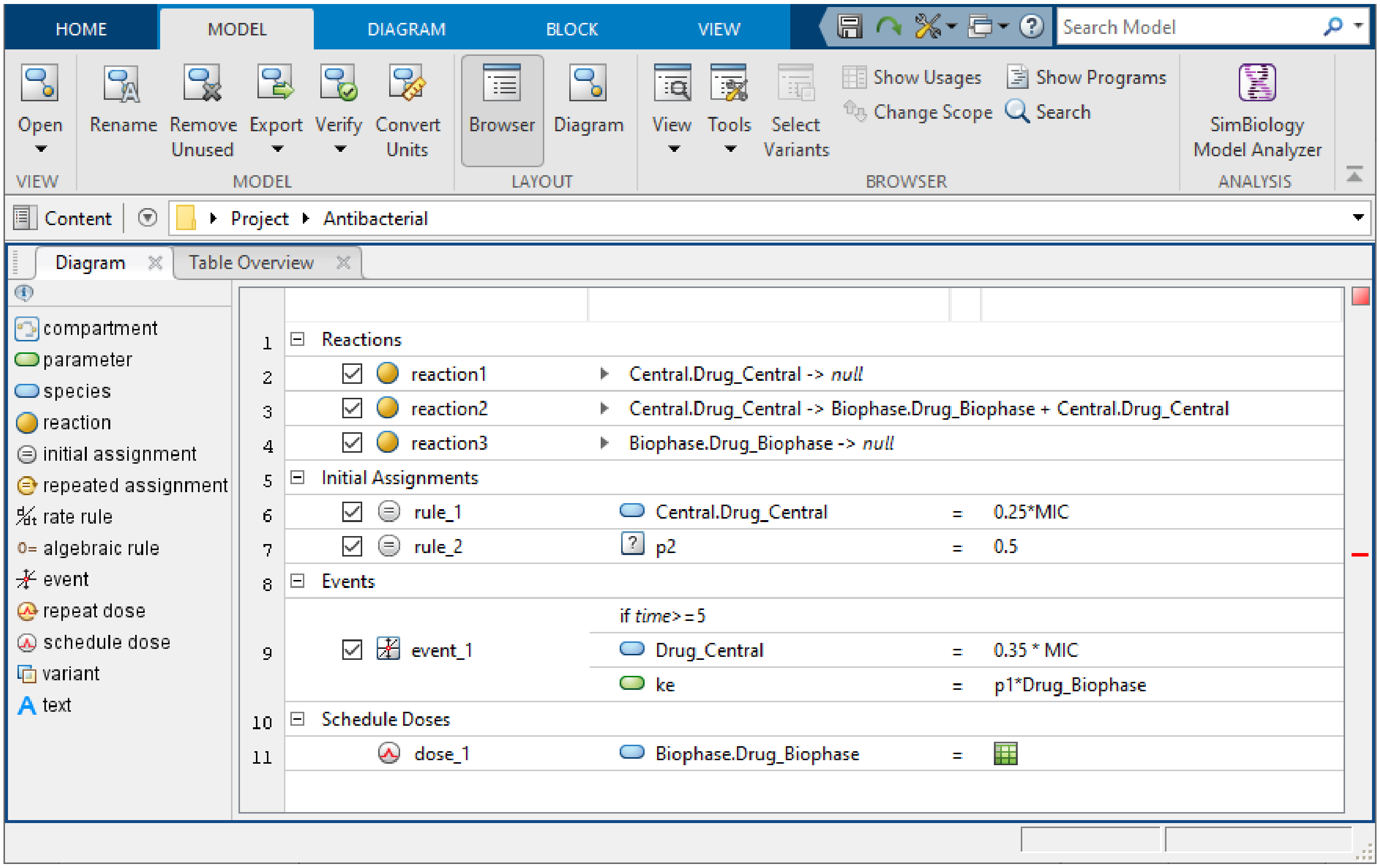
The kinetic law for a newly added reaction is configured to **MassAction** by default, and the app automatically creates and maps the species and parameters needed by the reaction rate. For other kinetic laws, only parameters are created and mapped. You need to create and map the species manually. To change the default kinetic law and reaction building settings, select **Tools > Reaction Building Preferences** on the **Model** tab.

This view also displays the assignment rules grouped according to their types such as initial assignment, repeated assignment, rate rule, and algebraic rule. For each rule, you can see its left-hand-side (LHS) and right-hand-side (RHS). The LHS is the quantity that is being modified by a rule, except for the algebraic rule. The algebraic rule takes the form `0 = Expression`, and the rule is specified as the `Expression`. For details, see “Definitions and Evaluations of Rules in SimBiology Models” on page 2-13.

This view also shows events. For each event, its trigger and event functions are grouped together. Each trigger is started with the word `if` followed by an expression such as `if time>=3`. You can also configure the LHS and RHS of each event function.

In addition to all the expressions, you can also see doses on page 2-30 that are grouped into either schedule doses or repeat doses. For each dose, the corresponding dosed species, that is, dose target, is shown right next to it. You can double-click the dose icon to edit all dose properties or the table icon to edit just the dose schedule.

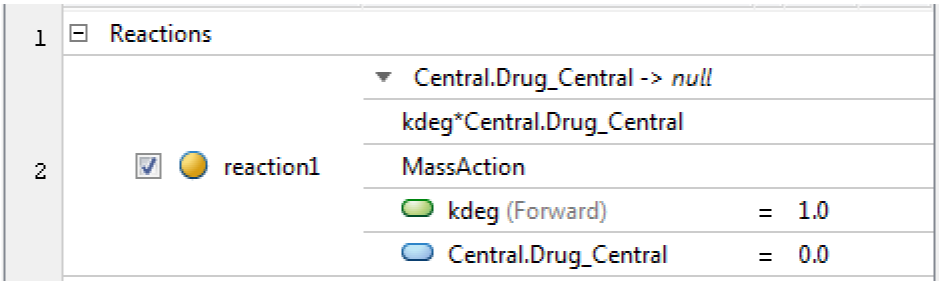
The following figure shows a model open in the expressions view of the browser.



1	⊟ Reactions			
2	☑ ● reaction1	▸ Central.Drug_Central -> *null*		
3	☑ ● reaction2	▸ Central.Drug_Central -> Biophase.Drug_Biophase + Central.Drug_Central		
4	☑ ● reaction3	▸ Biophase.Drug_Biophase -> *null*		
5	⊟ Initial Assignments			
6	☑ ⊜ rule_1	Central.Drug_Central	=	0.25*MIC
7	☑ ⊜ rule_2	p2	=	0.5
8	⊟ Events			
9	☑ event_1	if *time*>=5		
		Drug_Central	=	0.35 * MIC
		ke	=	p1*Drug_Biophase
10	⊟ Schedule Doses			
11	dose_1	Biophase.Drug_Biophase	=	▦

Adding and configuring reactions. There are three reactions in this example. Suppose the first and second reaction follow the mass action kinetics and the third reaction follows the Michaelis-Menten kinetics. For illustration purposes, the configuration of *reaction1* and *reaction3* is described below.

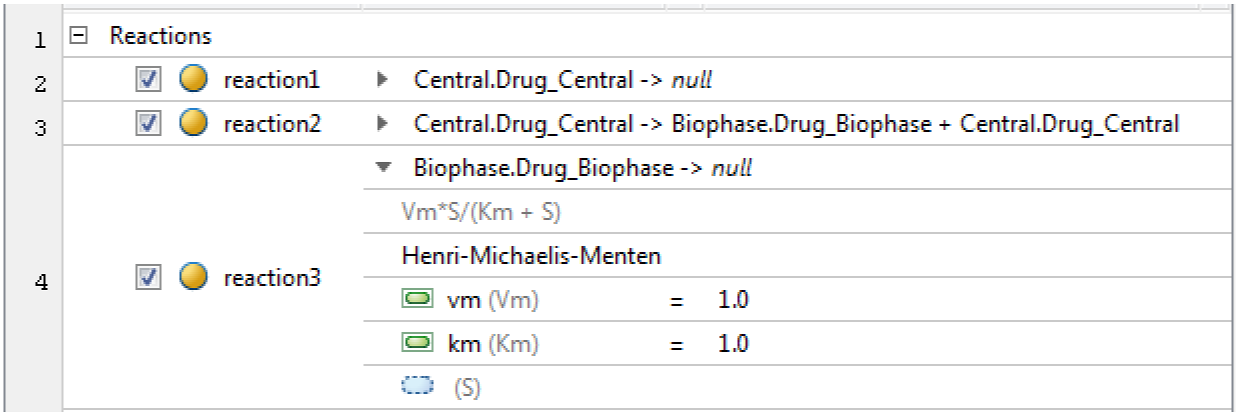
The following figure shows the details of *reaction1*. It follows the mass action kinetics with the reaction rate `kdeg*Central.Drug_Central`.



1	⊟ Reactions			
2	☑ ● reaction1	▾ Central.Drug_Central -> *null*		
		kdeg*Central.Drug_Central		
		MassAction		
		kdeg (Forward)	=	1.0
		Central.Drug_Central	=	0.0

To add a reaction, drag and drop a reaction block onto the browser. You can then enter the reaction string by double-clicking the default `null -> null` string. Since there are species in two compartments, qualify the species name with the compartment name. For example, *Central.Drug_Central* indicates that the *Drug_Central* species is inside the *Central* compartment. By default, the reaction's kinetic law is configured to **MassAction**. The parameter *kf* is automatically created and mapped to the forward rate constant of the reaction rate. If you want to use a different parameter, you can change it by double-clicking *kf* and entering the name of another parameter such as *kdeg*. You can then delete the parameter *kf* which is no longer used in the reaction rate.

The following figure shows the details of *reaction3* that is configured to follow Michaelis-Menten enzyme kinetics with the reaction rate `Vm*S/(Km+S)`.

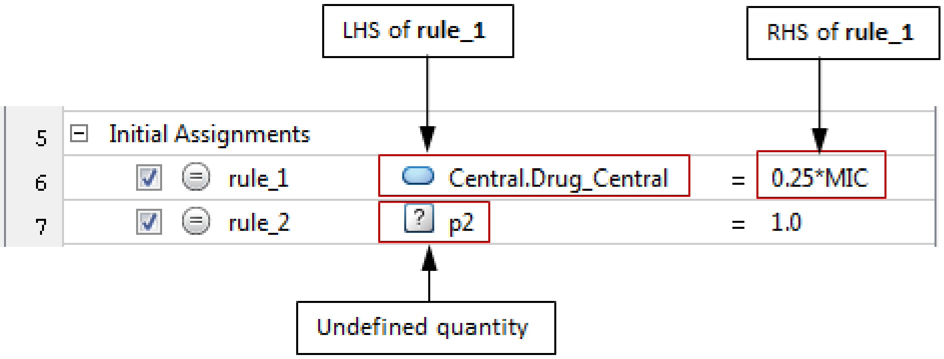


1	⊟ Reactions	
2	☑ ● reaction1	▸ Central.Drug_Central -> *null*
3	☑ ● reaction2	▸ Central.Drug_Central -> Biophase.Drug_Biophase + Central.Drug_Central
4	☑ ● reaction3	▾ Biophase.Drug_Biophase -> *null* Vm*S/(Km + S) Henri-Michaelis-Menten vm (Vm) = 1.0 km (Km) = 1.0 (S)

Drag and drop a reaction block onto the browser. By default, it uses the mass action kinetic law. The parameter *kf* is automatically created and mapped to the forward rate constant. To specify the reaction to follow Michaelis-Menten kinetics, double-click the row labeled **MassAction** and select **Henri-Michaelis-Menten**. SimBiology automatically updates the reaction rate to `Vm*S/(Km+S)`. The parameters *vm* and *km* are also automatically added and mapped to *Vm* and *Km*, which are the parameters referenced in the reaction rate. However, the *S* species referenced in the reaction rate is not mapped yet as indicated by a dotted species block. Double-click the *(S)* row to enter a model species that corresponds to the species *S*. The reaction rate is then automatically updated using the selected species. To avoid confusion, you may delete the parameter *kf*, which is no longer used in the reaction rate.

Adding and configuring assignment rules. In this example, there are two initial assignment rules on page 2-13. The first initial assignment *rule_1* initializes the concentration of species *Drug_Central*. The second initial assignment *rule_2* initializes an undefined quantity *p2*.

The next figure shows how the expressions view displays such assignment rules.



5	⊟ Initial Assignments			
6	☑ ⊜ rule_1	Central.Drug_Central	=	0.25*MIC
7	☑ ⊜ rule_2	? p2	=	1.0

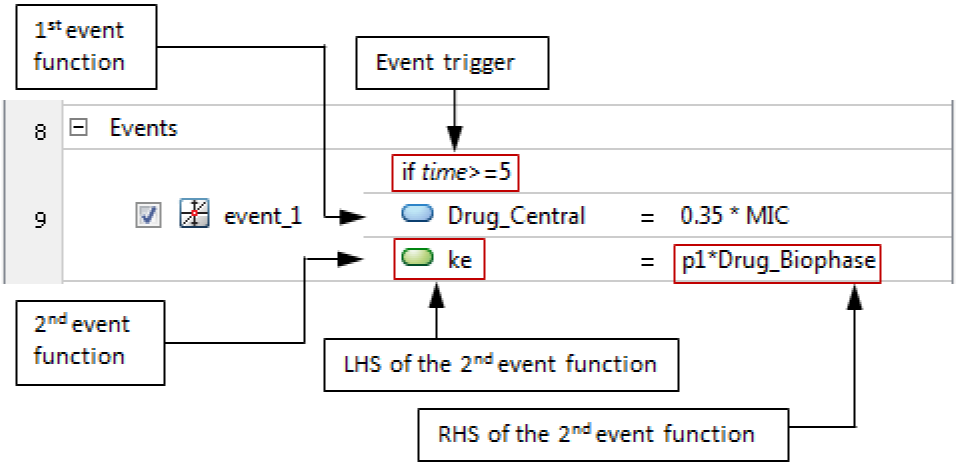
Drag and drop the initial assignment rule block onto the browser. Define the LHS of the rule by double-clicking the string `null` and entering the name of a quantity that the rule is modifying. Double-click the RHS of the equation to edit it. If the LHS of the rule is referring to a quantity that is not yet defined in the model, such as *p2* in this example, right-click the undefined quantity and select **Define Quantities** to define it as a species, parameter, or compartment.

Adding and configuring events. In this example, there is one event *event_1* with two event functions changing the concentration of species *Drug_Central* and parameter *ke* respectively. Specifically,

```
if (time >= 5)
    Drug_Central  = 0.35*MIC;
    ke = p1*Drug_Biophase;
```

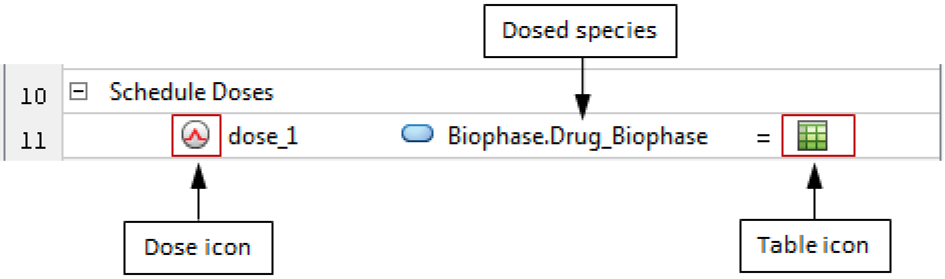
where *p1* is a parameter.

The next figure shows how the expressions view displays the event.



Drag and drop the event block onto the browser. Double-click the default trigger `if time>=1` to edit it. The default event function, `null = 0.0`, is listed below the trigger. Double-click its LHS and RHS to edit. To add more event functions, right-click the event and select **Add EventFcn**.

Adding and configuring doses. You can increase the amount or concentration of a species using a dose during simulation. In this example, the concentration of species *Drug_Biophase* is being increased by a schedule dose *dose_1*. The next figure shows how the expressions view displays the dose.



Drag and drop a dose block onto the browser. Define the dosed species by double-clicking the LHS of the table icon. Double-click the dose icon to edit all dose properties or the table icon to edit just the dose schedule.

Block Library Panel

The block library panel contains blocks from one or more block libraries. SimBiology provides a default block library which contains blocks that represent all modeling elements that you can use to build models. You can also create a library of custom blocks with different block appearance settings. For instance, you can customize a receptor protein to have a different block shape or color than other species whenever you use it in the diagram. For details on the SimBiology libraries, see “SimBiology Model Component Libraries” on page 1-27.

You can drag and drop most blocks from the panel to both the browser and diagram, and there are some blocks that can be dropped only in the browser, not in the diagram. Some blocks do not appear as blocks in the diagram but as contextual icons. For instance, a dose block appears as a contextual icon above a species block that is being dosed. The following table summarizes the behaviors of different built-in block types.

Block Name	Block Graphics	Drag and Drop onto the Browser	Drag and Drop onto the Diagram	Block appears as	Description
Species	⬭	Yes	Yes	Block	Quantity that participates in expressions, such as reactions and represents an amount or concentration.
Compartment	▢	Yes	Yes	Block	Physically bounded region that contains species in a model. All species in a model must belong to a compartment. A compartment can also belong to another compartment. For instance, a compartment that represents a cell can contain other cellular components such as mitochondria and nucleus as separate compartments within the cell compartment.

Block Name	Block Graphics	Drag and Drop onto the Browser	Drag and Drop onto the Diagram	Block appears as	Description
Parameter	🟩	Yes	Yes	Block	Quantity that is referenced by expressions. For instance, you can use it to define a rate constant of a reaction. By default, if you drop a parameter block onto the browser or diagram, it is available to all expressions including reactions. This type of parameter is called a model-scoped parameter and listed under the **Model Scoped Parameters** section in the quantities view of the browser. Another type is called a reaction-scoped parameter, where the parameter is only available to one particular reaction. These parameters are listed under the **Reaction Scoped Parameters** section. To add a reaction-scoped parameter, switch to the expressions view of the browser and then drop a parameter block onto a particular reaction. To switch between the two types, right-click a parameter block and select **Change Scope**.
Reaction	🟡	Yes (only in the expressions view of the browser)	Yes	Block	Process such as a transformation, transport, or the binding and unbinding of reactants and products.
Initial assignment	⊜	Yes	Yes	Block	Expression to assign the initial value of a quantity. For example, an initial assignment rule `s4 = 0.75` means that the amount of species *s4* is initialized to 0.75 at simulation time = 0.
Repeated assignment	⊜	Yes	Yes	Block	Expression to assign a quantity value repeatedly during simulation. For example, a repeated assignment rule `x = y1 + y2` means that the amount of species *x* is repeatedly assigned the total amount of species *y1* and *y2* throughout the simulation.

Block Name	Block Graphics	Drag and Drop onto the Browser	Drag and Drop onto the Diagram	Block appears as	Description
Rate rule	d/dt	Yes	Yes	Block	Differential equation to specify the time derivative of a model quantity. For example, a differential equation `x = k * (y + z)` specifies that the time derivative of species *x* (`dx/dt`) is evaluated continuously during the simulation according to the equation.
Algebraic rule	0=	Yes	Yes	Block	Expression to specify a mathematical constraint (for example, a nonlinear equation) on one or more quantities that must hold during simulation. For example, if you have an equation such as `x = log(x+time)`, write the corresponding algebraic rule as `log(x+time)-x`. **Tip** Use algebraic rules only for nonlinear equations that cannot be solved analytically to get closed-form solutions. For details, see “Algebraic Rules” on page 2-14.
Event		Yes	Yes	Block	Discrete transition in a quantity value. This discrete transition occurs when a specified condition becomes true.
Repeat dose		Yes	No	Contextual icon above a species block that is being dosed	Modeling element that increases the value of a species by a certain amount at predefined time intervals. To add a dose to a species, drag and drop the dose block onto the species in the quantities view of the browser. Double-click the dose icon to edit the dose value.
Schedule dose		Yes	No	Contextual icon above a species block that is being dosed	Modeling element that increases the value of a species by certain amount at specific time points. To add a dose to a species, drag and drop the dose block onto the species in the quantities view of the browser. Double-click the dose icon to edit the dose value.

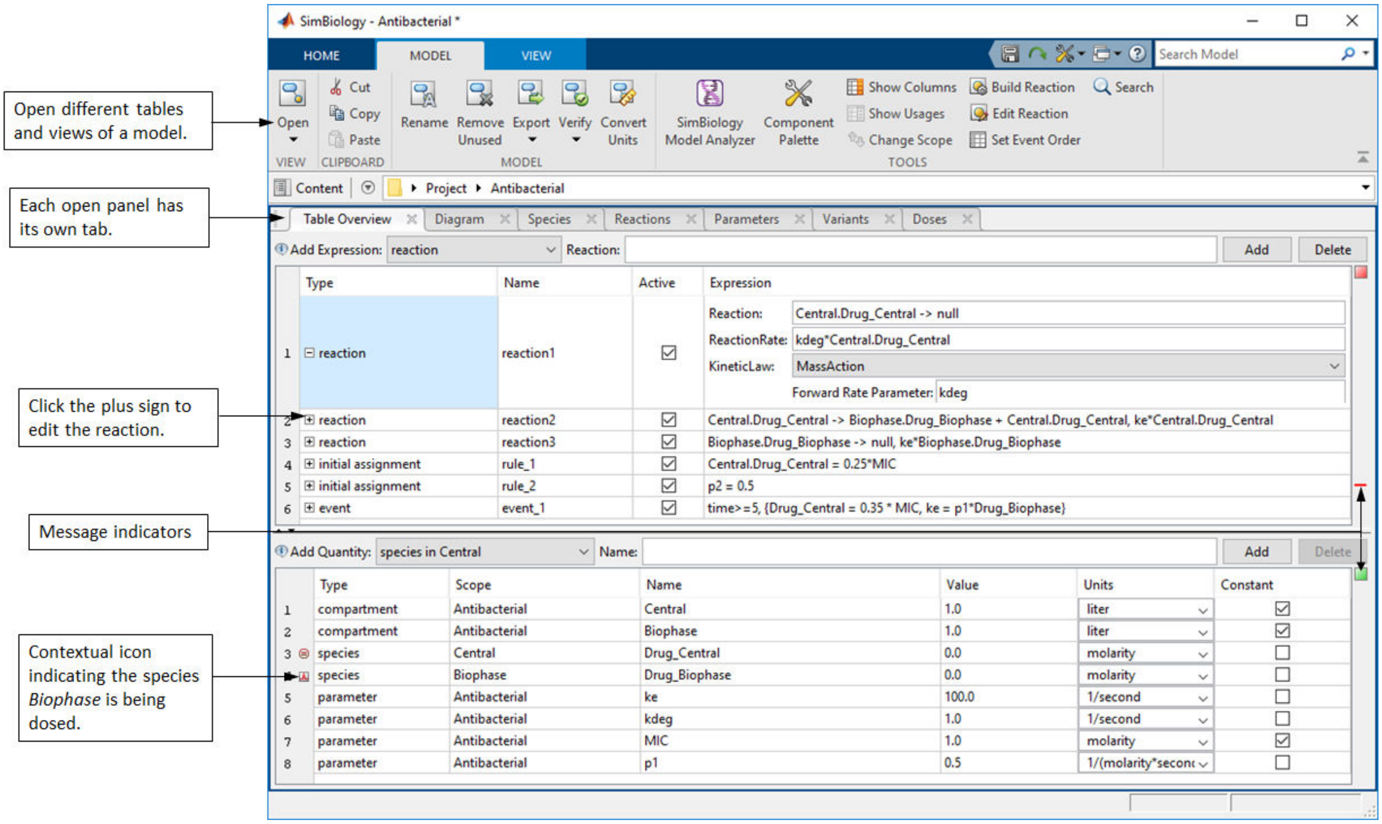
Block Name	Block Graphics	Drag and Drop onto the Browser	Drag and Drop onto the Diagram	Block appears as	Description
Variant	⧉	Yes (only in the quantities view of the browser)	No	Column in the Browser	Modeling element that is a collection of quantities with alternate values from the original model values.
Text	A	No	Yes	Block	Text block to annotate the model in the diagram.

Table View

This view presents a model in a tabular format, and there are many tables that organize the model information differently. For instance, the Table Overview gives an overview of the entire model by showing the complete list of model quantities, expressions, and some of their properties. There are also tables dedicated to each type of quantity or expression, doses, and variants, and these individual tables provide more information specific for each type.

You can open different tables from the **Open** drop-down list to edit and add elements to each table. For instance, you can enter a reaction string such as `y2 -> z` in the Table Overview or Reactions table.

The context menu, which you can open using a right-click, provides options to customize the table and open other modeling tools. For instance, to check which expressions are referencing a quantity, right-click the quantity and select **Show Usages** from the context menu.



Open different tables and views of a model.

Each open panel has its own tab.

Click the plus sign to edit the reaction.

Message indicators

Contextual icon indicating the species *Biophase* is being dosed.

Add Expression: reaction Reaction:

	Type	Name	Active	Expression
1	⊟ reaction	reaction1	☑	Reaction: Central.Drug_Central -> null ReactionRate: kdeg*Central.Drug_Central KineticLaw: MassAction Forward Rate Parameter: kdeg
2	⊞ reaction	reaction2	☑	Central.Drug_Central -> Biophase.Drug_Biophase + Central.Drug_Central, ke*Central.Drug_Central
3	⊞ reaction	reaction3	☑	Biophase.Drug_Biophase -> null, ke*Biophase.Drug_Biophase
4	⊞ initial assignment	rule_1	☑	Central.Drug_Central = 0.25*MIC
5	⊞ initial assignment	rule_2	☑	p2 = 0.5
6	⊞ event	event_1	☑	time>=5, {Drug_Central = 0.35 * MIC, ke = p1*Drug_Biophase}

Add Quantity: species in Central Name:

	Type	Scope	Name	Value	Units	Constant
1	compartment	Antibacterial	Central	1.0	liter	☑
2	compartment	Antibacterial	Biophase	1.0	liter	☑
3	species	Central	Drug_Central	0.0	molarity	☐
	species	Biophase	Drug_Biophase	0.0	molarity	☐
5	parameter	Antibacterial	ke	100.0	1/second	☐
6	parameter	Antibacterial	kdeg	1.0	1/second	☐
7	parameter	Antibacterial	MIC	1.0	molarity	☑
8	parameter	Antibacterial	p1	0.5	1/(molarity*seconc	☐

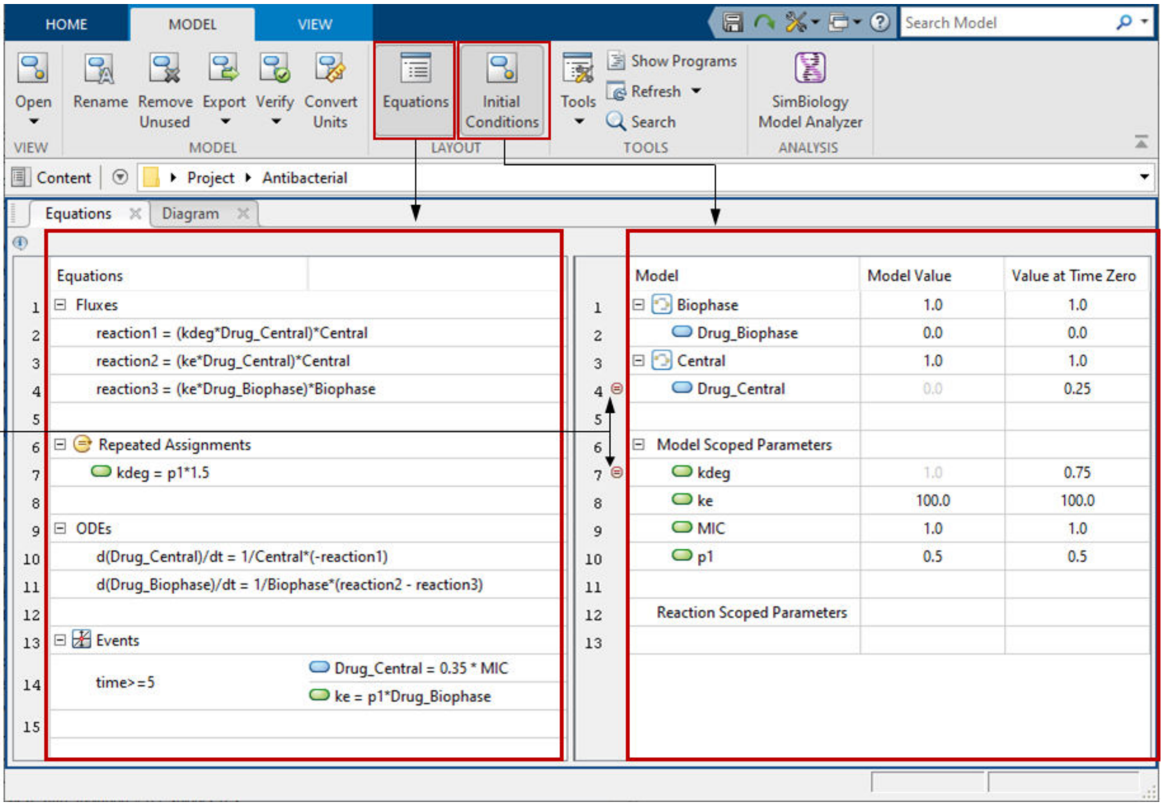
Equations View

This view describes a model in the form of mathematical equations. It shows a system of ordinary differential equations (ODEs) that represent a model. Specifically, the ODEs are derived from model reactions and define what quantities are being integrated during model simulation. For details about the simulation process and how SimBiology constructs ODEs, see “Model Simulation” on page 4-2.

You can use this view to help debug a model. For instance, you can check the initial conditions of ODEs to see if the quantity values are initialized as you expect. You can also see how SimBiology corrects the dimensions of ODEs by dividing the right-hand-sides of equations with compartment volumes. The volume-correction information can help debug unexpected simulation results, especially when you have a multi-compartment model with different compartment volumes. You can also compare the ODEs to the ones from a publication to reproduce the reported results.

The initial conditions of ODEs are used as starting points to simulate the dynamics of a model. The initial conditions are the quantity values at simulation time = 0. The **Value at Time Zero** column of the view shows these values. You can check this column to see if the quantity values are initialized as you expect based on initial assignment and repeated assignment rules, and debug the rules as necessary. For details about how SimBiology evaluates the assignment rules, see “Model Simulation” on page 4-2.

The next figure shows a model open in the equations view, which has the **Equations** and **Initial Conditions** sections. The **Equations** section contains the expressions and ODEs that SimBiology evaluates during model simulation. The **Initial Conditions** section contains the quantity values at simulation time = 0. Since initial assignments are evaluated just once at time = 0, they are only accessible in the **Initial Conditions** section via a contextual icon. Repeated assignments are evaluated both at time = 0 and during simulation. Hence they are shown in the **Equations** section and also indicated by contextual icons.



Equations	
⊟ Fluxes	
reaction1 = (kdeg*Drug_Central)*Central	
reaction2 = (ke*Drug_Central)*Central	
reaction3 = (ke*Drug_Biophase)*Biophase	
⊟ Repeated Assignments	
kdeg = p1*1.5	
⊟ ODEs	
d(Drug_Central)/dt = 1/Central*(-reaction1)	
d(Drug_Biophase)/dt = 1/Biophase*(reaction2 - reaction3)	
⊟ Events	

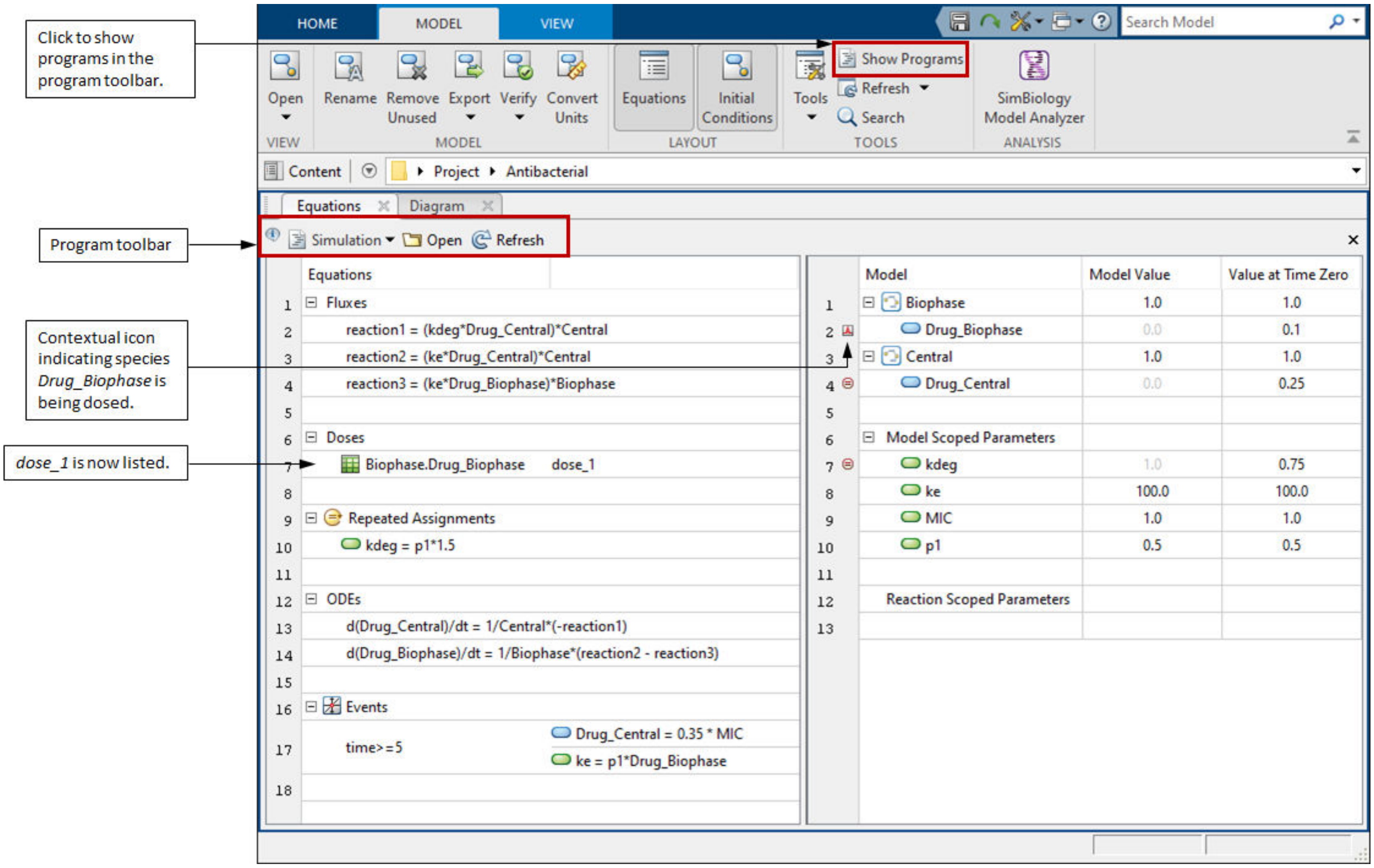
time>=5	Drug_Central = 0.35 * MIC
	ke = p1*Drug_Biophase

Model	Model Value	Value at Time Zero
⊟ Biophase	1.0	1.0
Drug_Biophase	0.0	0.0
⊟ Central	1.0	1.0
Drug_Central	0.0	0.25
⊟ Model Scoped Parameters		
kdeg	1.0	0.75
ke	100.0	100.0
MIC	1.0	1.0
p1	0.5	0.5
Reaction Scoped Parameters		

Contextual icons indicating species *Drug_Central* and parameter *kdeg* are being updated at simulation time = 0.

In this model, the concentration of drug in the central compartment *Drug_Central* is initialized to `0.25*`*`MIC`*, resulting in the amount of 0.25 as shown in the **Value at Time Zero** column. To see the initial assignment equation, click the contextual icon next to *Drug_Central*. In addition, a repeated assignment equation sets the value of parameter *kdeg* to *`p1`*`*1.5`, resulting in the value of 0.75. You can see the equation listed under the **Repeated Assignments** section. The desktop also shows a contextual icon next to the parameter *kdeg* to indicate the application of the assignment rule.

To perform an analysis on a model such as simulation or sensitivity calculation, you need to run a program. You can specify which doses and variants to use for the analysis in the program configuration. When you select a program, the app applies the doses and variants specified in the program, hence updating the corresponding quantity values in the **Value at Time Zero** column. The dose is also added to the **Equations** section. The next figure shows the equations view of the same model with a program selected. In this example, the program is a simulation program that uses a dose *dose_1* to increase the concentration of drug in the *Biophase* compartment *Drug_Biophase* at simulation time = 0.



Click to show programs in the program toolbar.

Program toolbar

Contextual icon indicating species *Drug_Biophase* is being dosed.

dose_1 is now listed.

	Equations	
1	⊟ Fluxes	
2	reaction1 = (kdeg*Drug_Central)*Central	
3	reaction2 = (ke*Drug_Central)*Central	
4	reaction3 = (ke*Drug_Biophase)*Biophase	
5		
6	⊟ Doses	
7	Biophase.Drug_Biophase	dose_1
8		
9	⊟ Repeated Assignments	
10	kdeg = p1*1.5	
11		
12	⊟ ODEs	
13	d(Drug_Central)/dt = 1/Central*(-reaction1)	
14	d(Drug_Biophase)/dt = 1/Biophase*(reaction2 - reaction3)	
15		
16	⊟ Events	

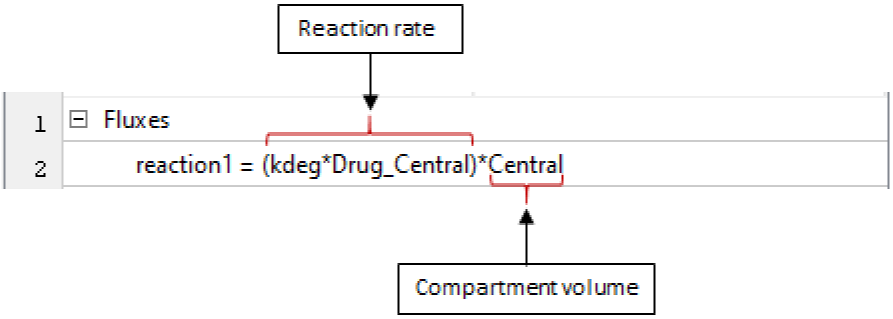
17	time>=5	Drug_Central = 0.35 * MIC
		ke = p1*Drug_Biophase

18		

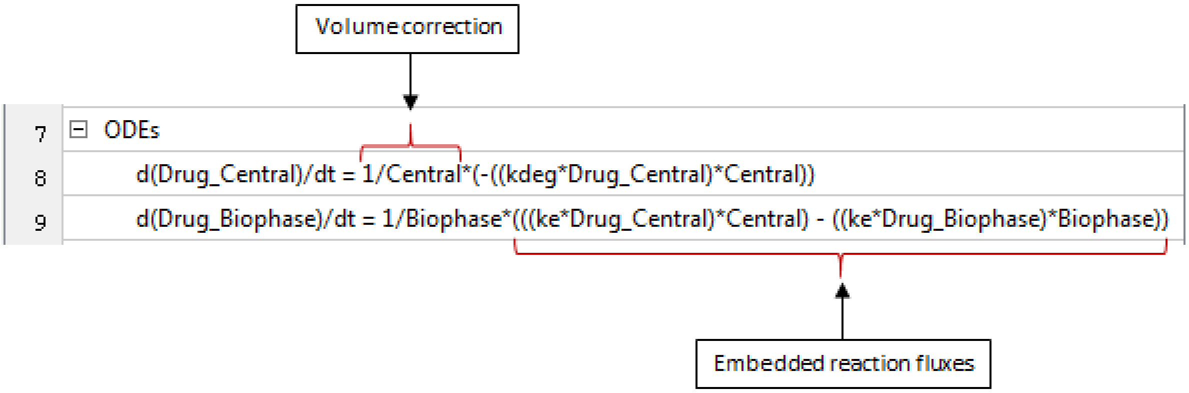
	Model	Model Value	Value at Time Zero
1	⊟ Biophase	1.0	1.0
2	Drug_Biophase	0.0	0.1
3	⊟ Central	1.0	1.0
4	Drug_Central	0.0	0.25
5			
6	⊟ Model Scoped Parameters		
7	kdeg	1.0	0.75
8	ke	100.0	100.0
9	MIC	1.0	1.0
10	p1	0.5	0.5
11			
12	Reaction Scoped Parameters		
13			

The grayed-out numbers in the **Model Value** column indicates the quantity values that have been changed due to the program configuration. The model value and the value at time zero are considered different if the relative tolerance between the values is greater than 10^{-12}, that is, `|x-y| > 1e-12*min(|x|,|y|)` where *x* is the model value and *y* is the value at time zero.

The **Fluxes** section of the equations view contains reaction fluxes for all reactions in the order they appear in the model. Reaction fluxes are equivalent to reaction rates except that the dimensions of fluxes are always `amount/time`. Hence, if the dimension of a reaction rate is `concentration/time`, the expression for the reaction flux is equal to the reaction rate multiplied by a compartment volume. SimBiology uses the name of a compartment to represent its volume in reaction fluxes and ordinary differential equations (ODEs). For details, see “Deriving ODEs from Reactions” on page 4-4. The next figure illustrates an example where *reaction1* is volume-corrected.



While preparing a model for simulation and other analyses, SimBiology performs dimensional analysis to make sure the dimensions of the left-hand-side (LHS) and right-hand-side (RHS) of each ODE are consistent. The LHS of each ODE is the time-derivative of the amount or concentration of a species, and the RHS is defined using reaction fluxes. If you specify no units, the default dimension for a species is `concentration`, and the default dimension for a flux is `amount/time`. For such cases, SimBiology divides the RHS by a compartment volume to make the dimensions of LHS and RHS consistent. For details, see “Deriving ODEs from Reactions” on page 4-4. You can see all the ODEs in the **ODEs** section of the view as shown in the next figure. In this example, SimBiology uses the default dimensions for species and fluxes. Hence each RHS of ODE is divided by the corresponding compartment volume *Central* or *Biophase* for volume-correction. By default, the flux names are used in ODEs. To use the expressions explicitly instead of using the flux names, select **Tools** > **Embed fluxes** on the **Model** tab. The following figure shows ODEs after fluxes have been embedded.



Volume correction

```
ODEs
    d(Drug_Central)/dt = 1/Central*(-((kdeg*Drug_Central)*Central))
    d(Drug_Biophase)/dt = 1/Biophase*(((ke*Drug_Central)*Central) - ((ke*Drug_Biophase)*Biophase))
```

Embedded reaction fluxes

References

[1] Nielsen, E. I., Viberg, A., Lowdin, E., Cars, O., Karlsson, M.O., and Sandstrom, M. (2007) Semimechanistic pharmacokinetic/pharmacodynamic model for assessment of activity of antibacterial agents from time-kill curve experiments. Antimicrobial Agents and Chemotherapy. 51:128-136.

See Also

Copy SimBiology Blocks

When building models interactively in the “Diagram” on page 1-4 view of the **SimBiology** app, you can copy and paste blocks using **Ctrl + C** and **Ctrl + V**. You can also use the context (right-click) menu.

- Using **Ctrl + V** works only if there is enough space near the original blocks. The app does not automatically increase the compartment size if there is not enough space.
- Using the context menu gives you a better control of where you want the copied blocks to appear. The app uses the current location of the mouse pointer to place the new blocks. You must still make sure that there is enough space at the new location.

Generally, *only selected blocks are copied and blocks connected to the selected blocks are not copied.* However, there are a few different behaviors depending on the block type as explained next.

Compartment Blocks

If you select a compartment and copy it, the desktop copies the compartment block and any blocks that are inside the compartment, even if they are not visible or selected explicitly. You can display all hidden blocks within a compartment by right-clicking inside the compartment and selecting **Show Hidden Blocks and Lines in Compartment**. The app does not copy any blocks that lie outside the compartment, even if they are connected to the blocks within it.

Species Blocks

If you copy a species block that is dosed, the associated dose is not copied. You can add a dose to the copied species manually using the “Browser” on page 1-8 or create a duplicate dose from the dose table on page 1-20 and update its target name.

Reaction Blocks

If you select a reaction block, the app copies only the reaction block and does not copy the associated species participating in the reaction. The reaction string is `null -> null` and its reaction rate is set to empty.

Parameter Blocks

Parameters can be model-scoped or reaction-scoped.

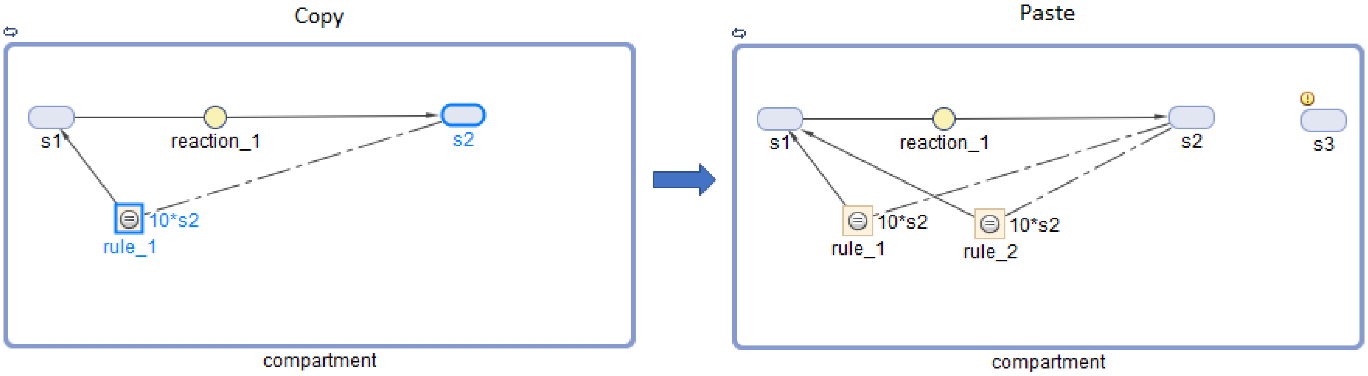
- If you select and copy an entire reaction (that is, the reaction block, the reactant species, and product species), the reaction-scoped parameter (such as a forward rate parameter) is also copied even though the parameter is not visible or selected explicitly.
- However, if the parameter is model-scoped, the newly copied reaction uses the original parameter (that is, both the original reaction and copied reaction share the parameter). If you paste the entire reaction to a different model, then the app does not copy the parameter, and you must create one for the reaction rate in that model.

Rule and Events Blocks

If you copy a rule or event block, the app uses the original rule or event string for the copied block. In other words, the copied rule or event has the connections to the same blocks as the original block.

If you copy a rule (or event) block and its right-hand-side (RHS) species block, the app does not update the rule (or event) string to include the newly copied species block. The copied rule continues pointing to the original RHS species block.

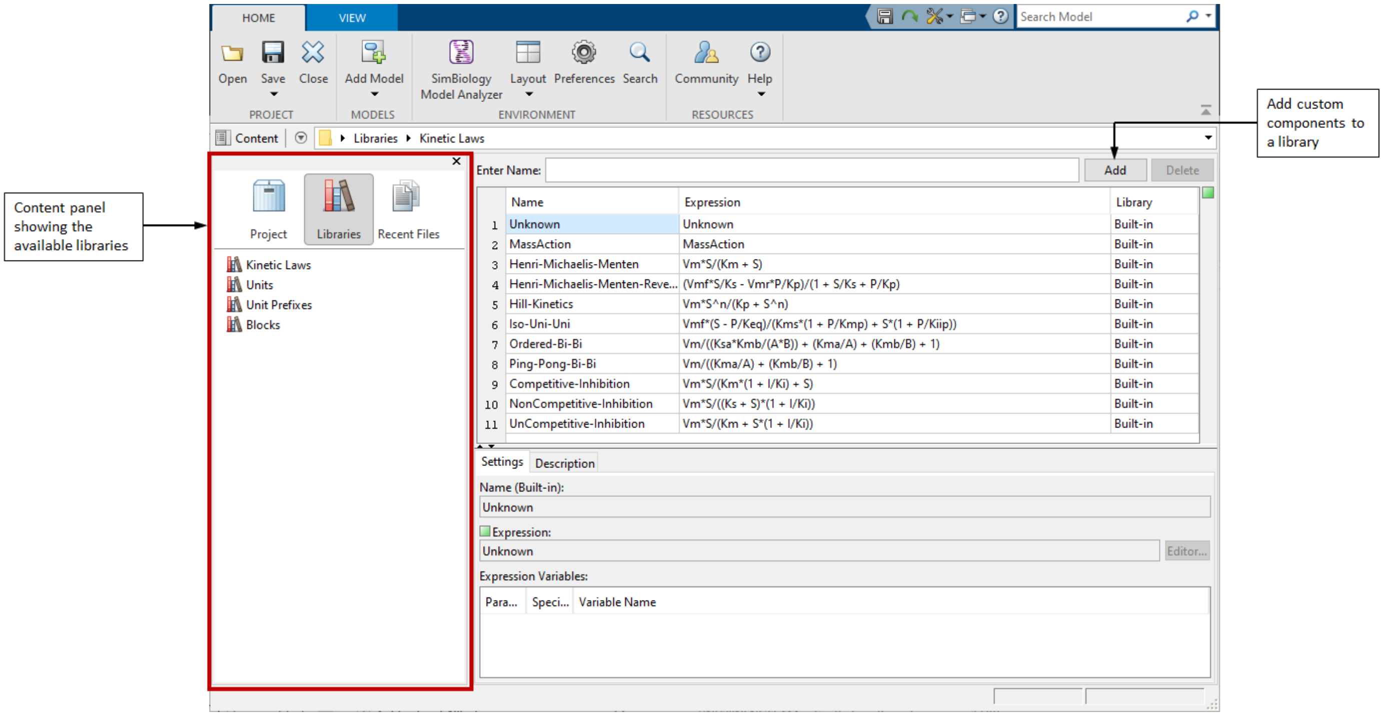
As an example, suppose that you have a rule (*rule_1*) defined as `s1 = 10 * s2`. You select *rule_1* and its RHS species block *s2* and copy them. After pasting, *rule_2* and *s3* are created, but *rule_2* continues pointing to the original *s2* block. The newly copied species *s3* is unused.



SimBiology Model Component Libraries

The SimBiology libraries are collections of built-in components that you can use to build and analyze models. For instance, you can use built-in units such as `mole` or `molecule` as amount units for species. As you perform model analyses, you can use built-in plots such as box plot or residuals plot to display the analysis results. You can also add custom components to any library. For instance, you can define a custom unit or block and use it as you build models. Similar to preferences, libraries are saved across sessions of MATLAB® and the SimBiology desktop. Libraries are available for all projects and are not part of any one project. You can also export libraries and share with others by selecting **Export All Libraries** from the action menu.

To open a library in the **SimBiology** app, select **Content** > **Libraries** and double-click a library. For instance, the following figure shows the kinetic laws library with all the available built-in kinetic laws.



Content panel showing the available libraries

Add custom components to a library

	Name	Expression	Library
1	Unknown	Unknown	Built-in
2	MassAction	MassAction	Built-in
3	Henri-Michaelis-Menten	Vm*S/(Km + S)	Built-in
4	Henri-Michaelis-Menten-Reve...	(Vmf*S/Ks - Vmr*P/Kp)/(1 + S/Ks + P/Kp)	Built-in
5	Hill-Kinetics	Vm*S^n/(Kp + S^n)	Built-in
6	Iso-Uni-Uni	Vmf*(S - P/Keq)/(Kms*(1 + P/Kmp) + S*(1 + P/Kiip))	Built-in
7	Ordered-Bi-Bi	Vm/((Ksa*Kmb/(A*B)) + (Kma/A) + (Kmb/B) + 1)	Built-in
8	Ping-Pong-Bi-Bi	Vm/((Kma/A) + (Kmb/B) + 1)	Built-in
9	Competitive-Inhibition	Vm*S/(Km*(1 + I/Ki) + S)	Built-in
10	NonCompetitive-Inhibition	Vm*S/((Ks + S)*(1 + I/Ki))	Built-in
11	UnCompetitive-Inhibition	Vm*S/(Km + S*(1 + I/Ki))	Built-in

Name (Built-in):
Unknown

Expression:
Unknown

Expression Variables:

Para...	Speci...	Variable Name

Kinetic Laws Library

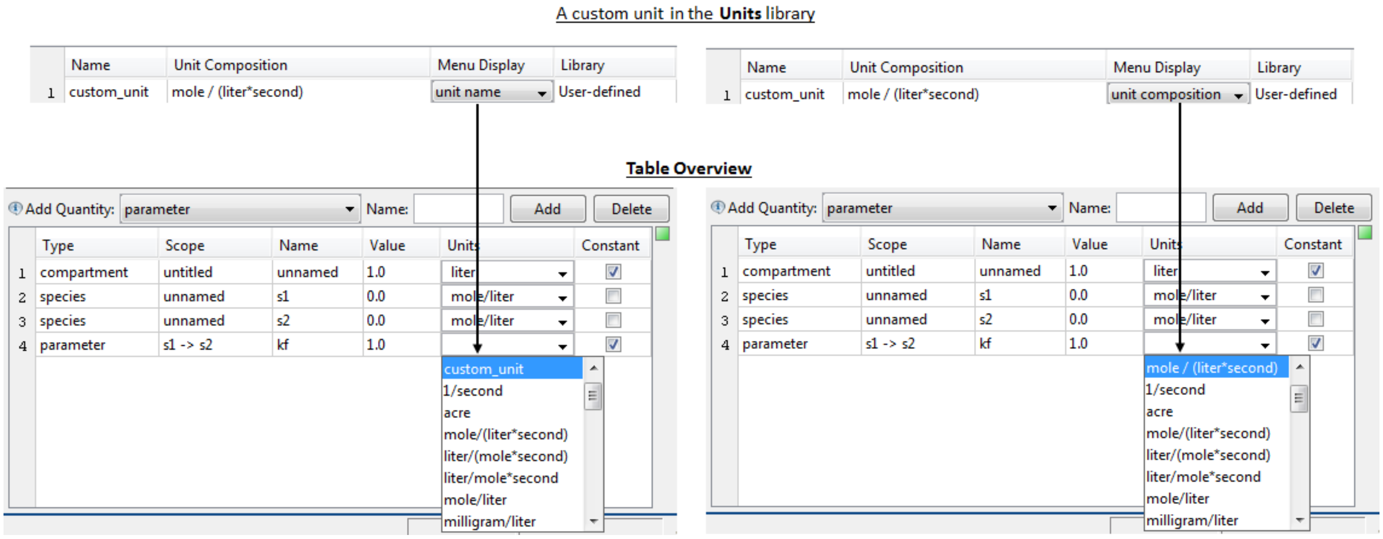
As you configure model reactions, you can use predefined reaction rates that follow particular kinetics such as mass action or Michaelis-Menten. SimBiology provides a list of such predefined rates in the **Kinetic Laws** library.

The kinetic law for a newly added reaction is configured to **MassAction** by default, and SimBiology automatically creates and maps the species and parameters needed by the reaction rate. For other kinetic laws, only parameters are created and mapped. You need to create and map the species manually. For details about configuring reaction rates in the SimBiology desktop, see “View by Expressions” on page 1-13. Use the **Unknown** kinetic law to define a custom reaction rate with its own parameters. You must define and add the species and parameters needed by the custom rate.

Note The **MassAction** and **Unknown** kinetic laws can have different simulation results even when the reaction rate is the same. This can happen when you have a reversible reaction with species in different compartments. The difference in simulation results is because of the volume-scaling performed by SimBiology during the dimensional analysis. For details, see “Deriving ODEs from Reactions” on page 4-4. Specifically, for **MassAction**, SimBiology uses corresponding compartment volumes to multiply the forward and reverse rates. However, for **Unknown** and other built-in kinetic laws, SimBiology multiplies the entire rate by only one compartment which contains the reactants. To see exactly what compartment volumes are used for scaling, open the “Equations View” on page 1-21 and check the **ODEs** section.

Units Library

It provides a collection of units that you can use. The library displays the unit composition for each unit. The **Menu Display** column controls how each unit is displayed when selecting quantity units in the Table on page 1-20 or Diagram on page 1-4 views. For instance, suppose you have a custom unit with the unit composition `mole/(liter*second)`. Depending on what you select in the **Menu Display**, SimBiology displays it differently in the **Units** column of Table Overview.



A custom unit in the **Units** library

	Name	Unit Composition	Menu Display	Library
1	custom_unit	mole / (liter*second)	unit name	User-defined

	Name	Unit Composition	Menu Display	Library
1	custom_unit	mole / (liter*second)	unit composition	User-defined

Table Overview

Add Quantity: parameter Name: Add Delete

	Type	Scope	Name	Value	Units	Constant
1	compartment	untitled	unnamed	1.0	liter	☑
2	species	unnamed	s1	0.0	mole/liter	☐
3	species	unnamed	s2	0.0	mole/liter	☐
4	parameter	s1 -> s2	kf	1.0		☑

Add Quantity: parameter Name: Add Delete

	Type	Scope	Name	Value	Units	Constant
1	compartment	untitled	unnamed	1.0	liter	☑
2	species	unnamed	s1	0.0	mole/liter	☐
3	species	unnamed	s2	0.0	mole/liter	☐
4	parameter	s1 -> s2	kf	1.0		☑

Unit Prefixes Library

It provides a list of all the available unit prefixes.

Blocks Library

It contains blocks that you can use to build models interactively. SimBiology provides a default block library called **ModelBuilding** that contains built-in blocks representing all modeling elements. In addition to built-in blocks, you can create custom blocks with different appearances. For instance, you can customize a receptor protein to have a different block shape or color than other species. You can use these blocks to build a model graphically in the diagram view on page 1-4. For details about different built-in block types, see Built-in Blocks on page 1-17.

See Also

SimBiology

Find Important Parameters with Sensitivity Analysis Using SimBiology Model Analyzer App

This example shows how to identify important model parameters for a tumor growth model [1]. In this example, you compute local time-dependent sensitivities of tumor growth with respect to model parameters for an anticancer drug.

Tumor Growth Model

The model used in this example is a SimBiology implementation of the pharmacokinetic/ pharmacodynamic (PK/PD) model by Simeoni et al. It quantifies the effect of anticancer drugs on tumor growth kinetics from *in vivo* animal studies. The drug pharmacokinetics are described by a two-compartment model with IV bolus dosing and linear elimination (*ke*) from the *Central* compartment. Tumor growth is a biphasic process with an initial exponential growth followed by linear growth. The growth rate of the proliferating tumor cells is described by

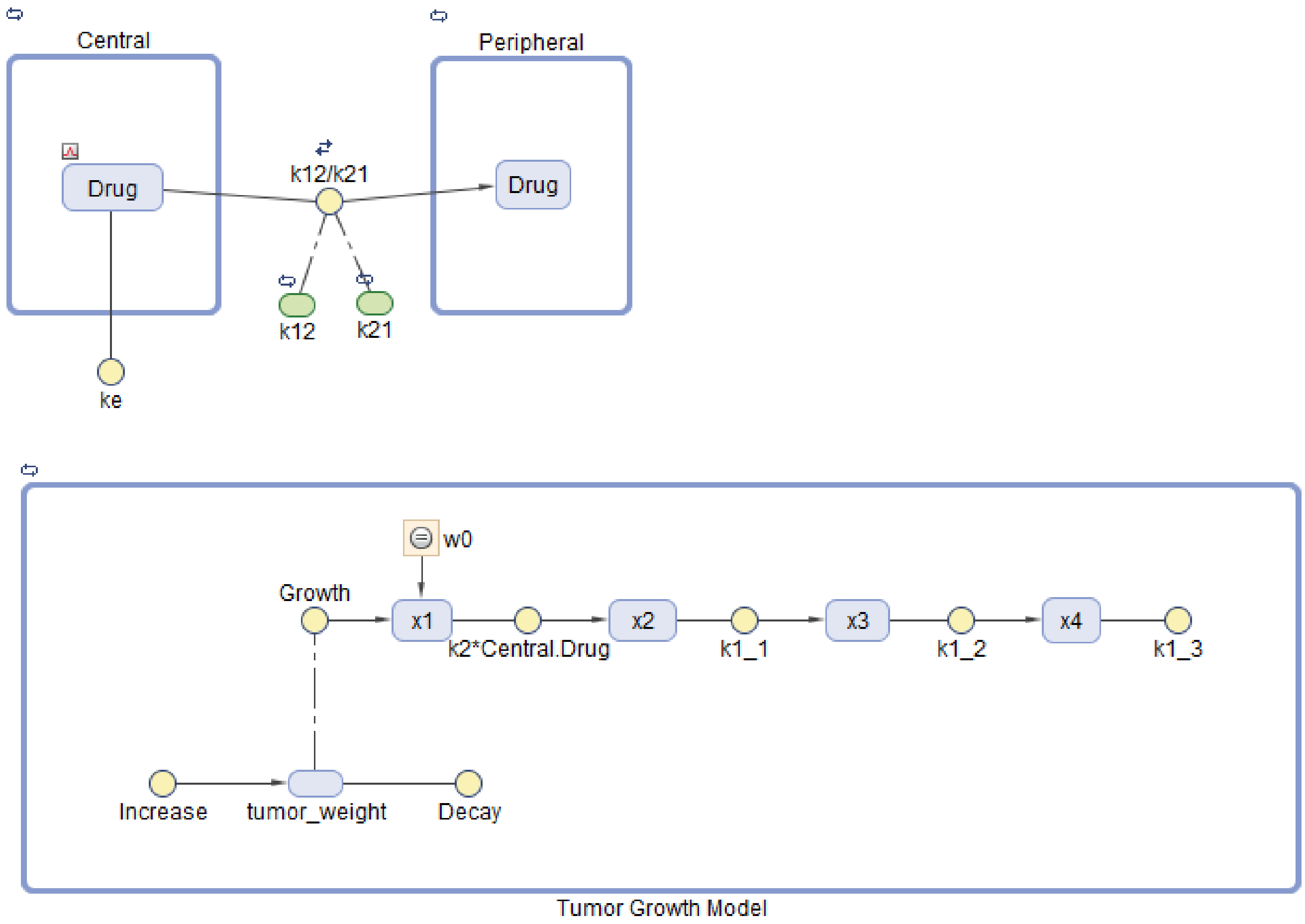
$$\frac{L_0 * x_1}{\left[1 + \left(\frac{L_0}{L_1} * w\right)^{\psi}\right]^{1/\psi}}$$

L_0, L_1, and Ψ are tumor growth parameters, x_1 is the weight of the proliferating tumor cells, and w is the total tumor weight. In the absence of any drugs, the tumor consists of proliferating cells only, that is, `w = x1`. In the presence of an anticancer agent, a fraction of the proliferating cells is transformed into nonproliferating cells. The rate of this transformation is assumed to be a function of the drug concentration in the plasma and an efficacy factor k_2. The nonproliferating cells *x2* go through a series of transit stages (*x3* and *x4*) and are eventually cleared from the system. The flow-through of the transit compartments is modeled as a first-order process with the rate constant k_1.

The SimBiology model makes these adjustments to the pharmacodynamics of tumor growth:

- Instead of defining the tumor weight as the sum of *x1*, *x2*, *x3*, and *x4*, the model defines the tumor weight by the reaction named *Increase*, `null → tumor_weight`, with the reaction rate $\left(2 * L_1 * L_0 * {x_1}^2 / L_1 + 2 * L_0 * x_1\right) * tumor\_weight$.

  tumor_weight is the total tumor weight, x_1 is the weight of the proliferating tumor cells, and L_0, and L_1 are tumor growth parameters.
- Similarly, the model defines the decrease in tumor weight by the reaction named *Decay*, `tumor_weight → null`, with the reaction rate $k_1 * x_4$. The constant k_1 is the forward rate parameter, and x_4 is the last species in the series of transit reductions in tumor weight.
- *ke* is a function of the clearance and the volume of the central compartment: `ke = Cl_Central/Central`.



Sensitivity Analysis

Sensitivity analysis on page 4-22 lets you determine which rate constants and concentrations in a model have significant influence on the overall behavior of the model. This example shows how to investigate which model parameters are sensitive to the tumor growth using the built-in sensitivity analysis program. Specifically, this example shows you how to calculate the sensitivity of *tumor_weight* species with respect to the model parameters as follows:

$$\frac{\partial(tumor\_weight)}{\partial(L_0)}, \frac{\partial(tumor\_weight)}{\partial(L_1)}, \frac{\partial(tumor\_weight)}{\partial(k_1)}, \ldots$$

Calculate Sensitivities

Suppose that you have already calibrated tumor growth model parameters on page 1-64 to experimental PK/PD data and saved parameter estimates and initial conditions as a variant called *parameterEstimates* in your model. As a follow-up analysis, you want to find out which model parameters are sensitive to tumor growth. The following steps show how to do so using the built-in sensitivity analysis program.

Load Tumor Growth Model

1 At the MATLAB command line, load the model (*m1*) by entering:

```
sbioloadproject tumor_growth_vpop_sa.sbproj
```

2 Open the **SimBiology Model Analyzer** app by typing `simBiologyModelAnalyzer` at the command line or by clicking **SimBiology Model Analyzer** on the **Apps** tab.

3 On the **Home** tab, select **Model** > `Import Model from MATLAB`. Select **Tumor Growth Model**.

Configure Calculate Sensitivities Program

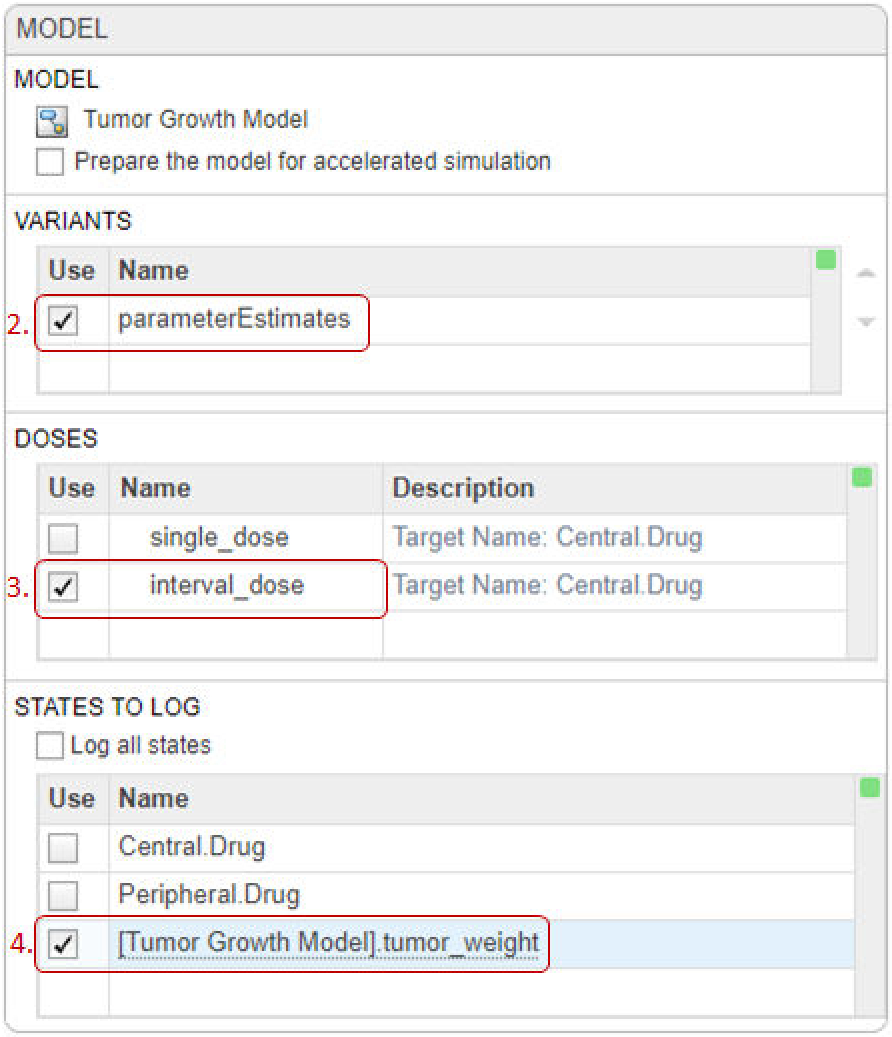
1 On the **Home** tab, select **Program** > **Calculate Sensitivities**. A new program opens.

2 In the **Variants** section of the **Model** step (a *setup* step) of the program, select **parameterEstimates**. This variant contains previously estimated parameter values and initial conditions for the drug.

3 In the **Doses** section, select **interval_dose**. This repeat dose applies 30 mg of the drug every 4 days starting at day 7 for a total of five times.

4 In the **States To Log** section, select **[Tumor Growth Model].tumor_weight** only.

Tip The context menu of the table contains options to add species, parameters, or compartments, and lets you make selections. Access the context menu by right-clicking the table.



MODEL

MODEL

Tumor Growth Model

□ Prepare the model for accelerated simulation

VARIANTS

	Use	Name
2.	☑	parameterEstimates

DOSES

	Use	Name	Description
	□	single_dose	Target Name: Central.Drug
3.	☑	interval_dose	Target Name: Central.Drug

STATES TO LOG

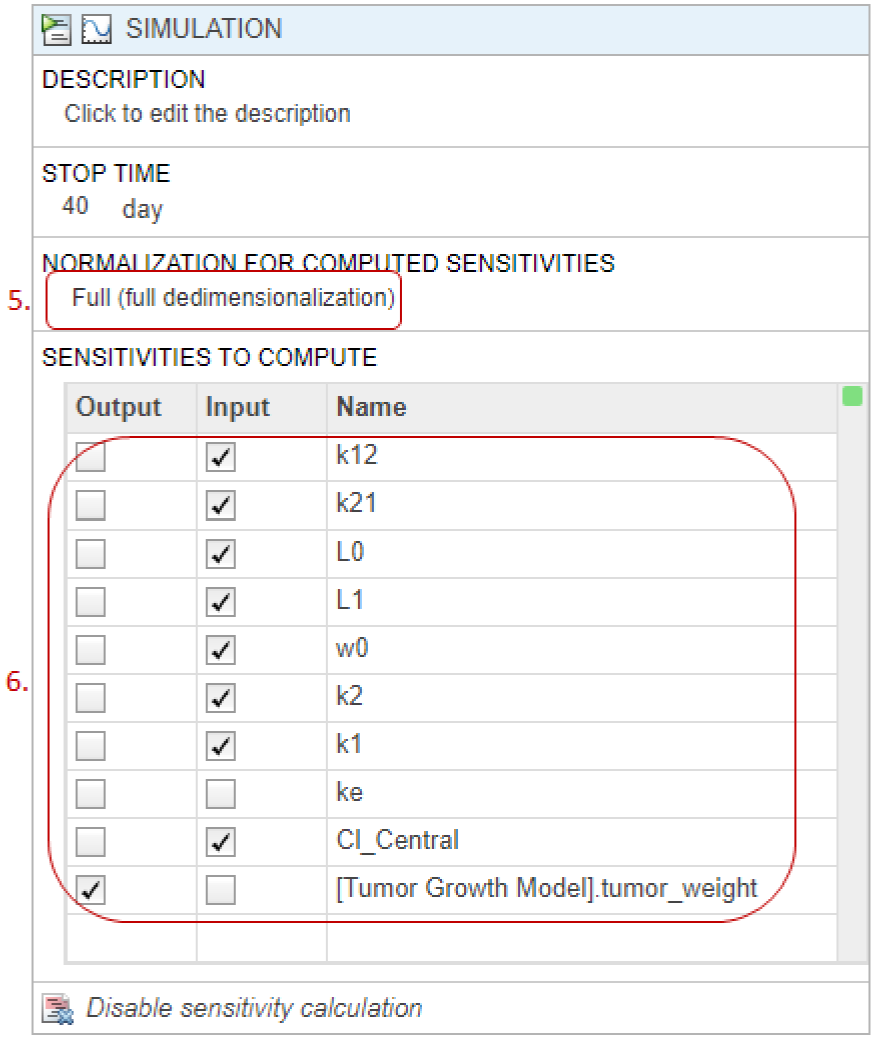
□ Log all states

	Use	Name
	□	Central.Drug
	□	Peripheral.Drug
4.	☑	[Tumor Growth Model].tumor_weight

5 In the **Simulation** step (an *execution* step), by default, the normalization method is set to **Full (full dedimensionalization)**, meaning that the app fully normalize the sensitivities so that they can be compared to each other. For details, see `Normalization`.

Tip You can run every program *execution* step separately. An execution step includes the run button next to the name the step. Running an individual step is especially helpful if the program contains multiple steps and you want to see the intermediate results from a particular step. By doing so, you can make adjustments as needed before running the next step or the whole program. To run the whole program, click the **Run** button on the **Home** tab.

6 In the **Sensitivities to Compute** section, right-click anywhere in the table and select **Add All Constant Parameters**. Clear *ke* because it is defined as a function of the `Central` volume and *Cl_Central*, which is already an input. Click the last empty row and type `tumor`. Then select `[Tumor Growth Model].tumor_weight` from the list. The *tumor_weight* species is then added as the output.



SIMULATION

DESCRIPTION
Click to edit the description

STOP TIME
40 day

NORMALIZATION FOR COMPUTED SENSITIVITIES
Full (full dedimensionalization)

SENSITIVITIES TO COMPUTE

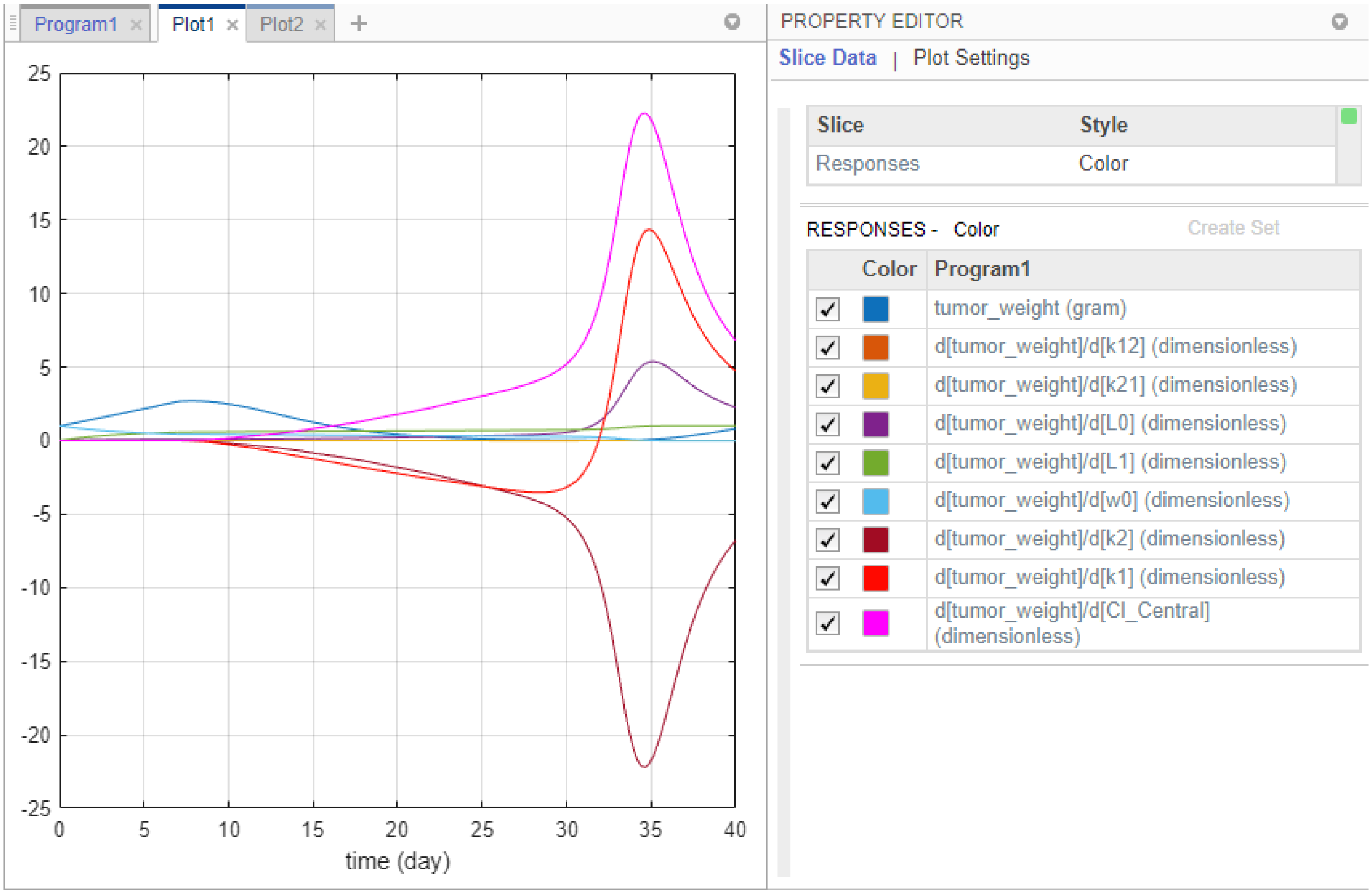
Output	Input	Name
□	☑	k12
□	☑	k21
□	☑	L0
□	☑	L1
□	☑	w0
□	☑	k2
□	☑	k1
□	□	ke
□	☑	Cl_Central
☑	□	[Tumor Growth Model].tumor_weight

Disable sensitivity calculation

7 On the **Home** tab, click **Run**.

Visualize Sensitivity Results

1 After the sensitivity analysis, the program automatically generates two plots. Click the **Plot1** tab to view a plot of the time course of the tumor weight and other sensitivity values `d[tumor_weight]/d[parameter]`. The time plots for `d[tumor_weight]/d[k21]` and `d[tumor_weight]/d[k12]` stay relatively flat compared to other sensitivities, indicating that the tumor growth is not sensitive to *k21* and *k12* parameters. The **Property Editor** lets you select which plots to display.



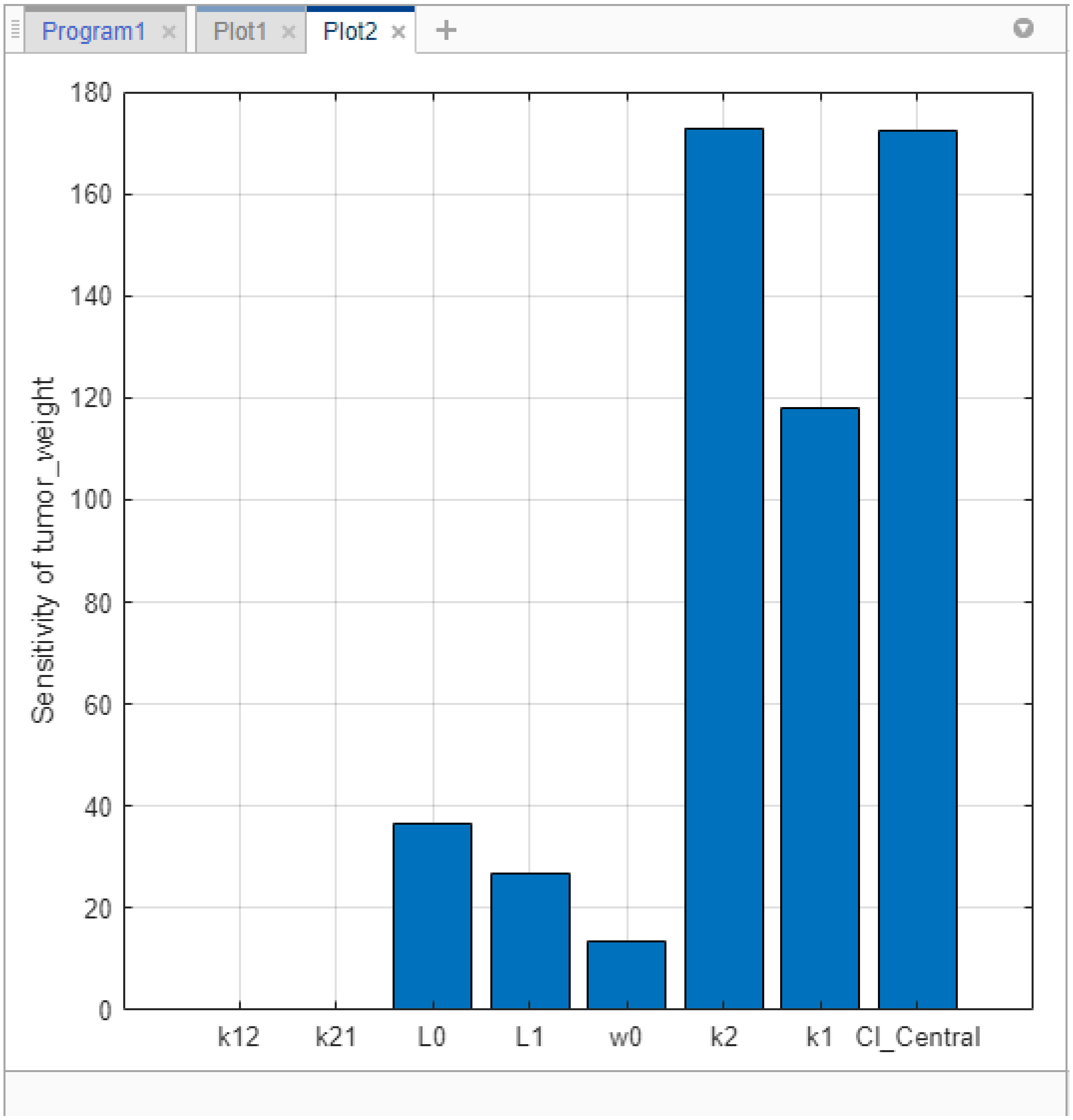
Slice	Style
Responses	Color

RESPONSES - Color

	Color	Program1
☑	🟦	tumor_weight (gram)
☑	🟧	d[tumor_weight]/d[k12] (dimensionless)
☑	🟨	d[tumor_weight]/d[k21] (dimensionless)
☑	🟪	d[tumor_weight]/d[L0] (dimensionless)
☑	🟩	d[tumor_weight]/d[L1] (dimensionless)
☑	🟦	d[tumor_weight]/d[w0] (dimensionless)
☑	🟥	d[tumor_weight]/d[k2] (dimensionless)
☑	🟥	d[tumor_weight]/d[k1] (dimensionless)
☑	🟪	d[tumor_weight]/d[Cl_Central] (dimensionless)

Tip Plots are backed by data that are currently present in the app workspace. Plots are not snapshots. When the data (either experimental data or simulation results) is removed or changed, the plots are also updated according to the changes in the underlying data.

2 You can also examine the same sensitivity data by checking the magnitude of the computed sensitivities integrated over time. The second generated plot (**Plot2**) shows a bar plot containing such information. The plot shows that the tumor weight is sensitive to *L0*, *L1*, *w0*, *k2*, *k1*, and *Cl_Central*, but not to *k12* and *k21*.



3 Save the sensitivity analysis results in a separate folder.

   a Right-click the **LastRun** folder of the program in the **Browser** pane.
   b Select `Save Data`.
   c In the **Save Data** dialog, enter `sa_data` as the data name.

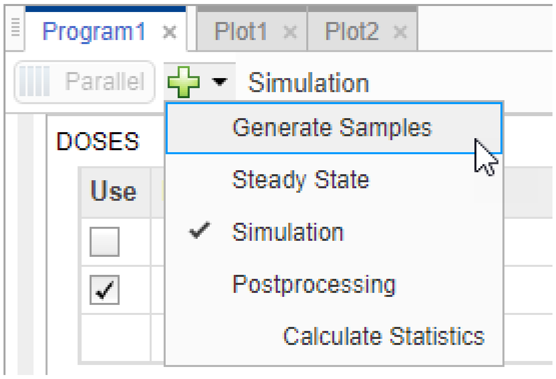
Investigate Variability of k12 and k21

The sensitivity analysis results indicate that tumor weight is not sensitive to the model values of *k12* and *k21*. Explore whether varying these parameter values results in different sensitivity results.

Vary k12

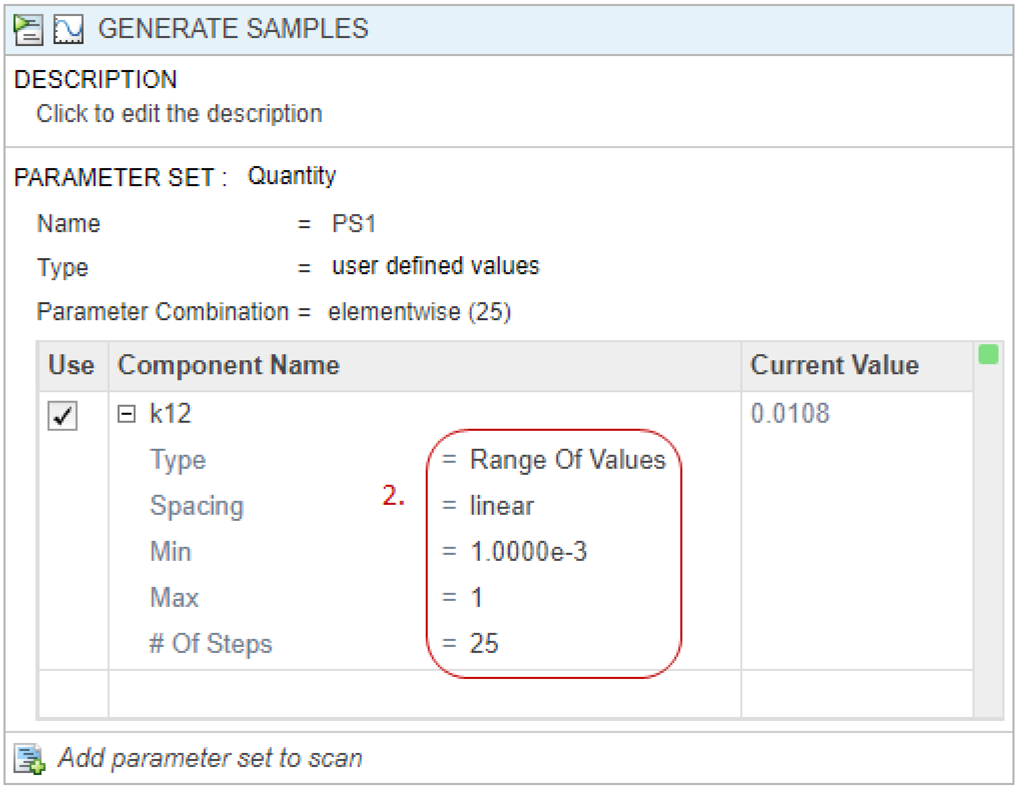
First, vary values for the *k12* parameter.

1 Click the **Program1** tab. Click the (+) plus at the top of the program and select `Generate Samples`.



2 The **Generate Samples** step appears. In the **Parameter Set** section of the step, double-click the empty cell in the **Component Name**, and enter *k12*. Set the following options:

- **Type** — Range Of Values
- **Spacing** — linear
- **Min** — 1e-3
- **Max** — 1
- **# Of Steps** — 25



GENERATE SAMPLES

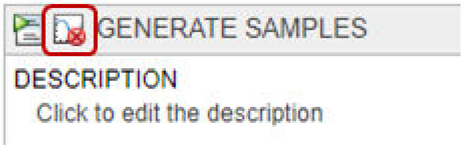
DESCRIPTION
Click to edit the description

PARAMETER SET : Quantity

Name = PS1
Type = user defined values
Parameter Combination = elementwise (25)

Use	Component Name		Current Value
☑	⊟ k12		0.0108
	Type	= Range Of Values	
	Spacing	= linear	
	Min	= 1.0000e-3	
	Max	= 1	
	# Of Steps	= 25	

3 Disable default plot generation by clicking the plot button at the top of the step.

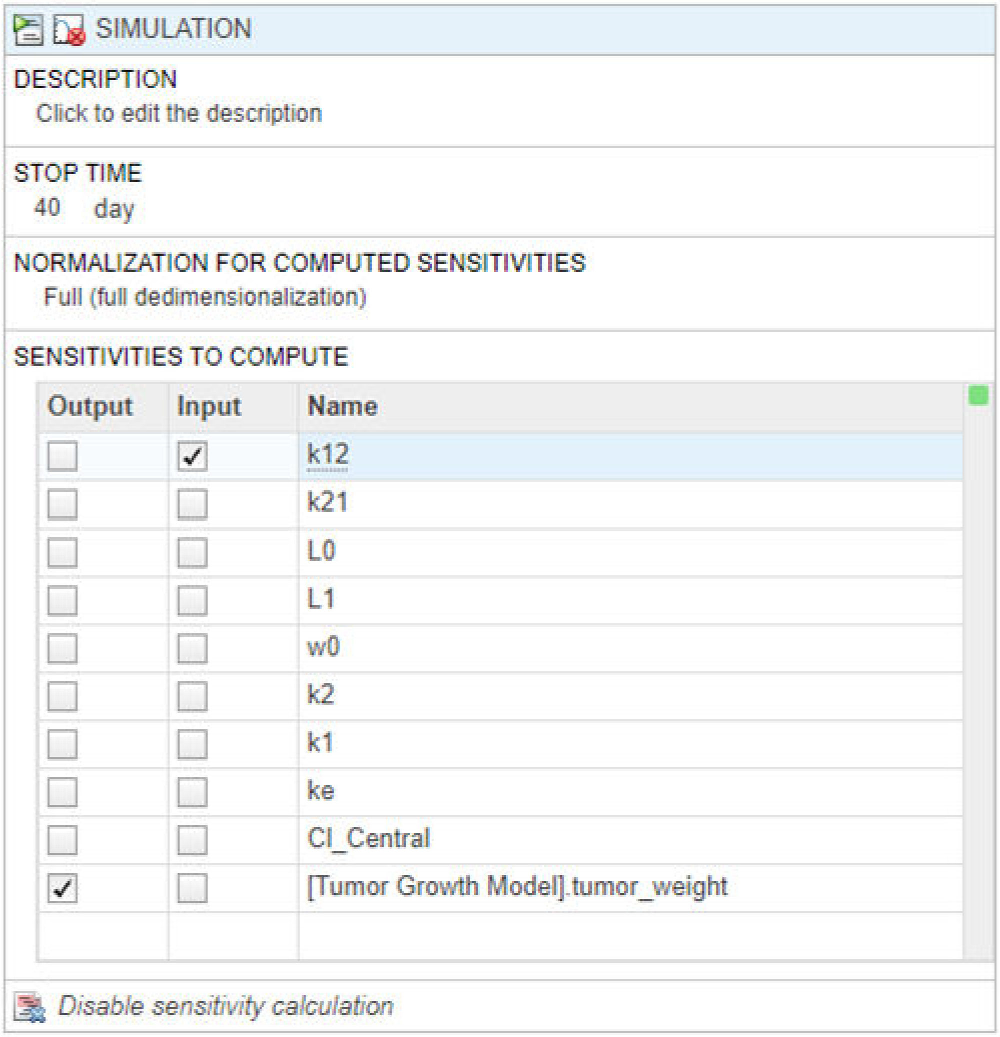


GENERATE SAMPLES

DESCRIPTION

Click to edit the description

4 In the **Simulation** step, under **Sensitivities to Compute**, clear all the inputs except *k12*. Keep *tumor_weight* as the output. Also, disable the default plot generation by clicking the plot button at the top of the step.



SIMULATION

DESCRIPTION

Click to edit the description

STOP TIME

40 day

NORMALIZATION FOR COMPUTED SENSITIVITIES

Full (full dedimensionalization)

SENSITIVITIES TO COMPUTE

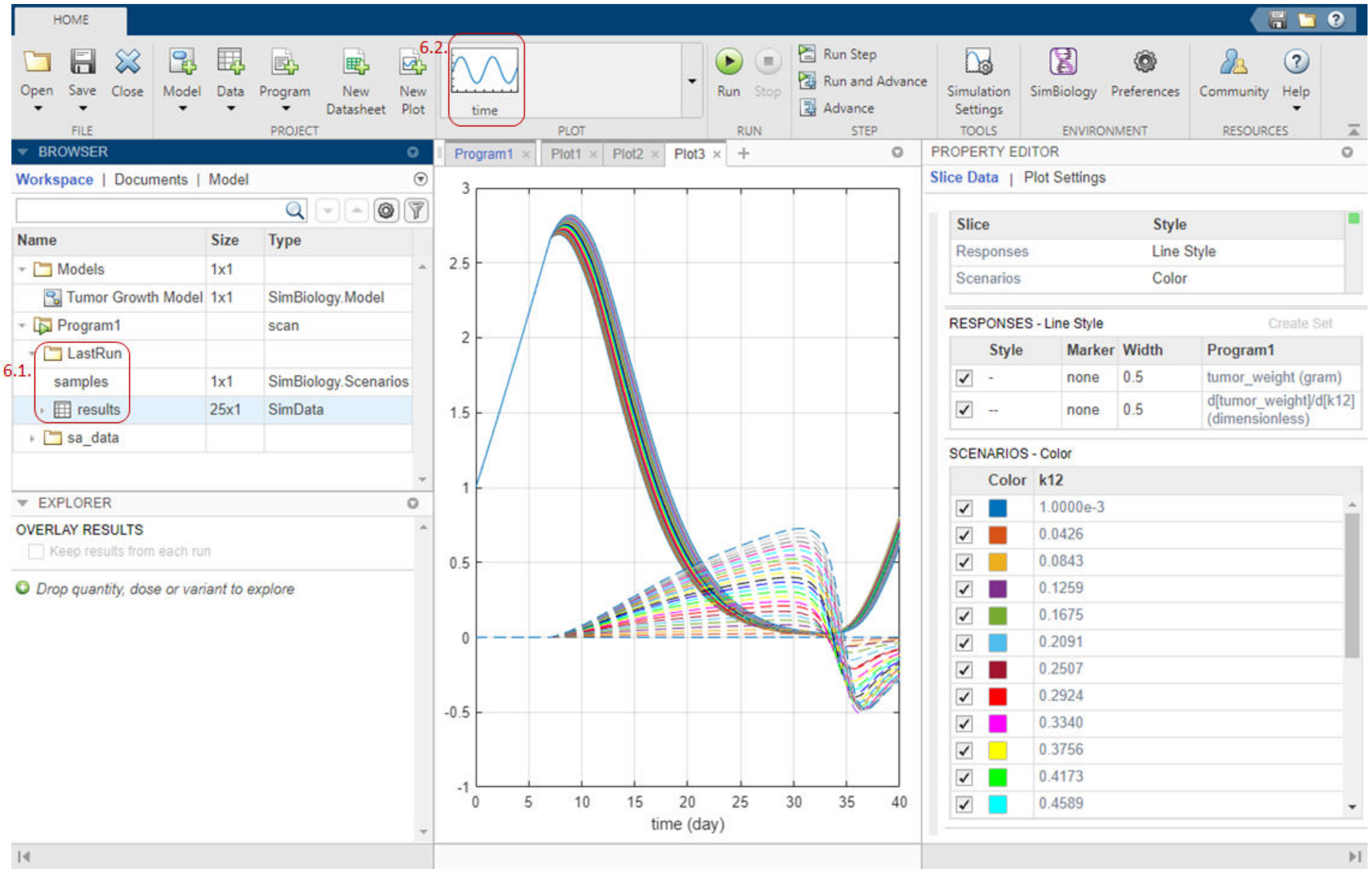
Output	Input	Name
☐	☑	k12
☐	☐	k21
☐	☐	L0
☐	☐	L1
☐	☐	w0
☐	☐	k2
☐	☐	k1
☐	☐	ke
☐	☐	Cl_Central
☑	☐	[Tumor Growth Model].tumor_weight

Disable sensitivity calculation

5 On the **Home** tab, click the **Run** button to run the whole program.

6 Once the simulation finishes, the sensitivity **results** are stored in the **LastRun** folder. Expand the **LastRun** folder. Click **results**. Then select **time** from the **Plot** section on the **Home** tab.

The **Responses** in the plot correspond to the tumor weight and the sensitivity of the tumor weight with respect to *k12*. Each response is plotted using a different line style. The **Scenarios** correspond to each parameter scan (simulation scenario). Each scenario is plotted using a different color.



Name	Size	Type
Models	1x1	
Tumor Growth Model	1x1	SimBiology.Model
Program1		scan
LastRun		
samples	1x1	SimBiology.Scenarios
results	25x1	SimData
sa_data		

Slice	Style
Responses	Line Style
Scenarios	Color

RESPONSES - Line Style

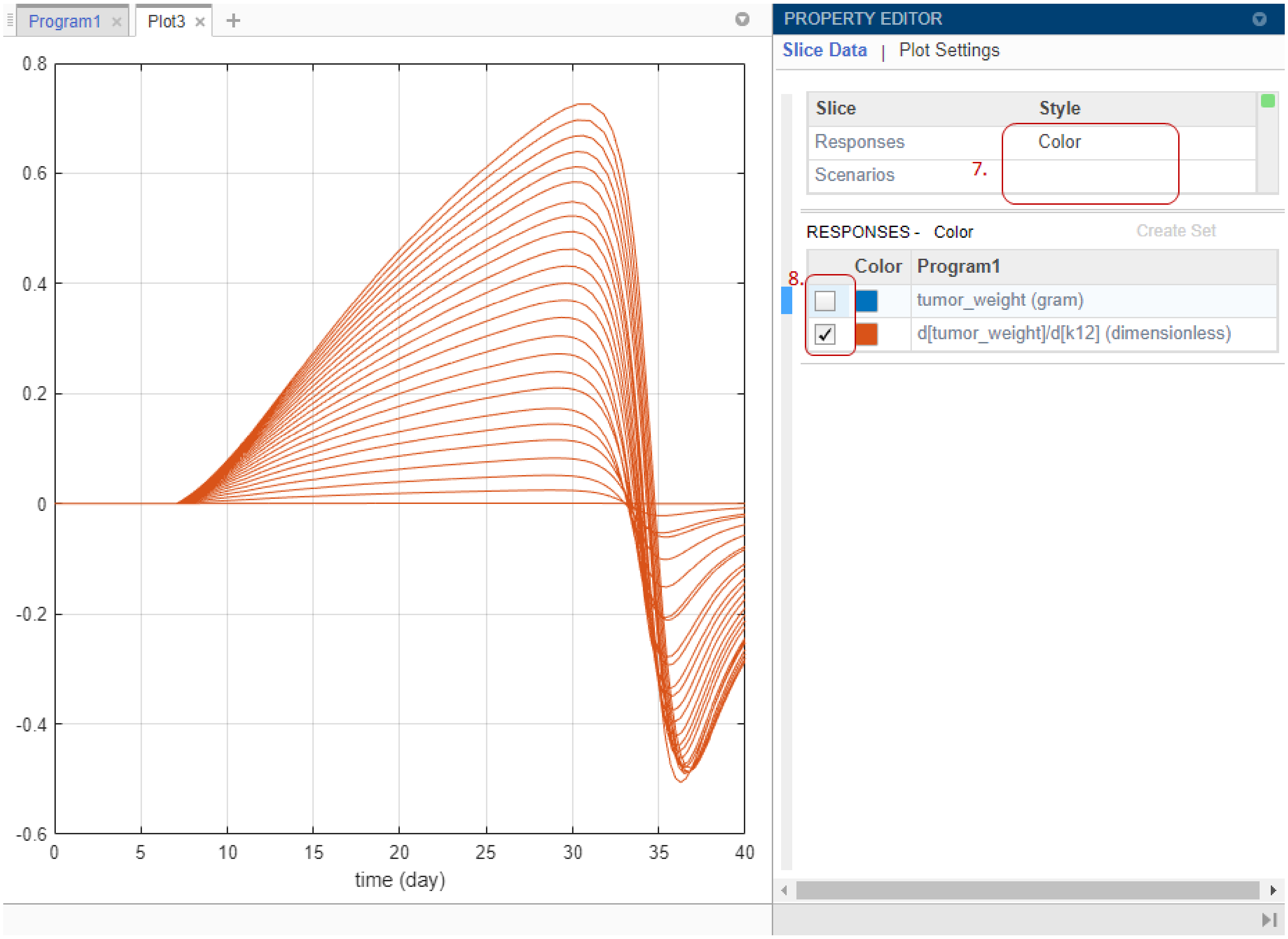
	Style	Marker	Width	Program1
☑	-	none	0.5	tumor_weight (gram)
☑	--	none	0.5	d[tumor_weight]/d[k12] (dimensionless)

SCENARIOS - Color

	Color	k12
☑	■	1.0000e-3
☑	■	0.0426
☑	■	0.0843
☑	■	0.1259
☑	■	0.1675
☑	■	0.2091
☑	■	0.2507
☑	■	0.2924
☑	■	0.3340
☑	■	0.3756
☑	■	0.4173
☑	■	0.4589

7 Customize the plot so that the plot uses a different color for each response instead. In the **Slice Data** table, clear the **Style** of **Scenarios** by selecting empty. Set **Responses** to **Color**.

8 In the **Responses** table, clear `tumor_weight` and keep only the sensitivity results.



Slice	Style
Responses	Color
Scenarios	

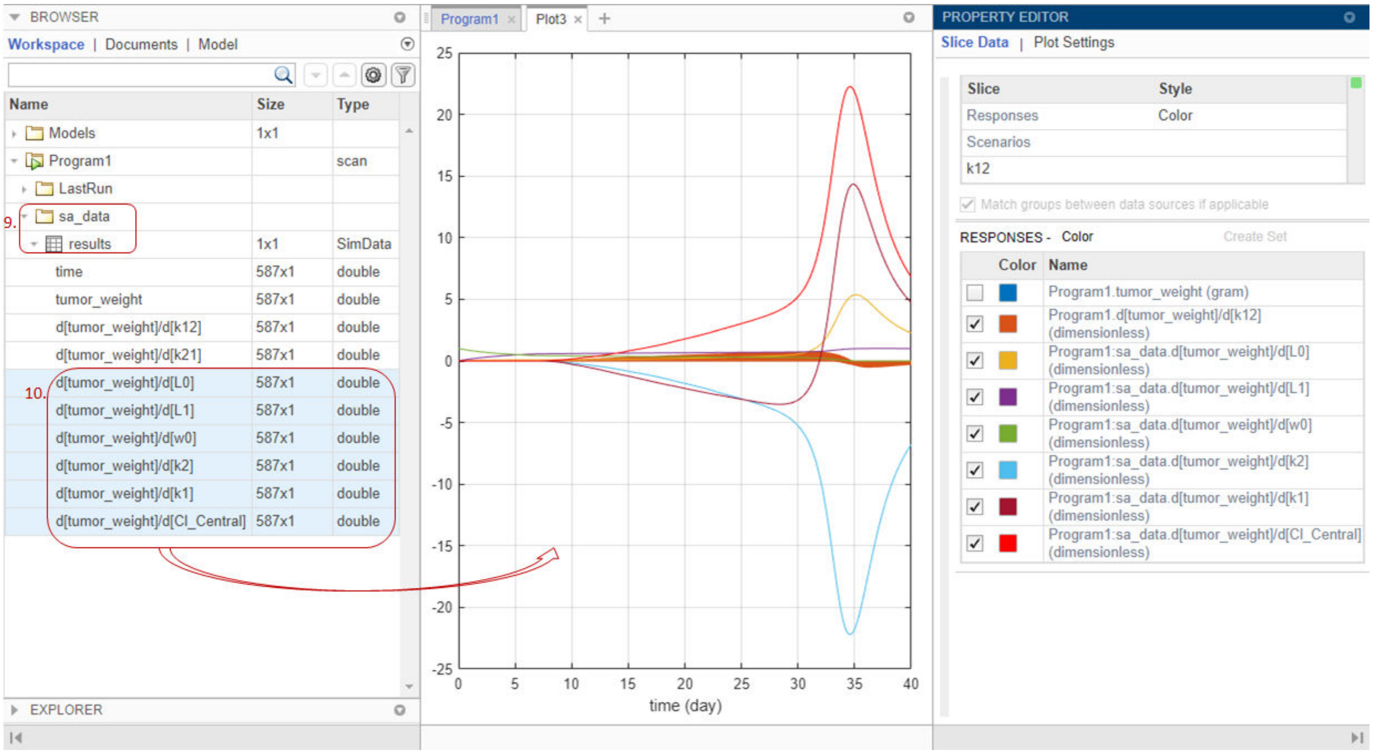
RESPONSES - Color

	Color	Program1
□	🟦	tumor_weight (gram)
☑	🟧	d[tumor_weight]/d[k12] (dimensionless)

The plot suggests that the tumor weight is sensitive to some *k12* parameter values when model variants different than the estimated parameter values are explored.

9 Plot the *k12* sensitivities together with the previous sensitivity results to see how sensitive the tumor growth is to *k12* relative to other model parameters. In the **Browser** pane, expand the folder **sa_data** that you saved previously. Then expand **results**.

10 Multiselect (**Ctrl + Click**) the sensitivity data entries for *L0*, *L1*, *w0*, *k2*, *k1*, and *Cl_Central*. Then drag them on to the *k12* sensitivity plot **Plot3**.



Name	Size	Type
Models	1x1	
Program1		scan
LastRun		
sa_data		
results	1x1	SimData
time	587x1	double
tumor_weight	587x1	double
d[tumor_weight]/d[k12]	587x1	double
d[tumor_weight]/d[k21]	587x1	double
d[tumor_weight]/d[L0]	587x1	double
d[tumor_weight]/d[L1]	587x1	double
d[tumor_weight]/d[w0]	587x1	double
d[tumor_weight]/d[k2]	587x1	double
d[tumor_weight]/d[k1]	587x1	double
d[tumor_weight]/d[CI_Central]	587x1	double

Slice	Style
Responses	Color
Scenarios	
k12	

☑ Match groups between data sources if applicable

RESPONSES - Color   Create Set

	Color	Name
□	🟦	Program1.tumor_weight (gram)
☑	🟧	Program1.d[tumor_weight]/d[k12] (dimensionless)
☑	🟨	Program1:sa_data.d[tumor_weight]/d[L0] (dimensionless)
☑	🟪	Program1:sa_data.d[tumor_weight]/d[L1] (dimensionless)
☑	🟩	Program1:sa_data.d[tumor_weight]/d[w0] (dimensionless)
☑	🟦	Program1:sa_data.d[tumor_weight]/d[k2] (dimensionless)
☑	🟥	Program1:sa_data.d[tumor_weight]/d[k1] (dimensionless)
☑	🟥	Program1:sa_data.d[tumor_weight]/d[CI_Central] (dimensionless)

The plot suggests that the tumor growth is still more sensitive to most of the other parameters than *k12*.

11 Save the results in a separate folder. Right-click **LastRun** and select `Save Data`. Enter `k12_data` as the data name.

Vary k21

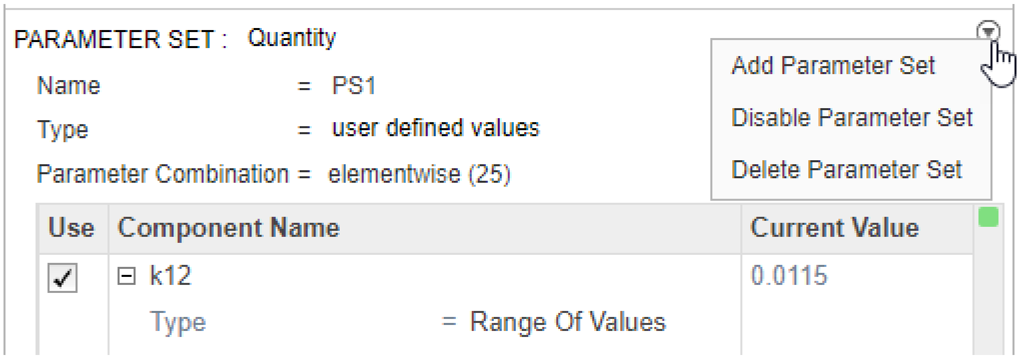
You can perform a similar analysis by varying *k21* and comparing its results with other parameter sensitivities.

1 Return to **Program1**. In the **Generate Samples** step, click **Add parameter set to scan**. A new **Parameter Set** section appears.

2 Double-click the empty cell in the **Component Name**, and enter *k21*. Set the same options as *k12*:

- **Type** — Range of Values
- **Spacing** — linear
- **Min** — 1e-3
- **Max** — 1
- **# Of Steps** — 25

3 Disable the first parameter set (**PS1**). Point to the top corner of **Parameter Set** section of **PS1** and click the action button. Then select `Disable Parameter Set`.



PARAMETER SET : Quantity

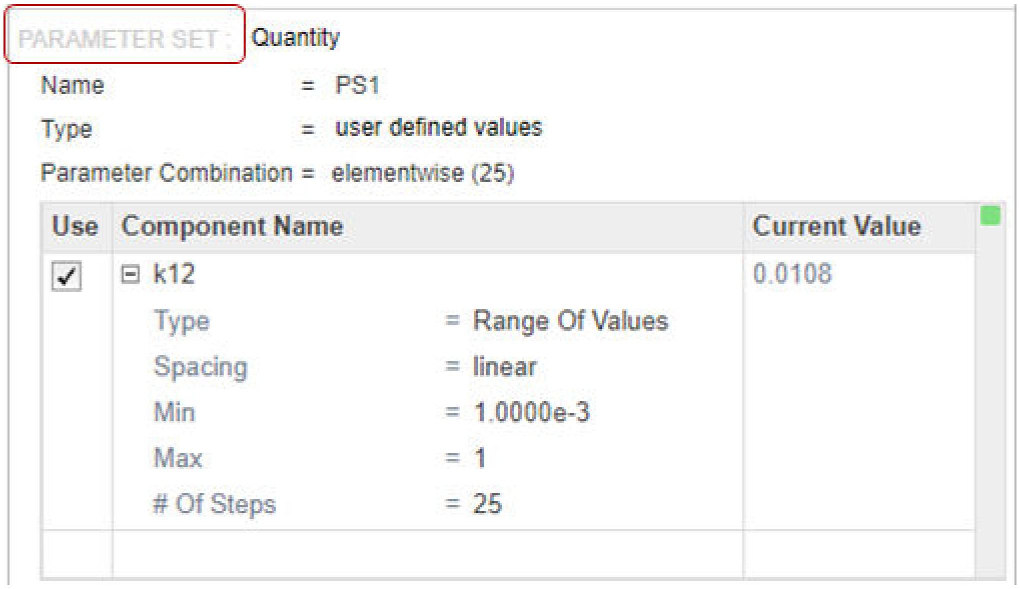
Name = PS1

Type = user defined values

Parameter Combination = elementwise (25)

Use	Component Name	Current Value
☑	⊟ k12	0.0115
	Type = Range Of Values	

The **Parameter Set** is then grayed out to indicate you have successfully disabled it.



PARAMETER SET : Quantity

Name = PS1

Type = user defined values

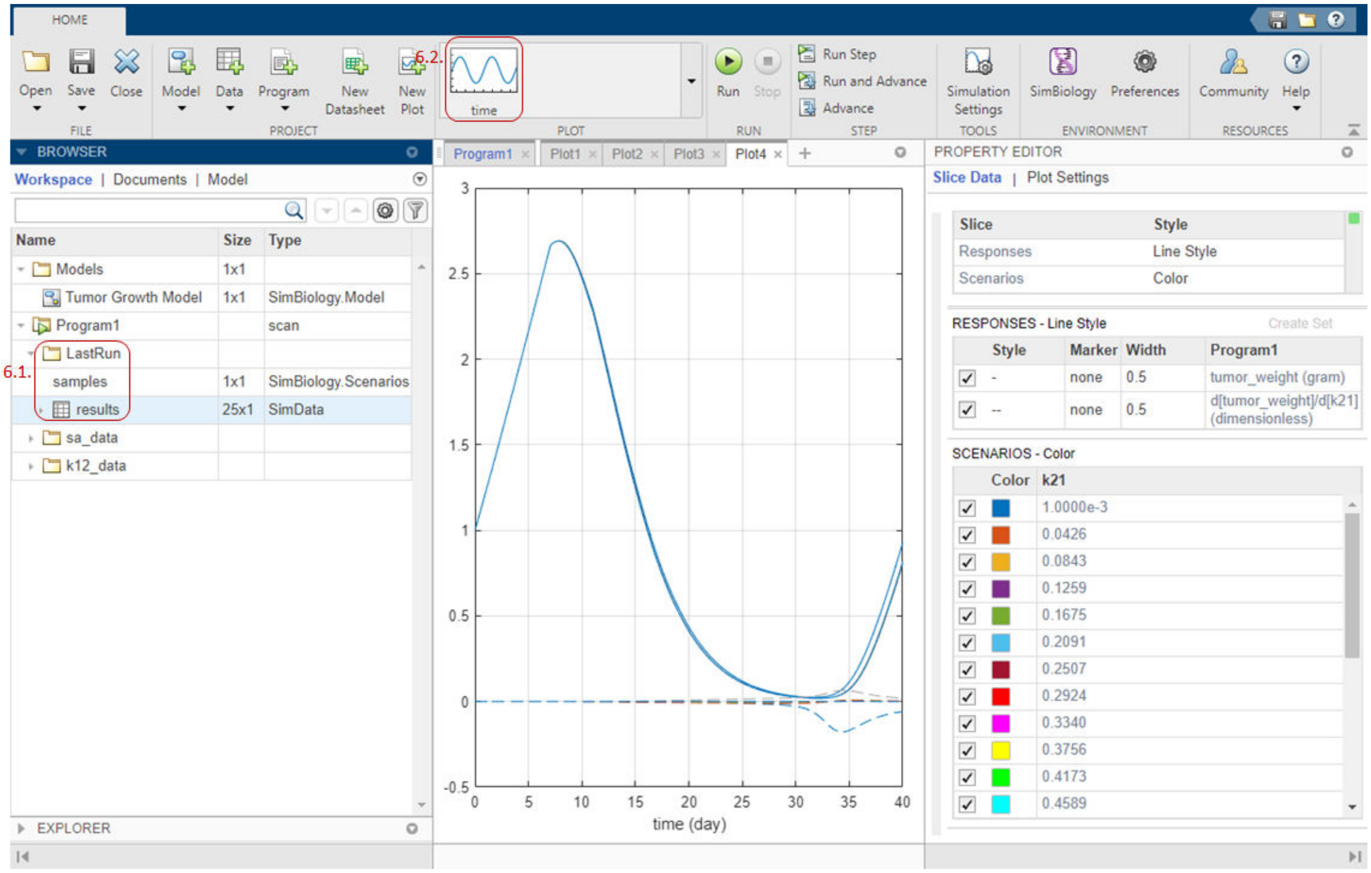
Parameter Combination = elementwise (25)

Use	Component Name	Current Value
☑	⊟ k12	0.0108
	Type = Range Of Values	
	Spacing = linear	
	Min = 1.0000e-3	
	Max = 1	
	# Of Steps = 25	

4 In the **Simulation** step, select *k21* as the only input and *tumor_weight* as the output.

5 Hit **Run** from the **Home** tab to run the program again.

6 Once the simulation finishes, the sensitivity **results** are stored in the **LastRun** folder. Expand the **LastRun** folder. Click **results**. Then select **time** from the **Plot** section on the **Home** tab.



Name	Size	Type
Models	1x1	
Tumor Growth Model	1x1	SimBiology.Model
Program1		scan
LastRun		
samples	1x1	SimBiology.Scenarios
results	25x1	SimData
sa_data		
k12_data		

Slice	Style
Responses	Line Style
Scenarios	Color

RESPONSES - Line Style

Create Set

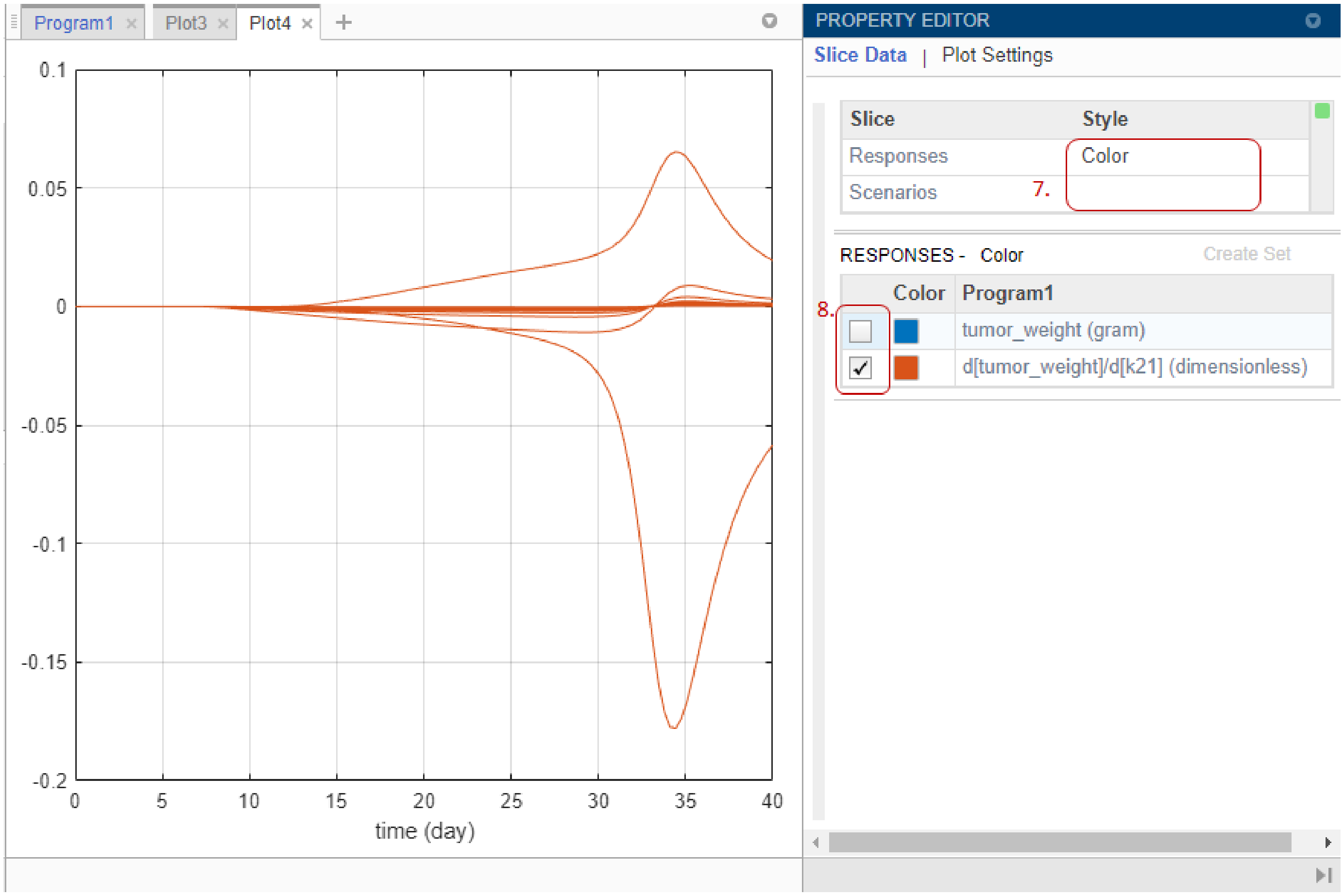
	Style	Marker	Width	Program1
☑	-	none	0.5	tumor_weight (gram)
☑	--	none	0.5	d[tumor_weight]/d[k21] (dimensionless)

SCENARIOS - Color

	Color	k21
☑	🟦	1.0000e-3
☑	🟧	0.0426
☑	🟨	0.0843
☑	🟪	0.1259
☑	🟩	0.1675
☑	🟦	0.2091
☑	🟥	0.2507
☑	🟥	0.2924
☑	🟪	0.3340
☑	🟨	0.3756
☑	🟩	0.4173
☑	🟦	0.4589

7 In the **Slice Data** table, clear the **Style** of **Scenarios** by selecting empty. Set **Responses** to **Color**.

8 In the **Responses** table, clear `tumor_weight` and keep only the sensitivity results.



Slice	Style
Responses	Color
Scenarios	

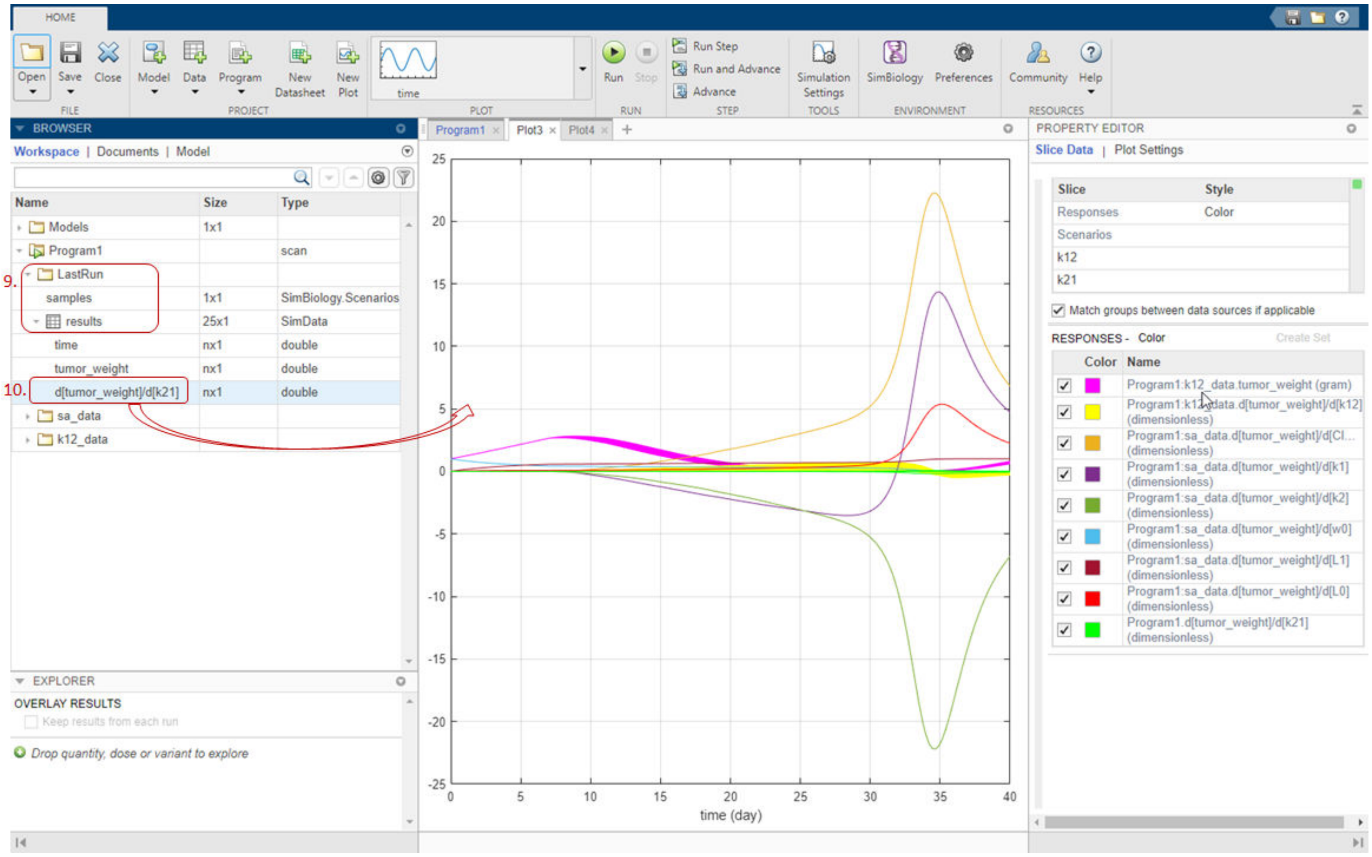
RESPONSES - Color

	Color	Program1
☐	🟦	tumor_weight (gram)
☑	🟧	d[tumor_weight]/d[k21] (dimensionless)

The plot suggests that the tumor weight is sensitive to some *k21* parameter values when model variants different than the estimated parameter values are explored.

9 Plot *k21* sensitivities together with the previous sensitivity results to compare. Click **Plot3**. In the **Browser** pane, expand the **LastRun** folder. Then expand **results**.

10 Drag the *k21* sensitivity data **d[tumor_weight]/d[k21]** on to **Plot3**.



Name	Size	Type
Models	1x1	
Program1		scan
LastRun		
samples	1x1	SimBiology.Scenarios
results	25x1	SimData
time	nx1	double
tumor_weight	nx1	double
d[tumor_weight]/d[k21]	nx1	double
sa_data		
k12_data		

□ Keep results from each run

Drop quantity, dose or variant to explore

Slice	Style
Responses	Color
Scenarios	
k12	
k21	

☑ Match groups between data sources if applicable

	Color	Name
☑	🟪	Program1:k12_data.tumor_weight (gram)
☑	🟨	Program1:k12_data.d[tumor_weight]/d[k12] (dimensionless)
☑	🟧	Program1:sa_data.d[tumor_weight]/d[Cl... (dimensionless)
☑	🟪	Program1:sa_data.d[tumor_weight]/d[k1] (dimensionless)
☑	🟩	Program1:sa_data.d[tumor_weight]/d[k2] (dimensionless)
☑	🟦	Program1:sa_data.d[tumor_weight]/d[w0] (dimensionless)
☑	🟥	Program1:sa_data.d[tumor_weight]/d[L1] (dimensionless)
☑	🟥	Program1:sa_data.d[tumor_weight]/d[L0] (dimensionless)
☑	🟩	Program1.d[tumor_weight]/d[k21] (dimensionless)

The plot suggests that the tumor growth is relatively more sensitive to most other parameters than *k21* and *k12*.

This example shows how to perform sensitivity analysis to find important model parameters to which tumor growth is sensitive. The initial analysis concludes that the tumor weight is not sensitive to some of the parameters. The example then explores the parameter space of these less sensitive parameters and compares the sensitivity results with those of other model parameters. You can use such sensitivity information for other analyses, such as to perform Monte Carlo simulations on page 1-46 by varying sensitive parameters to explore the model dynamics and biological variability.

References

[1] Simeoni, M., P. Magni, C. Cammia, G. De Nicolao, V. Croci, E. Pesenti, M. Germani, I. Poggesi, and M. Rocchetti. 2004. Predictive pharmacokinetic-pharmacodynamic modeling of tumor growth kinetics in xenograft models after administration of anticancer agents. *Cancer Research*. 64:1094-1101.

See Also

SimBiology Model Analyzer | `simbiology`

Model Biological Variability with Virtual Patients Using SimBiology Model Analyzer App

This example shows how to generate and simulate virtual patients using the **SimBiology Model Analyzer** app. In this example, a virtual patient is represented as a single realization of model parameters. The example uses a tumor growth model [1] to explore the variability of some model parameters that influence the tumor growth and investigate various dosing regimens to control it.

This example requires Statistics and Machine Learning Toolbox™.

Tumor Growth Model

The model used in this example is a SimBiology implementation of the pharmacokinetic/pharmacodynamic (PK/PD) model by Simeoni et al. It quantifies the effect of anticancer drugs on tumor growth kinetics from *in vivo* animal studies. The drug pharmacokinetics are described by a two-compartment model with IV bolus dosing and linear elimination (*ke*) from the *Central* compartment. Tumor growth is a biphasic process with an initial exponential growth followed by linear growth. The growth rate of the proliferating tumor cells is described by

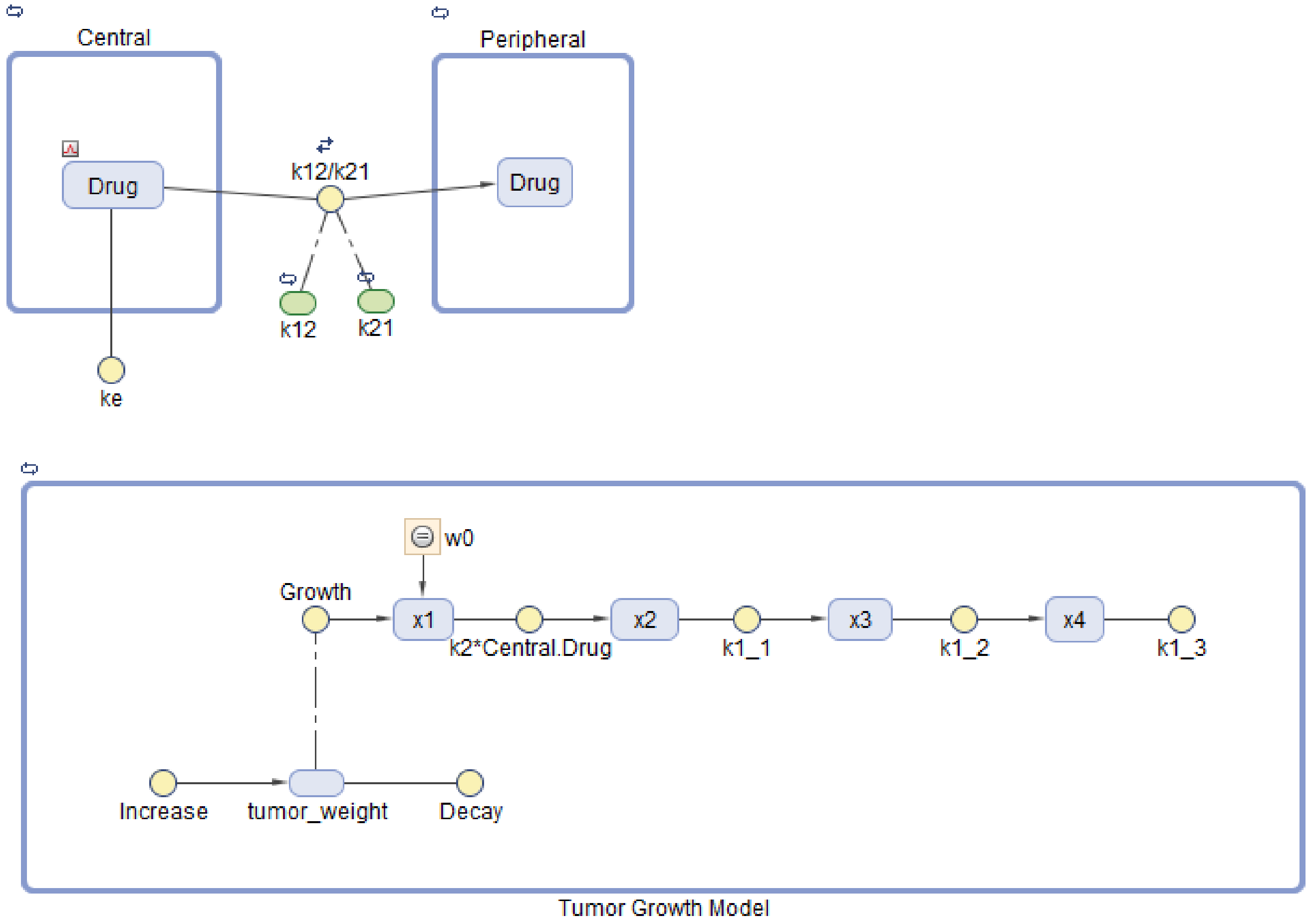
$$\frac{L_0 * x_1}{\left[1 + \left(\frac{L_0}{L_1} * w\right)^{\psi}\right]^{1/\psi}}$$

L_0, L_1, and Ψ are tumor growth parameters, x_1 is the weight of the proliferating tumor cells, and *w* is the total tumor weight. In the absence of any drugs, the tumor consists of proliferating cells only, that is, $w = x_1$. In the presence of an anticancer agent, a fraction of the proliferating cells is transformed into nonproliferating cells. The rate of this transformation is assumed to be a function of the drug concentration in the plasma and an efficacy factor k_2. The nonproliferating cells *x2* go through a series of transit stages (*x3* and *x4*) and are eventually cleared from the system. The flow-through of the transit compartments is modeled as a first-order process with the rate constant k_1.

The SimBiology model makes these adjustments to the pharmacodynamics of tumor growth:

- Instead of defining the tumor weight as the sum of *x1*, *x2*, *x3*, and *x4*, the model defines the tumor weight by the reaction named *Increase*, `null → tumor_weight`, with the reaction rate $\left(2 * L_1 * L_0 * {x_1}^2 / L_1 + 2 * L_0 * x_1\right) * tumor\_weight$.

  tumor_weight is the total tumor weight, x_1 is the weight of the proliferating tumor cells, and L_0, and L_1 are tumor growth parameters.
- Similarly, the model defines the decrease in tumor weight by the reaction named *Decay*, `tumor_weight → null`, with the reaction rate $k_1 * x_4$. The constant k_1 is the forward rate parameter, and x_4 is the last species in the series of transit reductions in tumor weight.
- *ke* is a function of the clearance and the volume of the central compartment: `ke = Cl_Central/Central`.



Description of Virtual Population

The virtual population in this example is represented by virtual patients, specified as distinct sets of patient-specific parameter values. Suppose that, based on prior knowledge or sensitivity analysis on page 1-30 of the model, the tumor growth is sensitive to these model parameters: *L0*, *L1*, *w0*, *k1*, *k2*, and *Cl_Central*. Assuming that these biological parameters follow the lognormal distribution (and must always be positive), you can generate virtual patients that represent different parameter values drawn from the joint lognormal distribution.

The lognormal distribution uses these parameters:

- *mu* – Mean of logarithmic values
- *sigma* – Standard deviation of logarithmic values

For details, see “Lognormal Distribution” (Statistics and Machine Learning Toolbox).

In this example, *mu* for each parameter is the log of the model value, and *sigma* is assumed to be 0.01 for all the parameters.

Dosing Strategies

In this example, an anticancer drug is used to control the tumor growth. Each virtual patient receives the same amount of the drug on the same schedule. The tumor growth response is simulated for each patient. Dose amounts are then varied to find the range of dose amounts that can suppress the tumor growth of many virtual patients in the population.

Generate Samples to Represent Virtual Patients

The following steps show how to draw sample values from the joint probability distribution for model parameters that are sensitive to tumor growth, to represent a group of virtual patients.

Load Tumor Growth Model

1 At the MATLAB command line, load the model (*m1*) by entering:

```
sbioloadproject tumor_growth_vpop_sa.sbproj
```

2 Open the **SimBiology Model Analyzer** app by typing `simBiologyModelAnalyzer` at the command line or by clicking **SimBiology Model Analyzer** on the **Apps** tab.

3 On the **Home** tab, select **Model** > `Import Model from MATLAB`. Select **Tumor Growth Model**.

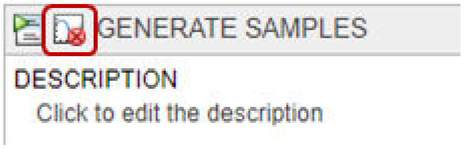
Define Joint Probability Distribution

Create a program to draw sample values from the joint lognormal distribution for tumor growth sensitive parameters: *L0*, *L1*, *w0*, *k1*, *k2*, and *Cl_Central*.

1 On the **Home** tab, select **Program** > **Generate Samples**. A new program opens.

2 In the **Variants** section in the **Model** step of the program, click the option to view the variants to be applied. Then select **parameterEstimates**, which contains the estimated parameter values.

3 In the **Generate Samples** step, click the plot button to disable default plot generation. You will plot the samples later on.



4 In the **Parameter Set** section, set **Type** to `values from a distribution`.

5 Set **Number Of Samples** to 25.

6 Double-click the empty cell in the **Component Name** column, and enter **L0**.

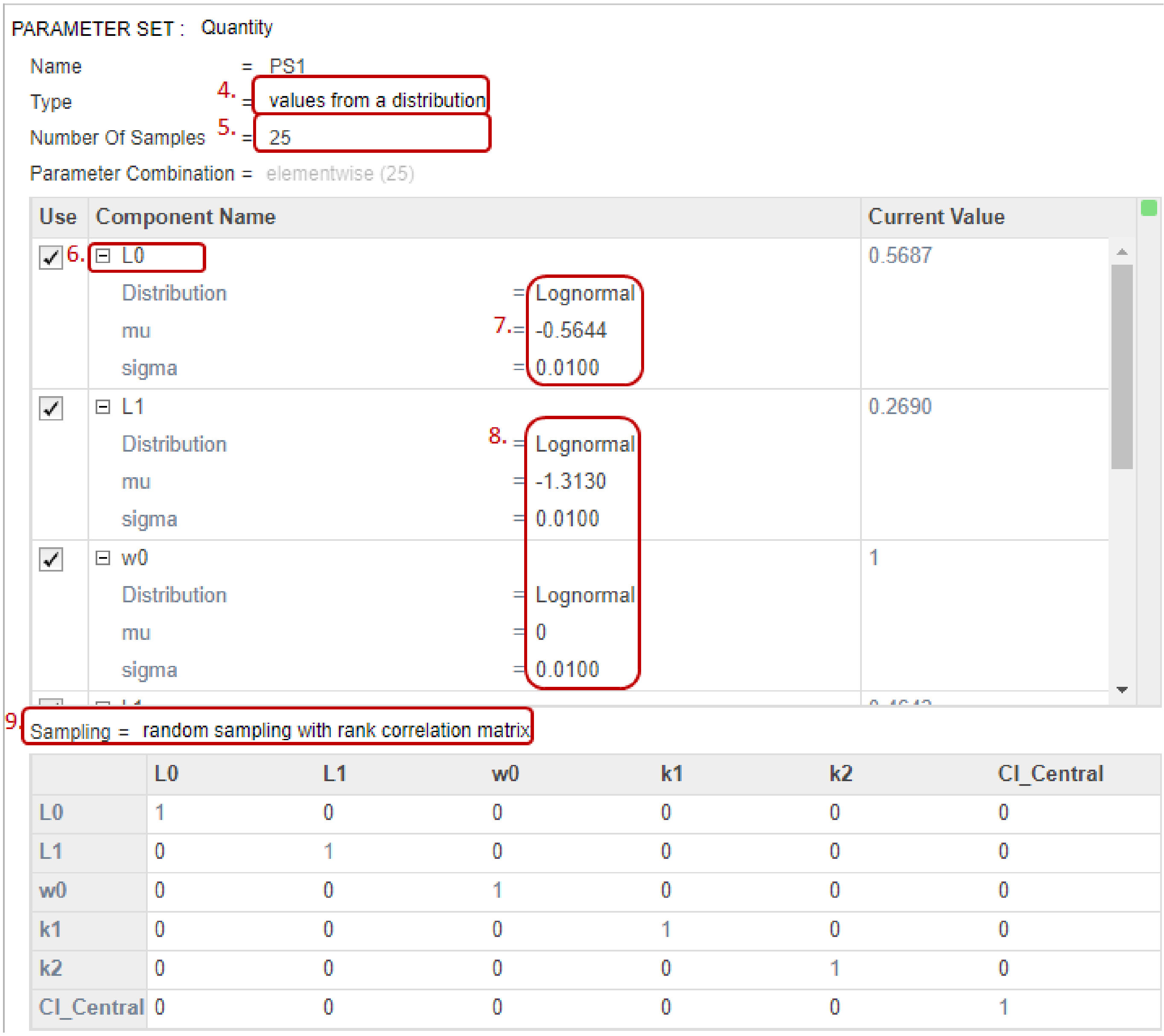
7 Change **Distribution** for **L0** to `Lognormal`. By definition, *mu* is the mean of logarithmic values. So, change *mu* to -0.5644, which is `log(Current Value)` or `log(0.5687)`. Change *sigma* to 0.01.

8 Double-click the empty cell in the **Component Name** column, and add *L1*. Repeat the same process to change the distribution to lognormal and set the *mu* and *sigma* values. Similarly, add *w0*, *k1*, *k2*, and *ke*. This table lists the corresponding *mu* values of these parameters that you can copy and paste in the software. Change *sigma* to 0.01 for each parameter.

Parameter	Current Value	*mu* or Log(Current Value)
L0	0.5687	-0.5644
L1	0.2690	-1.3130
w0	1	0
k1	0.4643	-0.7672
k2	8.4200e-4	-7.0797
Cl_Central	0.6672	-0.4047

9 For **Sampling**, use the default option: `random sampling with rank correlation matrix`, where the matrix is an identity matrix.

The following figure shows the **Parameter Set** section with all parameters configured. The red numbers correspond to the preceding steps.



PARAMETER SET : Quantity

Name = PS1

Type 4. = values from a distribution

Number Of Samples 5. = 25

Parameter Combination = elementwise (25)

Use	Component Name		Current Value
☑	6. ⊟ L0		0.5687
	Distribution	= Lognormal	
	mu	7. = -0.5644	
	sigma	= 0.0100	
☑	⊟ L1		0.2690
	Distribution	8. = Lognormal	
	mu	= -1.3130	
	sigma	= 0.0100	
☑	⊟ w0		1
	Distribution	= Lognormal	
	mu	= 0	
	sigma	= 0.0100	

9. Sampling = random sampling with rank correlation matrix

	L0	L1	w0	k1	k2	Cl_Central
L0	1	0	0	0	0	0
L1	0	1	0	0	0	0
w0	0	0	1	0	0	0
k1	0	0	0	1	0	0
k2	0	0	0	0	1	0
Cl_Central	0	0	0	0	0	1

10 (Optional) Save the project under a new name by selecting **Save > Save Project As** on the **Home** tab.

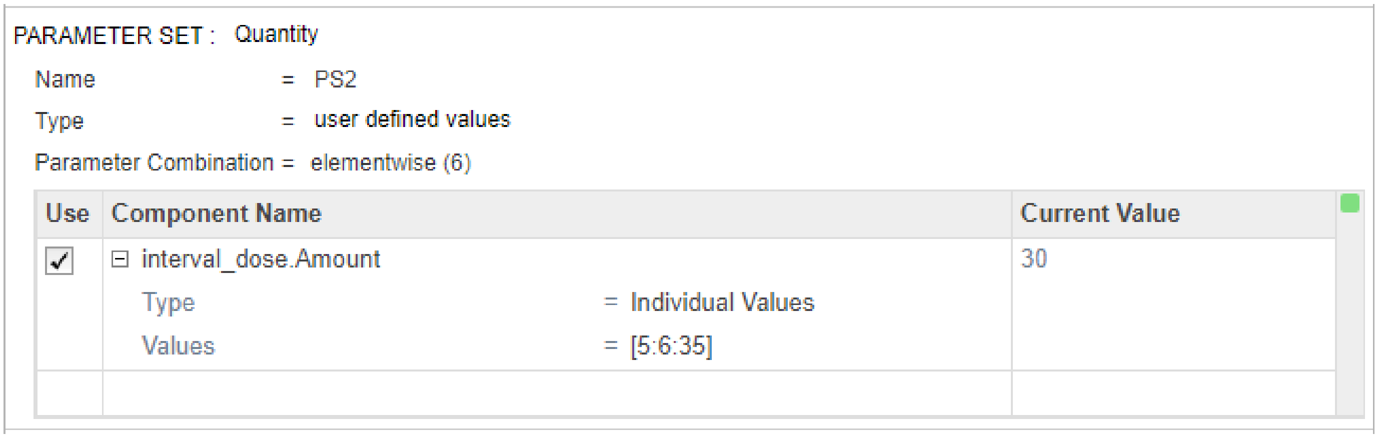
Define Dosing Strategies

Add the dosing information as another parameter set by specifying different dose amount—specifically, six dose amounts that are equally spaced from 5 to 35 mg. Then combine the doses with the first parameter set (virtual patients) using the Cartesian combination method. This method combines every virtual patient with every dose amount to generate a total of 150 iterations (or simulation scenarios). For details, see “Combine Simulation Scenarios in SimBiology” on page 3-23.

1 At the bottom of the **Generate Samples** step, click **Add parameter set to scan**. Another parameter set (**PS2**) appears.

2 Double-click the empty cell in the **Component Name** column, and type `interval`. A list of choices appears. Select `interval_dose.Amount`.

3 Set **Type** to `Individual Values` and set **Values** to `[5:6:35]`.



PARAMETER SET : Quantity

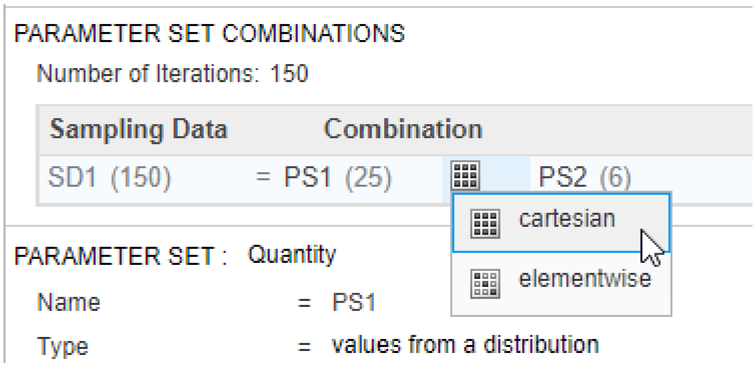
Name = PS2

Type = user defined values

Parameter Combination = elementwise (6)

Use	Component Name	Current Value
☑	⊟ interval_dose.Amount	30
	Type = Individual Values	
	Values = [5:6:35]	

4 Near the top of the **Generate Samples** step, under **Parameter Set Combinations**, ensure that the **Combination** type is set to `cartesian` to combine PS1 and PS2.



PARAMETER SET COMBINATIONS

Number of Iterations: 150

Sampling Data	Combination
SD1 (150)	= PS1 (25) ▦ PS2 (6)

PARAMETER SET : Quantity

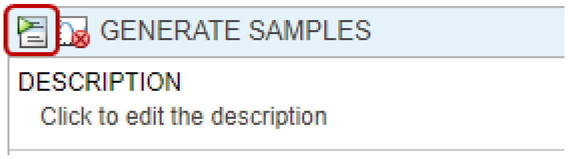
Name = PS1

Type = values from a distribution

Generate and Visualize Parameter and Dose Combinations

Once you have defined the joint lognormal distribution for model parameters, range of dose amounts, and the combination method, you can run the step to generate the different parameters and dose combinations.

1 Click the **Run this program step** button to generate samples.



GENERATE SAMPLES

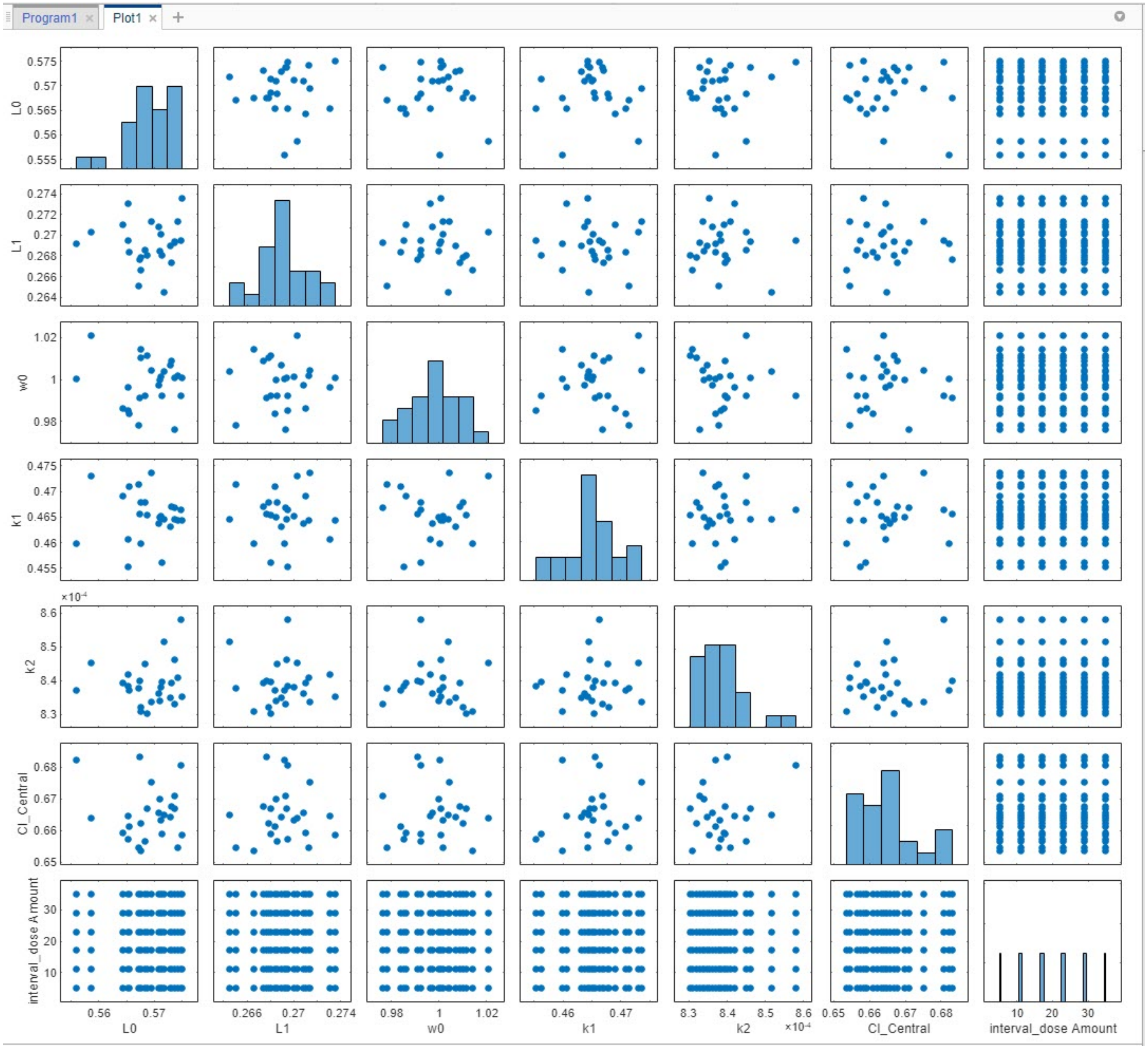
DESCRIPTION

Click to edit the description

Tip You can run every program *execution* step separately. An execution step includes the run button next to the name the step. Running an individual step is especially helpful if the program contains multiple steps and you want to see the intermediate results from a particular step. By doing so, you can make adjustments as needed before running the next step or the whole program. To run the whole program, click the **Run** button on the **Home** tab.

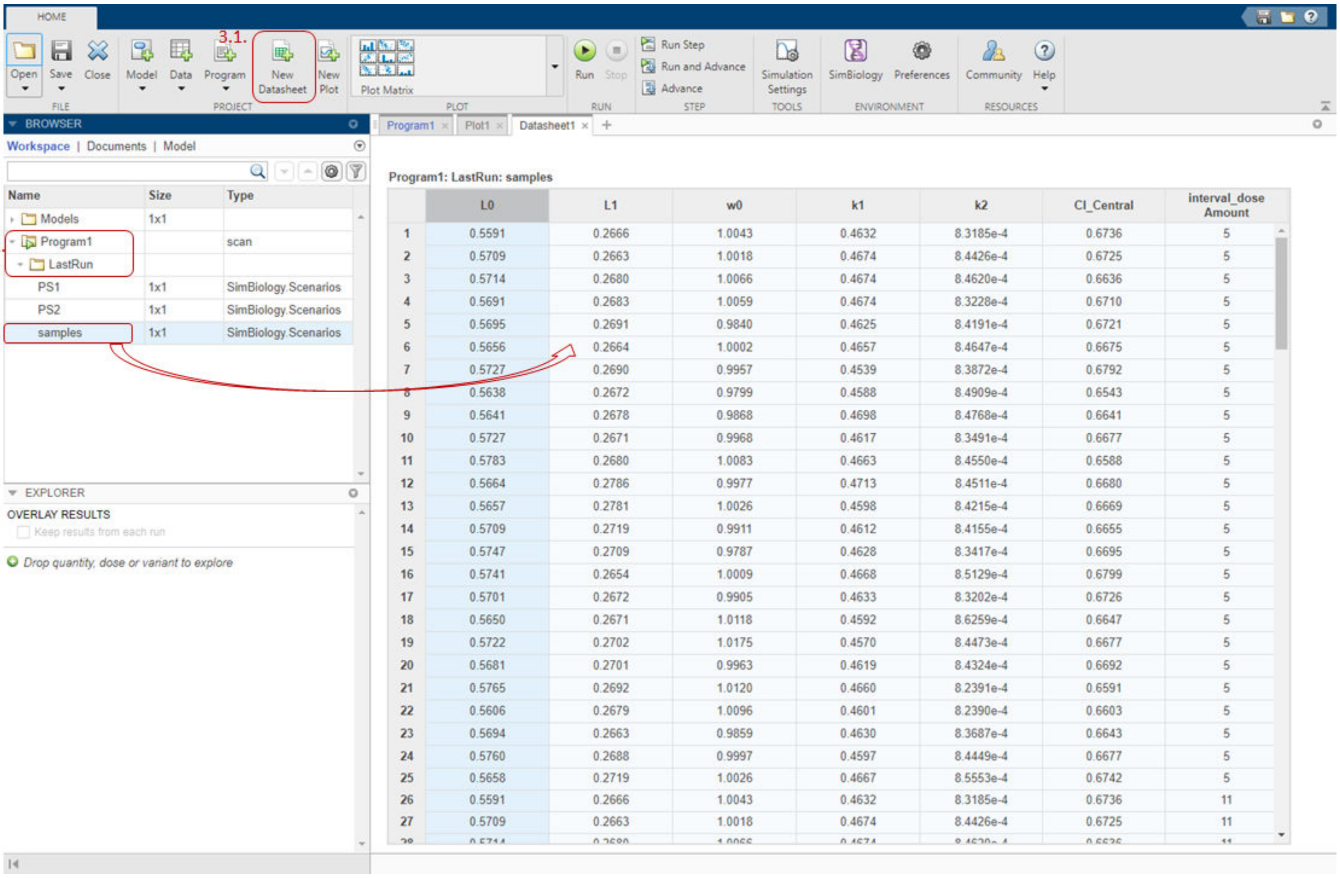
By default, the app stores the generated **samples** in the **LastRun** folder of the program.

2 Visualize the generated samples. In the **Browser** pane on the left, expand the **Program1** folder. Expand the **LastRun** folder, and then click **samples**. In the **Plot** section on the **Home** tab, click **Plot Matrix**. The plot shows the distribution of each parameter varied around its model value, except the dose amounts, which appear in a uniform range. Note that your plot might vary from the plot shown here due to randomness.



Note You can enable default plot generation to display the plot automatically every time you run the step. To do so, click the plot button at the top of the **Generate Samples** step. Keep in mind that when the program step has a lot of samples, plotting all the samples can be time consuming.

3 Display the generated virtual patients and dose combinations numerically in a tabular format. On the **Home** tab, click **New Datasheet**. From the **LastRun** folder of the program, drag **samples** into the new datasheet. Each row of the table represents a simulation scenario with different model parameter values and dose amounts.



Name	Size	Type
Models	1x1	
Program1		scan
LastRun		
PS1	1x1	SimBiology.Scenarios
PS2	1x1	SimBiology.Scenarios
samples	1x1	SimBiology.Scenarios

Program1: LastRun: samples

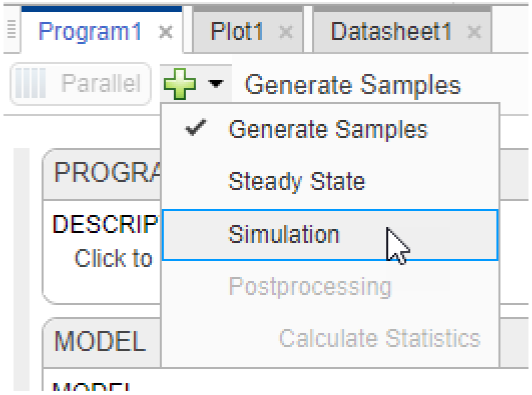
	L0	L1	w0	k1	k2	Cl_Central	interval_dose Amount
1	0.5591	0.2666	1.0043	0.4632	8.3185e-4	0.6736	5
2	0.5709	0.2663	1.0018	0.4674	8.4426e-4	0.6725	5
3	0.5714	0.2680	1.0066	0.4674	8.4620e-4	0.6636	5
4	0.5691	0.2683	1.0059	0.4674	8.3228e-4	0.6710	5
5	0.5695	0.2691	0.9840	0.4625	8.4191e-4	0.6721	5
6	0.5656	0.2664	1.0002	0.4657	8.4647e-4	0.6675	5
7	0.5727	0.2690	0.9957	0.4539	8.3872e-4	0.6792	5
8	0.5638	0.2672	0.9799	0.4588	8.4909e-4	0.6543	5
9	0.5641	0.2678	0.9868	0.4698	8.4768e-4	0.6641	5
10	0.5727	0.2671	0.9968	0.4617	8.3491e-4	0.6677	5
11	0.5783	0.2680	1.0083	0.4663	8.4550e-4	0.6588	5
12	0.5664	0.2786	0.9977	0.4713	8.4511e-4	0.6680	5
13	0.5657	0.2781	1.0026	0.4598	8.4215e-4	0.6669	5
14	0.5709	0.2719	0.9911	0.4612	8.4155e-4	0.6655	5
15	0.5747	0.2709	0.9787	0.4628	8.3417e-4	0.6695	5
16	0.5741	0.2654	1.0009	0.4668	8.5129e-4	0.6799	5
17	0.5701	0.2672	0.9905	0.4633	8.3202e-4	0.6726	5
18	0.5650	0.2671	1.0118	0.4592	8.6259e-4	0.6647	5
19	0.5722	0.2702	1.0175	0.4570	8.4473e-4	0.6677	5
20	0.5681	0.2701	0.9963	0.4619	8.4324e-4	0.6692	5
21	0.5765	0.2692	1.0120	0.4660	8.2391e-4	0.6591	5
22	0.5606	0.2679	1.0096	0.4601	8.2390e-4	0.6603	5
23	0.5694	0.2663	0.9859	0.4630	8.3687e-4	0.6643	5
24	0.5760	0.2688	0.9997	0.4597	8.4449e-4	0.6677	5
25	0.5658	0.2719	1.0026	0.4667	8.5553e-4	0.6742	5
26	0.5591	0.2666	1.0043	0.4632	8.3185e-4	0.6736	11
27	0.5709	0.2663	1.0018	0.4674	8.4426e-4	0.6725	11
28	0.5714	0.2680	1.0066	0.4674	8.4620e-4	0.6636	11

Perform Monte Carlo Simulations

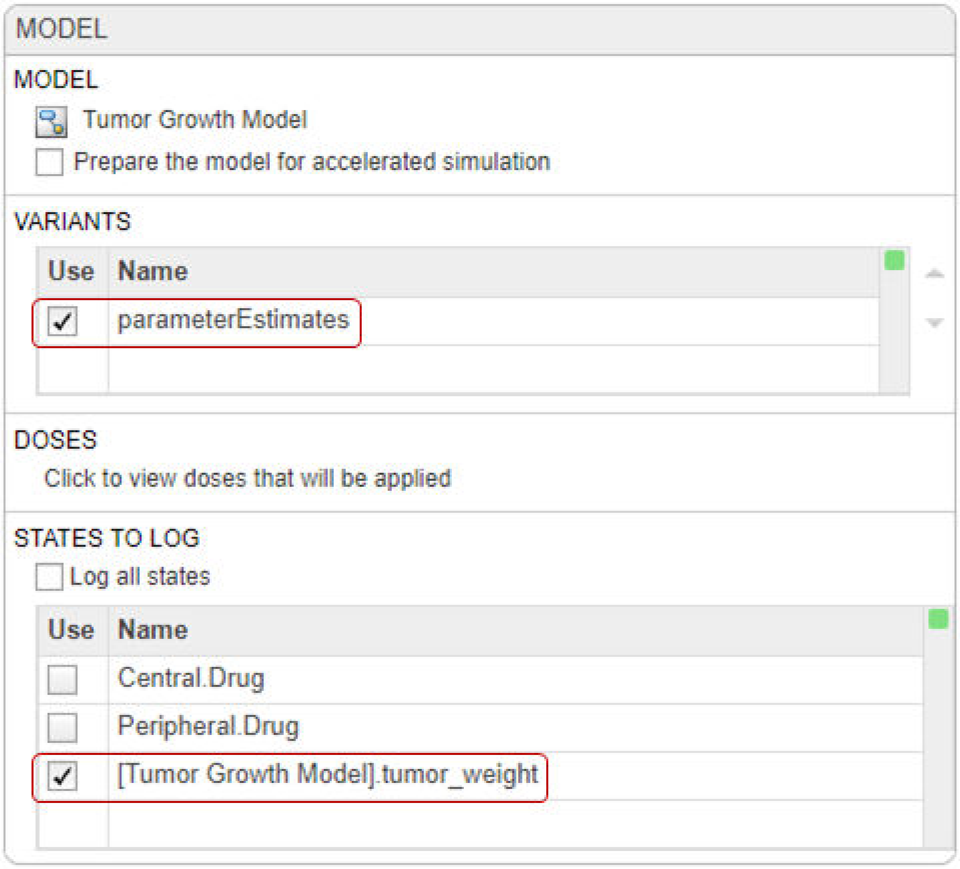
Once you have the samples ready, you can simulate the model to explore the tumor growth of virtual patients receiving different dose amounts.

1 Go back to the program by clicking the **Program1** tab.

2 Click the (+) icon at the upper left and click **Simulation**.



3 In the **Model** setup step, in the **Variants** section, make sure that **parameterEstimates** is selected. In the **States To Log** section, click the option for viewing the states, and then clear all check boxes except **[Tumor Growth Model].tumor_weight**.



MODEL

MODEL

Tumor Growth Model

□ Prepare the model for accelerated simulation

VARIANTS

Use	Name
☑	parameterEstimates

DOSES

Click to view doses that will be applied

STATES TO LOG

□ Log all states

Use	Name
□	Central.Drug
□	Peripheral.Drug
☑	[Tumor Growth Model].tumor_weight

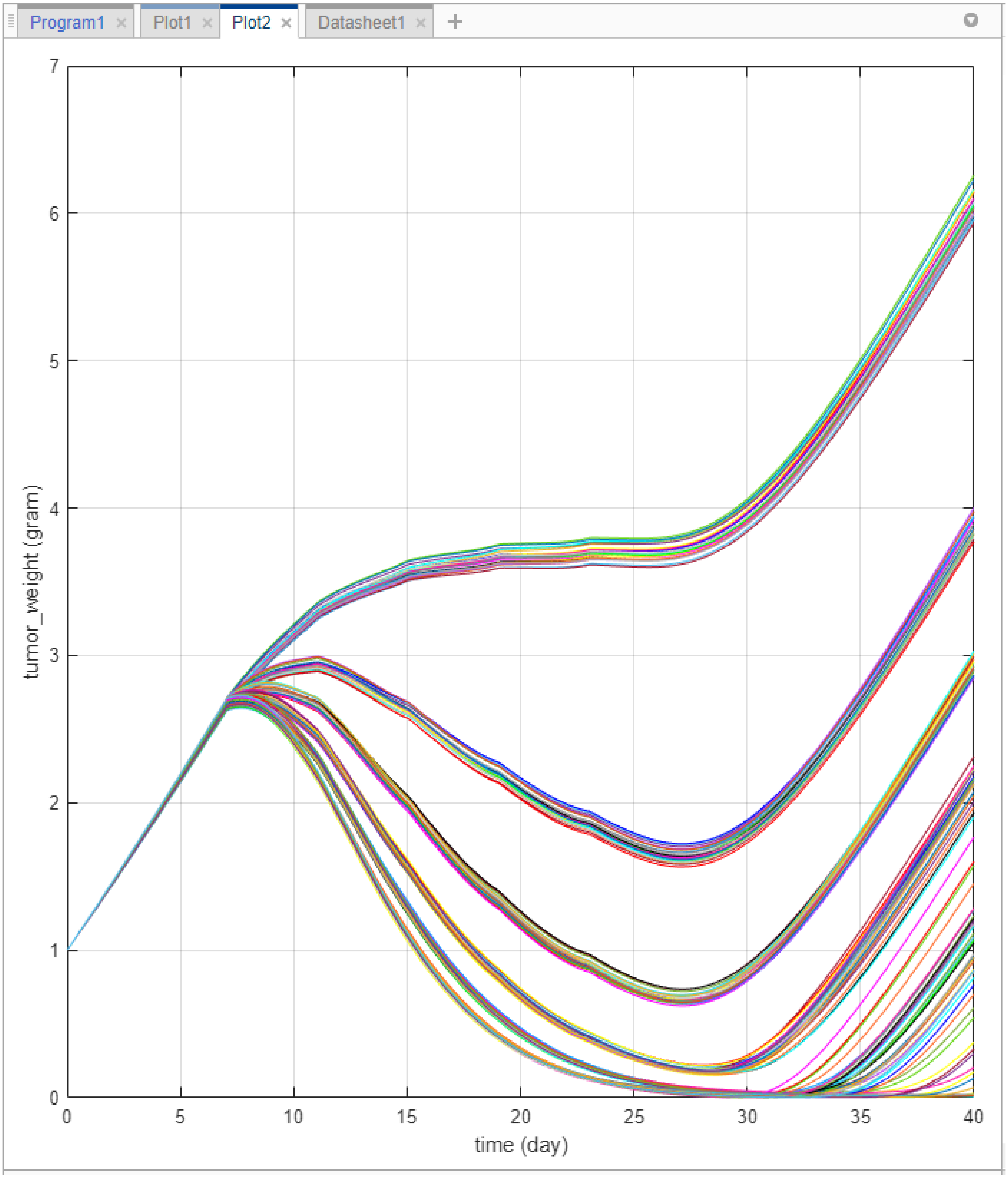
4 At the top of the **Simulation** step (scroll down to see this step), click the plot button to disable default plot generation.

5 Click the **Run this program step** button to simulate the model.

The app simulates the generated scenarios from the **Generate Samples** step that you ran previously. You do not need to rerun the step.

Once the simulation finishes, the app saves the simulation **results** in the **LastRun** folder.

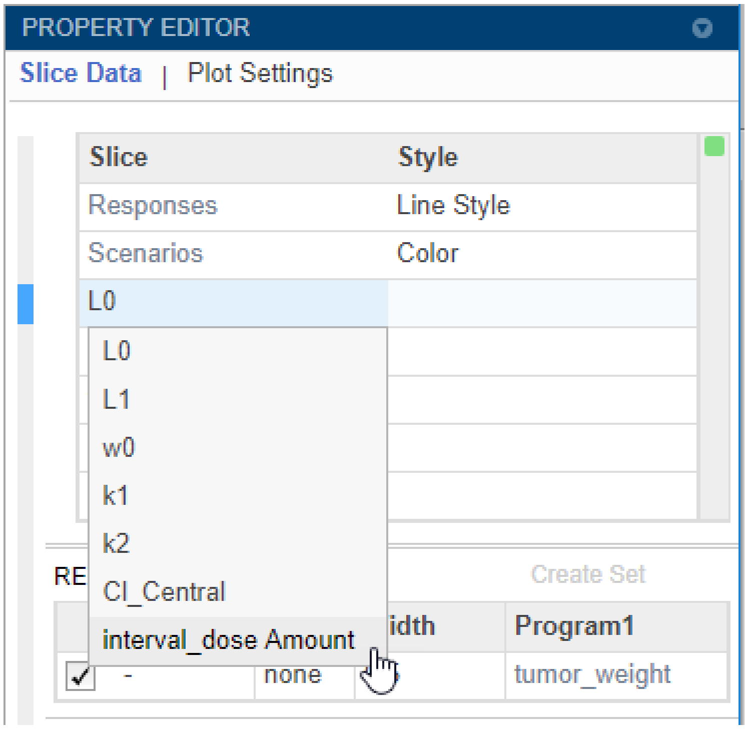
6 In the **LastRun** folder in the **Browser** pane, click **results**. In the **Plot** section on the **Home** tab, click **time**. Each line in the plot represents the tumor weight profile of each simulation scenario.


Program1 ×
Plot1 ×
Plot2 ×
Datasheet1 ×
+
tumor_weight (gram)
0
1
2
3
4
5
6
7
time (day)
0
5
10
15
20
25
30
35
40

The plot shows that the tumor weight profiles appear in groups, corresponding to different dose amounts. To better visualize this observation, you can slice the data by dose amounts. To do so, you can use the **Slice Data** table, which contains a summary of slicing variables that are currently being used in the plot and their corresponding plot styles.

Tip Plots are backed by data that are currently present in the app workspace. Plots are not snapshots. When the data (either experimental data or simulation results) is removed or changed, the plots are also updated according to the changes in the underlying data.

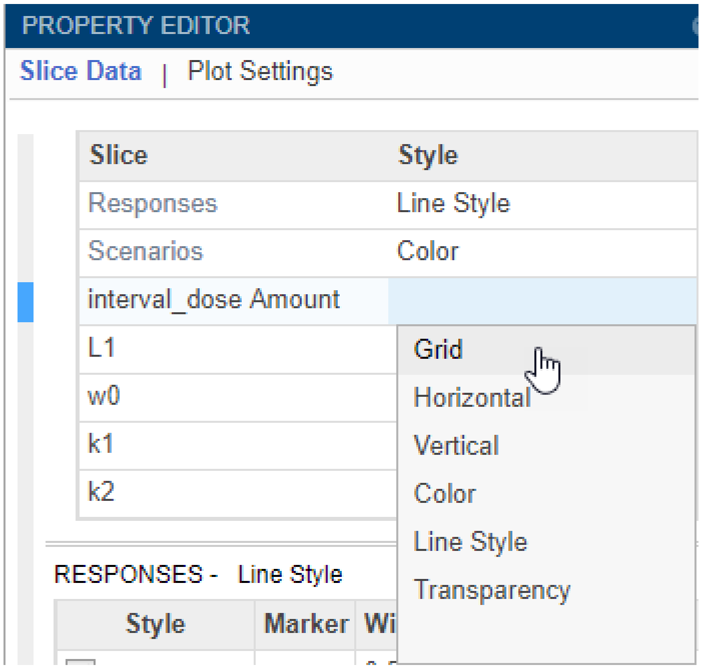
7 In the **Slice Data** table, double-click `L0` and select `interval_dose Amount`.



Slice	Style
Responses	Line Style
Scenarios	Color
L0	

Tip You can slice data using different slicing variables. Each slicing variable appears in the plot with a different visual *style* (or channel) such as color, line style, and axes position. Slicing variables can represent attributes of data, such as responses or scenarios (that is, groups or simulation runs). Slicing variables can also be covariates or parameter values associated with a scenario or group. By default, the app provides slicing variables for different response variables and different scenarios in the plotted data. You can add other visual styles (or channels) for sets of responses and associated parameter or covariate variables.

8 Double-click the empty cell in the **Style** column and select `Grid`.

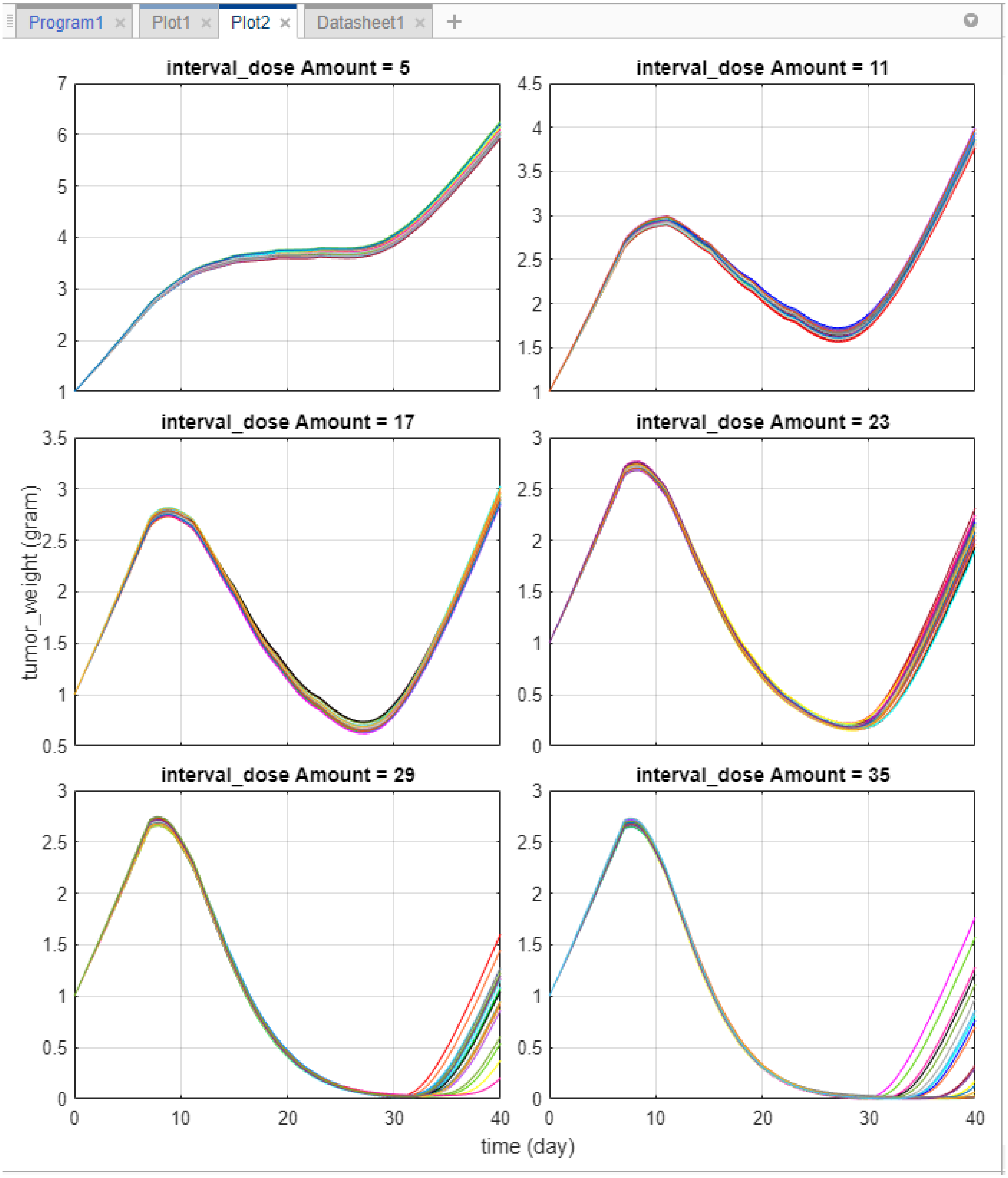


Slice	Style
Responses	Line Style
Scenarios	Color
interval_dose Amount	
L1	
w0	
k1	
k2	

RESPONSES - Line Style

Style	Marker	Wi

The app updates **Plot2** to show each dose amount on its own axes.

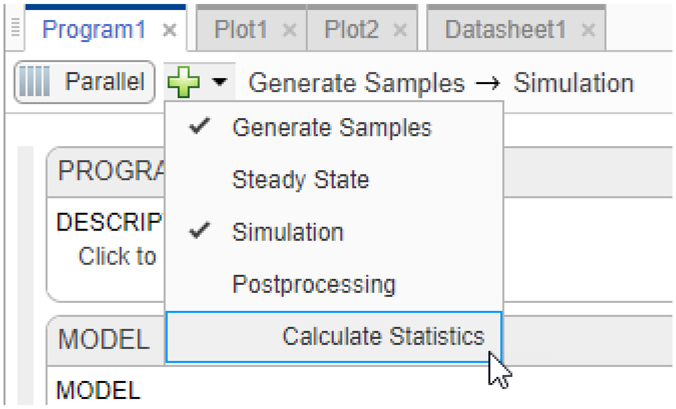

Program1
Plot1
Plot2
Datasheet1
+
interval_dose Amount = 5
7
6
5
4
3
2
1
interval_dose Amount = 11
4.5
4
3.5
3
2.5
2
1.5
1
interval_dose Amount = 17
3.5
3
2.5
2
1.5
1
0.5
interval_dose Amount = 23
3
2.5
2
1.5
1
0.5
0
interval_dose Amount = 29
3
2.5
2
1.5
1
0.5
0
interval_dose Amount = 35
3
2.5
2
1.5
1
0.5
0
tumor_weight (gram)
0
10
20
30
40
0
10
20
30
40
time (day)

The plot shows that different dose amounts play critical roles in the tumor growth of virtual patients. From this plot, you can obtain some initial insights into the optimal dose amounts and dose scheduling. For instance, suppose that the target tumor weight to reach is 0.5 grams. The simulation results indicates that a dose amount of 23 milligrams or larger can achieve that goal (you can fine-tune the range of dose amounts further by tweaking it in the **Generate Samples** step of the program). You can use this information in combination with existing drug toxicity information (not discussed in this example) to get a dosing regimen that satisfies both efficacy and safety requirements, for instance.

Postprocess Simulation Data

You can also postprocess the simulation results to look at the correlation between dose amounts and the tumor weight in another way.

1 Go back to the program by clicking the **Program1** tab. Click the (+) icon near the top of the program and select **Calculate Observables** under **Postprocessing**.



A **Postprocessing: Calculate Observables** step appears below the **Simulation** step.

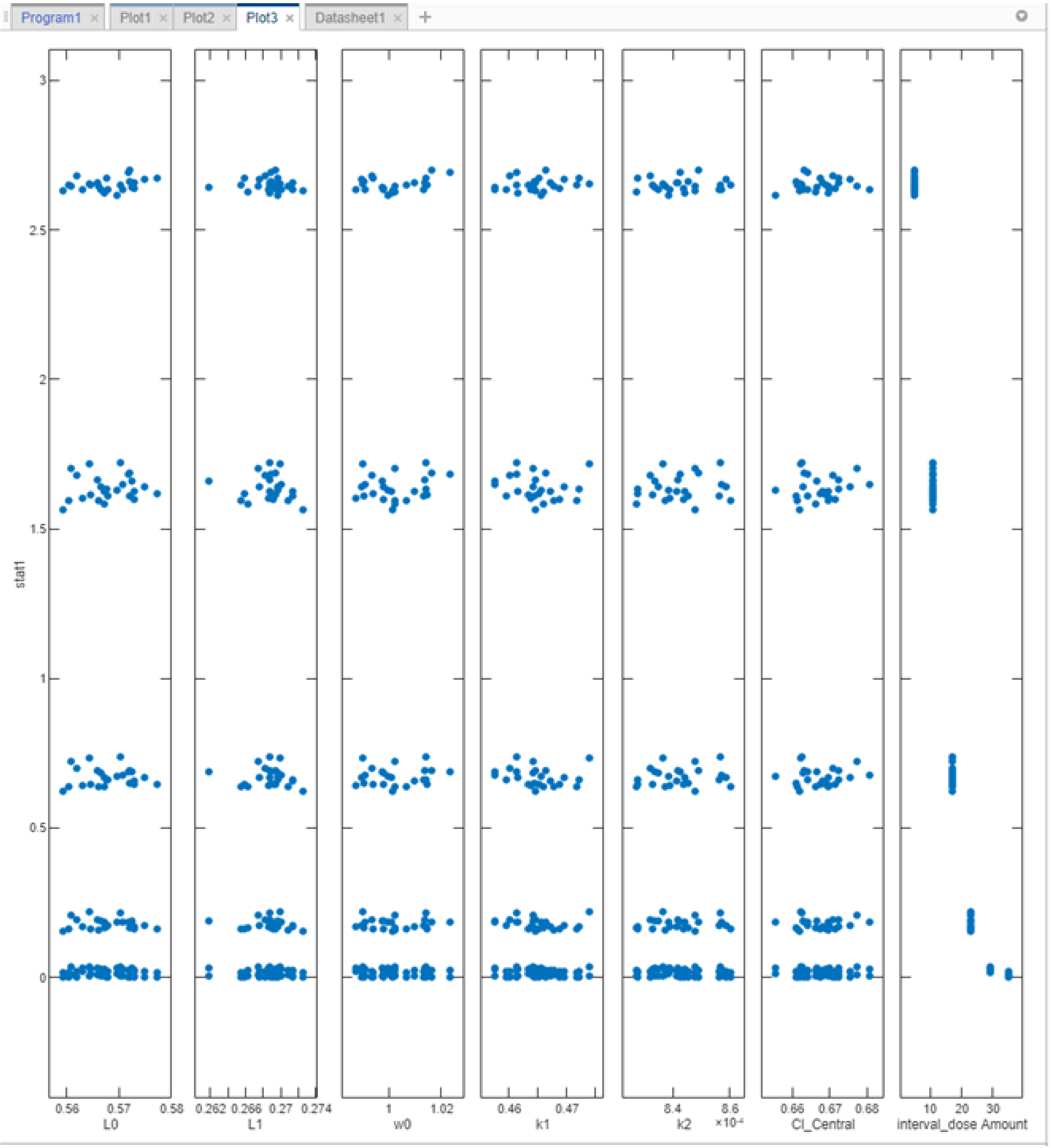
2 Double-click the first cell in the **Name** column. Enter `stat1`.

3 In the **Expression** column, enter the following expression: `min(vertcat(nan,tumor_weight(time>=7)))`. The expression returns the minimum tumor weight after the first dose applied at day 7 from each simulation.

Note Anytime you add an expression to the **Observables** table in the step, the expression is automatically added as an `observable` object to the corresponding model.

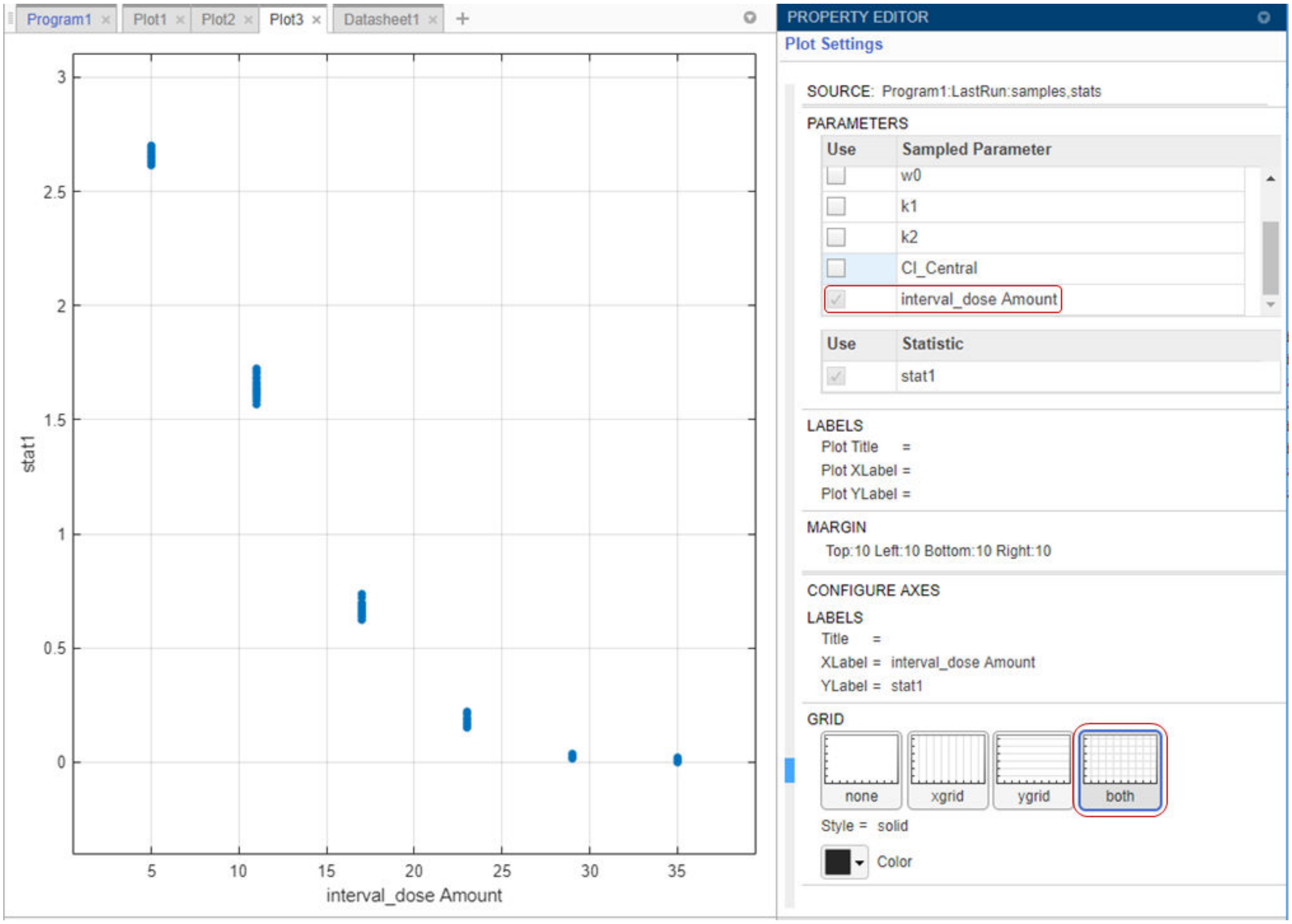
4 In the **Units** column, enter `gram`.

5 Rerun the simulation step by clicking the run button at the top of the step. The app simulates and evaluates the `stat1` expression for each simulation scenario or iteration, and generates the following plot. The *x*-axis represents the parameters and the *y*-axis represents the minimum tumor weight. Each dot represents a simulation scenario. The plot also shows that there is no correlation between the tumor size and the values of various model parameters, except for the dose amounts (the rightmost subplot).



6 Show only the dose amount subplot. In the **Plot Settings**, clear all check boxes in the **Parameters** table, except `interval_dose Amount`.

In the **Grid** section, select **both** to show the grids for both the *x*- and *y*-axis.



SOURCE: Program1:LastRun:samples,stats

PARAMETERS

Use	Sampled Parameter
□	w0
□	k1
□	k2
□	Cl_Central
☑	interval_dose Amount

Use	Statistic
☑	stat1

LABELS
Plot Title =
Plot XLabel =
Plot YLabel =

MARGIN
Top:10 Left:10 Bottom:10 Right:10

CONFIGURE AXES
LABELS
Title =
XLabel = interval_dose Amount
YLabel = stat1

GRID
Style = solid
Color

The plot confirms that higher dose amounts control the tumor growth better. These initial simulation results indicate that the dose amounts of 23 mg or larger could reach the hypothetical tumor weight threshold of 0.5 grams or lower. You can further adjust the range of dose amounts in the **Generate Samples** step.

Tip To interactively explore the plotted data, export the plot to a separate figure window by selecting **Export Plot** from the context (right-click) menu of the plot.

This example shows you how to generate samples to represent virtual patients and perform Monte Carlo simulations to explore the model response on tumor growth under different dose amounts. The simulation results indicate a range for dose amounts and dose schedules that controls the tumor growth for various virtual patients. You could further adjust the dosing regimens so that the doses stay within some known efficacy and toxicity thresholds. For a similar analysis, see the example “Scan Dosing Regimens Using SimBiology Model Analyzer App” on page 1-92.

References

[1] Simeoni, M., P. Magni, C. Cammia, G. De Nicolao, V. Croci, E. Pesenti, M. Germani, I. Poggesi, and M. Rocchetti. 2004. Predictive pharmacokinetic-pharmacodynamic modeling of tumor growth

kinetics in xenograft models after administration of anticancer agents. *Cancer Research*. 64:1094-1101.

See Also

`Observable` | **SimBiology Model Analyzer**

More About

- “Calculate NCA Parameters and Fit Model to PK/PD Data Using SimBiology Model Analyzer App” on page 1-64
- “Find Important Parameters with Sensitivity Analysis Using SimBiology Model Analyzer App” on page 1-30
- “Scan Dosing Regimens Using SimBiology Model Analyzer App” on page 1-92

Calculate NCA Parameters and Fit Model to PK/PD Data Using SimBiology Model Analyzer App

This example shows how to perform noncompartmental analysis to calculate NCA parameters and estimate the tumor growth model [1] parameters from experimental data using nonlinear regression in the **SimBiology Model Analyzer** app.

Tumor Growth Model

The model used in this example is a SimBiology implementation of the pharmacokinetic/pharmacodynamic (PK/PD) model by Simeoni et al. It quantifies the effect of anticancer drugs on tumor growth kinetics from *in vivo* animal studies. The drug pharmacokinetics are described by a two-compartment model with IV bolus dosing and linear elimination (*ke*) from the *Central* compartment. Tumor growth is a biphasic process with an initial exponential growth followed by linear growth. The growth rate of the proliferating tumor cells is described by

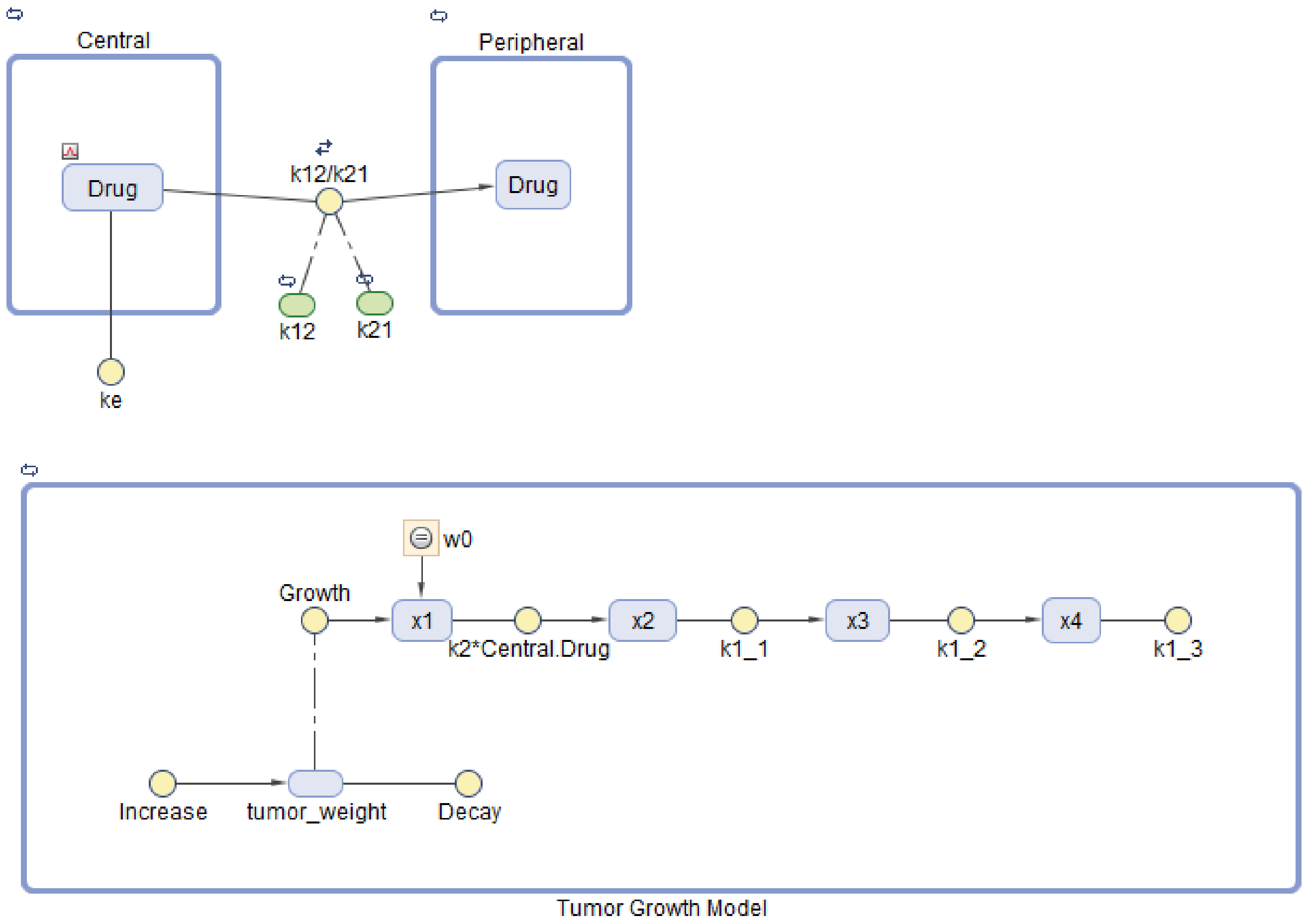
$$\frac{L_0 * x_1}{\left[1 + \left(\frac{L_0}{L_1} * w\right)^{\psi}\right]^{1/\psi}}$$

L_0, L_1, and Ψ are tumor growth parameters, x_1 is the weight of the proliferating tumor cells, and w is the total tumor weight. In the absence of any drugs, the tumor consists of proliferating cells only, that is, `w = x`$_1$. In the presence of an anticancer agent, a fraction of the proliferating cells is transformed into nonproliferating cells. The rate of this transformation is assumed to be a function of the drug concentration in the plasma and an efficacy factor k_2. The nonproliferating cells *x2* go through a series of transit stages (*x3* and *x4*) and are eventually cleared from the system. The flow-through of the transit compartments is modeled as a first-order process with the rate constant k_1.

The SimBiology model makes these adjustments to the pharmacodynamics of tumor growth:

- Instead of defining the tumor weight as the sum of *x1*, *x2*, *x3*, and *x4*, the model defines the tumor weight by the reaction named *Increase*, `null → tumor_weight`, with the reaction rate $\left(2 * L_1 * L_0 * {x_1}^2 / L_1 + 2 * L_0 * x_1\right) * tumor\_weight$.

  tumor_weight is the total tumor weight, x_1 is the weight of the proliferating tumor cells, and L_0, and L_1 are tumor growth parameters.
- Similarly, the model defines the decrease in tumor weight by the reaction named *Decay*, `tumor_weight → null`, with the reaction rate k_1*x_4. The constant k_1 is the forward rate parameter, and x_4 is the last species in the series of transit reductions in tumor weight.
- *ke* is a function of the clearance and the volume of the central compartment: `ke = Cl_Central/Central`.



PK/PD Data Description

The experimental (synthetic) data contains measurements from eight patients for three responses: measured drug concentrations in the central compartment, in the peripheral compartment, and measured tumor weight. The data also contains the dosing information, and each patient receives an IV dose at day 7.

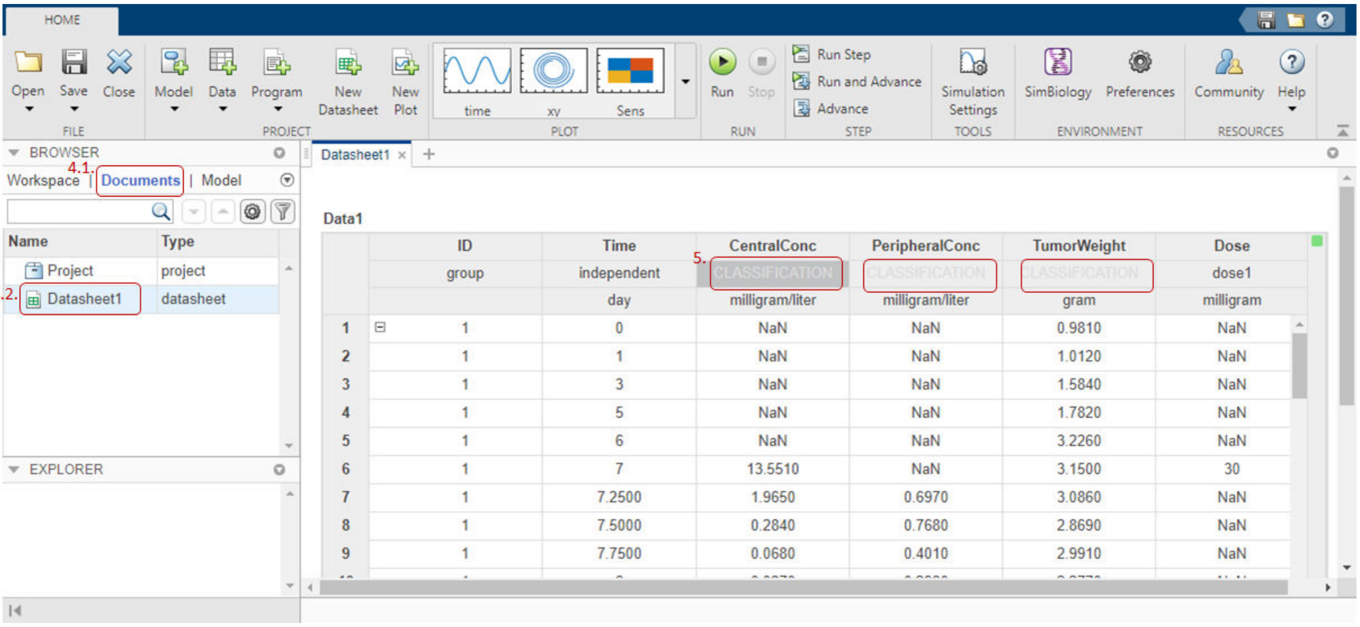
The data set contains the following columns.

- *ID* — Patient IDs
- *Time* — Times when measurements are taken
- *CentralConc* — Drug concentration in the central compartment
- *PeripheralConc* — Drug concentration in the peripheral compartment
- *Dose* — Dosing information for each patient

NaN values are used whenever there is no measurement or no dose is given.

Load Tumor Growth Model and Data

1. Open the **SimBiology Model Analyzer** app by typing `simBiologyModelAnalyzer` at the command line or by clicking the app icon on the **Apps** tab.
2. On the **Home** tab of the app, select **Open**.
3. Navigate to the folder *`matlabroot`*`\examples\simbio\data\`. *matlabroot* is the folder where you have installed MATLAB. Select the project file named `tumor_growth_fitPKPD.sbproj`. In the **Browser** pane, the **Models** folder contains the `Tumor Growth Model` and the **Data1** folder contains the experimental data.
4. Classify the measured responses of the data as the dependent variables. In the **Browser** pane, click **Documents** and double-click **Datasheet1**.
5. In the **Data1** table, double-click **Classification** under *CentralConc*. Select `dependent`. Repeat the same process for *PeripheralConc* and *TumorWeight*. Now all the data columns have proper classifications and the data is ready for use.



Name	Type
Project	project
Datasheet1	datasheet

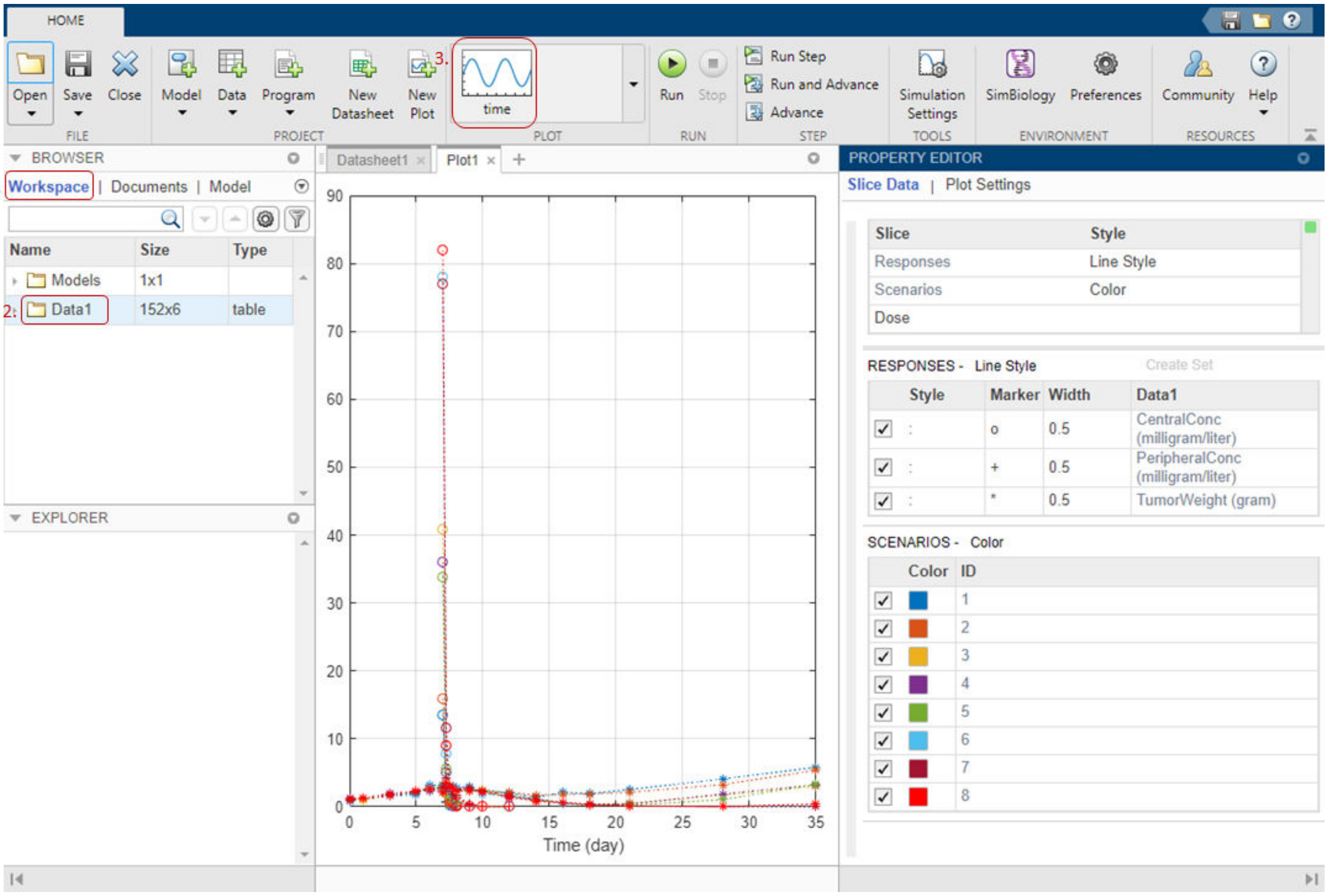
Data1

	ID	Time	CentralConc	PeripheralConc	TumorWeight	Dose
	group	independent	CLASSIFICATION	CLASSIFICATION	CLASSIFICATION	dose1
		day	milligram/liter	milligram/liter	gram	milligram
1	1	0	NaN	NaN	0.9810	NaN
2	1	1	NaN	NaN	1.0120	NaN
3	1	3	NaN	NaN	1.5840	NaN
4	1	5	NaN	NaN	1.7820	NaN
5	1	6	NaN	NaN	3.2260	NaN
6	1	7	13.5510	NaN	3.1500	30
7	1	7.2500	1.9650	0.6970	3.0860	NaN
8	1	7.5000	0.2840	0.7680	2.8690	NaN
9	1	7.7500	0.0680	0.4010	2.9910	NaN

Visualize Experimental Data

After you load the data, you can visualize the measured responses.

1. In the **Browser** pane, click **Workspace**.
2. Click **Data1**.
3. On the **Home** tab, in the **Plot** section, click the **time** plot. The app generates a time plot of all three responses, namely: *CentralConc*, *PeripheralConc*, and *TumorWeight*.



	Name	Size	Type
	Models	1x1	
2.	Data1	152x6	table

Slice	Style
Responses	Line Style
Scenarios	Color
Dose	

RESPONSES - Line Style

	Style	Marker	Width	Data1
☑	:	o	0.5	CentralConc (milligram/liter)
☑	:	+	0.5	PeripheralConc (milligram/liter)
☑	:	*	0.5	TumorWeight (gram)

SCENARIOS - Color

	Color	ID
☑	🟦	1
☑	🟧	2
☑	🟨	3
☑	🟪	4
☑	🟩	5
☑	🟦	6
☑	🟥	7
☑	🟥	8

In the default time plot, **Responses** correspond to the measured responses and are plotted using different line styles. **Scenarios** refers to different groups (eight patients) in the data and are plotted using different colors.

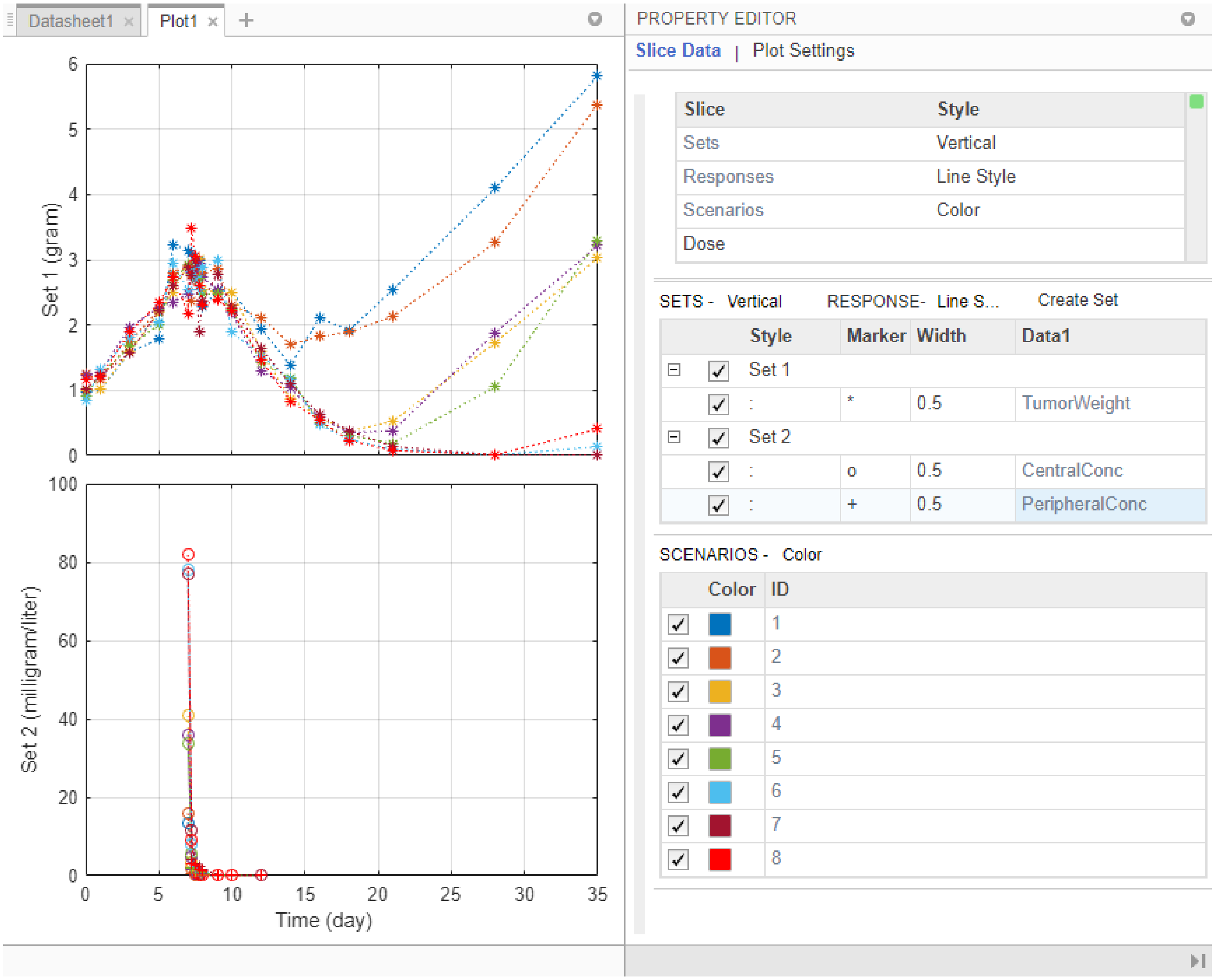
Tip Plots are backed by data that are currently present in the app workspace. Plots are not snapshots. When the data (either experimental data or simulation results) is removed or changed, the plots are also updated according to the changes in the underlying data.

Customize Data Visualization

You can customize the plot to make it clearer. For example, you can plot the PD data (*TumorWeight*) on a different axis than the PK data (*CentralConc* and *PeripheralConc*). To do so, create two diffferent groups (*sets*) of responses, where the first set contains only *TumorWeight* and the second set contains *CentralConc* and *PeripheralConc*.

- Right-click **TumorWeight (gram)** in the **Responses** table (the middle table in the **Property Editor**) and select `Create New Set`. The app creates **Set 1** and **Set 2**. **Set 1** contains only *TumorWeight*, which is now plotted on a different axis than **Set 2**, which contains *CentralConc* and *PeripheralConc*.

Note You can create a set only if there are *two* or more responses.



Slice	Style
Sets	Vertical
Responses	Line Style
Scenarios	Color
Dose	

SETS - Vertical   RESPONSE- Line S...   Create Set

	Style	Marker	Width	Data1
⊟	☑ Set 1			
	☑ :	*	0.5	TumorWeight
⊟	☑ Set 2			
	☑ :	o	0.5	CentralConc
	☑ :	+	0.5	PeripheralConc

SCENARIOS - Color

	Color	ID
☑	🟦	1
☑	🟧	2
☑	🟨	3
☑	🟪	4
☑	🟩	5
☑	🟦	6
☑	🟥	7
☑	🟥	8

The **Slice** table (the top table in the **Property Editor**) now contains **Sets**. This table is a summary table of all slicing variables that are currently present in the plot and their corresponding plot styles. In this current plot, the slicing variables are **Sets**, **Responses**, **Scenarios**, and **Dose**.

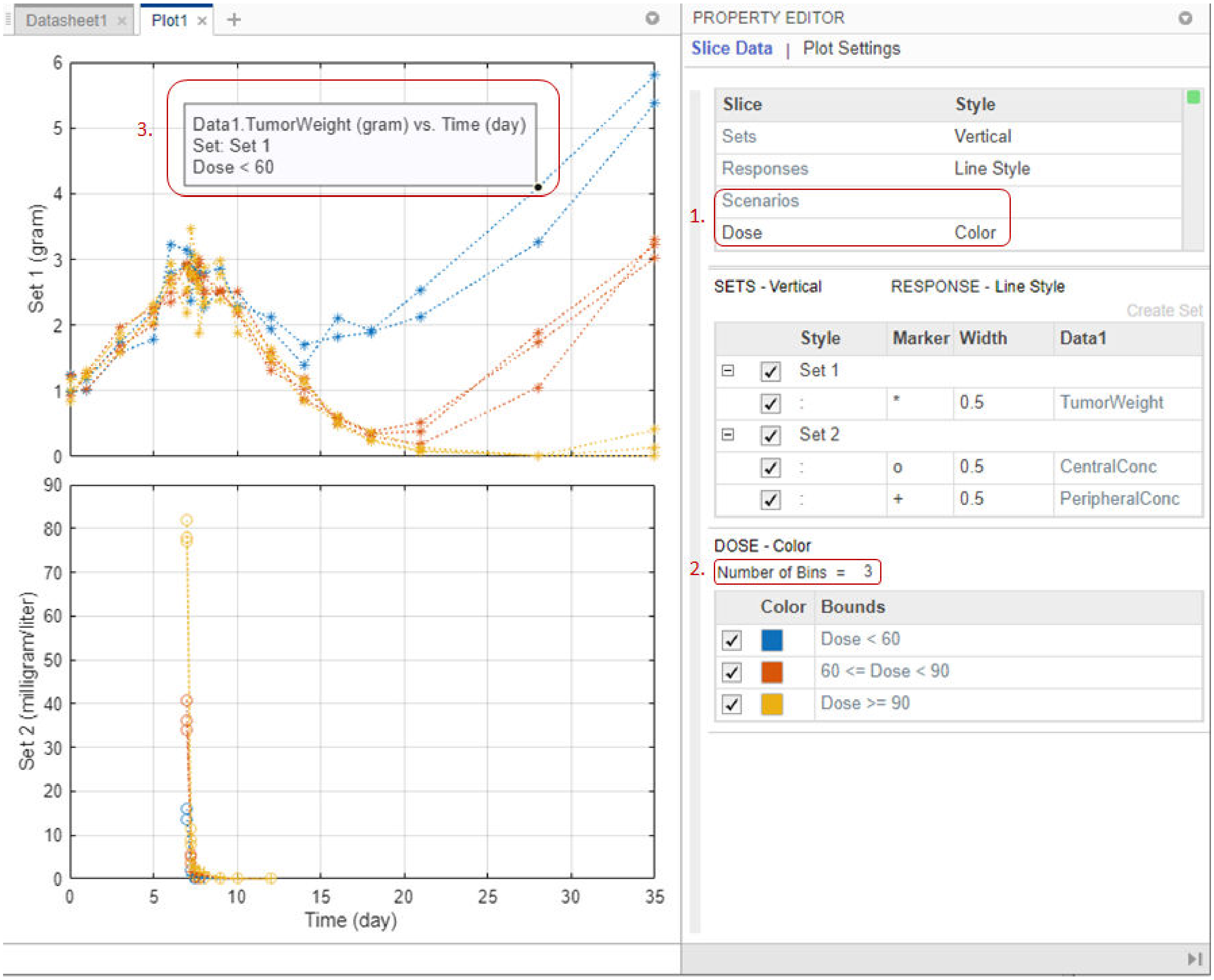
Tip You can slice data using different slicing variables. Each slicing variable appears in the plot with a different visual *style* (or channel) such as color, line style, and axes position. Slicing variables can represent attributes of data, such as responses or scenarios (that is, groups or simulation runs). Slicing variables can also be covariates or parameter values associated with a scenario or group. By default, the app provides slicing variables for different response variables and different scenarios in the plotted data. You can add other visual styles (or channels) for sets of responses and associated parameter or covariate variables.

You can also group the responses based on different dose amounts that the patients receive. There are three different dose groups: 30, 75, and 150 mg.

1 In the **Slice** table, at the **Dose** row, double-click the empty cell and select `Color`. A red indicator appears because another slicing variable (**Scenarios**) has the same plot style. Clear the style (visual channel) for **Scenarios** by selecting empty.

2 In the **Dose** table (the bottom table), the app has automatically binned the dose amounts. Set **Number of Bins** to 3. You can now see that the dose amount has an impact on the tumor size. The higher the dose is, the smaller the tumor becomes.

3 You can also query the corresponding dose group from each line by showing its data tip. Press **Ctrl** and click a blue line to display its data tip. To remove it, **Ctrl + Click** again anywhere on the same line.



Data1.TumorWeight (gram) vs. Time (day)
Set: Set 1
Dose < 60

Slice	Style
Sets	Vertical
Responses	Line Style
Scenarios	
Dose	Color

SETS - Vertical   RESPONSE - Line Style

		Style	Marker	Width	Data1
⊟	☑	Set 1			
	☑	:	*	0.5	TumorWeight
⊟	☑	Set 2			
	☑	:	o	0.5	CentralConc
	☑	:	+	0.5	PeripheralConc

DOSE - Color

Number of Bins = 3

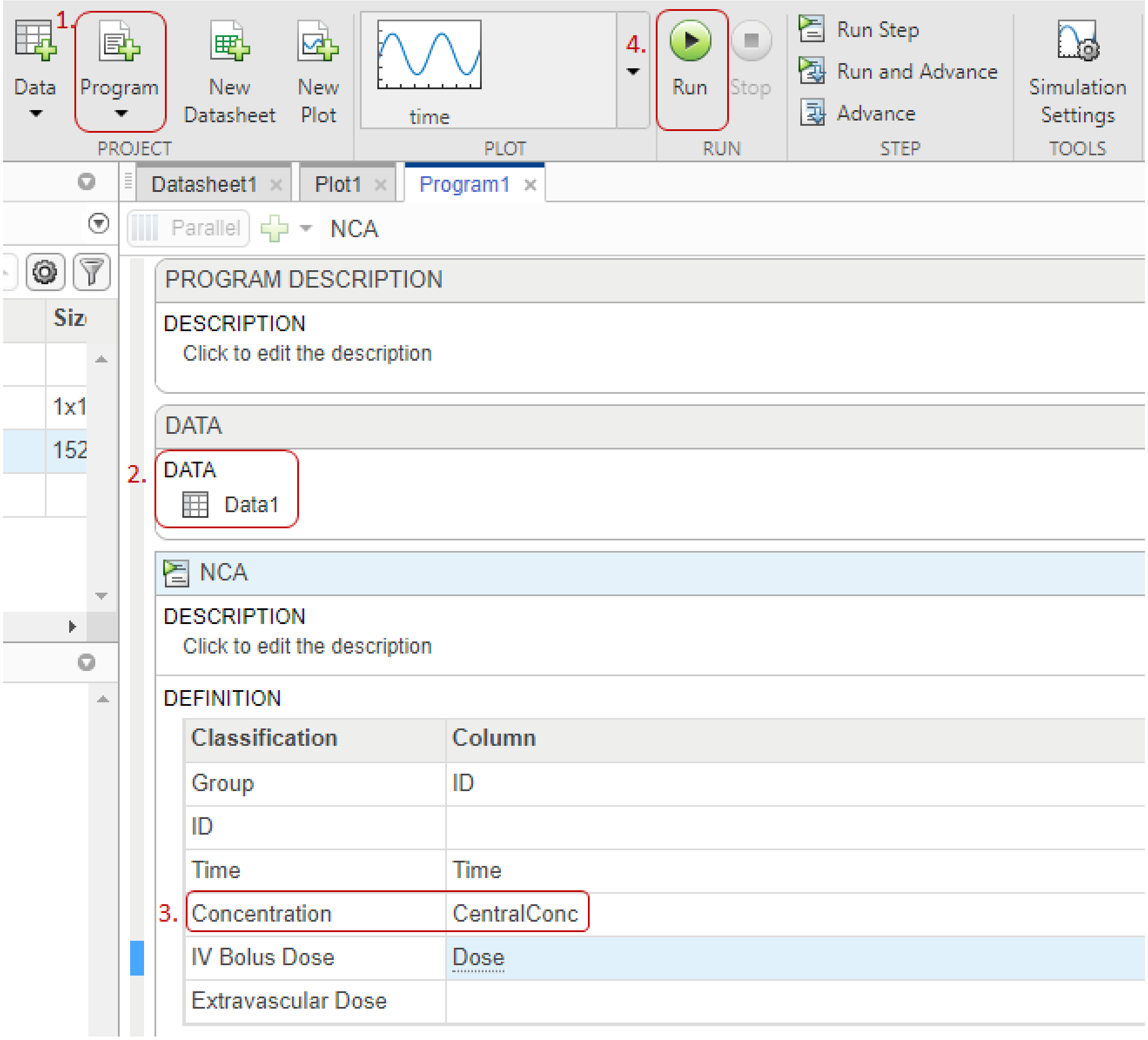
	Color	Bounds
☑	🟦	Dose < 60
☑	🟧	60 <= Dose < 90
☑	🟨	Dose >= 90

Perform Noncompartmental Analysis (NCA)

Using the drug pharmacokinetic data, you can estimate NCA parameters. NCA is model agnostic and can give insights into the drug pharmacokinetics without any underlying assumptions. You can use some of the NCA results as initial estimates when calibrating the model to the data, as discussed later in this example. For details on the list of available NCA parameters and their formulas, see “Noncompartmental Analysis” on page 4-88.

NCA Program Setup

1. On the **Home** tab, select **Program > Non-Compartmental Analysis**. A new program (**Program1**) appears.
2. The **Data** *setup* step of the program defines the data set to use for the NCA analysis. In this example, the program automatically selects **Data1**.
3. The **NCA** *execution* step defines the data column associations and algorithm details. In the **Definition** table, set **Concentration** to `CentralConc`. Leave the other settings unchanged.
4. On the **Home** tab, click **Run**.



PROGRAM DESCRIPTION

DESCRIPTION
Click to edit the description

DATA

DATA
Data1

NCA

DESCRIPTION
Click to edit the description

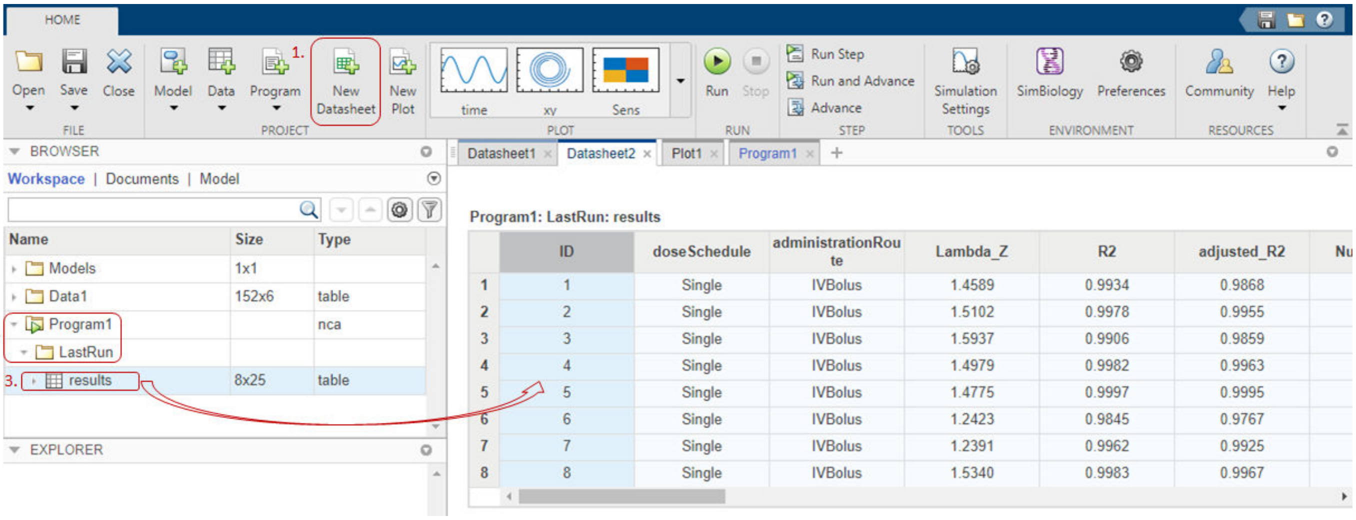
DEFINITION

Classification	Column
Group	ID
ID	
Time	Time
Concentration	CentralConc
IV Bolus Dose	Dose
Extravascular Dose	

Display NCA Results

Once the NCA analysis is complete, the app saves the results in the **LastRun** folder of the program by default.

1. On the **Home** tab, select **New Datasheet**. A new datasheet (**Datasheet2**) opens on a new tab.
2. In the **Browser** pane, expand the **Program1** folder. Then expand **LastRun** folder. The NCA results are stored in the table named **results**.
3. Drag **results** onto **Datasheet2**. The calculated NCA parameter values are now shown in the datasheet. Use the scroll bar at the bottom of the table to scroll and view the NCA data columns. For details about calculated NCA parameters, see “Noncompartmental Analysis” on page 4-88.



Name	Size	Type
Models	1x1	
Data1	152x6	table
Program1		nca
LastRun		
results	8x25	table

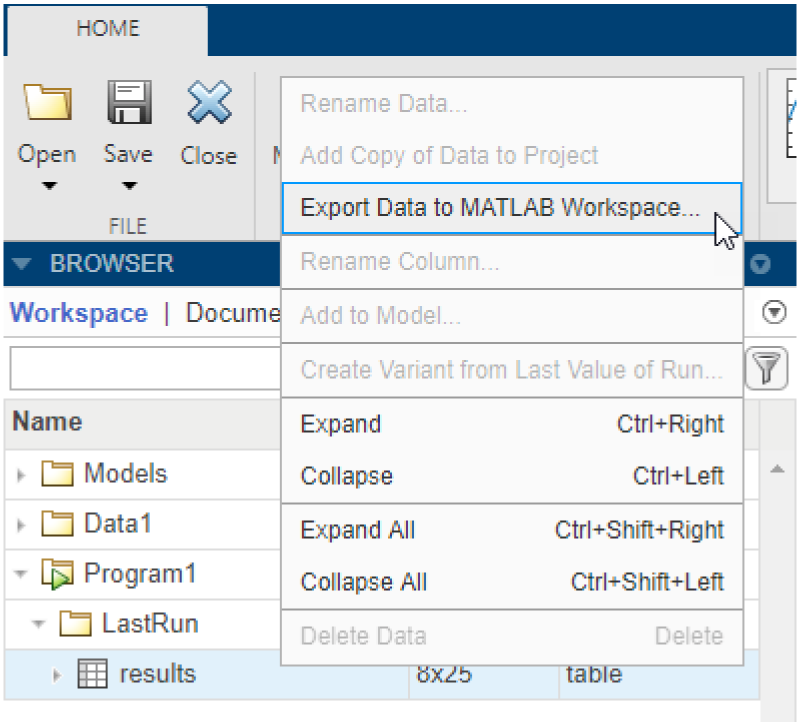
Program1: LastRun: results

	ID	doseSchedule	administrationRoute	Lambda_Z	R2	adjusted_R2	Nu
1	1	Single	IVBolus	1.4589	0.9934	0.9868	
2	2	Single	IVBolus	1.5102	0.9978	0.9955	
3	3	Single	IVBolus	1.5937	0.9906	0.9859	
4	4	Single	IVBolus	1.4979	0.9982	0.9963	
5	5	Single	IVBolus	1.4775	0.9997	0.9995	
6	6	Single	IVBolus	1.2423	0.9845	0.9767	
7	7	Single	IVBolus	1.2391	0.9962	0.9925	
8	8	Single	IVBolus	1.5340	0.9983	0.9967	

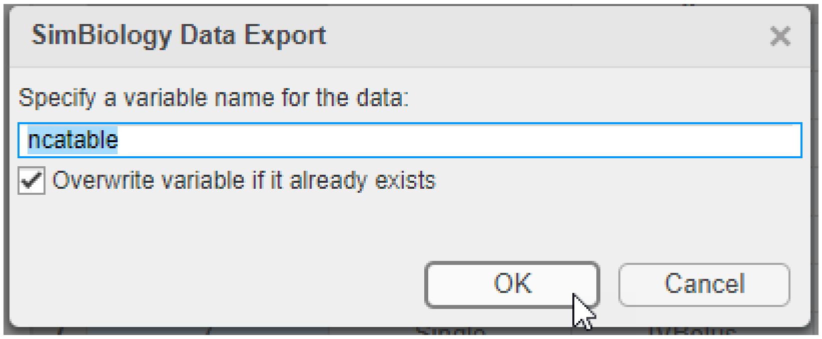
Export Results to MATLAB Workspace

You can export the NCA results to the MATLAB workspace and perform further data analyses at the command line.

1. Right-click **results**. Select **Export Data to MATLAB Workspace**.



2 The **SimBiology Data Export** dialog box opens. Change the name of the variable to **ncatable**. Click **OK**.



Specify a variable name for the data: ncatable

☑ Overwrite variable if it already exists

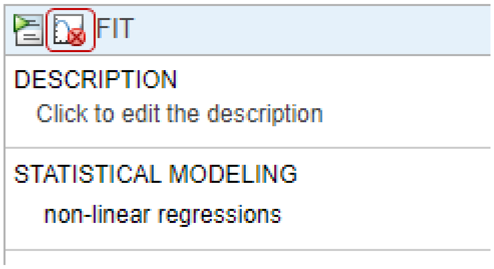
After you export the data to the MATLAB workspace, you can analyze the data at the command-line. For instance, you can calculate the average drug clearance from the NCA data and use it as the model parameter value.

Estimate Model Parameters Using Nonlinear Regression

SimBiology provides different regression techniques to estimate model parameters based on experimental data. This example details the steps for using the nonlinear regression method `lsqnonlin` (requires Optimization Toolbox™) to fit the model to the data. If you do not have Optimization Toolbox, the app uses `fminsearch` instead. For the purposes of the example, only some of the PK/PD model parameters are estimated, namely: *k1*, *L0*, *L1*, *Cl_Central*, *k12*, and *k21*.

Fit Program Setup

1. From the **Home** tab, select **Program > Fit Data**. A new program (**Program2**) appears on a new tab. The **Data** and **Model** steps have been prepopulated with **Data1** and **Tumor Growth Model**, respectively.
2. By default, the **Fit** step autogenerates plots after the fitting is complete. Disable the plot generation by clicking the plot icon at the top of the **Fit** program step for now. The plots will be explored later in the example.



FIT

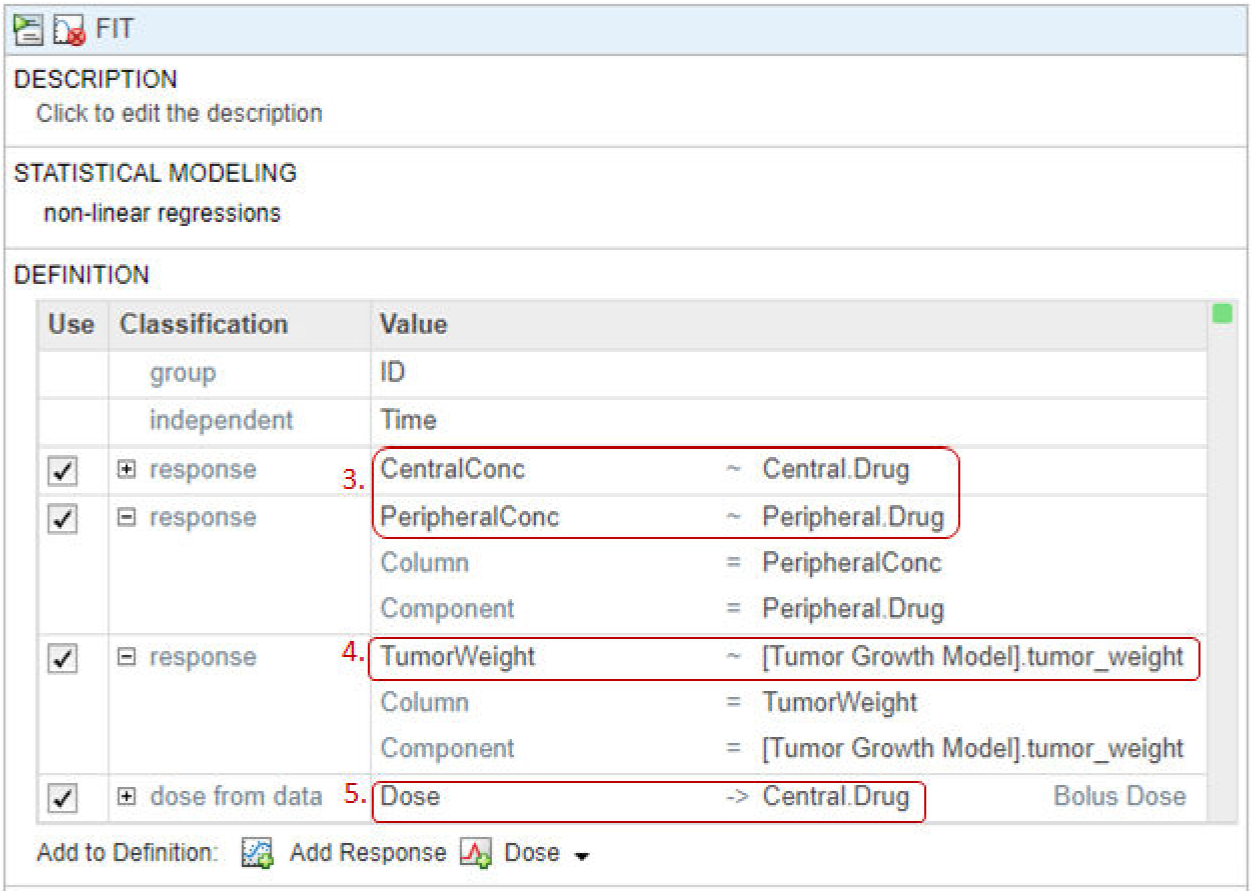
DESCRIPTION

Click to edit the description

STATISTICAL MODELING

non-linear regressions

3. Define the mapping between the model components and the data columns from the experimental data. Specifically, map *CentralConc* to the model species *Drug* in the central compartment (*Central.Drug*). Similarly, map *PeripheralConc* to *Peripheral.Drug*.
4. Map *TumorWeight* to *[Tumor Growth Model].tumor_weight*.
5. Map the *Dose* column to *Central.Drug* to indicate that the *Drug* species in the *Central* compartment is being dosed.



FIT

DESCRIPTION

Click to edit the description

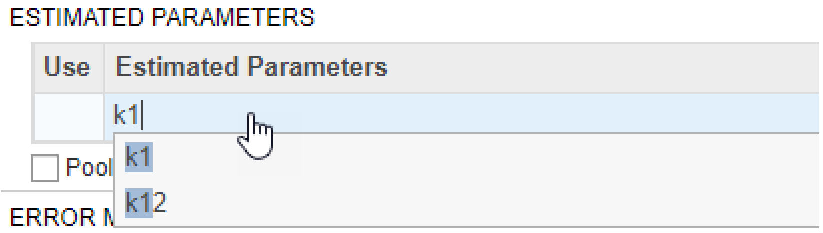
STATISTICAL MODELING

non-linear regressions

DEFINITION

Use	Classification		Value			
	group		ID			
	independent		Time			
☑	⊞ response	3.	CentralConc	~	Central.Drug	
☑	⊟ response		PeripheralConc	~	Peripheral.Drug	
			Column	=	PeripheralConc	
			Component	=	Peripheral.Drug	
☑	⊟ response	4.	TumorWeight	~	[Tumor Growth Model].tumor_weight	
			Column	=	TumorWeight	
			Component	=	[Tumor Growth Model].tumor_weight	
☑	⊞ dose from data	5.	Dose	->	Central.Drug	Bolus Dose

6 Define the model parameters to estimate in the **Estimated Parameters** table. Double-click the empty cell in the **Estimated Parameters** column and type *k1*. The app shows model components with matching names. Select `k1` from the list.



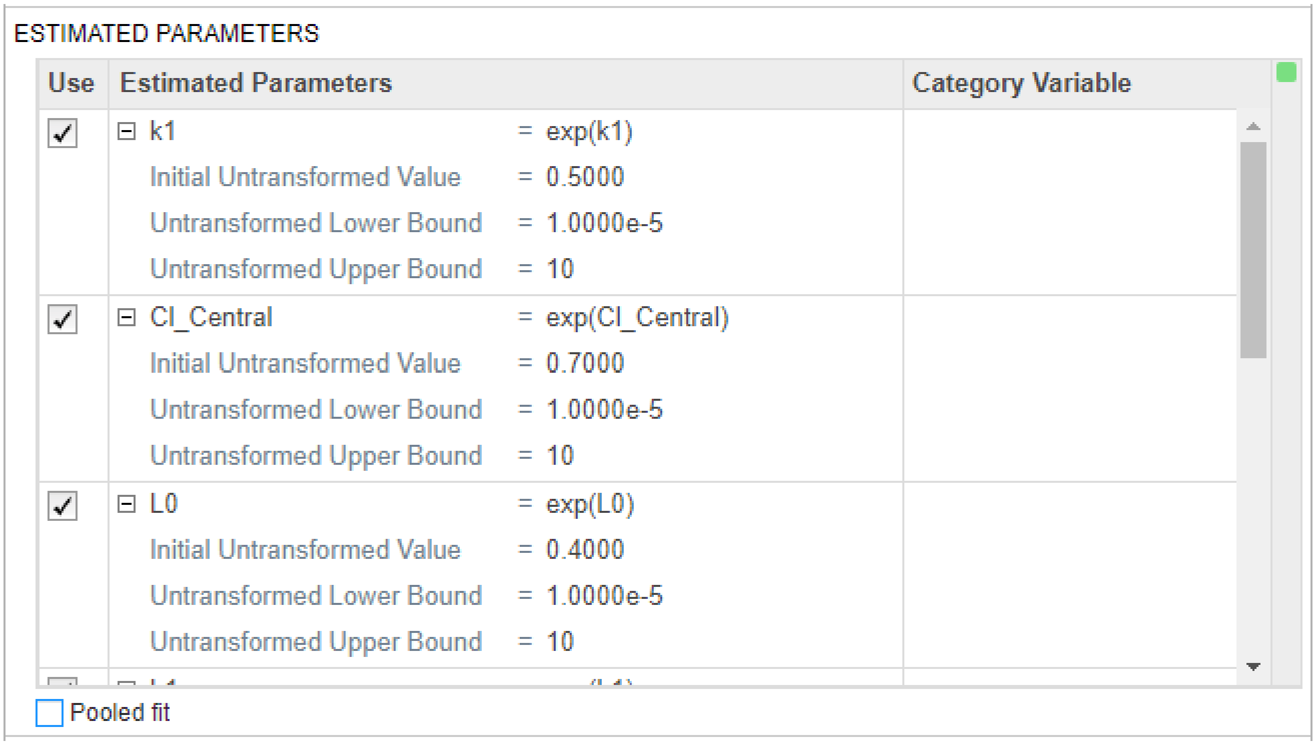
ESTIMATED PARAMETERS

Use	Estimated Parameters
	k1

By default, the parameter is log-transformed as indicated by the expression **exp(k1)**. You can change the expression to no transformation, `probit`, or `logit` transformation. For this example, keep the default log transform because it often improves the convergence. The **Initial Untransformed Value** is automatically set to the model value which is 0.5.

7 Enforce the biological parameters to stay positive by specifying the **Untransformed Lower Bound** and **Untransformed Upper Bound** as `1e-5` and `10`, respectively.

8 Similarly, add the following parameters: *Cl_Central*, *L0*, *L1*, *k12*, and *k21*.

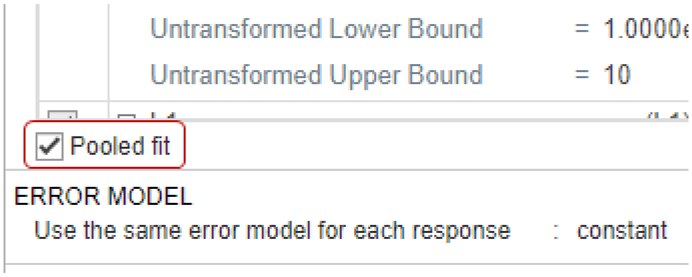


ESTIMATED PARAMETERS

Use	Estimated Parameters		Category Variable
☑	⊟ k1	= exp(k1)	
	Initial Untransformed Value	= 0.5000	
	Untransformed Lower Bound	= 1.0000e-5	
	Untransformed Upper Bound	= 10	
☑	⊟ Cl_Central	= exp(Cl_Central)	
	Initial Untransformed Value	= 0.7000	
	Untransformed Lower Bound	= 1.0000e-5	
	Untransformed Upper Bound	= 10	
☑	⊟ L0	= exp(L0)	
	Initial Untransformed Value	= 0.4000	
	Untransformed Lower Bound	= 1.0000e-5	
	Untransformed Upper Bound	= 10	

□ Pooled fit

9 Select **Pooled fit** to estimate one set of parameters for all patients (population fit). If you do not select **Pooled fit**, the app estimates one set of parameters for each patient (individual fit).



Untransformed Lower Bound = 1.0000e

Untransformed Upper Bound = 10

☑ Pooled fit

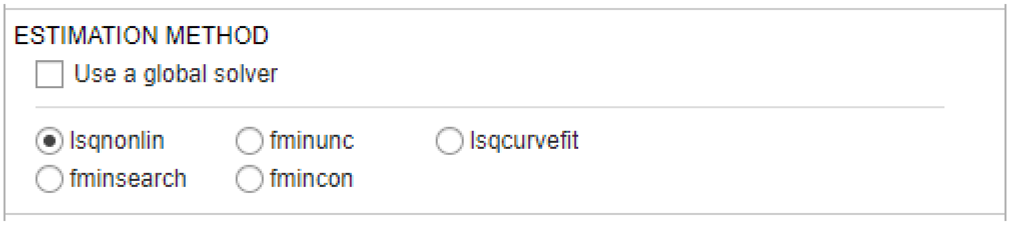
ERROR MODEL

Use the same error model for each response : constant

10 The default error model is the constant error model. SimBiology supports constant, proportional, exponential, and combined error models. For details, see “Error Models” on page 4-44. For now, use the constant error model.

11 Keep the rest of the fitting settings unchanged. These settings are

- Estimation Method — The default method is `lsqnonlin` if you have Optimization Toolbox. If you do not, the app uses `fminsearch`.



ESTIMATION METHOD

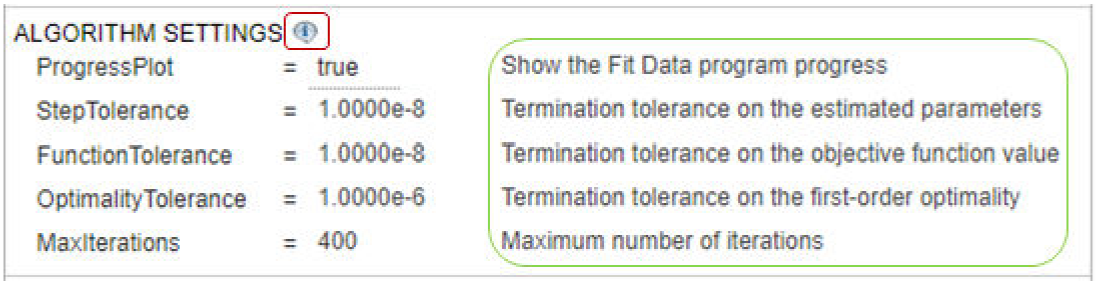
□ Use a global solver

● lsqnonlin ○ fminunc ○ lsqcurvefit

○ fminsearch ○ fmincon

For more information, see “Supported Methods for Parameter Estimation in SimBiology” on page 4-42.

- Algorithm Settings — Most common options for the estimation method. Click to expand the section and see the options. To see the description of each option, click the info icon to the right of the header.



ALGORITHM SETTINGS

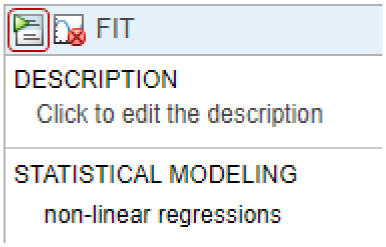
ProgressPlot	=	true	Show the Fit Data program progress
StepTolerance	=	1.0000e-8	Termination tolerance on the estimated parameters
FunctionTolerance	=	1.0000e-8	Termination tolerance on the objective function value
OptimalityTolerance	=	1.0000e-6	Termination tolerance on the first-order optimality
MaxIterations	=	400	Maximum number of iterations

- Advanced Algorithm Settings — Advanced settings for the estimation method. The table is empty by default.

Run Fit Program

After you set the fitting options, you can run the **Fit** step.

1. At the top of the **Fit** step, click the **Run this program step** button.



FIT

DESCRIPTION

Click to edit the description

STATISTICAL MODELING

non-linear regressions

By default, the **Fit** step shows the progress of parameter estimation in a separate figure. The progress plot shows the live status of parameter estimation and fitting quality measures such as log likelihood. For details, see “Progress Plot” on page 4-45.

2. The progress plot shows that the fit converged. You can close the progress plot.

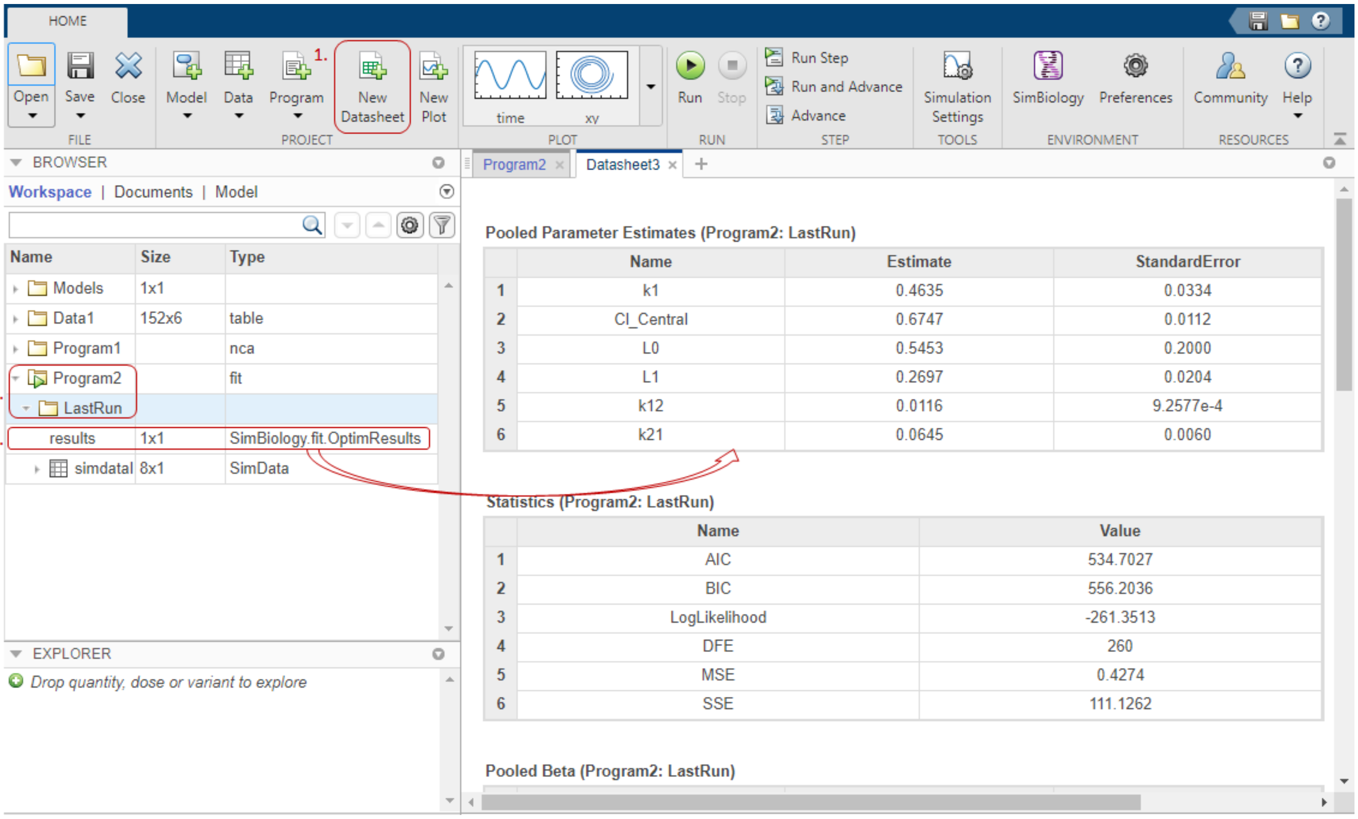
   If you are using `fminsearch`, the fit can fail to converge due to reaching the maximum number of iterations. You can increase **MaxIter** in the **Algorithm Settings**, but for the purposes of this example, you can continue completing the steps without doing so.

Visualize Fit Results

Once the parameter estimation is complete, the fit results are stored in the **LastRun** folder of the program.

1. On the **Home** tab, select **New Datasheet**. A new datasheet opens in a new tab.
2. In the **Browser** pane, expand the **Program2** folder. Then expand **LastRun** folder, which contains **results** and **simdataI**. **results** contains estimated parameter values and fit statistics. **simdataI** contains the simulated model responses for each individual (patient or group) using the estimated parameter values.

3 Drag **results** onto the new datasheet.



Name	Size	Type
Models	1x1	
Data1	152x6	table
Program1		nca
Program2		fit
LastRun		
results	1x1	SimBiology.fit.OptimResults
simdataI	8x1	SimData

Pooled Parameter Estimates (Program2: LastRun)

	Name	Estimate	StandardError
1	k1	0.4635	0.0334
2	Cl_Central	0.6747	0.0112
3	L0	0.5453	0.2000
4	L1	0.2697	0.0204
5	k12	0.0116	9.2577e-4
6	k21	0.0645	0.0060

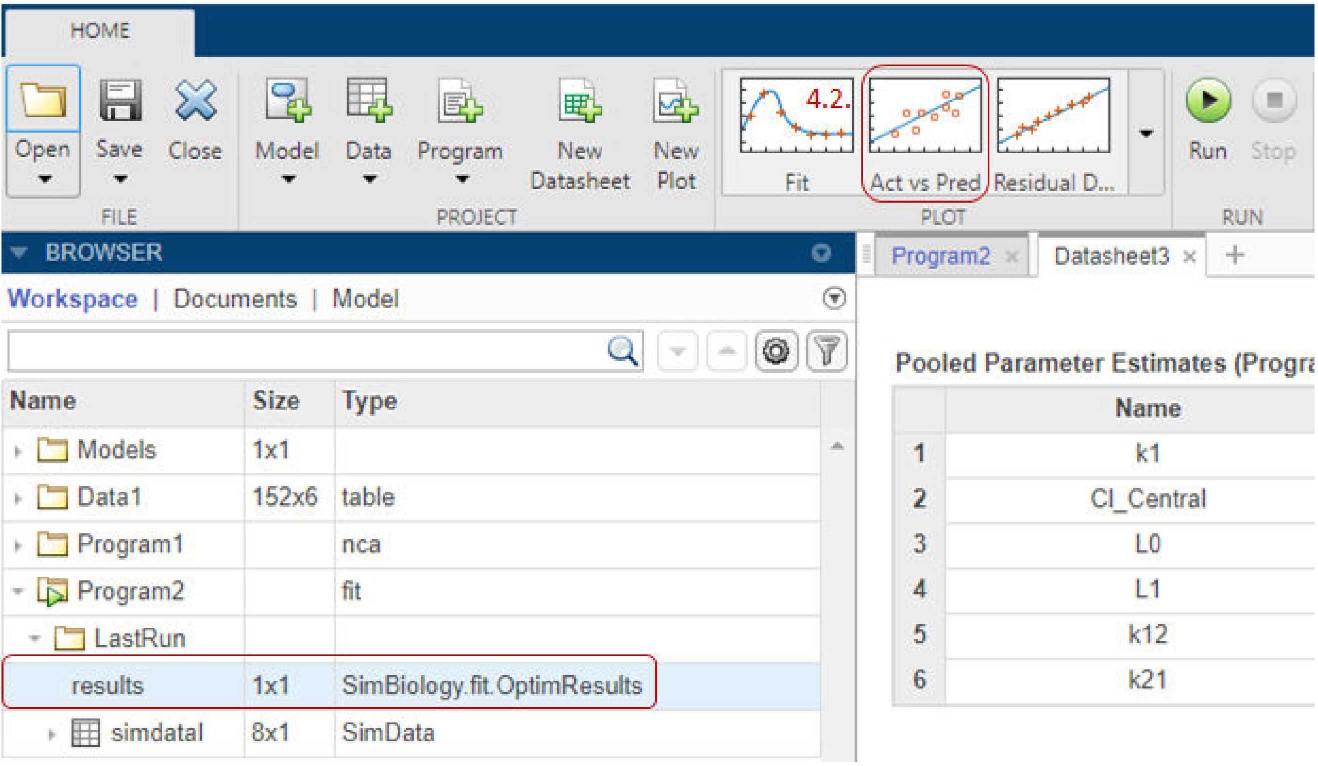
Statistics (Program2: LastRun)

	Name	Value
1	AIC	534.7027
2	BIC	556.2036
3	LogLikelihood	-261.3513
4	DFE	260
5	MSE	0.4274
6	SSE	111.1262

Pooled Beta (Program2: LastRun)

The **Statistics** table contains fit quality measures such as AIC, BIC, and log likelihood. These measures can be useful, for example, to compare the performance of different error models.

4 In addition to quality statistics, you can also view various fit plots, such as actual versus predicted plots and residual distribution plots. First, select **results** in the **Browser** pane. Available fit plots are automatically listed in the **Plot** section on the **Home** tab. Then select **Act vs Pred** from the list.

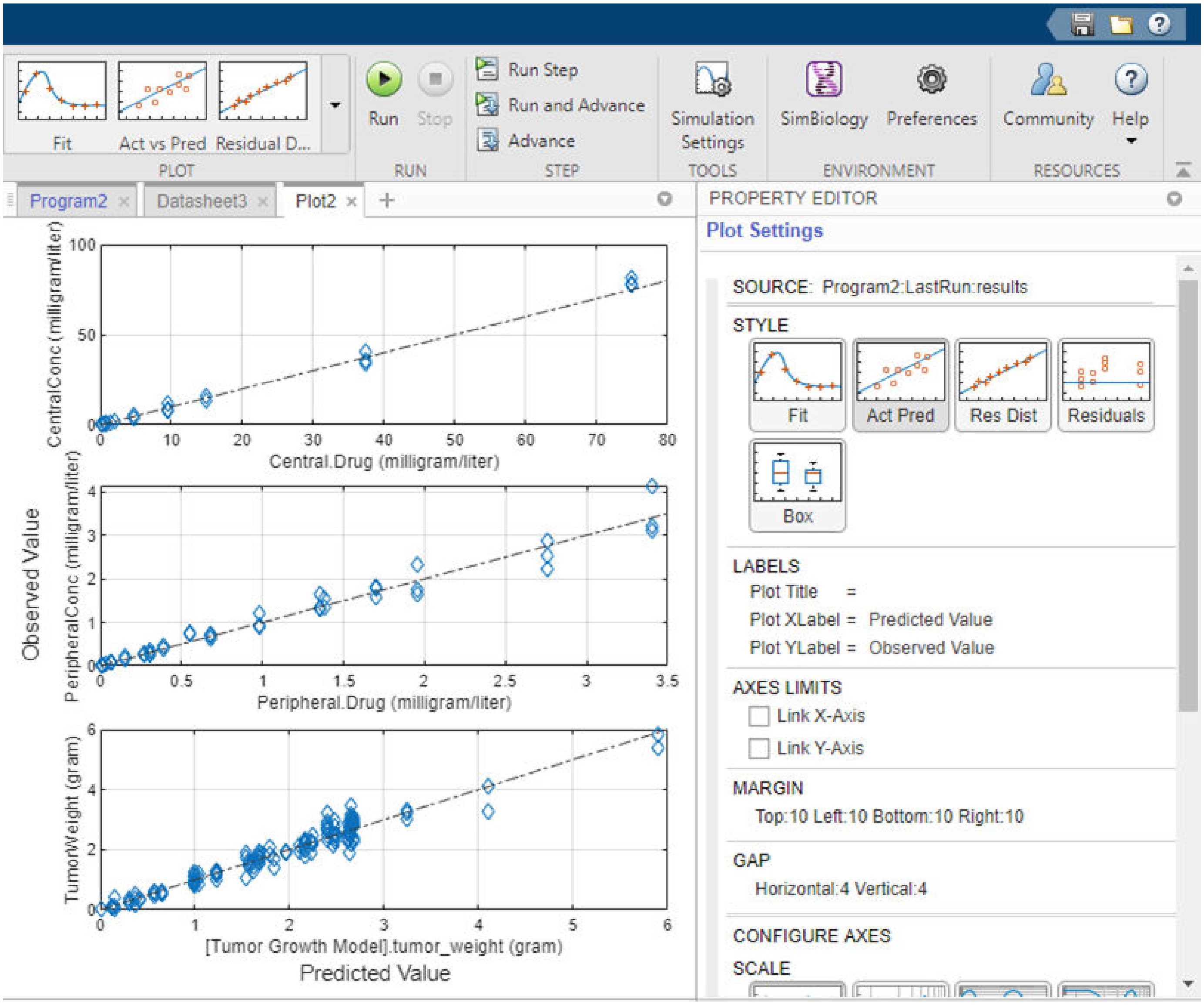


Name	Size	Type
Models	1x1	
Data1	152x6	table
Program1		nca
Program2		fit
LastRun		
results	1x1	SimBiology.fit.OptimResults
simdatal	8x1	SimData

Pooled Parameter Estimates (Progra

	Name
1	k1
2	Cl_Central
3	L0
4	L1
5	k12
6	k21

The actual versus predicted plot appears on a separate tab. The predicted responses are plotted on the *x*-axis and the observed (experimental) responses are plotted on the *y*-axis.



SOURCE: Program2:LastRun:results

Plot Title =

Plot XLabel = Predicted Value

Plot YLabel = Observed Value

□ Link X-Axis

□ Link Y-Axis

MARGIN

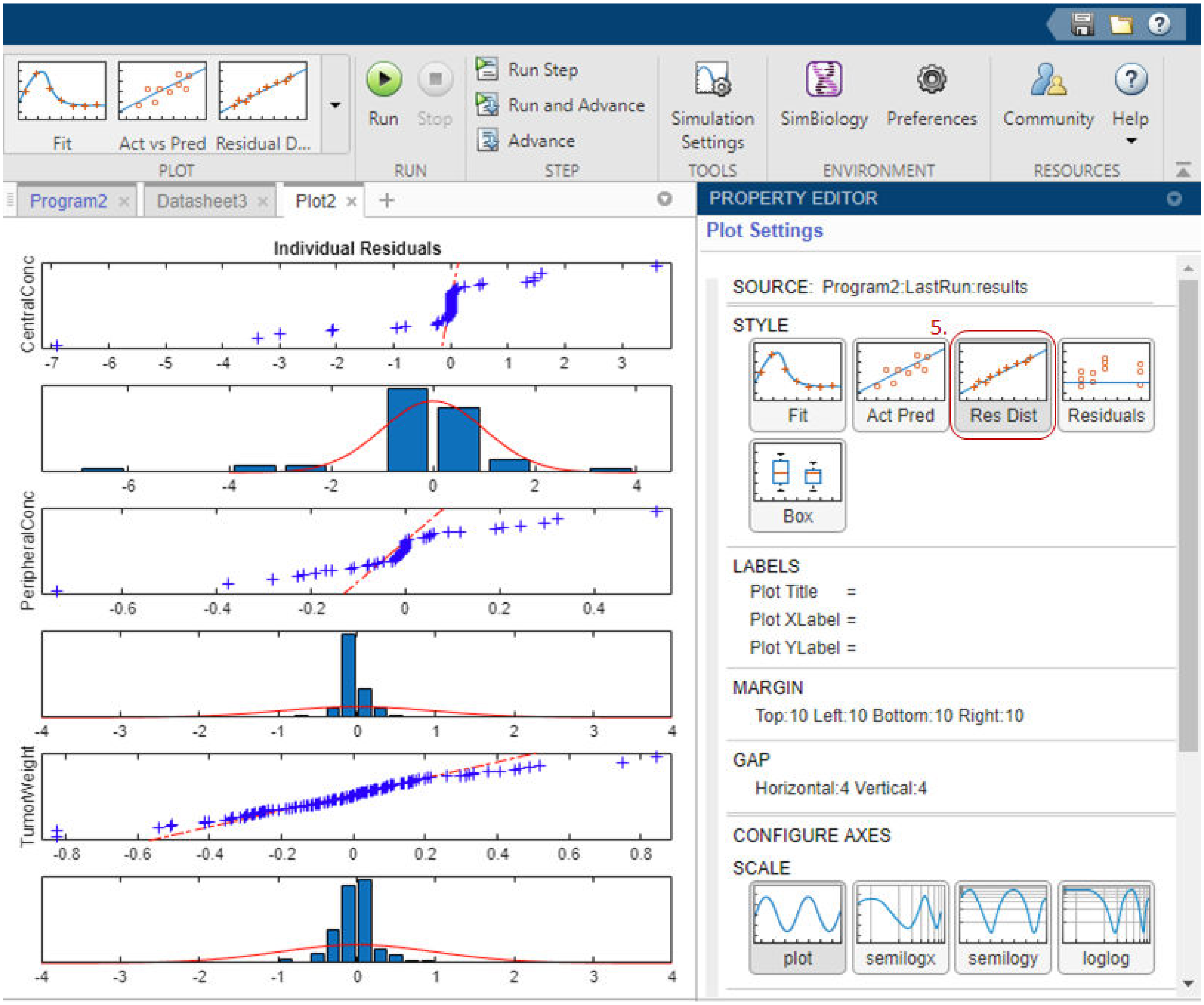
Top:10 Left:10 Bottom:10 Right:10

GAP

Horizontal:4 Vertical:4

You can change the plot to other supported plots by selecting one of the plots from the **Style** section in the **Property Editor**. If you want the new plot on its own separate tab and you do not want to reuse the existing plot tab, select the plot from the **Plot** section on the **Home** tab.

5 Change the plot to a residual distribution plot by selecting **Res Dist** in the **Style** section.



SOURCE: Program2:LastRun:results

Plot Title =

Plot XLabel =

Plot YLabel =

MARGIN

Top:10 Left:10 Bottom:10 Right:10

GAP

Horizontal:4 Vertical:4

The plot shows whether the residuals for each response are normally distributed. In an ideal normal probability plot of residuals, the residuals line up along the diagonal line across the plot and the histograms indicate a normal fit. However, from the plot, the residuals for all three responses, especially *CentralConc* and *PeripheralConc*, do not seem to be normally distributed. It could be an indication that the constant error model assumption is incorrect.

Compare Different Error Models

The following steps show how to change the error model to an exponential error model to fit the data again and compare the fit statistics of two different error models.

Save Fit Results

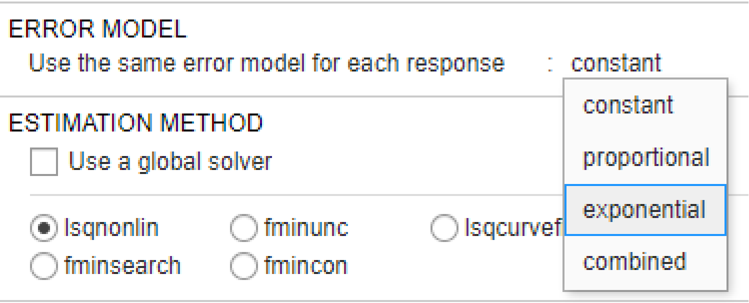
Before fitting the data again using the exponential error model, save the constant error model result in a separate folder. Otherwise, the program, by default, overwrites results from the **LastRun** folder every time you run the fit.

1. Right-click the **LastRun** folder of the fit program in the **Browser** pane.
2. Select `Save Data`.
3. In the **Save Data** dialog, enter `fit_constant` as the data name.

Rerun Fit With Exponential Error Model

After you save the data, you can rerun the fit program with a different error model.

1. Return to the fit program by clicking the **Program2** tab. In the **Error Model** section, select **exponential**.



ERROR MODEL

Use the same error model for each response : constant

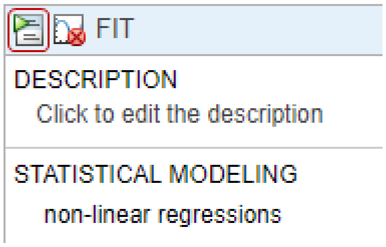
ESTIMATION METHOD

□ Use a global solver

● lsqnonlin ○ fminunc ○ lsqcurvef

○ fminsearch ○ fmincon

2. At the top of the **Fit** step, click the **Run this program step** button.



FIT

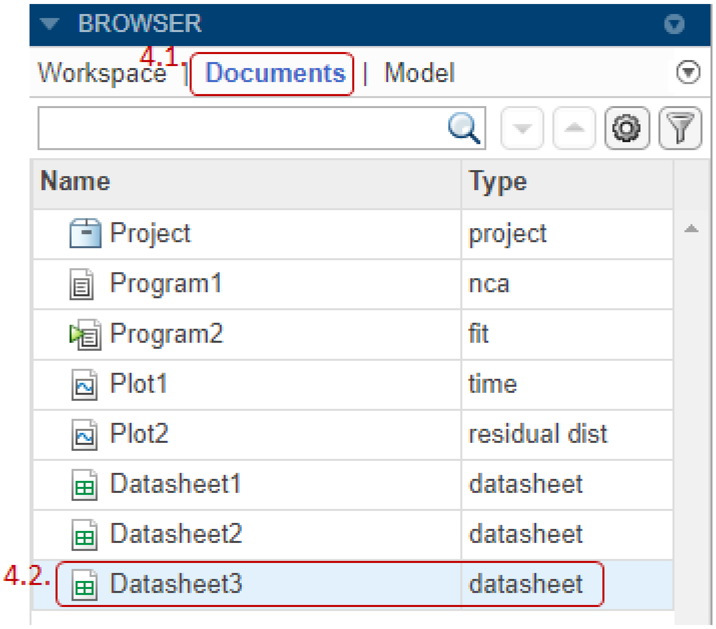
DESCRIPTION

Click to edit the description

STATISTICAL MODELING

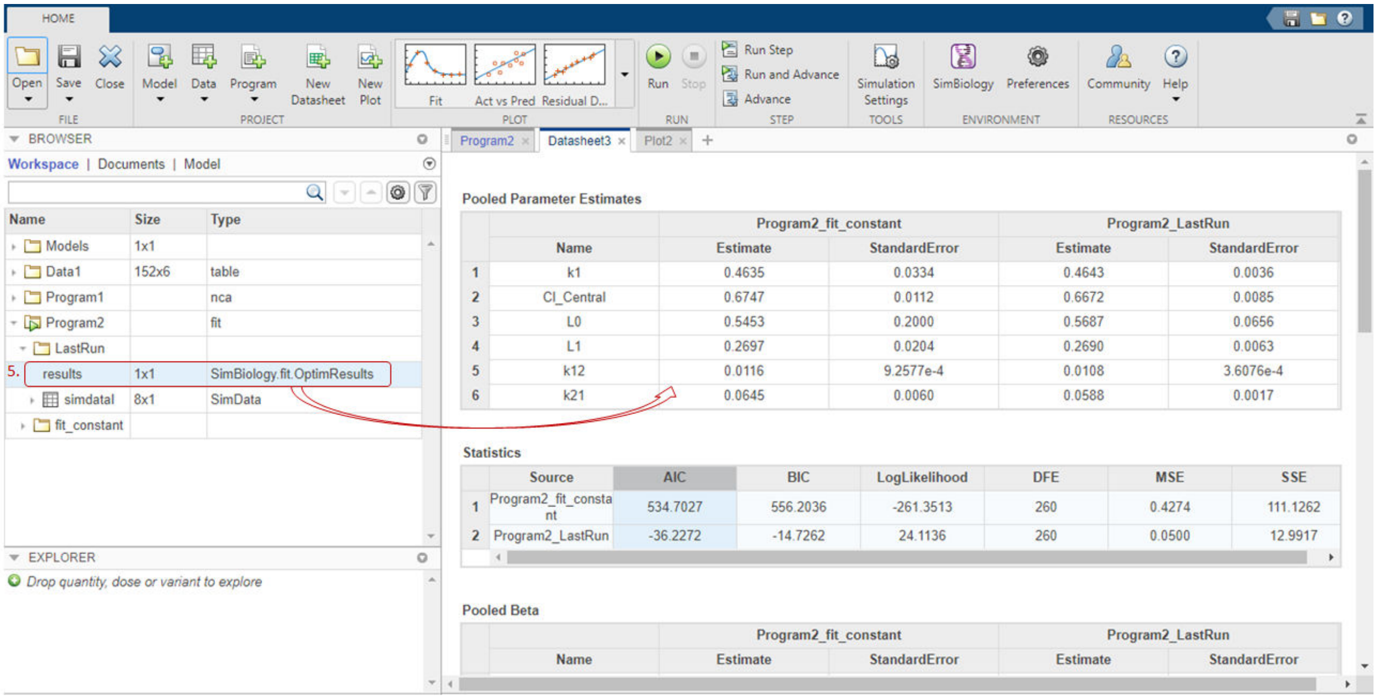
non-linear regressions

3. Close the progress plot after the fit completes.
4. If you closed the datasheet (**Datasheet3**) that contains the fit statistics from the previous fit, reopen the datasheet. In the **Browser** pane, click **Documents**. Then double-click **Datasheet3**.



Name	Type
Project	project
Program1	nca
Program2	fit
Plot1	time
Plot2	residual dist
Datasheet1	datasheet
Datasheet2	datasheet
Datasheet3	datasheet

5 Click **Workspace** to go back to the fit program results. Expand **LastRun** folder of the fit program. Then drag **results** onto **Datasheet3**. New columns (**Program2_LastRun**) containing the latest fit results are added next to the previous fit results (**Program2_fit_constant**).



Name	Size	Type
Models	1x1	
Data1	152x6	table
Program1		nca
Program2		fit
LastRun		
results	1x1	SimBiology.fit.OptimResults
simdataI	8x1	SimData
fit_constant		

Pooled Parameter Estimates

		Program2_fit_constant		Program2_LastRun	
	Name	Estimate	StandardError	Estimate	StandardError
1	k1	0.4635	0.0334	0.4643	0.0036
2	Cl_Central	0.6747	0.0112	0.6672	0.0085
3	L0	0.5453	0.2000	0.5687	0.0656
4	L1	0.2697	0.0204	0.2690	0.0063
5	k12	0.0116	9.2577e-4	0.0108	3.6076e-4
6	k21	0.0645	0.0060	0.0588	0.0017

Statistics

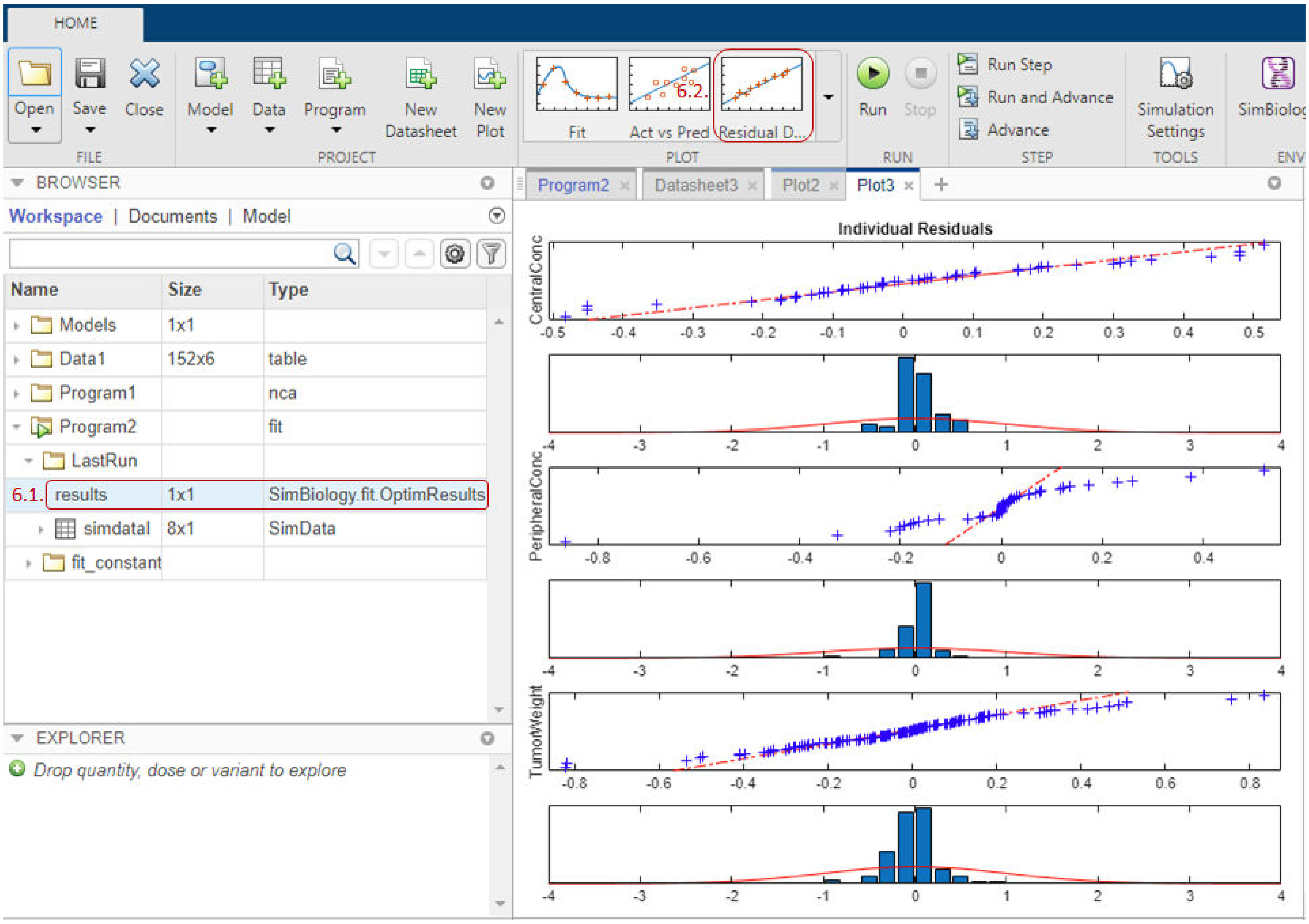
	Source	AIC	BIC	LogLikelihood	DFE	MSE	SSE
1	Program2_fit_constant	534.7027	556.2036	-261.3513	260	0.4274	111.1262
2	Program2_LastRun	-36.2272	-14.7262	24.1136	260	0.0500	12.9917

Pooled Beta

		Program2_fit_constant		Program2_LastRun	
	Name	Estimate	StandardError	Estimate	StandardError

The **Statistics** table compares the fit quality measures. From the comparison, both the AIC and BIC of the fit using the exponential error model are smaller than those of the previous fit. This indicates that the exponential error model fits the data better than the constant error model. The larger log likelihood of the exponential error model also indicates it is a better fit.

6 Next, look at the residual distribution plot. Click **results** from the **LastRun** folder. Then click **Residual Dist** from the **Plot** section on the **Home** tab.



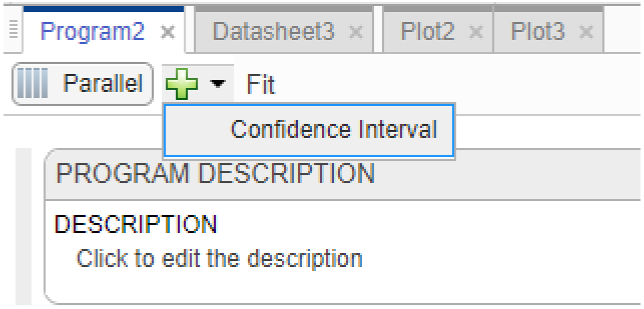
Name	Size	Type
Models	1x1	
Data1	152x6	table
Program1		nca
Program2		fit
LastRun		
results	1x1	SimBiology.fit.OptimResults
simdataI	8x1	SimData
fit_constant		

Compared to the residual distributions of the constant error model, the residual distributions from the exponential error model look more normal, indicating that the exponential error model indeed fits the data better.

Calculate Confidence Intervals

Another way to assess the quality of fit results is to compute 95% confidence intervals for the estimated parameters and model predictions — that is, model simulation results using the estimated parameters. This step requires Statistics and Machine Learning Toolbox.

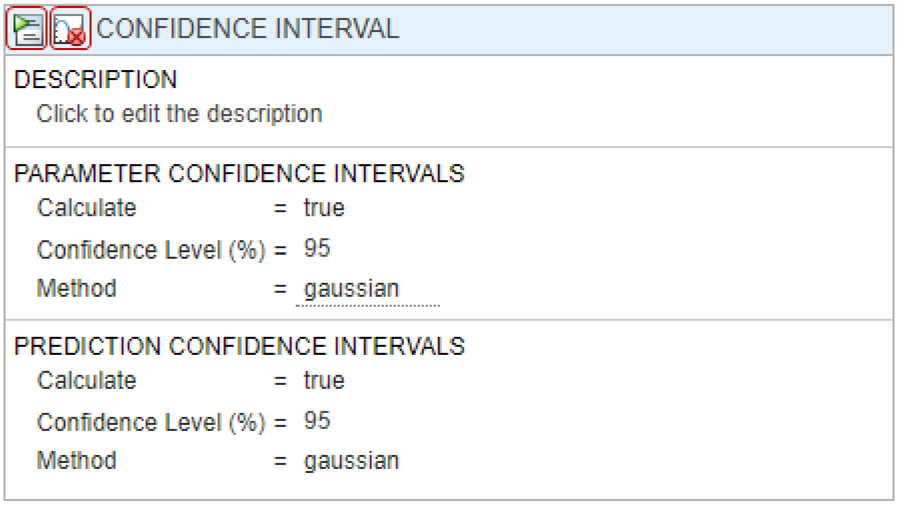
1 Return to the fit program. Click the (+) icon at the upper left and select `Confidence Interval`. A **Confidence Interval** step appears following the **Fit** step.



PROGRAM DESCRIPTION

DESCRIPTION
Click to edit the description

2 At the top of the **Confidence Interval** step, disable the autogeneration of plots by clicking the plot icon. For both **Parameter Confidence Intervals** and **Prediction Confidence Intervals**, use the default method `gaussian` and `95%` confidence level. Click the **Run this program step** button to compute confidence intervals.



CONFIDENCE INTERVAL

DESCRIPTION
Click to edit the description

PARAMETER CONFIDENCE INTERVALS
Calculate = true
Confidence Level (%) = 95
Method = gaussian

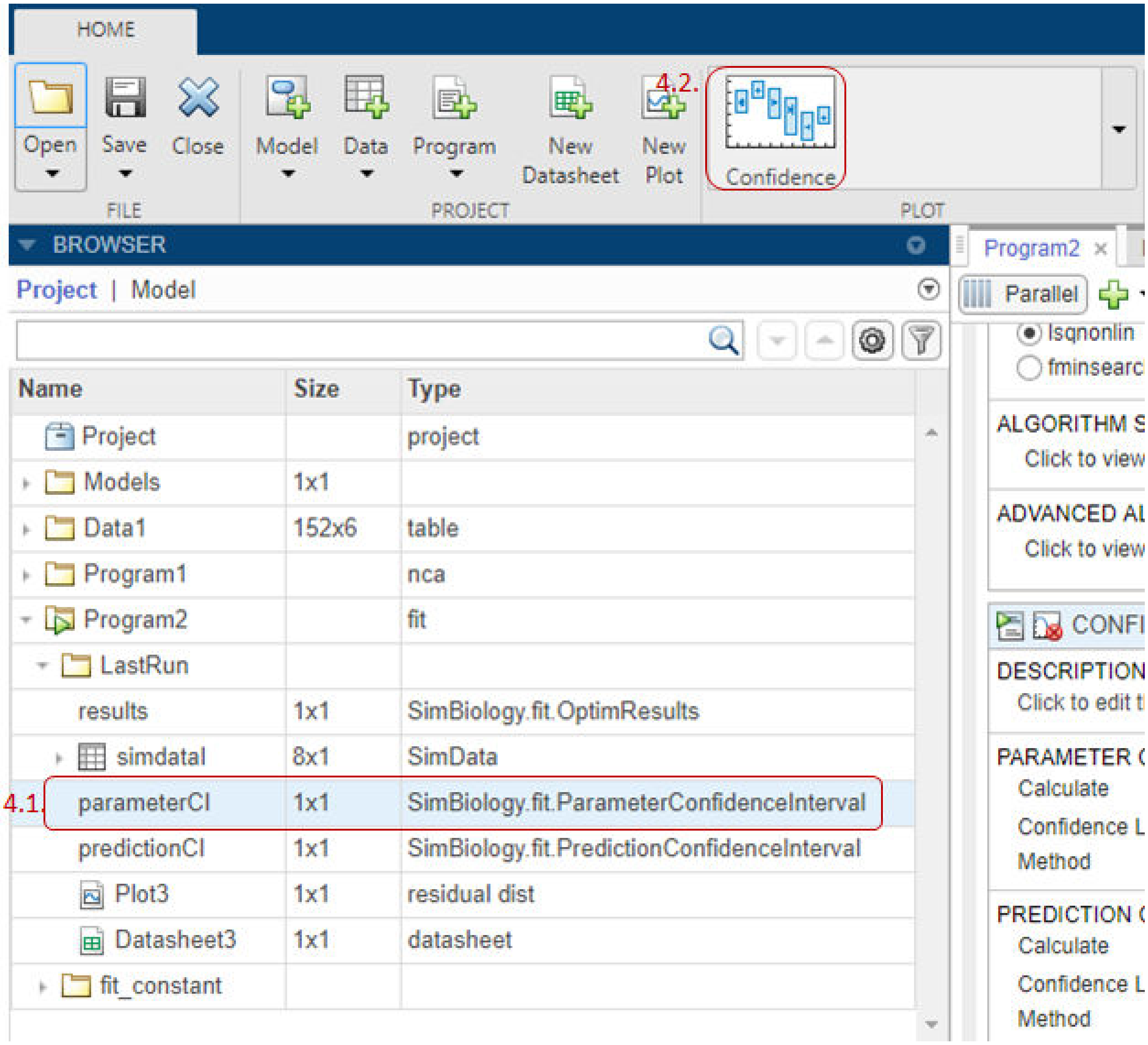
PREDICTION CONFIDENCE INTERVALS
Calculate = true
Confidence Level (%) = 95
Method = gaussian

For parameter confidence intervals, the supported methods are gaussian, profileLikelihood, and bootstrap.

For prediction confidence intervals, the supported methods are gaussian and bootstrap.

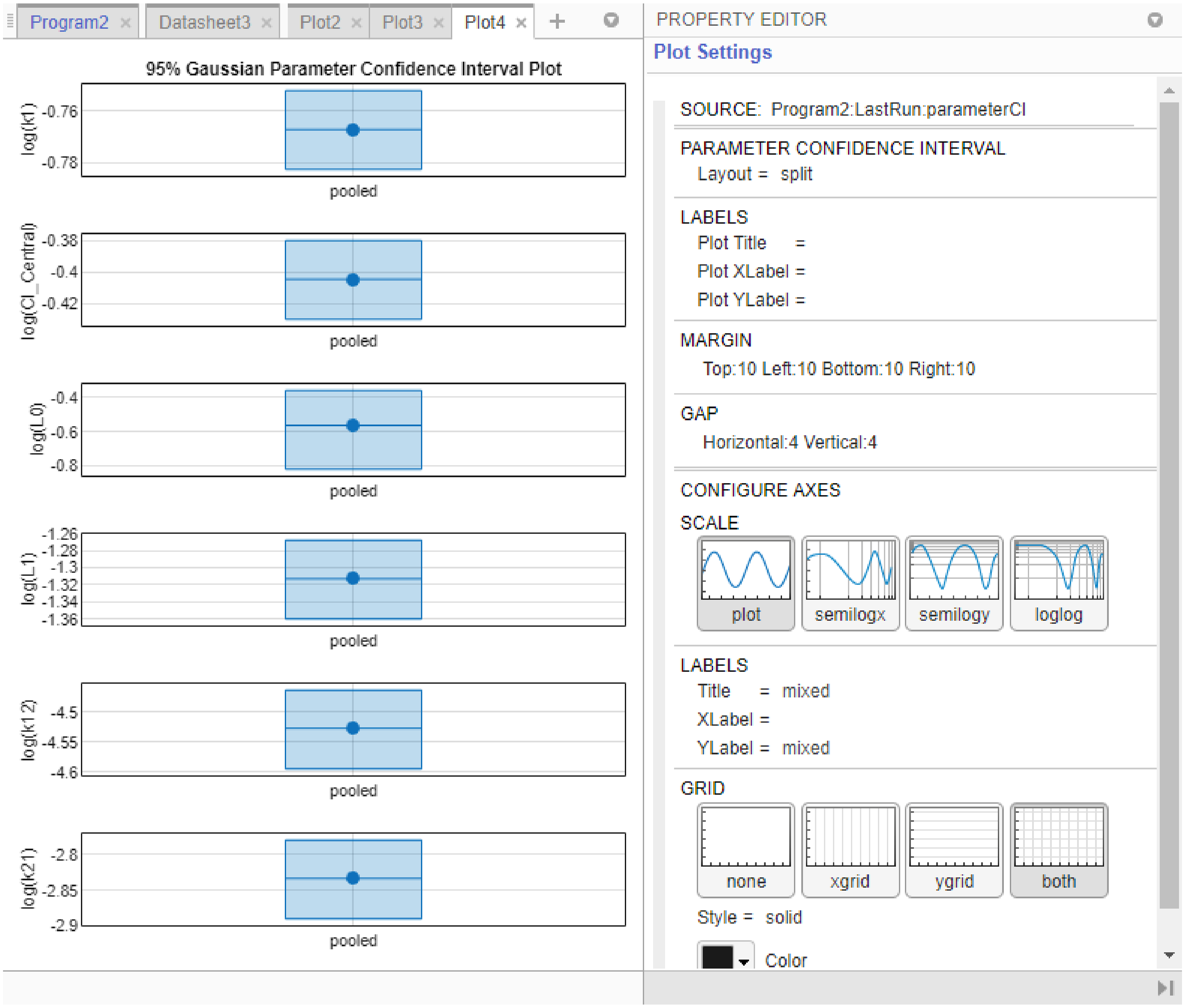
3 Once completed, the results are stored as *parameterCI* and *predictionCI* in the **LastRun** folder of the program. *parameterCI* contains 95% confidence intervals for the estimated parameters. *predictionCI* contains 95% confidence intervals for the model predictions.

4 Plot 95% confidence intervals for the estimated parameters. Click *parameterCI* in the **Browser** pane and select **Confidence** in the **Plot** section.



Name	Size	Type
Project		project
Models	1x1	
Data1	152x6	table
Program1		nca
Program2		fit
LastRun		
results	1x1	SimBiology.fit.OptimResults
simdata1	8x1	SimData
parameterCI	1x1	SimBiology.fit.ParameterConfidenceInterval
predictionCI	1x1	SimBiology.fit.PredictionConfidenceInterval
Plot3	1x1	residual dist
Datasheet3	1x1	datasheet
fit_constant		

The confidence interval for each estimated parameter is shown in a new plot. The plot indicates the successful computation of the confidence intervals for all estimated parameters.



Plot Settings

SOURCE: Program2:LastRun:parameterCI

PARAMETER CONFIDENCE INTERVAL

Layout = split

LABELS

Plot Title =

Plot XLabel =

Plot YLabel =

MARGIN

Top:10 Left:10 Bottom:10 Right:10

GAP

Horizontal:4 Vertical:4

CONFIGURE AXES

SCALE

LABELS

Title = mixed

XLabel =

YLabel = mixed

GRID

Style = solid

Depending on the outcome (status) of the confidence interval estimation, the app plots the results differently.

- If the status of confidence interval estimation is a *success* (as in the above plot), the app uses the first default color (blue) to plot a line and a centered dot for every parameter estimate. The app also plots a box to indicate the confidence intervals.
- If the status is *constrained* or *estimable*, the app uses the second default color (red) and plots a line, centered dot, and box to indicate the confidence intervals.
- If the status is *not estimable*, the app plots only a line and a centered cross in red.
- If there are any transformed parameters with estimated values that are 0 (for the `log` transform) and 0 or 1 (for the `probit` or `logit` transform), no confidence intervals are plotted for those parameter estimates.

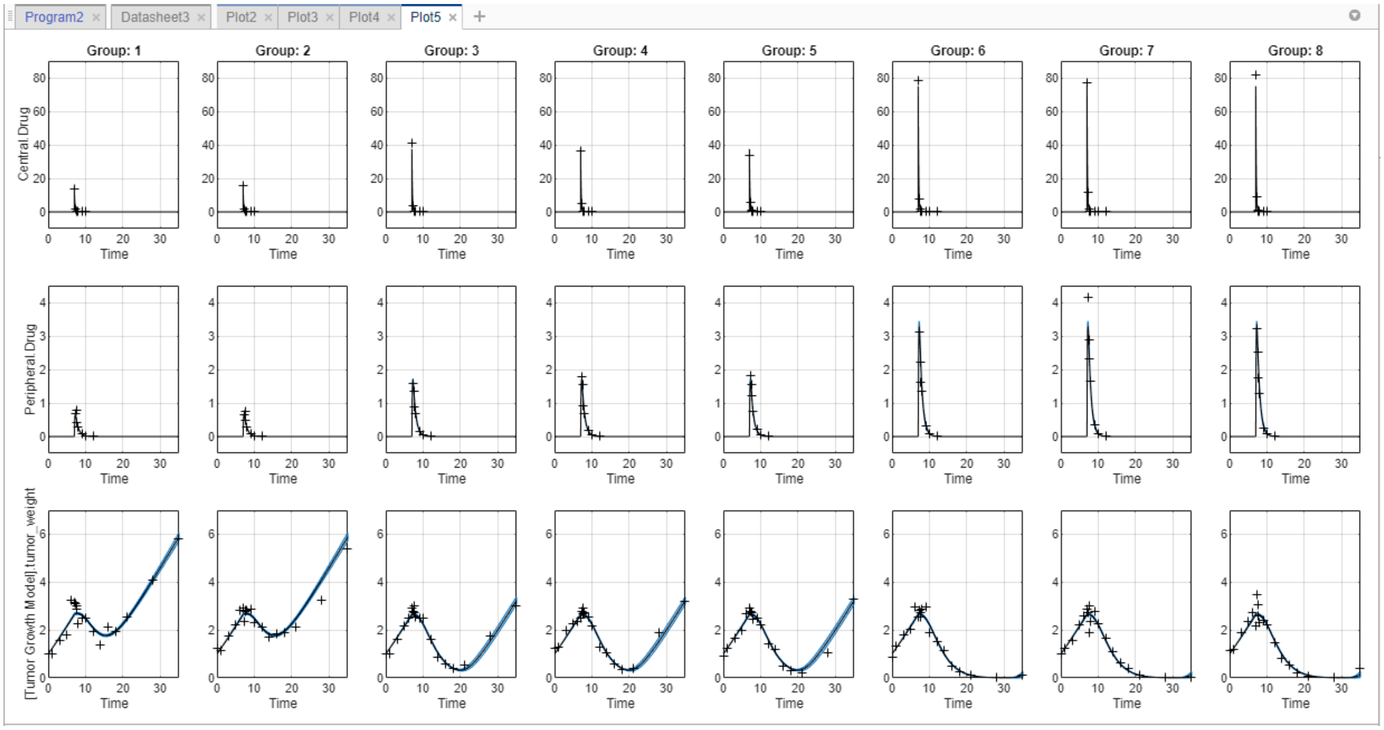
For more details on the definitions of different statuses, see “Parameter Confidence Interval Estimation Status”.

You can also change the **Layout** of the plot in **Plot Settings**.

- The `'split'` layout displays the confidence interval for each parameter estimate on a separate axis.
- The `'grouped'` layout displays all confidence intervals on one axis, grouped by parameter estimates. Each estimated parameter is separated by a vertical black line.

In both cases, the parameter bounds defined in the original fit are marked by square brackets. The app uses vertical dotted lines to group confidence intervals of parameter estimates that have been computed in a common fit.

5 Similarly, plot 95% confidence intervals for the model predictions. Click *predictionCI* in the **Browser** pane and select **Confidence** in the **Plot** section.



In this plot, the confidence interval for each group is plotted in a separate column, and each response is plotted in a separate row. The plot indicates the successful computation of the confidence intervals.

The plotting behavior differs depending on the outcome (status) of the confidence interval calculation.

- If the status is *constrained* or *not estimable*, the app uses the second default color (red) to plot the confidence intervals.
- Otherwise, the app uses the first default color (blue) and plots the confidence intervals as shaded areas (as in the above plot).

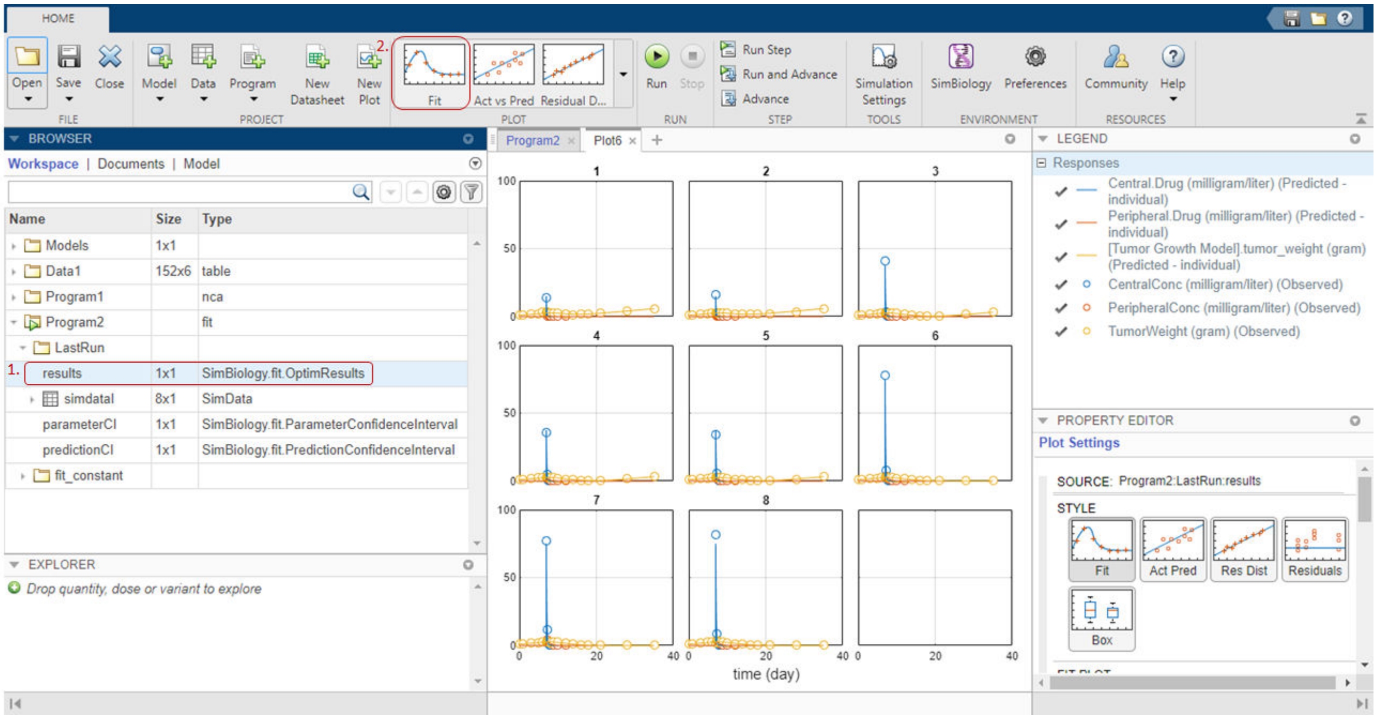
For details, see “Gaussian Confidence Interval Calculation for Model Predictions” and “Bootstrap Confidence Interval Calculation”.

Show Overlay of Fitted Results Over Experimental Data

Once the parameter estimation is complete, you can plot the simulation results on top of the experimental data. There are generally two ways to get the plot.

The first approach is to use the default **Fit** plot that shows the fitted results over experimental data for each group.

1. Click **results** from the **LastRun** folder of the fit program.
2. Select **Fit** in the **Plot** section on the **Home** tab.



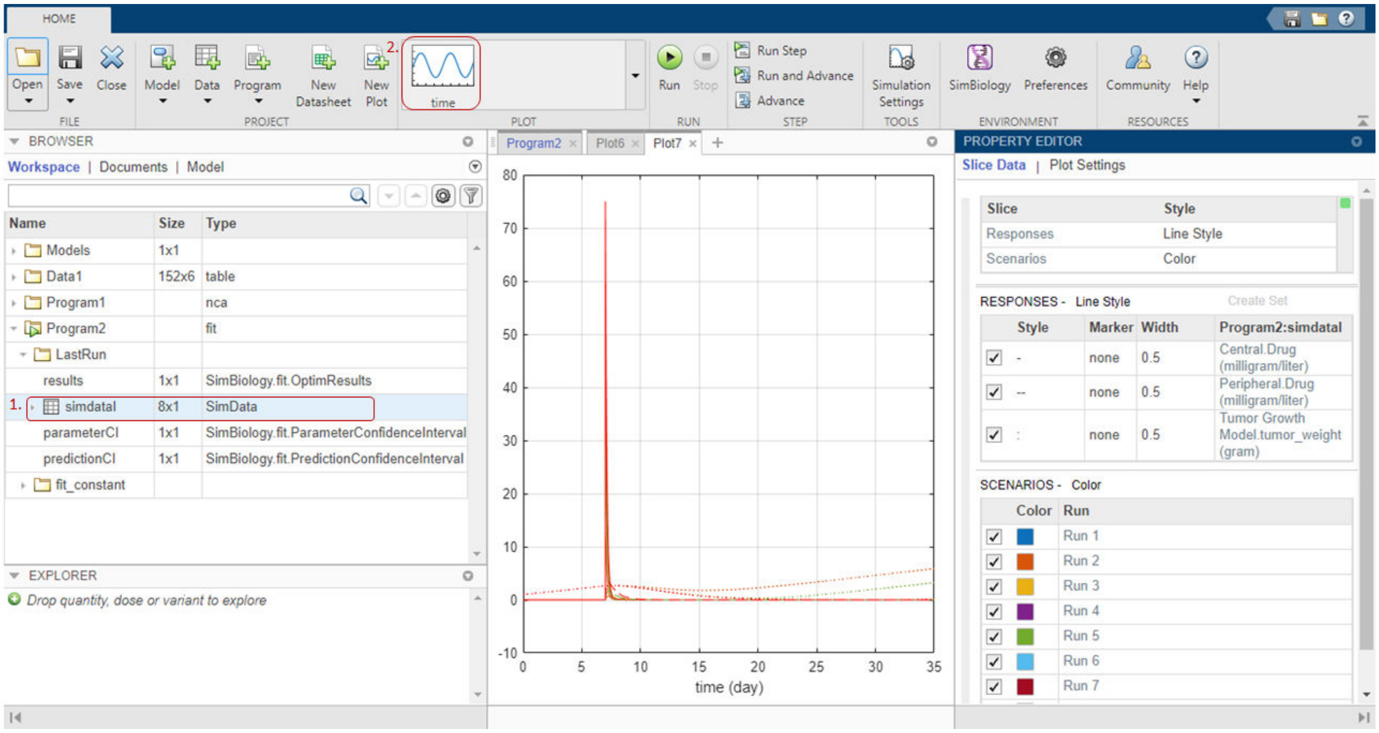
Name	Size	Type
Models	1x1	
Data1	152x6	table
Program1		nca
Program2		fit
LastRun		
1. results	1x1	SimBiology.fit.OptimResults
simdataI	8x1	SimData
parameterCI	1x1	SimBiology.fit.ParameterConfidenceInterval
predictionCI	1x1	SimBiology.fit.PredictionConfidenceInterval
fit_constant		

- ✓ Central.Drug (milligram/liter) (Predicted - individual)
- ✓ Peripheral.Drug (milligram/liter) (Predicted - individual)
- ✓ [Tumor Growth Model].tumor_weight (gram) (Predicted - individual)
- ✓ CentralConc (milligram/liter) (Observed)
- ✓ PeripheralConc (milligram/liter) (Observed)
- ✓ TumorWeight (gram) (Observed)

SOURCE: Program2:LastRun:results

The second approach is more customized and involved, but you have more control over how the plot looks like. The following steps show how to plot the tumor growth profiles of all groups on the same axes but use different line styles for experimental data and simulation data.

1. Click **simdataI** in the **LastRun** folder of the fit program.
2. Click **time** from the **Plot** section on the **Home** tab.



Name	Size	Type
Models	1x1	
Data1	152x6	table
Program1		nca
Program2		fit
LastRun		
results	1x1	SimBiology.fit.OptimResults
simdatal	8x1	SimData
parameterCI	1x1	SimBiology.fit.ParameterConfidenceInterval
predictionCI	1x1	SimBiology.fit.PredictionConfidenceInterval
fit_constant		

Slice	Style
Responses	Line Style
Scenarios	Color

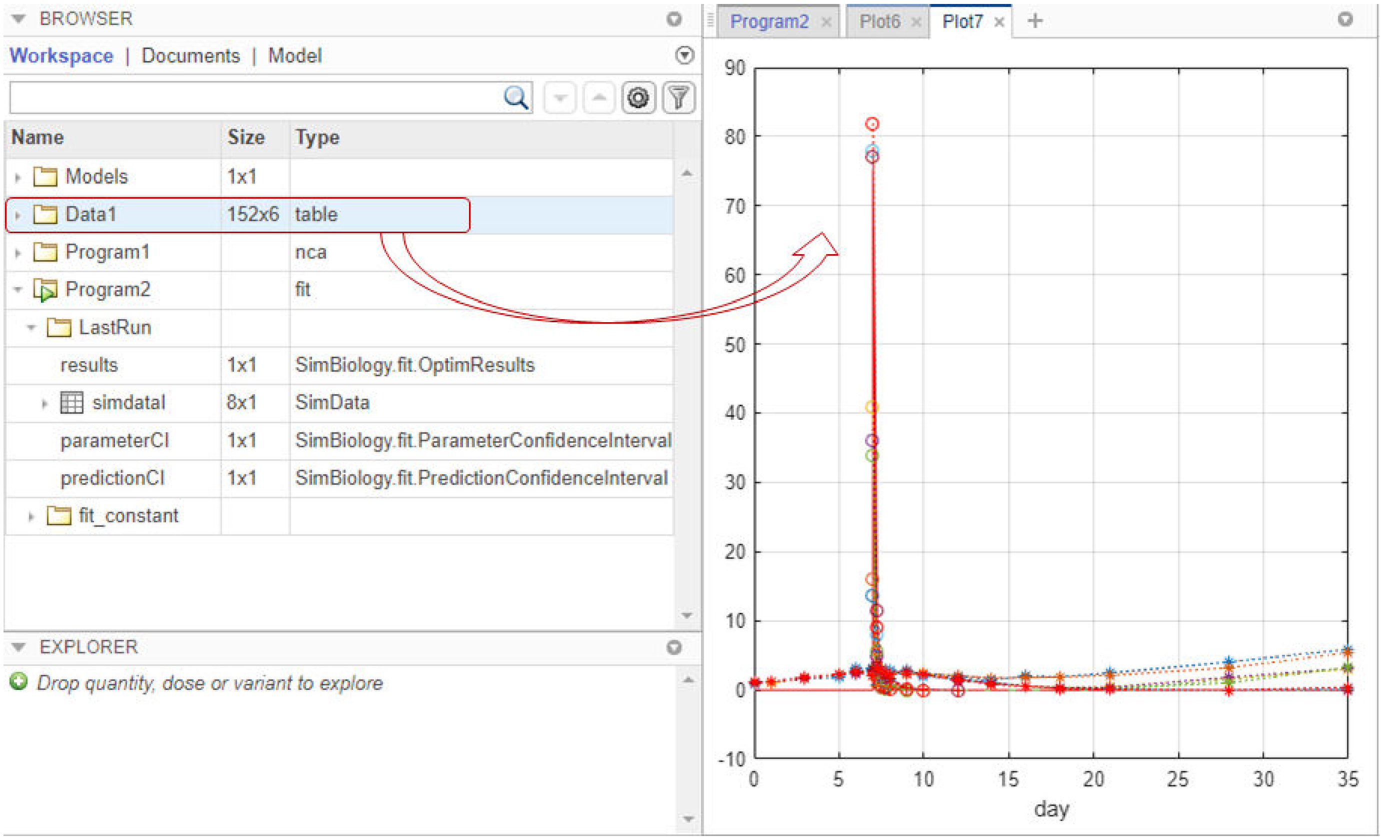
RESPONSES - Line Style

	Style	Marker	Width	Program2:simdatal
☑	-	none	0.5	Central.Drug (milligram/liter)
☑	--	none	0.5	Peripheral.Drug (milligram/liter)
☑	:	none	0.5	Tumor Growth Model.tumor_weight (gram)

SCENARIOS - Color

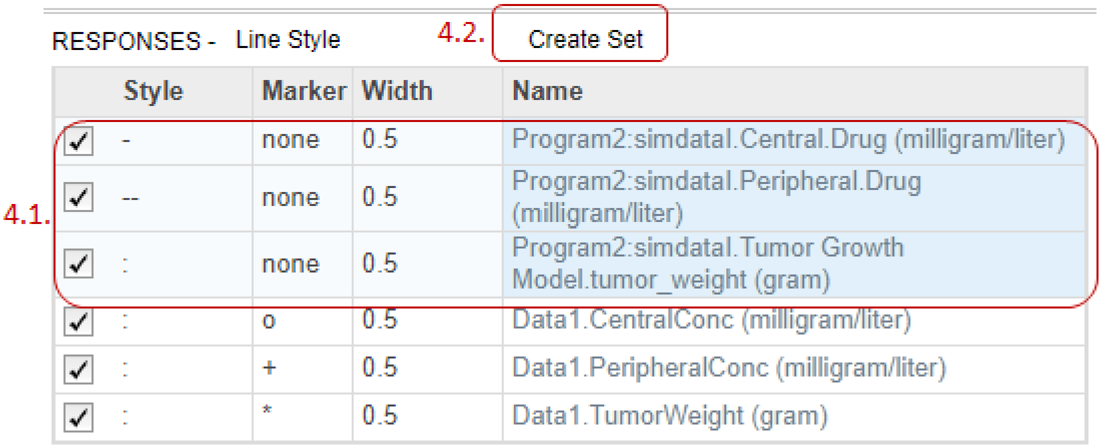
	Color	Run
☑	🟦	Run 1
☑	🟧	Run 2
☑	🟨	Run 3
☑	🟪	Run 4
☑	🟩	Run 5
☑	🟦	Run 6
☑	🟥	Run 7

3 Drag **Data1** (experimental data) on to the plot.



Name	Size	Type
Models	1x1	
Data1	152x6	table
Program1		nca
Program2		fit
LastRun		
results	1x1	SimBiology.fit.OptimResults
simdatal	8x1	SimData
parameterCI	1x1	SimBiology.fit.ParameterConfidenceInterval
predictionCI	1x1	SimBiology.fit.PredictionConfidenceInterval
fit_constant		

4 In the **Property Editor** of the plot, multiselect (**Ctrl + Click**) the first three rows (simulation data) of the **Responses** table and click **Create Set**.

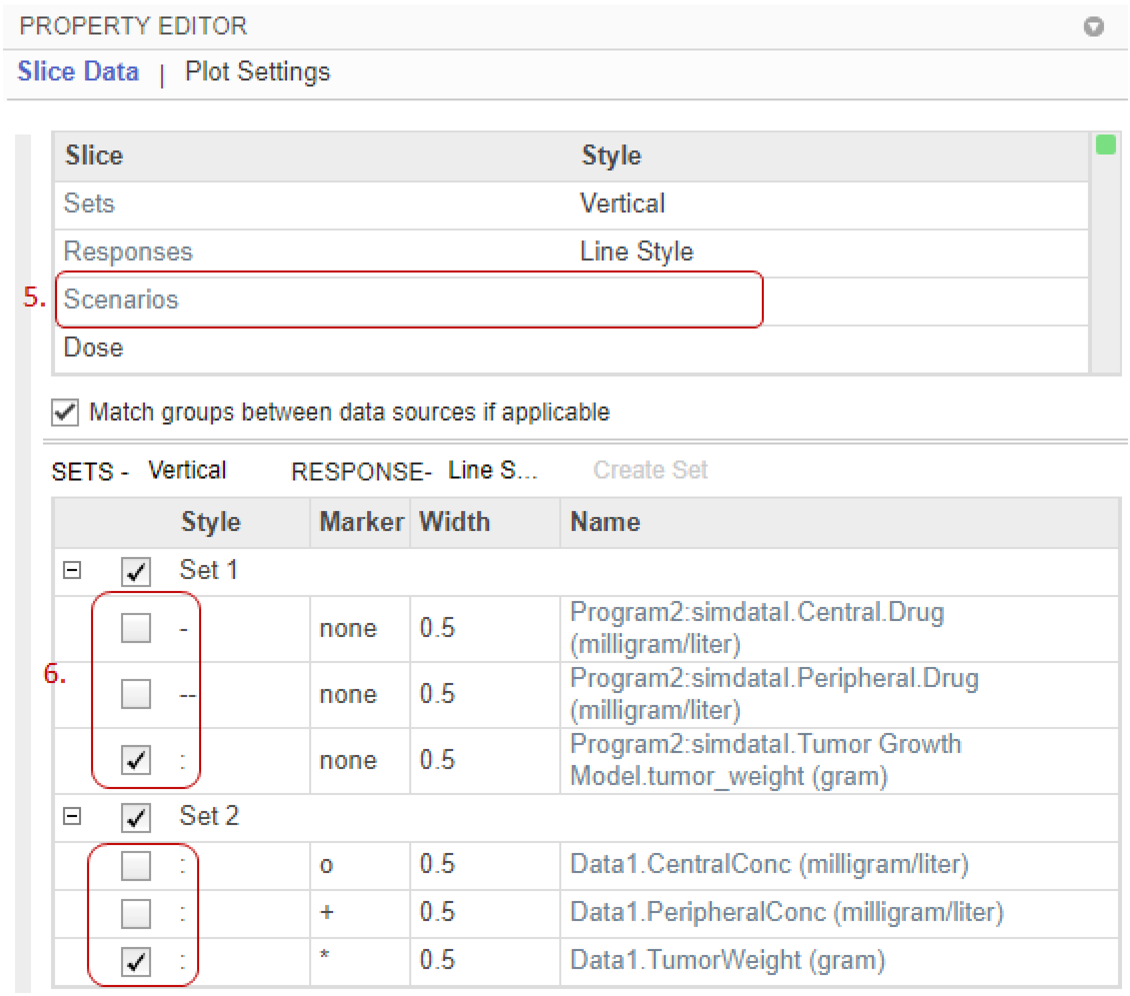


RESPONSES - Line Style Create Set

	Style	Marker	Width	Name
☑	-	none	0.5	Program2:simdata1.Central.Drug (milligram/liter)
☑	--	none	0.5	Program2:simdata1.Peripheral.Drug (milligram/liter)
☑	:	none	0.5	Program2:simdata1.Tumor Growth Model.tumor_weight (gram)
☑	:	o	0.5	Data1.CentralConc (milligram/liter)
☑	:	+	0.5	Data1.PeripheralConc (milligram/liter)
☑	:	*	0.5	Data1.TumorWeight (gram)

5 In the **Slice** table, use no style for **Scenarios** by selecting empty.

6 Clear the first two rows from each set. Keep only the tumor growth data from each set.



PROPERTY EDITOR

Slice Data | Plot Settings

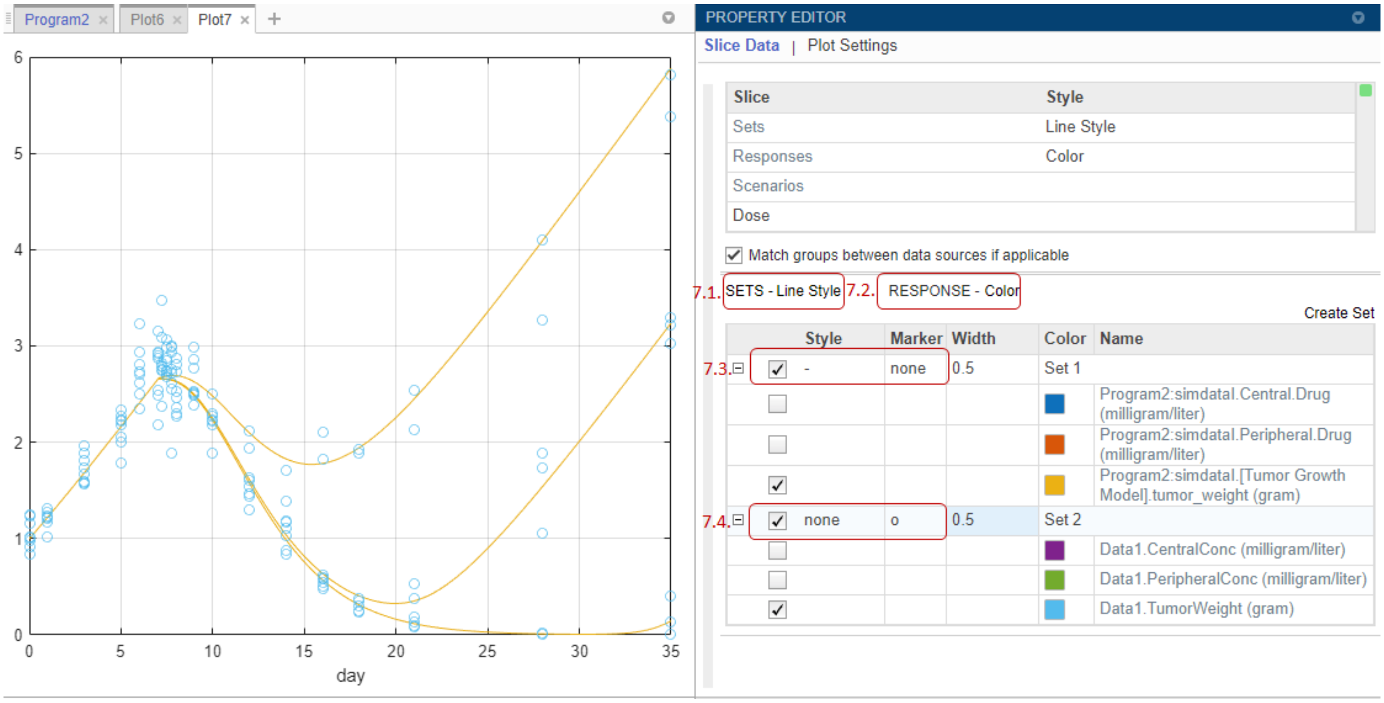
Slice	Style
Sets	Vertical
Responses	Line Style
Scenarios	
Dose	

☑ Match groups between data sources if applicable

SETS - Vertical RESPONSE- Line S... Create Set

	Style	Marker	Width	Name
⊟ ☑	Set 1			
□	-	none	0.5	Program2:simdata1.Central.Drug (milligram/liter)
□	--	none	0.5	Program2:simdata1.Peripheral.Drug (milligram/liter)
☑	:	none	0.5	Program2:simdata1.Tumor Growth Model.tumor_weight (gram)
⊟ ☑	Set 2			
□	:	o	0.5	Data1.CentralConc (milligram/liter)
□	:	+	0.5	Data1.PeripheralConc (milligram/liter)
☑	:	*	0.5	Data1.TumorWeight (gram)

7 Use `Line Style` for **Sets** and `Color` for **Response**. Also set the **Marker** of the simulation data (Program2) to none. Set the **Style** of the experimental data (Data1) to none. Select o as the **Marker**.



Slice	Style
Sets	Line Style
Responses	Color
Scenarios	
Dose	

☑ Match groups between data sources if applicable

7.1. SETS - Line Style 7.2. RESPONSE - Color

Create Set

		Style	Marker	Width	Color	Name
7.3.⊟	☑	-	none	0.5	Set 1	
	☐				🟦	Program2:simdata1.Central.Drug (milligram/liter)
	☐				🟧	Program2:simdata1.Peripheral.Drug (milligram/liter)
	☑				🟨	Program2:simdata1.[Tumor Growth Model].tumor_weight (gram)
7.4.⊟	☑	none	o	0.5	Set 2	
	☐				🟪	Data1.CentralConc (milligram/liter)
	☐				🟩	Data1.PeripheralConc (milligram/liter)
	☑				🟦	Data1.TumorWeight (gram)

After fitting to data, you can set the model values to the parameter estimates and perform other analyses. For instance, you can find out important model parameters using sensitivity analysis on page 1-30 and vary the sensitive parameters to investigate model variability by using virtual patients on page 1-46.

References

[1] Simeoni, M., P. Magni, C. Cammia, G. De Nicolao, V. Croci, E. Pesenti, M. Germani, I. Poggesi, and M. Rocchetti. 2004. Predictive pharmacokinetic-pharmacodynamic modeling of tumor growth kinetics in xenograft models after administration of anticancer agents. *Cancer Research*. 64:1094-1101.

See Also

SimBiology Model Analyzer

More About

- “Find Important Parameters with Sensitivity Analysis Using SimBiology Model Analyzer App” on page 1-30
- “Model Biological Variability with Virtual Patients Using SimBiology Model Analyzer App” on page 1-46
- “Scan Dosing Regimens Using SimBiology Model Analyzer App” on page 1-92

Scan Dosing Regimens Using SimBiology Model Analyzer App

This example shows how to assess the efficacy and toxicity of various dose amounts in the **SimBiology Model Analyzer** app by using the target (receptor) occupancy as a biomarker. The example uses the target-mediated drug disposition (TMDD) model [1] with slight modifications.

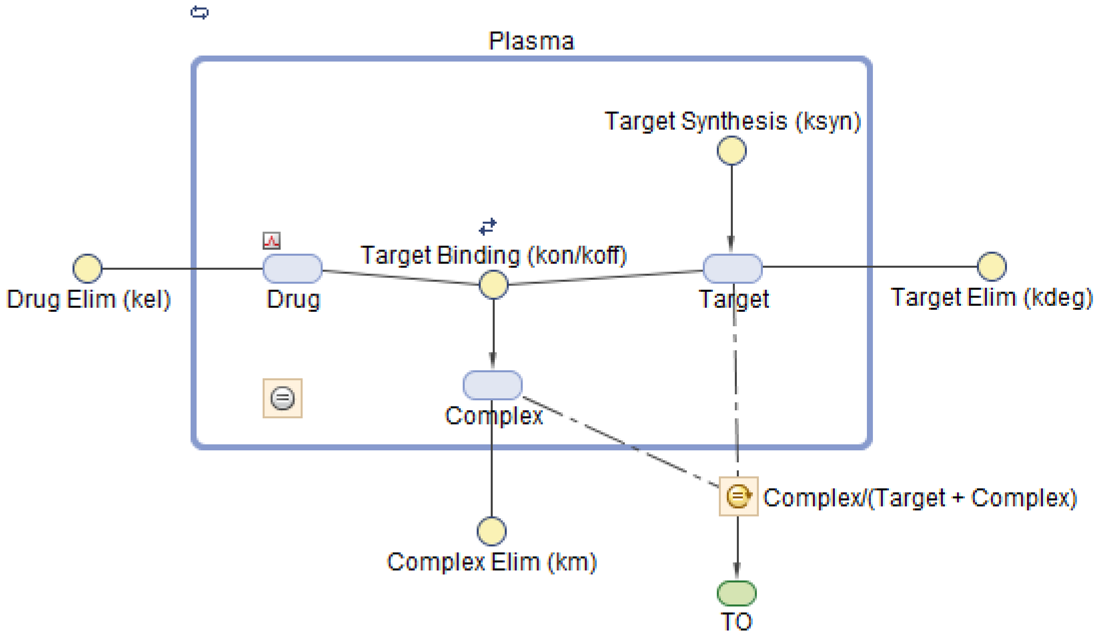
Target-Mediated Drug Disposition (TMDD) Model

Target-mediated drug disposition (TMDD) is a phenomenon in which a drug binds with high affinity to its pharmacologic target site, such as a receptor or enzyme, in an interaction that is reflected in the pharmacokinetic characteristics of the drug.

This example uses a modified TMDD model based on the model used by Mager and Jusko [1] with a minor adjustment. The authors proposed a generic TMDD model that accounted for saturable drug-target binding and target (or receptor) mediated elimination. `Drug` in the `Plasma` reversibly binds with the unbound `Target` to form the drug-target `Complex`. *kon* and *koff* are the association and dissociation rate constants, respectively. Clearance of free `Drug` and `Complex` from the `Plasma` is described by first-order processes with rate constants — *kel* and *km*, respectively. Free target turnover is described by the zero-order synthesis rate *ksyn* and a first-order elimination (the rate constant, *kdeg*). Variants of the TMDD model have been since used to characterize the pharmacokinetics of numerous drugs.

The adjustment made to the model presented by Mager and Jusko, is as follows.

- Target occupancy (*TO*) is modeled as `TO = Complex/(Target + Complex)`, where *TO* is a parameter and *Complex* and *Target* are species.



Explore Model Response Using Different Dose Amounts

Investigate the model response on the target occupancy (*TO*) using different dose amounts.

Load TMDD Model

1 At the MATLAB command line, load the model (*m1*) by entering:

```
sbioloadproject tmdd_with_TO.sbproj
```

2 Open the **SimBiology Model Analyzer** app by typing `simBiologyModelAnalyzer` at the command line or by clicking **SimBiology Model Analyzer** on the **Apps** tab.

3 On the **Home** tab, select **Model** > `Import Model from MATLAB`. Select **TMDD**.

Generate Array of Doses and Simulate Model

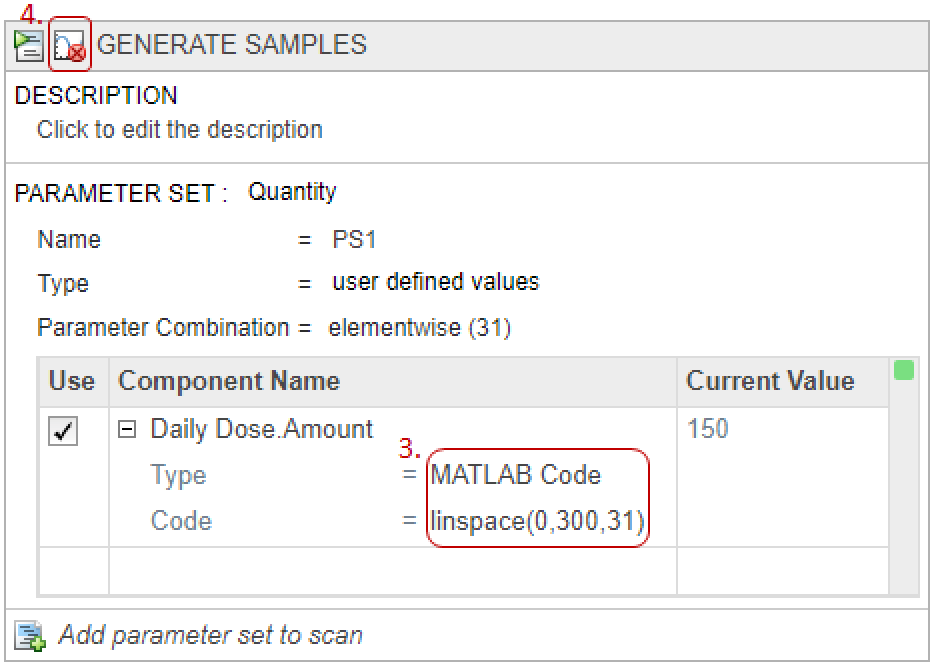
First, create a program to generate an array of doses with different dose amounts ranging from 0 to 300 nanomoles.

1 Select **Program > Generate Samples**.

2 In the **Generate Samples** step, double-click the empty cell under **Component Name** and enter **Daily Dose.Amount**.

3 Under `Daily Dose.Amount`, set **Type** to `MATLAB Code`, and set **Code** to `linspace(0,300,31)`. This code specifies the generation of 31 doses with amounts ranging from 0 to 300 nanomoles.

4 Disable the plotting of dose samples by clicking the plot button, as shown in the figure.



GENERATE SAMPLES

DESCRIPTION
Click to edit the description

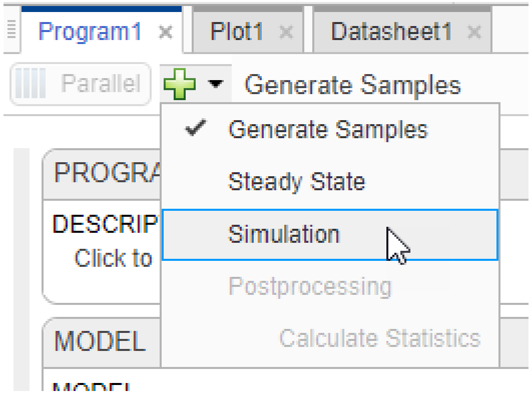
PARAMETER SET : Quantity

Name = PS1
Type = user defined values
Parameter Combination = elementwise (31)

Use	Component Name		Current Value
☑	⊟ Daily Dose.Amount		150
	Type	= MATLAB Code	
	Code	= linspace(0,300,31)	

Add parameter set to scan

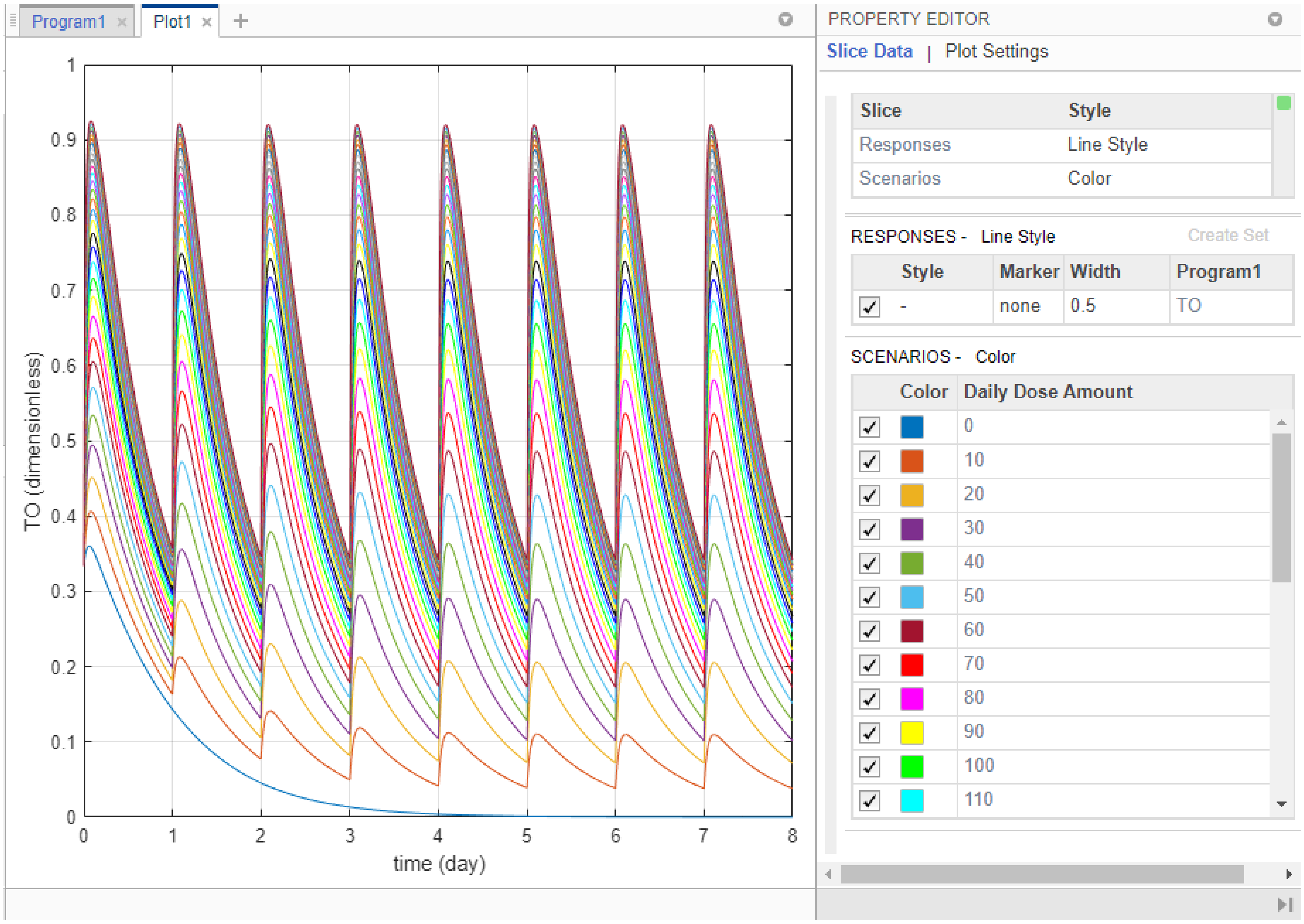
5 Add a simulation step to simulate the model using the defined doses. Click the (+) icon at the upper left of the program and select **Simulation**. A **Simulation** step appears following the **Generate Samples** step.



6 In the **Model** step, click **States to Log**. Select **TO** as the only state to log by clearing all other check boxes.

7 On the **Home** tab, click **Run**.

Once the simulation is complete, the program plots the results in **Plot1**, which shows the time course of the model response (*TO*) given different dose amounts.



Slice	Style
Responses	Line Style
Scenarios	Color

RESPONSES - Line Style

	Style	Marker	Width	Program1
☑	-	none	0.5	TO

SCENARIOS - Color

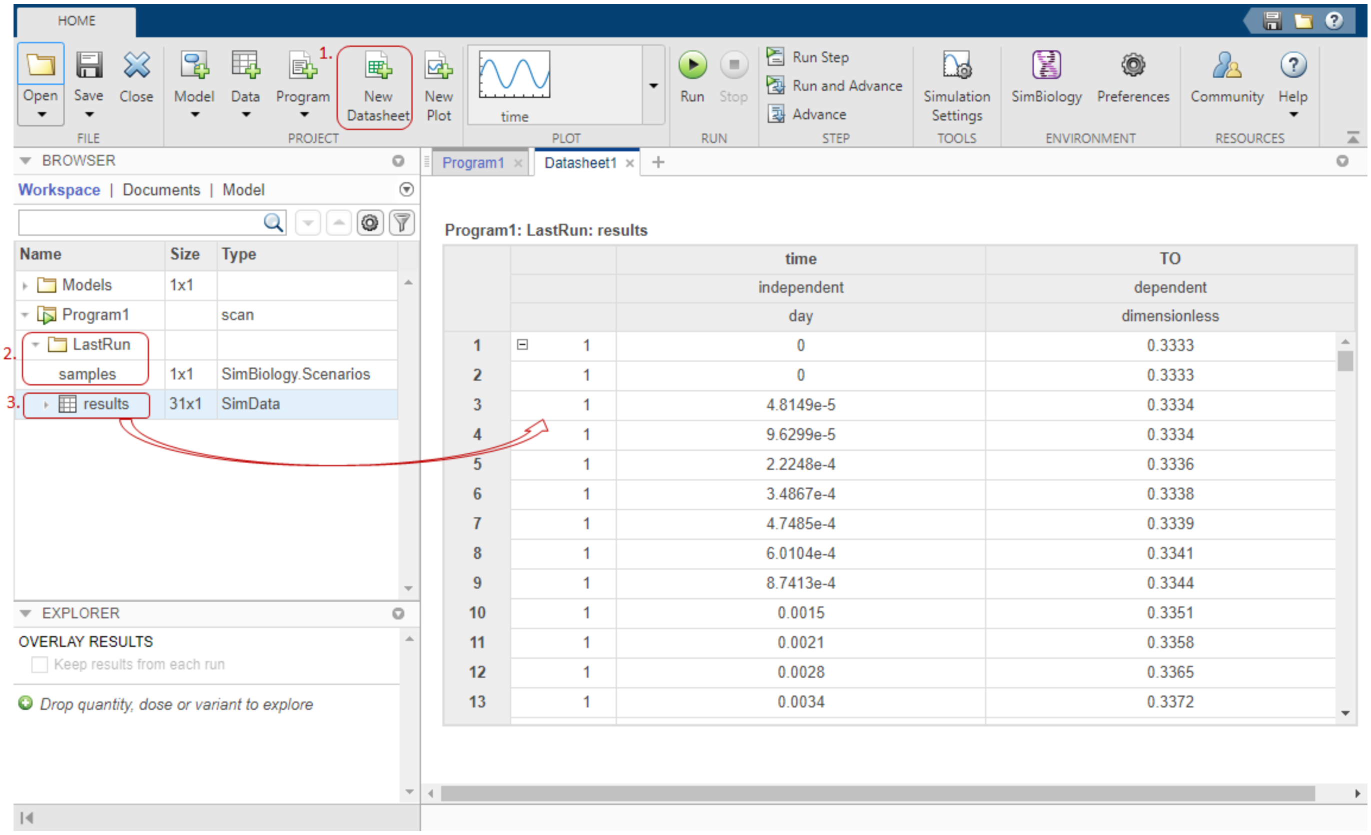
	Color	Daily Dose Amount
☑	🟦	0
☑	🟧	10
☑	🟨	20
☑	🟪	30
☑	🟩	40
☑	🟦	50
☑	🟥	60
☑	🟥	70
☑	🟪	80
☑	🟨	90
☑	🟩	100
☑	🟦	110

Tip Plots are backed by data that are currently present in the app workspace. Plots are not snapshots. When the data (either experimental data or simulation results) is removed or changed, the plots are also updated according to the changes in the underlying data.

Define Maximum and Minimum Target Occupancy Thresholds

Investigate which dose amounts correspond to the *TO* responses that lie within the safety (`TO = 0.85`) and efficacy (`TO = 0.15`) thresholds. One approach is to add two (horizontal) threshold lines to the *TO* response plot.

1. On the **Home** tab, in the **Project** section, click **New Datasheet**.
2. In the **Browser** pane, expand the **Program1** folder, then expand the **LastRun** folder.
3. Drag **results** into the new datasheet. The datasheet now shows **time** and **TO** columns with their corresponding values.

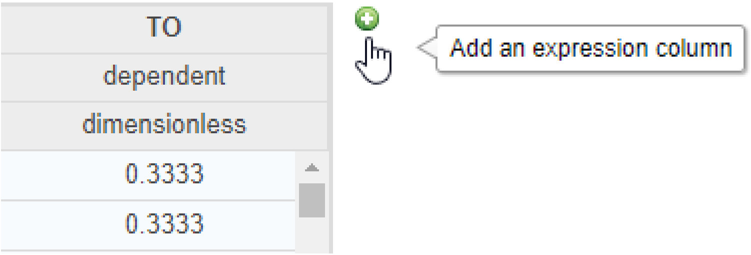


Name	Size	Type
Models	1x1	
Program1		scan
LastRun		
samples	1x1	SimBiology.Scenarios
results	31x1	SimData

Program1: LastRun: results

			time	TO
			independent	dependent
			day	dimensionless
1	⊟	1	0	0.3333
2		1	0	0.3333
3		1	4.8149e-5	0.3334
4		1	9.6299e-5	0.3334
5		1	2.2248e-4	0.3336
6		1	3.4867e-4	0.3338
7		1	4.7485e-4	0.3339
8		1	6.0104e-4	0.3341
9		1	8.7413e-4	0.3344
10		1	0.0015	0.3351
11		1	0.0021	0.3358
12		1	0.0028	0.3365
13		1	0.0034	0.3372

4. Point to the table and add an expression by clicking the (+) icon that appears at the top right.



TO
dependent
dimensionless
0.3333
0.3333

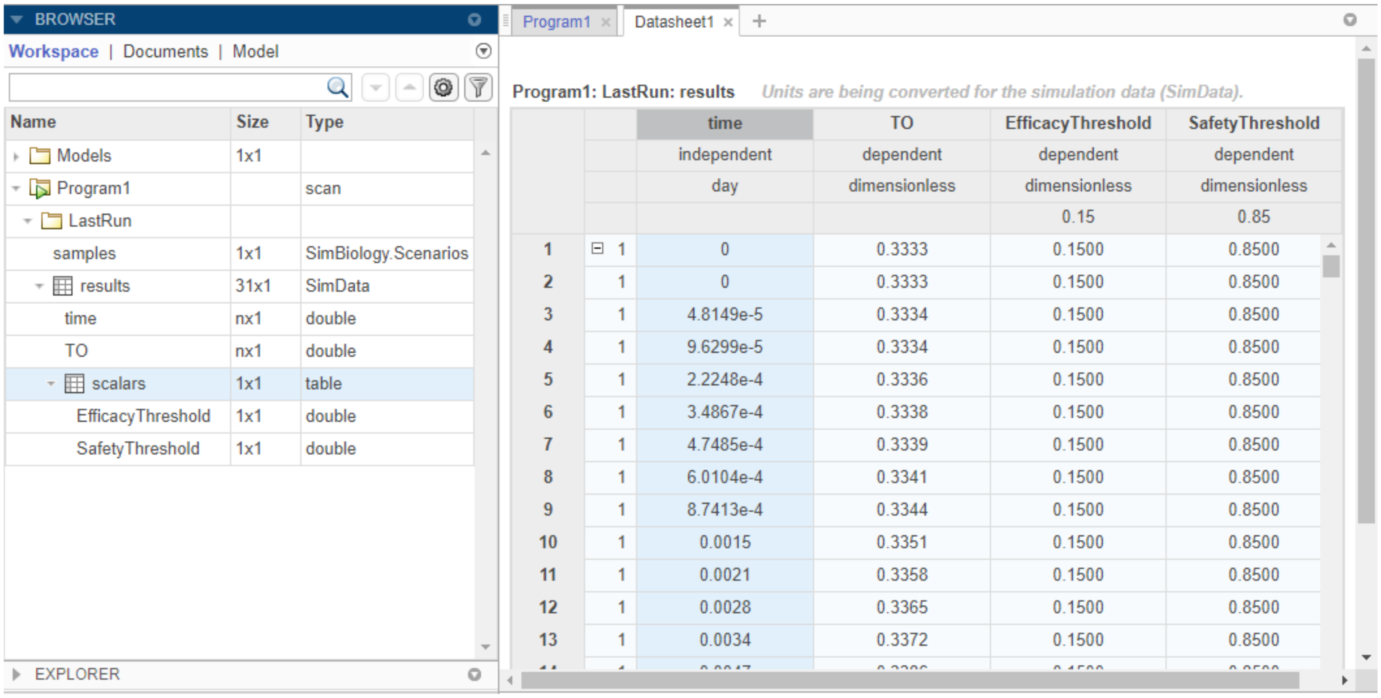
Note Anytime you add an expression to a datasheet containing results from **LastRun**, the expression is added as an `observable` object to the model. In addition, the **Postprocessing: Calculate Observables** step is also added to the corresponding program that generated the **LastRun** data.

5 Double-click **Name1** and rename it **EfficacyThreshold**.

6 Double-click **UNITS** and enter `dimensionless`.

7 Double-click the **Expression** cell and enter `0.15`. The expression fills the column with the same constant value (0.15) for every time point.

8 Similarly, add another expression column named **SafetyThreshold** with the expression `0.85`. Expand **results** in the **LastRun** folder. The values from these two expression columns are scalar-valued `observables` and now stored in the table named **scalars**.



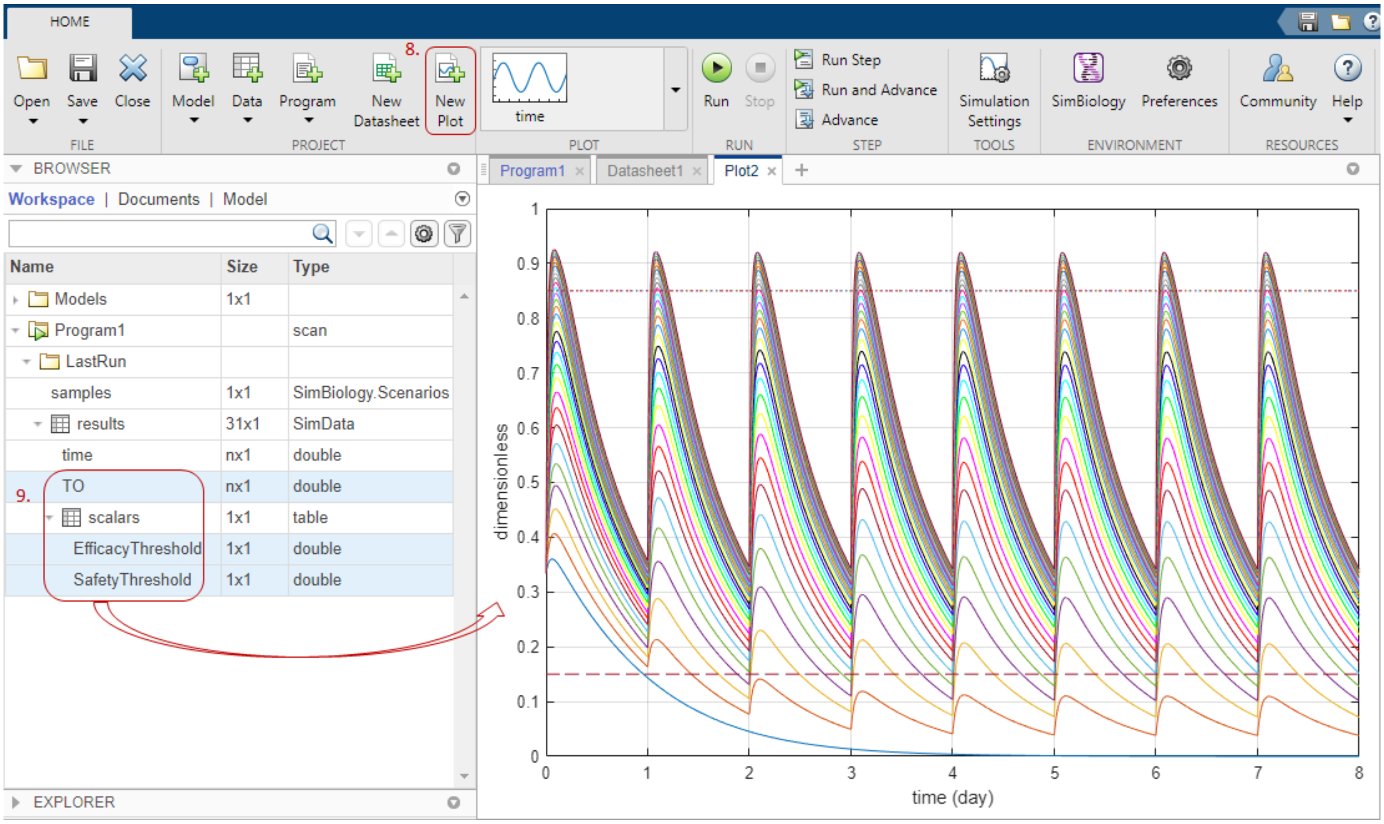
Name	Size	Type
Models	1x1	
Program1		scan
LastRun		
samples	1x1	SimBiology.Scenarios
results	31x1	SimData
time	nx1	double
TO	nx1	double
scalars	1x1	table
EfficacyThreshold	1x1	double
SafetyThreshold	1x1	double

Program1: LastRun: results *Units are being converted for the simulation data (SimData).*

		time	TO	EfficacyThreshold	SafetyThreshold
		independent	dependent	dependent	dependent
		day	dimensionless	dimensionless	dimensionless
				0.15	0.85
1	⊟ 1	0	0.3333	0.1500	0.8500
2	1	0	0.3333	0.1500	0.8500
3	1	4.8149e-5	0.3334	0.1500	0.8500
4	1	9.6299e-5	0.3334	0.1500	0.8500
5	1	2.2248e-4	0.3336	0.1500	0.8500
6	1	3.4867e-4	0.3338	0.1500	0.8500
7	1	4.7485e-4	0.3339	0.1500	0.8500
8	1	6.0104e-4	0.3341	0.1500	0.8500
9	1	8.7413e-4	0.3344	0.1500	0.8500
10	1	0.0015	0.3351	0.1500	0.8500
11	1	0.0021	0.3358	0.1500	0.8500
12	1	0.0028	0.3365	0.1500	0.8500
13	1	0.0034	0.3372	0.1500	0.8500

9 You can now plot the maximum and minimum thresholds along with the simulation data. On the **Home** tab, click **New Plot**.

10 Press **Ctrl** and select the **TO**, **EfficacyThreshold**, and **SafetyThreshold** variables from the **Browser** pane. Drag them into the plot. The plot now shows *TO* profiles along with the efficacy and safety threshold lines.

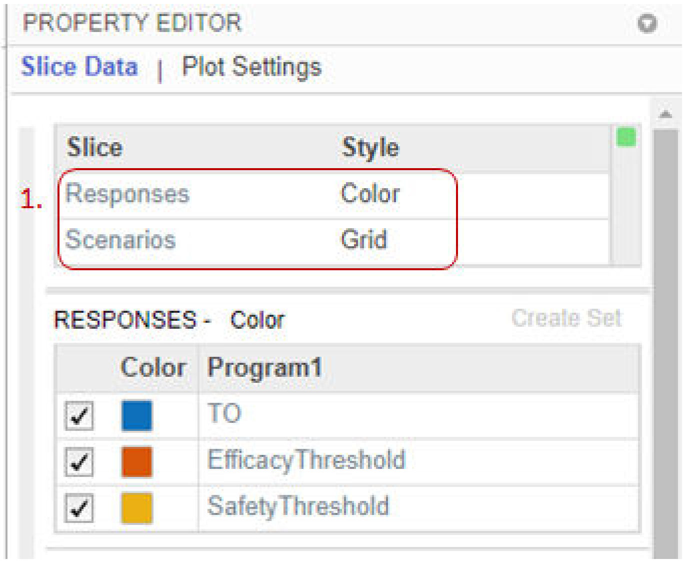


Name	Size	Type
Models	1x1	
Program1		scan
LastRun		
samples	1x1	SimBiology.Scenarios
results	31x1	SimData
time	nx1	double
TO	nx1	double
scalars	1x1	table
EfficacyThreshold	1x1	double
SafetyThreshold	1x1	double

Visualize Target Occupancy Responses on Separate Axes

The plot shows that certain *TO* responses either exceed the safety threshold or dip below the efficacy threshold. To better visualize this observation, you can customize the plot to see each dose amount and the corresponding *TO* response on separate axes.

1 In the **Property Editor** of the plot, in the **Slice Data** table, set **Responses** to `Color` and **Scenarios** to `Grid`.

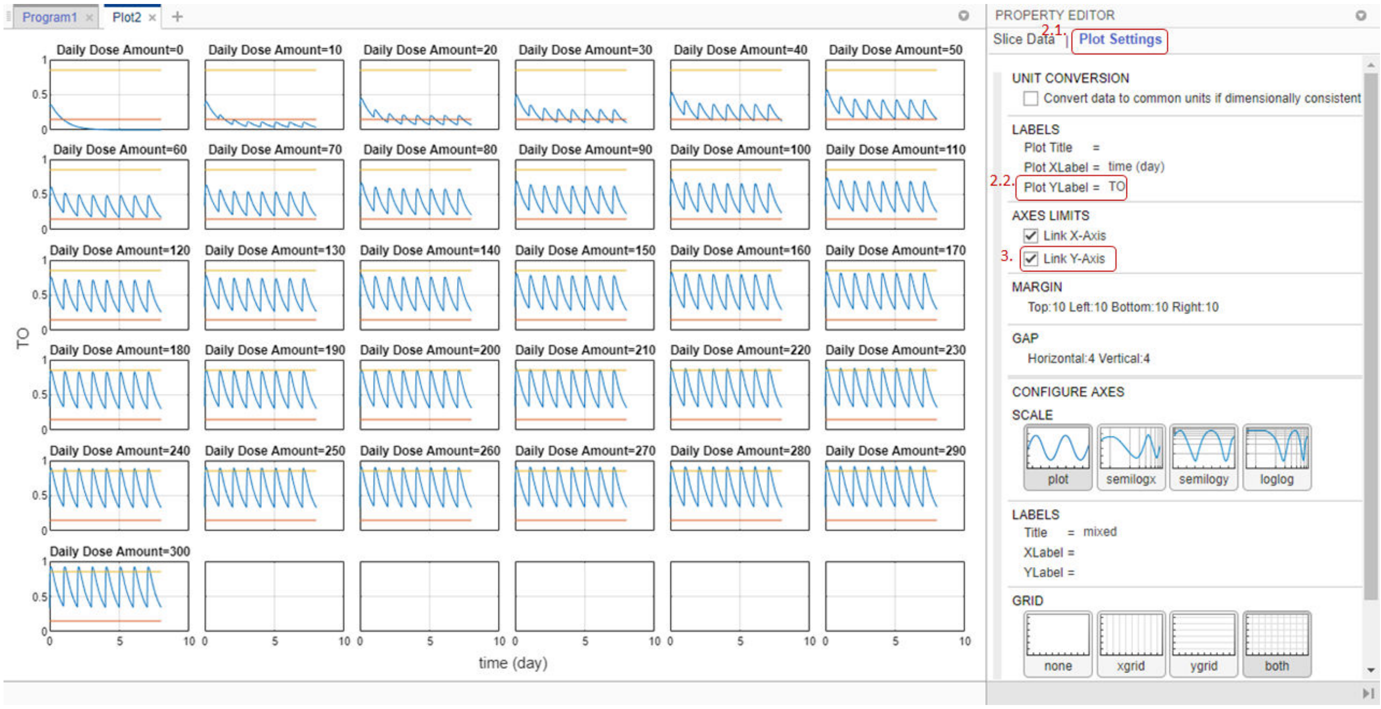


Slice	Style
Responses	Color
Scenarios	Grid

	Color	Program1
☑	🟦	TO
☑	🟧	EfficacyThreshold
☑	🟨	SafetyThreshold

2 Click **Plot Settings**. For **Plot YLabel**, enter `TO` as the value.

3 In the **Axes Limits** section, select **Link Y-Axis** to show the same set of labels on the *y*-axis for all subplots.



PROPERTY EDITOR

Slice Data | Plot Settings

UNIT CONVERSION

□ Convert data to common units if dimensionally consistent

LABELS

Plot Title =

Plot XLabel = time (day)

Plot YLabel = TO

AXES LIMITS

☑ Link X-Axis

☑ Link Y-Axis

MARGIN

Top:10 Left:10 Bottom:10 Right:10

GAP

Horizontal:4 Vertical:4

CONFIGURE AXES

SCALE

LABELS

Title = mixed

XLabel =

YLabel =

GRID

Postprocess Simulation Results

In addition to visually inspecting each response plot on separate axes, you can add a postprocessing step to query exactly which dose amounts stay within the thresholds.

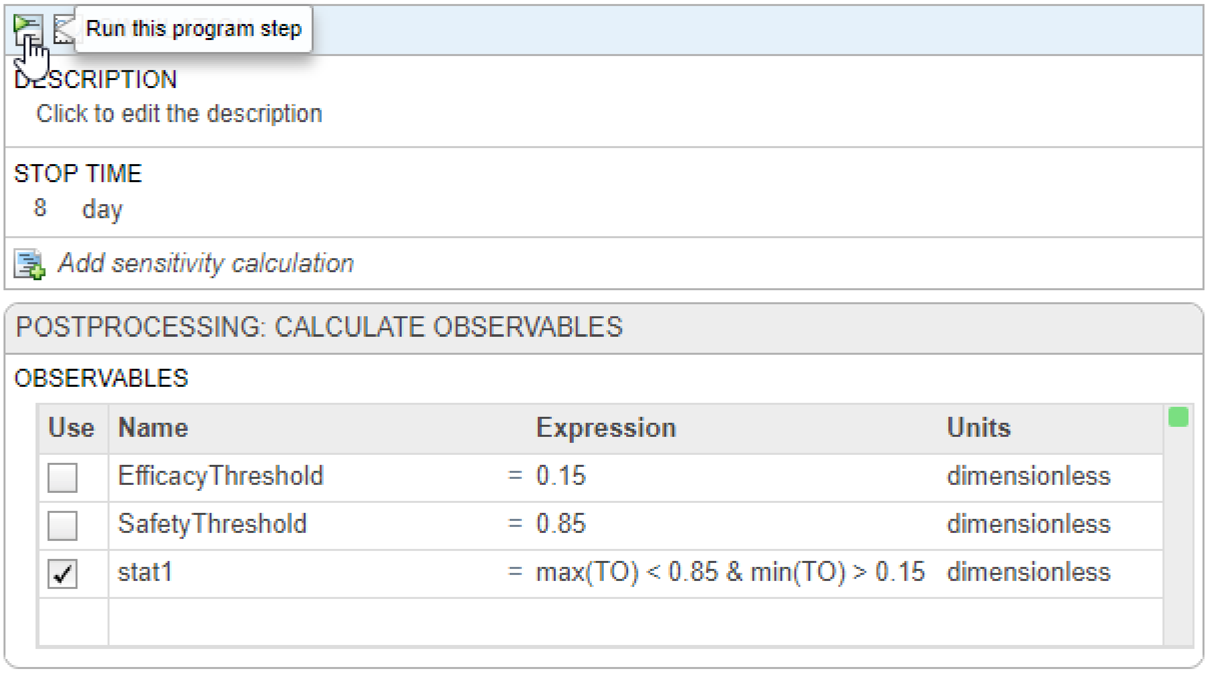
1 Click the **Program1** tab. In the **Postprocessing: Calculate Observables** step, double-click the first empty cell in the **Name** column and enter: `stat1`.

2 In the **Expression** column, enter `max(TO) < 0.85 & min(TO) > 0.15`.

Note Anytime you add an expression to the **Observables** table in the step, the expression is automatically added as an `observable` object to the corresponding model.

3 Select `stat1` as the only observable. Clear `EfficacyThreshold` and `SafetyThreshold`.

4 To evaluate `stat1`, run just the simulation step by clicking the run button at the top of the **Simulation** step.



DESCRIPTION

Click to edit the description

STOP TIME

8 day

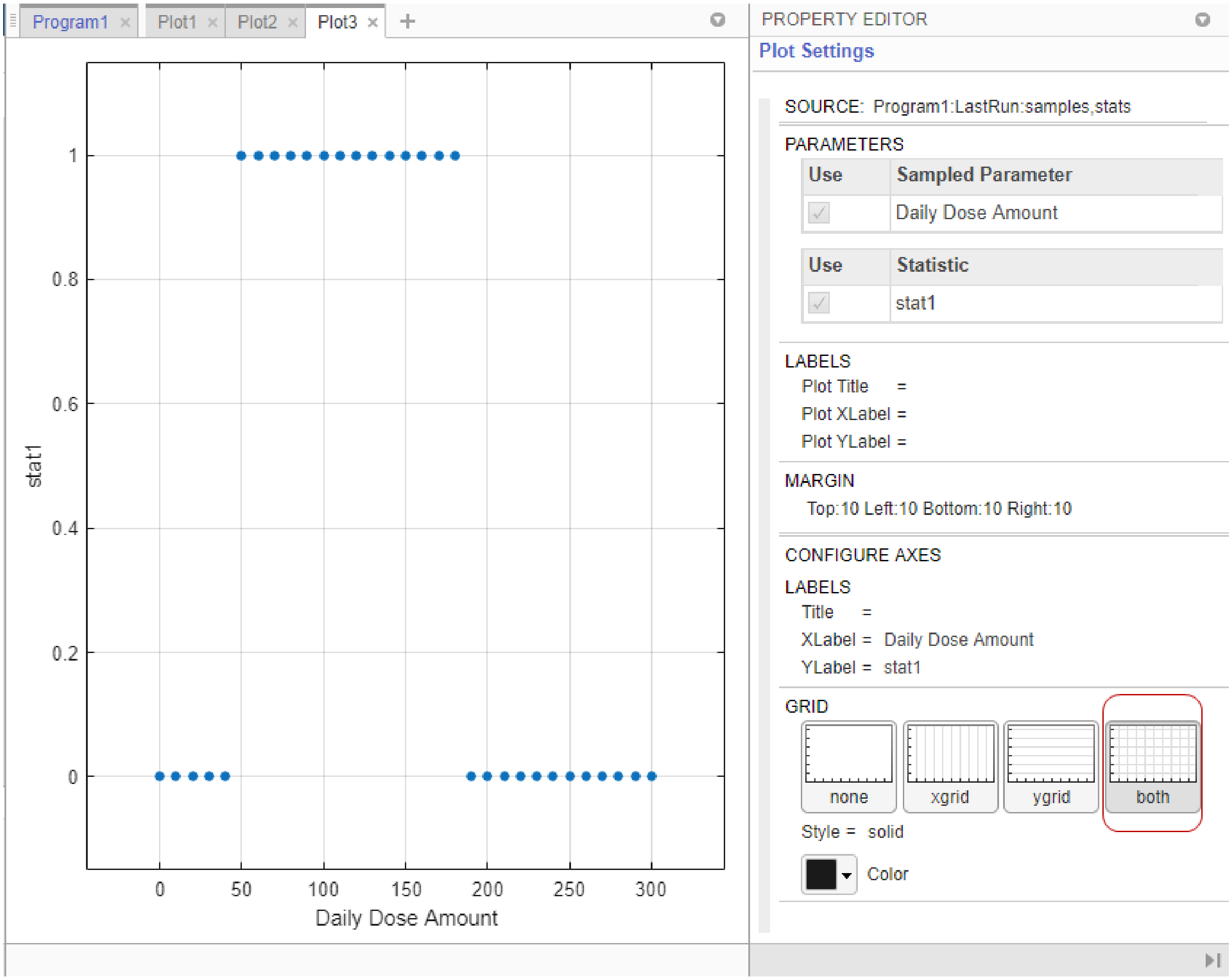
Add sensitivity calculation

POSTPROCESSING: CALCULATE OBSERVABLES

OBSERVABLES

Use	Name		Expression	Units
□	EfficacyThreshold	=	0.15	dimensionless
□	SafetyThreshold	=	0.85	dimensionless
☑	stat1	=	max(TO) < 0.85 & min(TO) > 0.15	dimensionless

Running the program step generates the following figure. Display both the x-grid and y-grid by clicking **both** in the **Grid** section. The *x*-axis represents the dose amounts and the *y*-axis represents whether the dose amount generates a *TO* response that stays within the safety and efficacy thresholds (with a value of 1) or not (with a value of 0).



Plot Settings

SOURCE: Program1:LastRun:samples,stats

PARAMETERS

Use	Sampled Parameter
☑	Daily Dose Amount

Use	Statistic
☑	stat1

LABELS

Plot Title =
Plot XLabel =
Plot YLabel =

MARGIN

Top:10 Left:10 Bottom:10 Right:10

CONFIGURE AXES

LABELS

Title =
XLabel = Daily Dose Amount
YLabel = stat1

GRID

Style = solid

The plot shows that dose amounts ranging from `50` to `180` nanomoles provide *TO* responses that lie within the desired efficacy and safety thresholds.

See Also

`Observable` | **SimBiology Model Analyzer** | `simbiology`

More About

- "Calculate NCA Parameters and Fit Model to PK/PD Data Using SimBiology Model Analyzer App" on page 1-64
- "Find Important Parameters with Sensitivity Analysis Using SimBiology Model Analyzer App" on page 1-30
- "Model Biological Variability with Virtual Patients Using SimBiology Model Analyzer App" on page 1-46

References

[1] Marger, D., and W. Jusko. 2001. General pharmacokinetic model for drugs exhibiting target-mediated drug disposition. *Journal of Pharmacokinetics and Pharmacodynamics*. 28: 507–532.

Modeling

What is a SimBiology Model?

In this section...

Model Definition

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

Expressions

There are four distinct types of expressions in SimBiology:

- Reactions
- Rules
- Events
- Observables

Reactions

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

Example reactions include:

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

Rules

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

For example, you can use a rule to:

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

Events

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

For example, you can use an event to:

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

Observable

An `Observable` is a mathematical expression that lets you perform post-simulation calculations. For example, you can use an observable to:

- Compute some statistics such as area under the curve (AUC) of a drug concentration profile.
- Compute the fraction of a ligand that is bound to a receptor at each time step.

You can also use an observable as a response in simulation, data fitting, and global sensitivity analysis.

Quantities

SimBiology uses three types of quantities in models:

- Compartments
- Species
- Parameters

Compartments

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

Species

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

Parameters

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

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

Model Hierarchy

Note the following conditions imposed on quantities in the model hierarchy:

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

Representing a Model

In SimBiology, models and their components are implemented as objects. These objects have properties and methods that you can use to access and configure them. Use the `get` method to list the property values of an object. Use the dot notation to change the property values of an object.

SimBiology objects are handle objects, which has implications for how they behave during copy operations. Handle objects are referenced by their handle variable, and copies of the handle variable refer to the same object. To learn how handle objects affect copy operations, see Copying Objects (MATLAB).

See Also

`Model`

Species Object

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

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

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

How Species Amounts Change During Simulations

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

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

Keeping a Species Amount Unchanged

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

ConstantAmount	BoundaryCondition	Reaction	Rule	Changed By
True	False	No	No	Never

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

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

Changing a Species Amount with a Reaction or Rule

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

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

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

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

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

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

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

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

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

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

ConstantAmount	BoundaryCondition	Reaction	Rule	Changed By
False	True	Yes	Yes	Rule

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

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

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

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

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

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

ConstantAmount	BoundaryCondition	Reaction	Rule	Changed By
True	True	Yes	No	Never

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

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

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

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

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

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

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

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

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

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

Definitions and Evaluations of Reactions in SimBiology Models

A reaction is a mathematical expression that describe a transformation, transport, or binding process that changes one or more species. Typically, an amount of a species is changed through a reaction.

In SimBiology, a reaction is represented by a `reaction object`, which has the following properties.

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

A reaction is scoped to a model.

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

Writing Reaction Expressions

Use standard chemistry reaction notation to create the mathematical expression for a reaction (`Reaction` property of a `reaction object`).

Following are rules for writing reaction expressions:

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

Examples of reaction expressions include:

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

Note Same species can be used multiple times in the list of reactions or products. The expression `'2 A'` is equivalent to `'A + A'`.

Writing Reaction Rate Expressions Explicitly

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

Following are rules for writing reaction rate expressions:

- The expression must be a single MATLAB statement that returns a scalar.
- In a model with a single compartment, specify species using *`speciesName`*. In a model with multiple compartments, specify species using qualified names: *`compartmentName`*.*`speciesName`*. For example, `nucleus.DNA` denotes the species `DNA` in the compartment `nucleus`.
- Enclose names with non-alphanumeric characters (including spaces) in brackets.
- Do not end the reaction rate expression with any of the following:
  - Semicolon
  - Comma
  - Comment text preceded by `%`
  - Line continuations indicated by `...`

For example, if you have the following reaction expression:

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

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

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

Tip If your reaction rate expression is not continuous and differentiable, see “Using Events to Address Discontinuities in Rule and Reaction Rate Expressions” on page 2-28 before simulating your model.

Creating Reaction Rate Expressions Using Kinetic Law Objects

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

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

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

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

$V_m * S/(K_m + S)$

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

```
A -> B
```

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

Va*A/(Ka + A)

Examples of Creating Reaction Rates

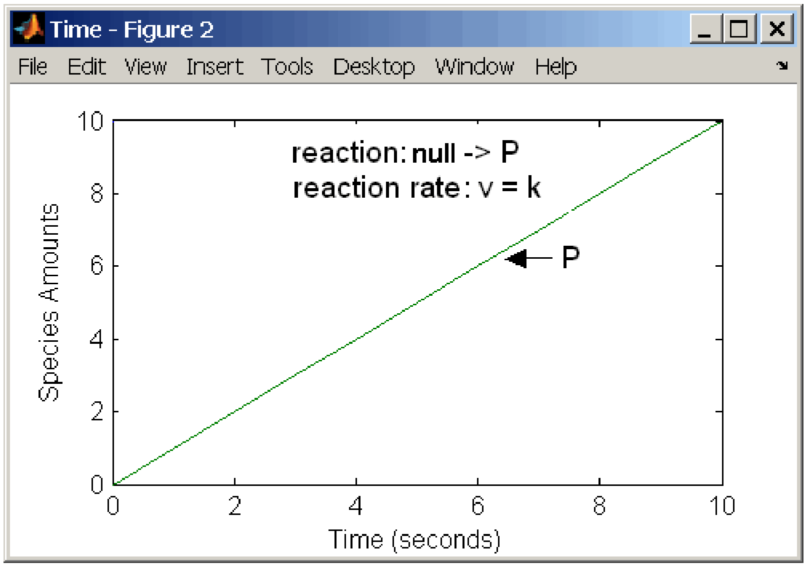
Example of Creating a Zero-Order Reaction

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

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

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

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



Zero-Order Mass Action Kinetics

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

Examples of Creating Other Reactions

For examples of creating other reaction rates, see “Define Reaction Rates with Mass Action Kinetics” on page A-2 and “Define Reaction Rates with Enzyme Kinetics” on page A-6.

How Reaction Rates Are Evaluated

Reaction Rate Dimensions

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

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

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

Reactions Spanning Multiple Compartments

Specify reactions that span compartments using the syntax *`compartment1Name`*.*`species1Name`* -> *`compartment2Name`*.*`species2Name`*. The reaction rate dimensions must resolve to amount/time if either of the following conditions are true:

- Reactant species are in different compartments.
- The reaction is reversible mass action and the products are in multiple compartments.

Note The `MassAction` and `Unknown` kinetic laws can have different simulation results even when the reaction rate is the same. This can happen when you have a reversible reaction with species in different compartments. The difference in simulation results is because of the volume-scaling performed by SimBiology during the dimensional analysis. For details, see “Deriving ODEs from Reactions” on page 4-4. Specifically, for `MassAction`, SimBiology uses corresponding compartment volumes to multiply the forward and reverse rates. However, for `Unknown` and other built-in kinetic laws, SimBiology multiplies the entire rate by only one compartment which contains the reactants. To see exactly what compartment volumes are used for scaling, use `getequations` or open the **Equations** view from the **SimBiology** app and check the ODEs section.

Examples

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

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

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

Viewing Equations for Reactions

You can view the system of equations that SimBiology creates when you build a model using reaction expressions. For details, see “View Model Equations” on page 2-42.

See Also

More About

- “Create and Simulate a Model with a Custom Function” on page 2-36
- “Model Simulation” on page 4-2
- “Deriving ODEs from Reactions” on page 4-4
- “What is a SimBiology Model?” on page 2-2
- “Definitions and Evaluations of Rules in SimBiology Models” on page 2-13
- “Events in SimBiology Models” on page 2-22
- “Component Usage” on page 2-43
- “Evaluation of Model Component Names in Expressions” on page 2-45

Definitions and Evaluations of Rules in SimBiology Models

In this section...

Overview

Rules are mathematical expressions that allow you to define or modify model quantities, namely compartment capacity, species amount, or parameter value.

Rules can take the form of initial assignments, assignments during the course of a simulation (repeated assignments), algebraic relationships, or differential equations (rate rules). Details of each type of rule are described next.

Initial Assignment

An initial assignment rule lets you specify the initial value of a model quantity as a numeric value or as a function of other model quantities. It is evaluated once at the beginning of a simulation.

An initial assignment rule is expressed as `Variable = Expression`, and the rule is specified as the `Expression`. For example, you could write an initial assignment rule to set the initial amount of `species2` to be proportional to `species1` as `species2 = k * species1` where `k` is a constant parameter.

Repeated Assignment

A repeated assignment rule lets you specify the value of a quantity as a numeric value or as a function of other quantities repeatedly during the simulation. It is evaluated at every time step, which is determined by the solver during the simulation process.

A repeated assignment rule is expressed as `Variable = Expression`, and the rule is specified as the `Expression`. For example, to repeatedly evaluate the total species amount by summing up the species in different compartments, you could enter: `xTotal = c1.X + c2.X`, where `xTotal` is a nonconstant parameter, `c1` and `c2` are the name of compartments where species x resides.

Algebraic Rules

An algebraic rule lets you specify mathematical constraints on one or more model quantities that must hold during a simulation. It is evaluated continuously during a simulation.

An algebraic rule takes the form `0 = Expression`, and the rule is specified as the `Expression`. For example, if you have a mass conservation equation such as `species_total = species1 + species2`, write the corresponding algebraic rule as `species1 + species2 - species_total`.

However, repeated assignment rules are mathematically equivalent to algebraic rules, but result in exact solutions instead of approximated solutions. Therefore, it is recommended that you use repeated assignment rules instead of algebraic rules whenever possible. Use algebraic rules only when:

- You cannot analytically solve the equations to get a closed-form solution. For example, there is no closed-form solution for `x^4 + ax^3 + bx^2 + cx + k = 0` whereas the closed-form solution for `kx – c = 0` is `x = c/k`.
- You have multiple equations with multiple unknowns, and they could be inconvenient to solve.

If you use an algebraic rule, rate rule, or repeated assignment to vary the value of a parameter or compartment during the simulation, make sure the `ConstantValue` property of the parameter or `ConstantCapacity` of the compartment is set to `false`.

Repeated Assignment vs. Algebraic Rules

Repeated assignment rules are mathematically equivalent to algebraic rules, but result in exact solutions. However, algebraic rules are solved numerically, and the accuracy depends on the error tolerances specified in the simulation settings. In addition, there are several advantages to repeated assignment rules such as better computational performance, more accurate results since no rules have to be solved numerically (hence no approximations), and sensitivity analysis support.

Tip

- If you can analytically solve for a variable, use a repeated assignment rule instead of an algebraic rule.
- In repeated assignment rules, the constrained variable is explicitly defined as the left-hand side, whereas in algebraic rules it is inferred from the degrees of freedom in the system of equations. See also “Considerations When Imposing Constraints” on page 2-16.

Rate Rules

A rate rule represents a differential equation and lets you specify the time derivative of a model quantity. It is evaluated continuously during a simulation.

A rate rule is represented as $\frac{dVariable}{dt} = Expression$, which is expressed in SimBiology as `Variable = Expression`. For example, if you have a differential equation for species x, $\frac{dx}{dt} = k(y + z)$, write the rate rule as: `x = k * (y + z)`.

For more examples, see “Rate Rule Examples” on page 2-16.

Evaluation Order of Rules

At the start of the simulation (that is, at simulation time = 0), SimBiology evaluates the initial assignment and repeated assignment rules as a set of simultaneous constraints. SimBiology treats the rules as a unified system of constraints and automatically reorders and evaluates them. The order in which the rules appear in the model has no effect on the simulation results.

If a quantity is being modified by an assignment rule, the rule replaces initial value properties, such as `InitialAmount`, `Capacity`, or `Value`. Similarly, a variant altering such quantities has no effect because the value is superseded by the assignment rules.

SimBiology throws an error if the model has circular dependencies in the initial assignment and repeated assignment rules. In other words, initial assignments and repeated assignments cannot have a variable that is explicitly or implicitly referenced on both the left- and right-hand sides of the equation.

For instance, you cannot create circular sets of assignments such as `a = b + 1` and `b = a + 1`, where *a* and *b* are explicitly referenced on both sides of the equation. An example of an implicit reference is when an assignment rule references a species in concentration. In this case, the compartment that contains the species is implicitly referenced.

Compatibility Considerations

You might observe different simulation results with respect to initial assignments for previous releases of SimBiology (R2017a or earlier). To recover the same simulation results at time = 0, as in R2017a or earlier, use the `updateInitialAssignments` function in the command line. If you are using the SimBiology app, right-click the model from the **Project Workspace** and select **Remove Order Dependencies**.

Conservation of Amounts During Simulation

During a simulation (that is, at simulation time > 0), SimBiology conserves species amounts rather than concentrations if there are any changes to the volume of a compartment where the species reside. In other words, if you have a repeated assignment rule or an event that changes the volume, then you see the effect of conservation of species amounts at time > 0.

However, at the beginning of a simulation (that is, at simulation time = 0), the concept of amount conservation does not apply because there are no changes before time = 0. Only one set of initial conditions exists and SimBiology uses the conditions at the start of the simulation. Specifically, at time = 0, SimBiology:

1. Initializes variables for species, compartments, and parameters using the corresponding `InitialAmount`, `Capacity`, and `Value` properties.
2. Updates the values by replacing them with the corresponding alternate values from variants, if any.
3. Updates the values by evaluating initial assignment and repeated assignment rules as a set of simultaneous constraints. Therefore, the assignment rules replace initial values if model quantities are being modified by such rules or variants.

Compatibility Considerations

In previous releases (R2017a or earlier), if a repeated assignment changed a compartment volume, SimBiology used the compartment capacity to determine the initial amount and conserved it when the compartment volume changed at time = 0. In R2017b or later, SimBiology uses the InitialAmount property of the species as the initial condition at time = 0. Consider the following model.

```
m = sbiomodel('m1')
v = addcompartment(m,'v',10,'ConstantCapacity',0,'CapacityUnit','liter')
p = addparameter(m,'p','ValueUnit','liter')
r = addrule(m,'v = 100 * p','repeatedAssignment')
s = addspecies(v,'s',50,'InitialAmountUnit','milligram/liter')
```

In R2017a or earlier, SimBiology first calculated the initial amount of *s* as `50 milligram/liter * 10 liter = 500 milligram`, and then applied the repeated assignment rule `v = 100 liter`. So, the concentration of *s* was then calculated and reported as `500 milligram/100 liter = 5 milligram/liter` at time = 0.

In R2017b or later, SimBiology uses the InitialAmount property of species *s*, and reports the initial amount of *s* as `50 milligram/liter` instead.

Writing Rule Expressions

Use MATLAB syntax to write a mathematical expression for a rule. Note that no semicolon or comma is needed at the end of a rule expression. If your algebraic, repeated assignment, or rate rule expression is not continuous or differentiable, see “Using Events to Address Discontinuities in Rule and Reaction Rate Expressions” on page 2-28 before simulating your model.

Considerations When Imposing Constraints

Suppose that you have a species y whose amount is determined by the equation `y = m * x - c`. In SimBiology, the algebraic rule to describe this constraint is written as `m * x - c - y`. If you want to use this rule to determine the value of y, then m, x, and c must be variables or constants whose values are known or determined by other equations. Therefore, you must ensure that the system of equations is not overconstrained or underconstrained. For instance, if you have more equations than unknowns, then the system is overconstrained. Conversely, if you have more unknowns than the equations, then the system is underconstrained.

Tip The behavior of an underconstrained system could be fixed by adding additional rules or by setting the ConstantValue or ConstantCapacity or ConstantAmount property of some of the components in the model.

Rate Rule Examples

The following examples show how to create rate rules for different applications.

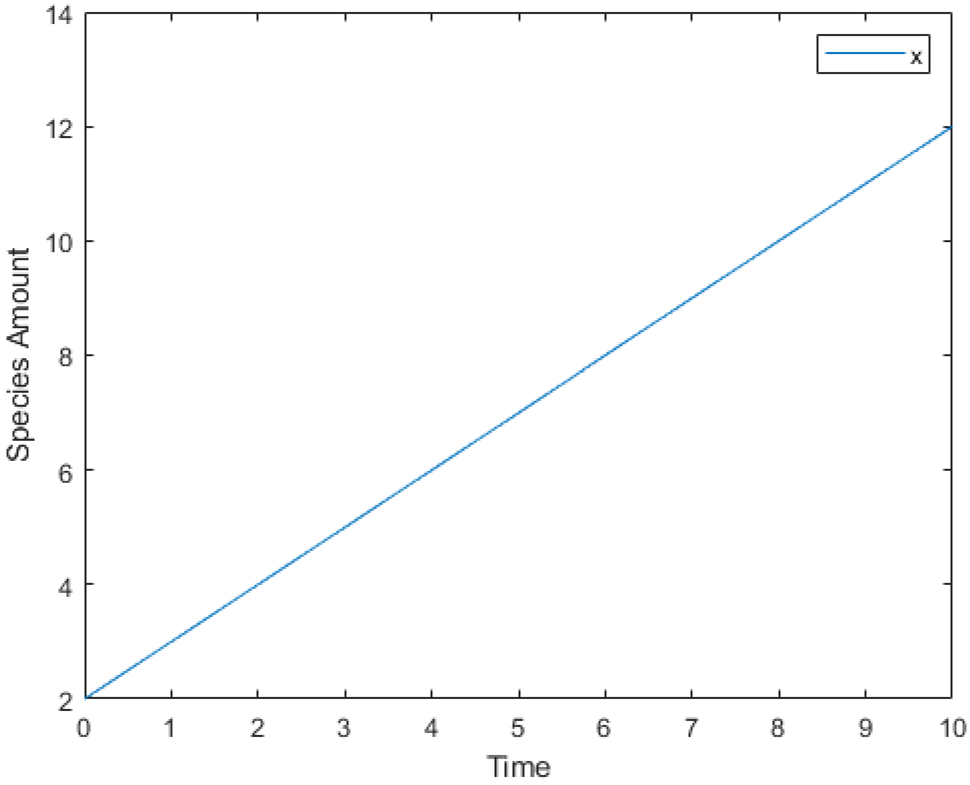
Create a Rate Rule for a Constant Rate of Change

This example shows how to increase the amount or concentration of a species by a constant value using the zero-order rate rule. For example, suppose species x increases by a constant rate k. The rate of change is:

$$dx/dt = k$$

Set the initial amount of species x to 2, and the value of parameter k to 1. Use the following commands to set up a SimBiology model accordingly and simulate it.

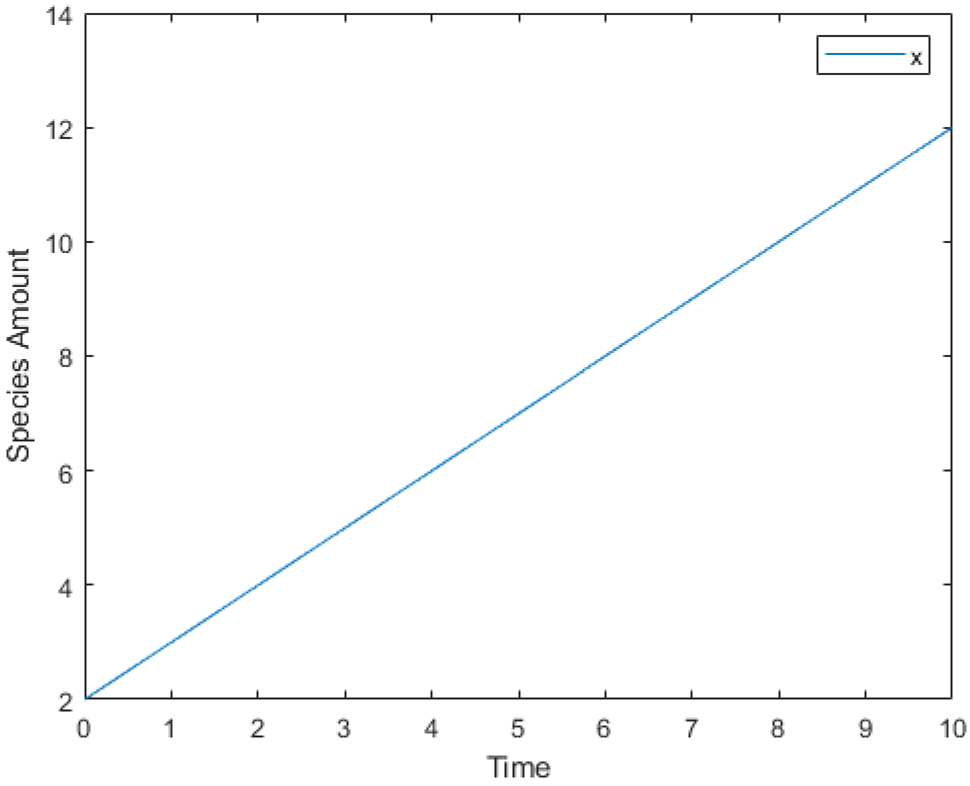
```
m = sbiomodel('m');
c = addcompartment(m,'comp');
s = addspecies(m,'x','InitialAmount',2);
p = addparameter(m,'k','Value',1);
r = addrule(m,'x = k','RuleType','rate');
[t,sd,species] = sbiosimulate(m);
plot(t,sd);
legend(species)
xlabel('Time');
ylabel('Species Amount');
```



Alternatively, you could model a constant increase in a species using the Mass Action reaction null -> x with the forward rate constant k.

```
clear
m = sbiomodel('m');
c = addcompartment(m,'comp');
s = addspecies(m,'x','InitialAmount',2);
r = addreaction(m,'null -> x');
kl = addkineticlaw(r,'MassAction');
p = addparameter(kl,'k','Value',1);
kl.ParameterVariableNames = 'k';
[t,sd,species] = sbiosimulate(m);
```

```
plot(t,sd);
legend(species)
xlabel('Time');
ylabel('Species Amount');
```



Create a Rate Rule for an Exponential Rate of Change

This example shows how to change the amount of a species similar to a first-order reaction using the first-order rate rule. For example, suppose the species x decays exponentially. The rate of change of species x is:

$$dx/dt = -k * x$$

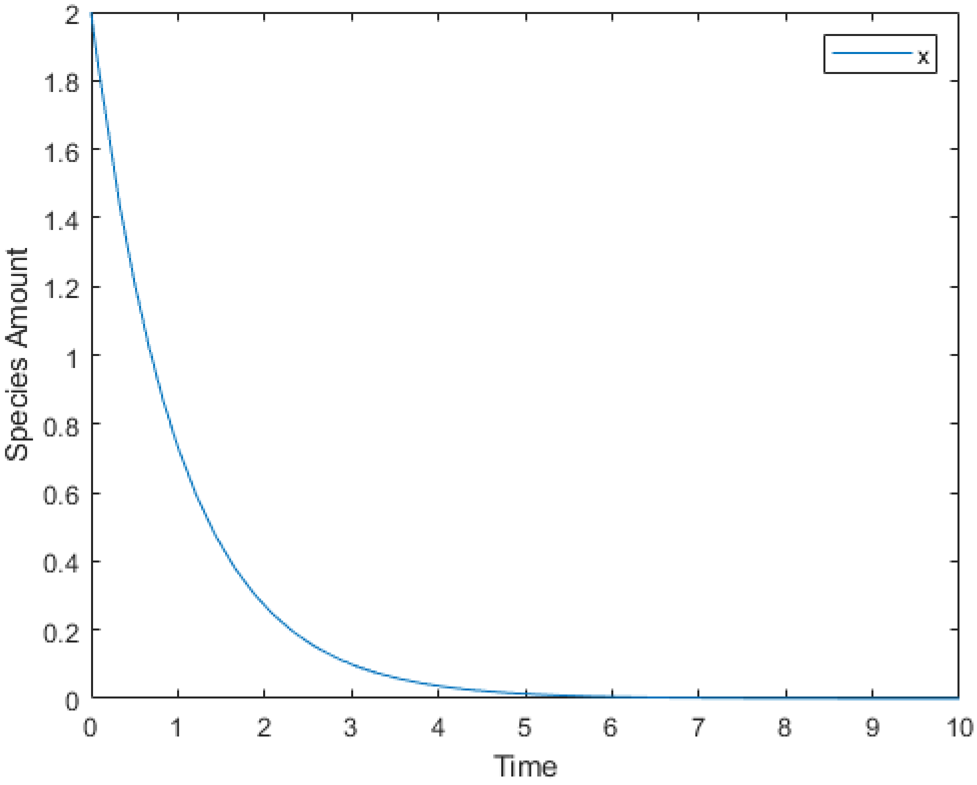
The analytical solution is:

$$C_t = C_0 * e^{-kt}$$

where C_t is the amount of species at time t, and C_0 is the initial amount. Use the following commands to set up a SimBiology model accordingly and simulate it.

```
m = sbiomodel('m');
c = addcompartment(m,'comp');
s = addspecies(m,'x','InitialAmount',2);
p = addparameter(m,'k','Value',1);
r = addrule(m,'x = -k * x','RuleType','rate');
[t,sd,species] = sbiosimulate(m);
```

```
plot(t,sd);
legend(species);
xlabel('Time');
ylabel('Species Amount');
```



If the amount of a species x is determined by a rate rule and x is also in a reaction, x must have its BoundaryCondition property set to true. For example, with a reaction a -> x and a rate rule $\frac{dx}{dt} = k*x$, set the BoundaryCondition property of species x to true so that a differential rate term is not created from the reaction. The amount of x is determined solely by a differential rate term from the rate rule. If the BoundaryCondition property is set to false, you will get the following error message such as Invalid rule variable 'x' in rate rule or reaction.

Create a Rate Rule to Define a Differential Rate Equation

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:

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

as a rate rule in SimBiology: C = vi - (vd*X*C)/(Kc + C) - kd*C

Create a Rate Rule for the Rate of Change That Is Determined by Another Species

This example shows how to create a rate rule where 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. Suppose you have a SimBiology model with three species (`a`, `b`, and `c`), one reaction (`a -> b`), and two parameters (`k1` and `k2`). The rate equation is defined as $b = -k_1 * a$, and rate rule is $dc/dt = k_2 * a$. The solution for the species in the reaction are:

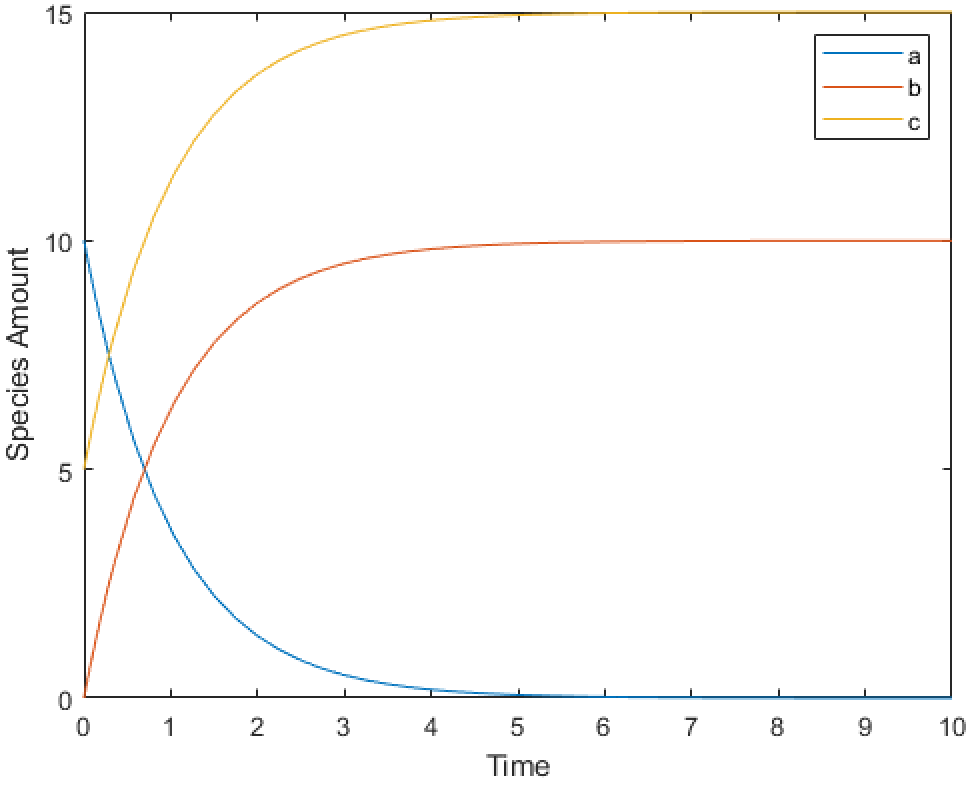
$\mathrm{a} = \mathrm{a_o e^{-k_1 t}}$, $\mathrm{b} = \mathrm{a_o(1 - e^{-k_1 t})}$.

Since the rate rule $dc/dt = k_2 * a$ is dependent on the reaction, $dc/dt = k_2(a_o e^{-k_1 t})$. The solution is:

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

Enter the following commands to set up a SimBiology model accordingly and simulate it.

```
m = sbiomodel('m');
c = addcompartment(m,'comp');
s1 = addspecies(m,'a','InitialAmount',10,'InitialAmountUnits','mole');
s2 = addspecies(m,'b','InitialAmount',0,'InitialAmountUnits','mole');
s3 = addspecies(m,'c','InitialAmount',5,'InitialAmountUnits','mole');
rxn = addreaction(m,'a -> b');
kl = addkineticlaw(rxn,'MassAction');
p1 = addparameter(kl,'k1','Value',1,'ValueUnits','1/second');
rule = addrule(m,'c = k2 * a','RuleType','rate');
kl.ParameterVariableNames = 'k1';
p2 = addparameter(m,'k2','Value',1,'ValueUnits','1/second');
[t,sd,species] = sbiosimulate(m);
plot(t,sd);
legend(species);
xlabel('Time');
ylabel('Species Amount');
```



See Also

More About

- “What is a SimBiology Model?” on page 2-2
- “Definitions and Evaluations of Reactions in SimBiology Models” on page 2-8
- “Events in SimBiology Models” on page 2-22
- “Component Usage” on page 2-43
- “Evaluation of Model Component Names in Expressions” on page 2-45

Events in SimBiology Models

In this section...

Overview

In SimBiology, an event is a discrete transition in value of a quantity or expression in a model. This discrete transition occurs when a customized condition becomes true. The condition can be a specific time and/or a time-independent condition. Such conditions are defined in an `Event object`.

Event Triggers

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

Typical event triggers are:

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

Note Currently, events cannot be triggered at time = 0. However, you can get the event to happen just after time = 0 by using `time > timeSmall` as the event trigger where `timeSmall` can be a tiny fraction of a second such as 1.0 picosecond.

Event Functions

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

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

Specifying Event Triggers

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

A trigger can contain the keyword `time` and relational operators to trigger an event that occurs at a specific time during the simulation. For example, `time >= x`. For more information see the `Trigger` property.

Use MATLAB syntax to write expressions for event triggers. Note that the expression must be a single MATLAB statement that returns a logical. No semicolon or comma is needed at the end of an 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 “Relational Operations” (MATLAB) and “Logical Operations” (MATLAB).

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 &) evaluates the first part of the expression for whether the statement is true or false, and skips evaluating the second statement if this statement is false.
`(time >= 5) || (speciesA < 1000)`	Execute the event when the following condition becomes true: Time is greater than or equal to `5`, or if `speciesA` is less than 1000.

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

Tip If UnitConversion is on and your model has any event, follow the recommendation below.

Non-dimensionalize any parameters used in the event `Trigger` if they are not already dimensionless. For example, suppose you have a trigger `x > 1`, where *x* is the species concentration in mole/liter. Non-dimensionalize *x* by scaling (dividing) it with a constant such as `x/x0 > 1`, where *x0* is a parameter defined as 1.0 mole/liter. Note that *x* does not have to have the same unit as the constant *x0*, but must be dimensionally consistent with it. For example, the unit of *x* can be picomole/liter instead of mole/liter.

Specifying Event Functions

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

Use MATLAB syntax to define expressions for event functions. The expression must be a single MATLAB assignment statement that includes =, or a cell array of such statements. No semicolon or comma is needed at the end of the expression.

Following are rules for writing expressions for event functions:

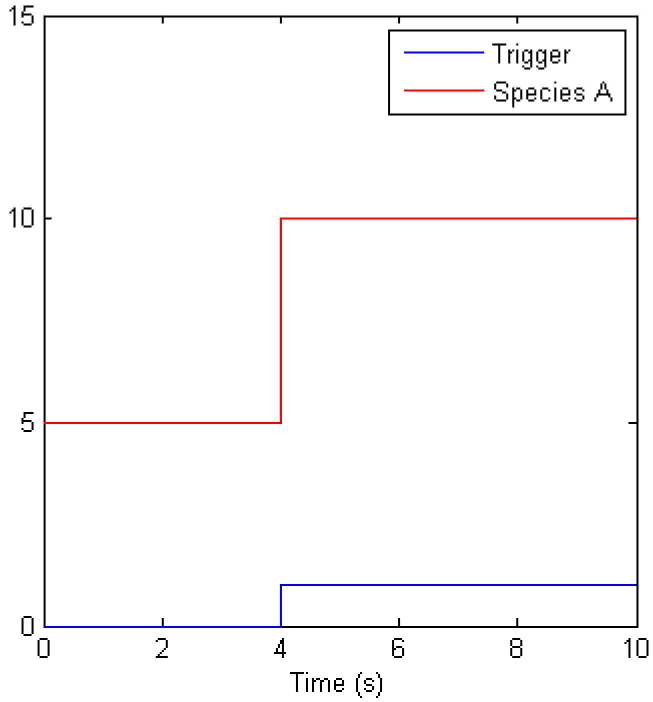
EventFcn	Explanation
speciesA = speciesB	When the event is executed, set the amount of speciesA equal to that of speciesB.
k = k/2	When the event is executed, halve the value of the rate constant k.
{'speciesA = speciesB','k = k/2'}	When the event is executed, set the amount of speciesA equal to that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

Simulation Solvers for Models Containing Events

To simulate models containing events, use a deterministic (ODE or SUNDIALS) solver or the stochastic ssa solver. Other stochastic solvers do not support events. For more information, see "Choosing a Simulation Solver" on page 4-6.

How Events Are Evaluated

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



At `time = 4 s` the trigger becomes true and the event executes. In the previous figure 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 s`. In practice there is a very minute delay (for example you might notice that the event is executed at `time = 4.00001 s`). Thus, you must specify that the trigger can become true at or after `4s`, which is `time >= 4 s`.

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

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

The trigger is evaluated at every time step to check whether the condition specified in the trigger transitions from false to true. The solver detects and tracks falling edges, which is when the trigger becomes false, so if another rising edge is encountered, the event is reexecuted. If a trigger is already true before a simulation starts, then the event does not execute at the start of the simulation. The event is not executed until the solver encounters a rising edge. Very rarely, the solver might miss a rising edge. An example of this is when a rising edge follows very quickly after a falling edge, and the step size results in the solver skipping the transition point.

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

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

Evaluation of Simultaneous Events

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

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 both 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, 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 might or might not occur simultaneously.

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

Evaluation of Multiple Event Functions

Consider an event function in which you specify that the value of a model component (`SpeciesB`) depends 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 this 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 next example, 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 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. Here is an example of cyclical events. This example assumes that `Species B <= 4` at the start of the cycle.

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

Using Events to Address Discontinuities in Rule and Reaction Rate Expressions

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

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

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

See Also

More About

- “What is a SimBiology Model?” on page 2-2
- “Definitions and Evaluations of Reactions in SimBiology Models” on page 2-8
- “Definitions and Evaluations of Rules in SimBiology Models” on page 2-13
- “Component Usage” on page 2-43
- “Evaluation of Model Component Names in Expressions” on page 2-45

Variants in SimBiology Models

A variant stores alternate values of model parameters and initial conditions. You can use variants to evaluate model behavior under different experimental or initial conditions, without having to change the existing values or create additional models with the new values.

A variant lets you store an alternate value for any of the following model elements:

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

Simulating using a variant does not alter the model original values. The values specified in the variant are temporarily applied to the model during simulation. You can permanently replace the values in your model with the values stored in the variant object by committing it to the model. When you use multiple variants during a simulation, and there are duplicate specifications for a property value, the last occurrence for the property value in the array of variants is used during simulation.

Creating Variants Programmatically

There are two ways to create variants or add variants to a model. To create a standalone variant that is not attached to any model, use `sbiovariant`. To add a variant to an existing model, use `addvariant`. Use the `commit` function to replace the values in your model with the variant values permanently.

For illustrated examples of using variants, see the following.

- “Simulate Biological Variability of the Yeast G Protein Cycle Using the Wild-Type and Mutant Strains” on page 2-34
- “Simulate Model of Glucose-Insulin Response with Different Initial Conditions” on page 3-18

Creating Variants Graphically

You can interactively create and add variants using the SimBiology app. For details, see “Diagram View” on page 1-4.

See Also

`Variant object` | `addcontent` | `addvariant` | `commit` | `sbiovariant`

More About

- “SimBiology Apps”

Doses in SimBiology Models

Doses let you increase the amount of a species in a SimBiology model during simulation, either at specific time points or regular intervals. For example, you can use a dose object to model an instantaneous supply of a drug regimen during the simulation of a model. The increase in the amount of a species occurs only during simulation and does not alter the species value permanently (that is, the value in the model is not changed).

Representing Doses

There are two types of dose objects.

- `ScheduleDose object` — Applies a dose to a single species at a predefined list of time points
- `RepeatDose object` — Applies a dose to a single species at regular intervals

SimBiology dose objects support the following dosing types.

Dosing Strategy	**Description**	**Dose Object Properties Configuration**
Bolus	Instantaneous increase in the amount of drug in the compartment	To create a bolus dose, set the Amount and TargetName properties of a dose object. You might also need to configure other properties such as RepeatCount, Interval, or scheduled dose times (Time) if you are applying a series of doses. For details on these properties, see `ScheduleDose object` and `RepeatDose object` .
Infusion	Increase of the drug at a fixed rate over a period of time, which is calculated from the dose amount	Unlike a bolus dose, you also need to specify the infusion rate (Rate property) of the dose object.
Zero-order	Increase of the drug at a fixed rate calculated from the dose amount and dose duration	Unlike a bolus dose, you also need to create a zero-order duration parameter and specify the duration parameter name (`DurationParameterName`property) of the dose object.
First-order	Increase of the drug via first-order absorption kinetics	Unlike bolus, infusion, or zero-order, you need to create an additional reaction for the drug absorption.

Creating Doses Programmatically

There are two common ways to create dose objects using the command-line interface. One way is to create a dose object using the `sbiodose` or `adddose` function. Another is to create dose objects automatically from data containing dosing information. This first approach is useful when you want to explore different dosing strategies through simulation. The second approach is useful if you already have a data set with dosing information and plan to use this dosing information in your simulation or parameter estimation.

Create a Dose Object Using sbiodose or adddose

sbiodose creates a standalone dose object that is not attached to any model. You can apply a standalone dose to different models during simulation by specifying it as a dosing argument for sbiosimulate, or attach it to any model using adddose. You can also use it during parameter estimation using sbiofit or sbiofitmixed.

adddose creates a dose object and adds it to a model. You must set its Active property to true to apply the dose to the model during simulation.

The following examples show how to add a dose object to a one-compartment PK model using sbiodose and set up the dose properties manually. Alternatively, you can use the built-in PK models with different dosing types. For details, see “Create Pharmacokinetic Models” on page 5-13.

Dosing Strategy	Example
Bolus	“Add a Series of Bolus Doses to a One-Compartment Model”
Infusion	“Add an Infusion Dose to a One-Compartment Model”
Zero-order	“Increase Drug Concentration in a One-Compartment Model via Zero-order Dosing”
First-order	“Increase Drug Concentration in a One-Compartment Model via First-order Dosing”

Create Dose Objects from Dosing Data

If you already have dosing data for one or more subjects or patients that you would like to use in your parameter estimation, first create a groupedData object from your data set. Use createDoses function to automatically generate an array of dose objects. You can then use the dose array during parameter estimation using sbiofit or sbiofitmixed. For a complete workflow, see “Modeling the Population Pharmacokinetics of Phenobarbital in Neonates”.

Creating Doses Graphically

You can interactively create and add doses using the SimBiology app. For details, see “Diagram View” on page 1-4.

Parameterized and Adaptive Doses

You can specify some of the properties of RepeatDose and ScheduleDose objects by using model parameters. This parameterization of dose properties gives you more flexibility in modeling different dosing applications, such as scaling the dose amount by body weight.

RepeatDose properties that you can parameterize are: Amount, Rate, Interval, StartTime, RepeatCount, LagParameterName, and DurationParameterName. ScheduleDose properties that can be parameterized are LagParameterName and DurationParameterName. You can set these RepeatDose properties, except LagParameterName and DurationParameterName, to either a numeric value or the name of a model-scoped parameter (as a character vector or string).

You can make doses adaptive to events, such as increasing the dose amount when the drug concentration drops below some threshold. This adaptive feature of doses is useful for doses that are not instantaneous. Consider an IV infusion for a drug being added at a fixed rate over a fixed duration. If an event modifies a dose parameter while this dose is in progress, you have two options:

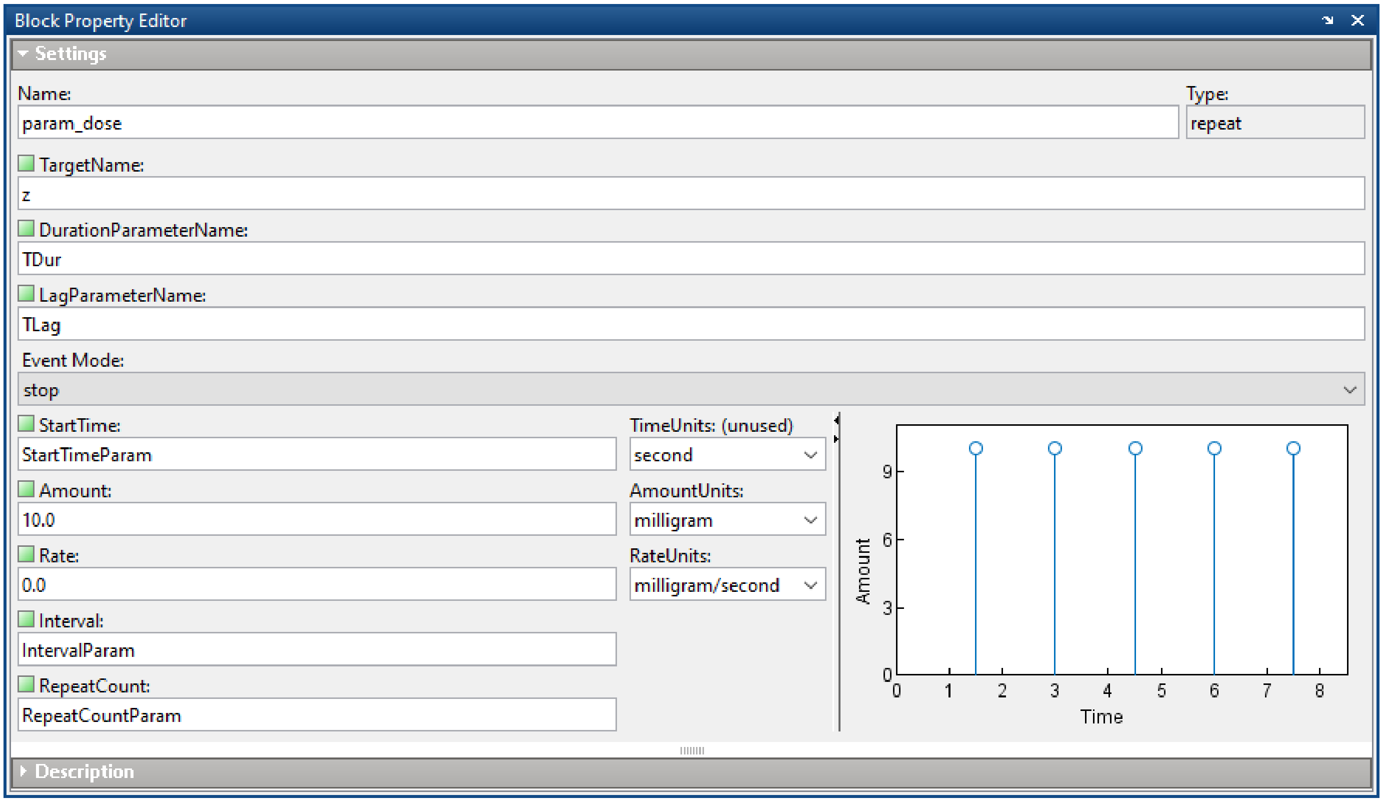
- Stop the ongoing dose if any relevant parameter values change by setting the `EventMode` property of the dose object to `'stop'`.
- Continue the ongoing dose to completion, and updated parameter values affect only subsequent doses by setting `EventMode` to `'continue'`.

For details, see the EventMode property. For illustrated examples, see “Scale Dose Amount by Body Weight” and “Change Dose Behavior In Response to Changes in Model Parameters”.

Units Validation on Parameterized Dose Properties

If you parameterize a dose property and enable dimensional analysis, the unit of the dose property (dose unit) is validated. The dose unit is valid either if it is empty or if it exactly matches the unit of the parameter. If the dose unit is invalid, SimBiology issues a warning in the command line and uses the unit of the parameter instead. To remove the warning, set the dose unit to empty (`''`) or to the same unit as the parameter unit.

In the SimBiology app, the text `(unused)` appears next to the corresponding unit field in the **Block Property Editor** or **Doses** tab, meaning that SimBiology is using the parameter unit instead of the dose unit during simulation. For example, this figure shows the details of the repeat dose object *param_dose* in the **Block Property Editor**. The **TimeUnits** option is designated as `unused`. Instead, SimBiology uses the unit of the parameter *StartTimeParam* during simulation. The **Amount** and **Rate** properties are not parameterized, and hence the selected **AmountUnits** and **RateUnits** options are used. To see the units of referenced parameters, click the indicator next to the corresponding property. The **TimeUnits** option corresponds to both the **StartTime** and **Interval** properties.



Block Property Editor

Settings

Name: param_dose
Type: repeat

🟩 TargetName: z

🟩 DurationParameterName: TDur

🟩 LagParameterName: TLag

Event Mode: stop

🟩 StartTime: StartTimeParam
TimeUnits: (unused) second

🟩 Amount: 10.0
AmountUnits: milligram

🟩 Rate: 0.0
RateUnits: milligram/second

🟩 Interval: IntervalParam

🟩 RepeatCount: RepeatCountParam

Description

Simulation Solvers for Models Containing Doses

To simulate models containing doses, use a deterministic (ODE or SUNDIALS) solver. Stochastic solvers do not support doses. For details, see “Choosing a Simulation Solver” on page 4-6.

See Also

`RepeatDose object` | `ScheduleDose object` | `adddose` | `sbiodose`

More About

- “SimBiology Apps”

Simulate Biological Variability of the Yeast G Protein Cycle Using the Wild-Type and Mutant Strains

This example shows how to create and apply a variant to the G protein model of a wild-type strain. The variant represents a parameter value for the G protein model of a mutant strain. Thus, when you simulate the model without applying the variant, you see results for the wild type strain, and when you simulate the model with the variant, you see results for the mutant strain. This example uses the model described in Model of the Yeast Heterotrimeric G Protein Cycle on page B-14.

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

For information on variants, see “Variants in SimBiology Models” on page 2-29.

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

```
sbioloadproject gprotein
```

You can create a variant of the original model by specifying a different parameter value for the `kGd` parameter of the model. First, add a variant to the `m1` model object.

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

Next, add a parameter `kGd` with a value of `0.004` to the variant object `v1`.

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

Simulate the wild type model.

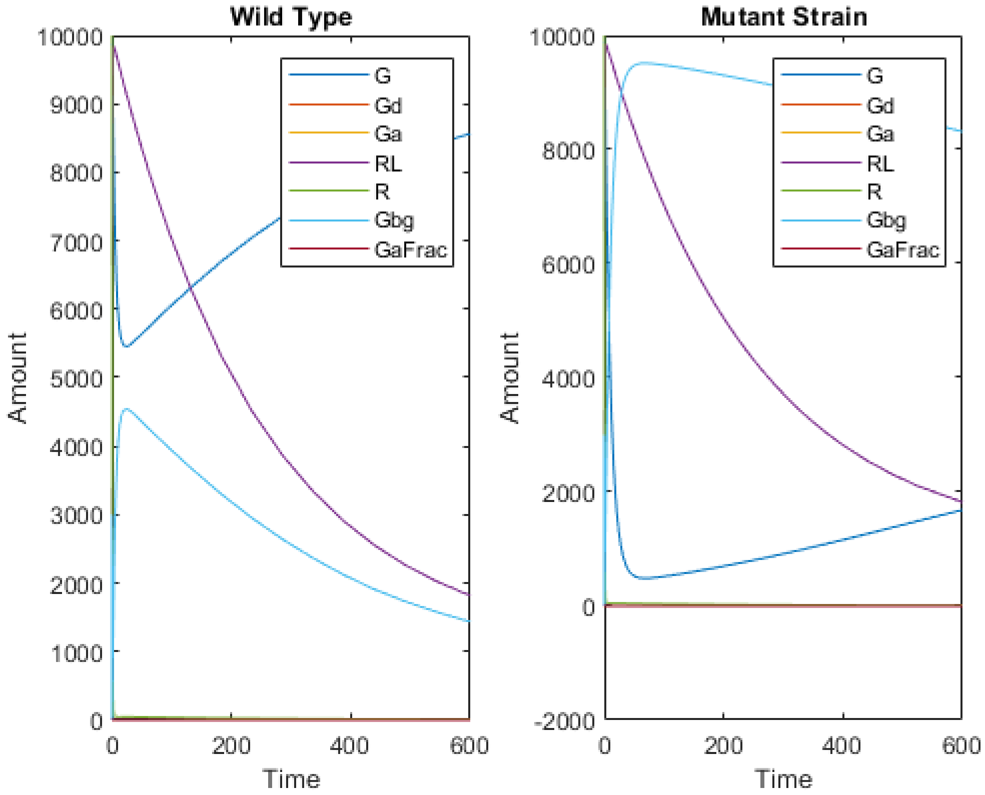
```
[t,x,names] = sbiosimulate(m1);
```

Simulate the mutant strain model by applying the variant.

```
[tV,xV,names] = sbiosimulate(m1,v1);
```

Plot and compare the simulated results.

```
subplot(1,2,1)
plot(t,x);
legend(names);
xlabel('Time');
ylabel('Amount');
title('Wild Type');

subplot(1,2,2)
plot(tV,xV);
legend(names);
xlabel('Time');
ylabel('Amount');
title('Mutant Strain');
```


Wild Type
Mutant Strain
Amount
Time
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
-2000
200
400
600
G
Gd
Ga
RL
R
Gbg
GaFrac

Create and Simulate a Model with a Custom Function

This example shows how to create a custom function and incorporate it in model simulation.

Overview

Prerequisites for the Example

This example assumes you have a working knowledge of:

- MATLAB app
- Creating and saving MATLAB programs

About the Example Model

This example uses the model described in Model of the Yeast Heterotrimeric G Protein Cycle on page B-14.

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

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

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

Assumptions of the Example

This example assumes that:

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

About the Example

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

About Using Custom Functions in SimBiology Expressions

You can use custom functions in:

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

The requirements for using custom functions in SimBiology expressions are:

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

Tip If your rule or reaction rate expression is not continuous and differentiable, see “Using Events to Address Discontinuities in Rule and Reaction Rate Expressions” on page 2-28 before simulating your model.

Create a Custom Function

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

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

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

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

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

3. Save the file (name the file `inhibvalex.m`) in a directory that is on the MATLAB search path, or to a directory that you can access.
4. If the location of the file is not on the MATLAB search path, change the working directory to the file location.

Load the Example Model

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

```
sbioloadproject gprotein
```

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

Add the Custom Function to the Example Model

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

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

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

   The `time` input is a SimBiology keyword recognized as simulation time
2. Create the two parameters used by the `rule1` rule and assign values to them:

   ```
   p1 = addparameter(m1, 'Cpo', 250);
   p2 = addparameter(m1, 'Kel', 0.01);
   ```
3. Create the species used by the `rule1` rule:

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

Define a Rule to Change Parameter Value

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

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

   ```
   p3 = sbioselect(m1, 'Type', 'parameter', 'Name', 'kGd');
   p3.ConstantValue = false;
   ```
2. Add a repeated assignment rule to the model to define how the `kGd` parameter is affected by the `Inhib` species.

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

Add an Event to Reset the Solver at a Discontinuity

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

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

Simulate the Modified Model

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

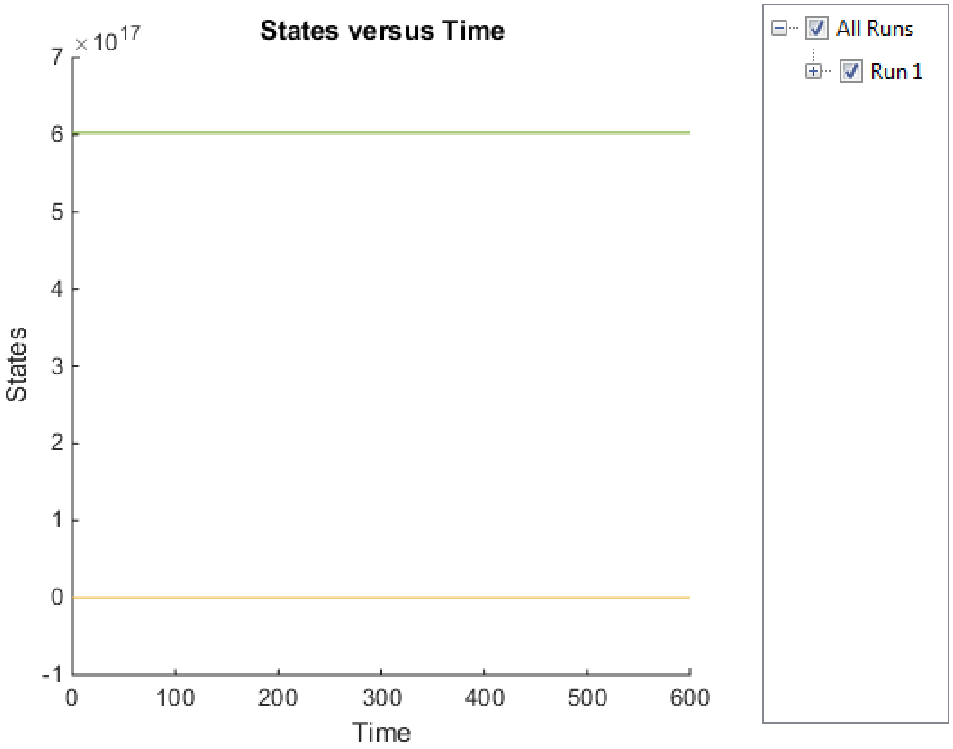
```
cs = getconfigset(m1);
cs.RuntimeOptions.StatesToLog = 'all';
```

2 Simulate the model.

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

3 Plot the results.

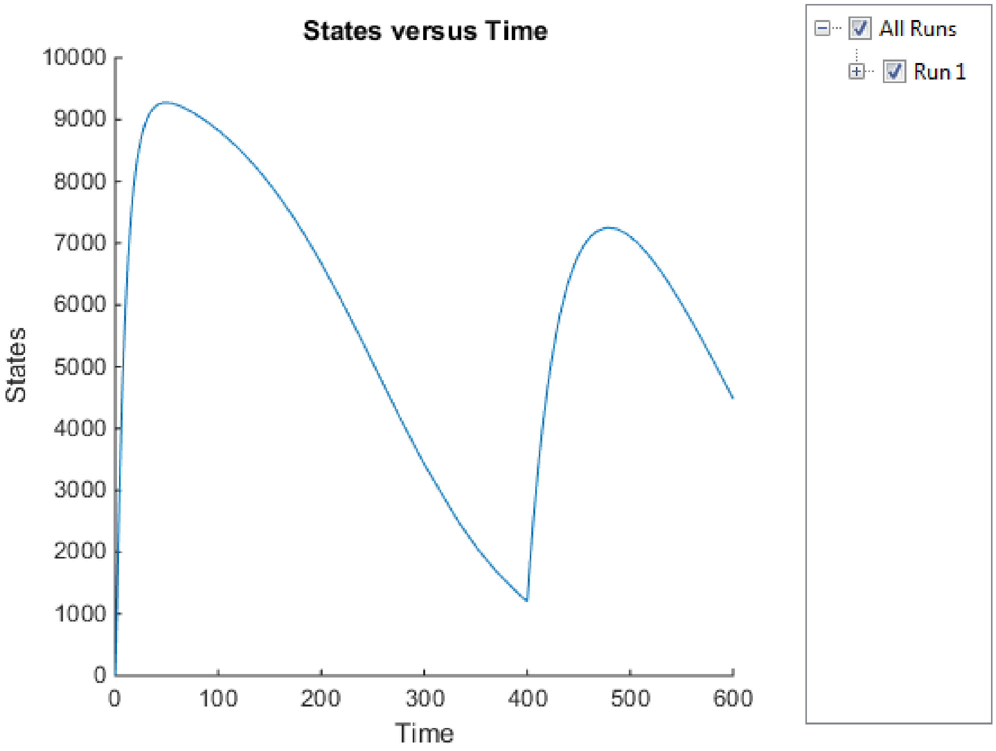
```
sbioplot(simDataObj);
```



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

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

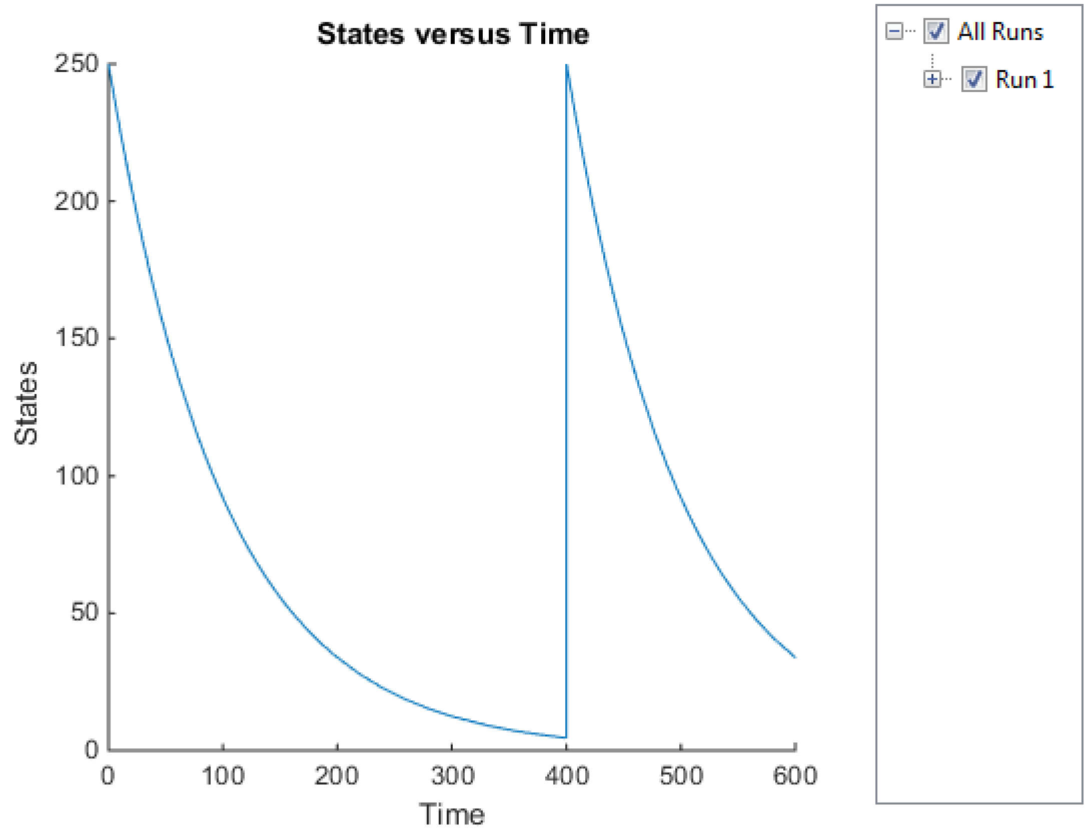
```
GaSimDataObj = selectbyname(simDataObj,'Ga');
sbioplot(GaSimDataObj);
```



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

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

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



See Also

`addpath` | `cd` | `function`

More About

- “Change Folders on the Search Path” (MATLAB)

View Model Equations

You can view the system of equations that SimBiology creates when you build a model using reactions, rules, events, variants, and doses. Viewing model equations is useful for:

- Publishing purposes
- Model debugging

For details, see the `getequations` method of a `Model` object or the “Equations View” on page 1-21 from the SimBiology app.

Component Usage

SimBiology lets you find species, parameters, and compartments that are not used in a model. You can also query how a particular quantity is used by other expressions such as a parameter being used as a reaction rate constant or species being used in an event.

From the command line, use the `findUnusedComponents` function to look for unused model components and the `findUsages` function to see how a component is used in expressions. From the SimBiology app on page 1-4, select **Remove Unused** on the **Model** tab to delete unused quantities. To look for usages of a quantity, select **Show Usages**.

Species Usage

A species is used when it is referenced in any of the following properties of other components:

- The Reaction or ReactionRate property of a `Reaction object`,
- The ParameterVariableNames or SpeciesVariableNames property of a `KineticLaw object`,
- The Rule property of a `Rule object`,
- The Trigger or EventFcns property of an `Event object`,
- The `Expression` property of an `observable` object, and
- The TargetName property of a `ScheduleDose object` or `RepeatDose object`.

Parameter Usage

A parameter is used when it is referenced in any of the following properties of other components:

- The ReactionRate property of a `Reaction object`,
- The ParameterVariableNames property of a `KineticLaw object`,
- The Rule property of a `Rule object`,
- The Trigger or EventFcns property of an `Event object`,
- The `Expression` property of an `observable` object,
- The Content property of a `Variant object`, and
- The `DurationParameterName` or `LagParameterName` property of a `ScheduleDose object` or `RepeatDose object`.
- The Amount, Rate, Interval, StartTime, or RepeatCount property of a `RepeatDose object`.

Compartment Usage

A compartment is used when it is referenced in any of the following properties of other components:

- The Parent property of a `Species object`,
- The Owner property of a `Compartment object`,
- The ReactionRate property of a `Reaction object`,
- The ParameterVariableNames property of a `KineticLaw object`,
- The Rule property of a `Rule object`,

- The Trigger or EventFcns property of an `Event object`,
- The `Expression` property of an `observable` object, and
- The Content property of a `Variant object`.

Observable Usage

An observable object is used when it is referenced in the `Expression` property of another observable.

Unit and UnitPrefix Usage

A unit or unit prefix is used when it is referenced in any of the following properties of other components:

- The Composition property of all units in the `BuiltInLibrary` and `UserDefinedLibrary`,
- The `Units` property of a species, compartment, parameter, or observable object in the specified models,
- The TimeUnits property of all specified doses,
- The AmountUnits property of all specified doses, and
- The RateUnits property of all specified doses.

Abstract Kinetic Law Usage

An abstract kinetic law object *aklObj* can only be used by a `Reaction object` *robj*. It is used when:

- The KineticLaw property of the reaction object is not empty, *and*
- `robj.KineticLaw.KineticLawName` matches the name of the abstract kinetic law `aklObj.Name`.

See Also

`findUnusedComponents` | `findUsages(AbstractKineticLaw)` | `findUsages(observable)` | `findUsages(species,parameter,compartment)` | `findUsages(unit,unitprefix)`

More About

- “What is a SimBiology Model?” on page 2-2
- “SimBiology Apps”
- “Model Views in SimBiology App” on page 1-4
- “Evaluation of Model Component Names in Expressions” on page 2-45

Evaluation of Model Component Names in Expressions

SimBiology model components on page 2-2 include quantities and expressions. You can refer to model quantities (compartments, species, and parameters) and observables by their names in an expression, such as a reaction or an assignment equation. Follow these guidelines when you name model components or referencing their names in expressions. When evaluating a name that matches different quantities, SimBiology resolves it by following precedence rules.

Guidelines for Naming Model Components

- Model, parameter, and observable names cannot contain brackets `[ ]` and cannot be empty, the word `time`, or all whitespace.
- Compartment, species, and observable names cannot contain the characters `->`, `<->`, `[` or `]` and cannot be empty, the word `null`, or the word `time`. However, a name can contain the words `null` and `time` within the name, such as `nullDrug`.
- Reaction, event, and rule names cannot contain brackets `[ ]` and cannot be the word `time`.

Guidelines for Referencing Names in Expressions

- If the quantity name is not a valid MATLAB variable name, you must enclose the name in brackets when referring to it in an expression. For example, if the name of a species is *DNA polymerase+*, write `[`*`DNA polymerase+`*`]`.
- If you have multiple species with the same name in different compartments, you must qualify the name by referring to the name of the compartment that contains the species. For example, the qualified name *`nucleus`*`.[`*`DNA polymerase+`*`]` refers to the *DNA polymerase+* species that resides in the *nucleus* compartment.

Precedence Rules for Evaluating Quantity Names

If a name referenced in an expression matches multiple quantities or `observable` objects, SimBiology evaluates the expression using precedence rules. The rules depend on whether the name is referenced in a reaction or other expressions that are not reactions.

For Reactions

When a reaction refers to a name that matches different quantities, SimBiology evaluates the name as the first quantity with a matching name in this order: species, parameter scoped to the reaction, compartment, or parameter scoped to the model.

For Other Expressions

There are different types of expressions that are not reactions, namely rules, events, and observables. Rules include initial and repeated assignment equations, algebraic equations, and differential rate equations. An event contains expressions that represent an event trigger and one or more event functions to model discrete transitions in the values of quantities or expressions in the model. An `observable` object is a mathematical expression that lets you perform post-simulation calculations.

When a nonreaction expression refers to a name that matches different quantities, SimBiology evaluates the name as the first quantity with a matching name in this order: species, compartment, or parameter scoped to the model. An active observable expression can reference another active observable object by its name as long as there are no circular dependencies (or algebraic loops).

See Also

Name

More About

- “What is a SimBiology Model?” on page 2-2
- “Definitions and Evaluations of Reactions in SimBiology Models” on page 2-8
- “Definitions and Evaluations of Rules in SimBiology Models” on page 2-13
- “Events in SimBiology Models” on page 2-22
- “Component Usage” on page 2-43

Structural Analysis

Model Verification

In this section...

What is Model Verification?

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

Model verification checks many aspects of the model including:

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

When to Verify a Model

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

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

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

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

Verifying That a Model Has No Warnings or Errors

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

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

Model Verification Example

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

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

2. Verify the model to check for warnings and errors:

   ```
   verify(model)
   ```

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

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

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

4. Verify the model again:

   ```
   verify(model)
   ```

Conserved Moiety Determination

In this section...

Introduction to Moiety Conservation

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

Consider this simple network:

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

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

Algorithms for Conserved Cycle Calculations

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

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

- `'qr'` — `sbioconsmoiety` uses an algorithm based on `QR` factorization. From a numerical standpoint, this is the most efficient and reliable approach.
- `'rreduce'` — `sbioconsmoiety` uses an algorithm based on row reduction, which yields better numbers for smaller models. This is the default.
- `'semipos'` — `sbioconsmoiety` returns conservation relations in which all the coefficients are greater than or equal to zero, permitting a more transparent interpretation in terms of physical quantities.

For larger models, the `QR`-based method is recommended. For smaller models, row reduction or the semipositive algorithm may be preferable. For row reduction and `QR` factorization, the number of conservation relations returned equals the row rank degeneracy of the model object's stoichiometry matrix. The semipositive algorithm can return a different number of relations. Mathematically speaking, this algorithm returns a generating set of vectors for the space of semipositive conservation relations.

In some situations, you may be interested in the dimensional reduction of your model via conservation relations. Recall the simple model, presented in “Introduction to Moiety Conservation” on page 3-4, that contained the conserved cycle A + B + C. Given A and B, C is determined by the conservation relation; the system can be thought of as having only two dynamic variables rather than three. The 'link' algorithm specification caters to this situation. In this case, sbioconsmoiety partitions the species in the model into independent and dependent sets and calculates the dependence of the dependent species on the independent species.

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

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

$$\mathrm{N} = \begin{pmatrix} NR \\ ND \end{pmatrix}$$

where R in the independent submatrix N_R denotes 'reduced'; the (n – k)-by-k link matrix L0 is defined so that $\mathrm{N}_\mathrm{D} = \mathrm{L0}*\mathrm{N}_\mathrm{R}$. 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.

More About

For examples of determining conserved moieties, see:

- “Determining Conserved Moieties” on page 3-6
- “Finding Conserved Quantities in a Pathway Model”

Determining Conserved Moieties

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

```
sbioloadproject Goldbeter_Mitotic_Oscillator_with_reactions
```

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

2 Display the species information:

```
m1.Compartments.Species

SimBiology Species Array

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

3 Display the reaction information:

```
m1.Reactions

SimBiology Reaction Array

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

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

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

g =

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


sp =

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

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

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

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

cons_rel =

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

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

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

7 Show the list of independent species:

```
SI

SI =

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

8 Show the list of dependent species:

```
SD

SD =

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

9 Show the link matrix relating `SD` and `SI` by converting the `L0` output from a sparse matrix to a `full` matrix:

```
L0_full = full(L0)

L0_full =

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

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

```
NR_full = full(NR)

NR_full =

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

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

```
ND_full = full(ND)

ND_full =

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

Determining the Adjacency Matrix for a Model

In this section…

What Is an Adjacency Matrix?

An adjacency matrix lets you easily determine:

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

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

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

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

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

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

Retrieving an Adjacency Matrix for a Model

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

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

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

2 Get the adjacency matrix for `m1`:

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

M =

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


Headings = 

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

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

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

```
M_full =

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

Determining the Stoichiometry Matrix for a Model

In this section...

What Is a Stoichiometry Matrix?

A stoichiometry matrix lets you easily determine:

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

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

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

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

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

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

Retrieving a Stoichiometry Matrix for a Model

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

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

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

2 Get the stoichiometry matrix for `m1`:

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

M =

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


objSpecies =

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


objReactions =

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

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

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

```
M_full =

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

Selecting Absolute Tolerance and Relative Tolerance for Simulation

In this section...

SimBiology uses AbsoluteTolerance and RelativeTolerance to control the accuracy of integration during simulation. Specifically, AbsoluteTolerance is used to control the largest allowable absolute error at any step during simulation. It controls the error when a solution is small. Intuitively, when the solution approaches 0, `AbsoluteTolerance` is the threshold below which you do not worry about the accuracy of the solution since it is effectively 0. `RelativeTolerance` controls the relative error of a single step of the integrator. Intuitively, it controls the number of significant digits in a solution, except when it is smaller than the absolute tolerance, and $-\log_{10}(RelativeTolerance)$ is the number of correct digits.

Algorithm

At each simulation step `i`, the solver estimates the local error `e` in the state `j` of the simulation. The solver reduces the size of time step `i` until the error of the state satisfies:

$$|e(i,j)| \leq \max(RelativeTolerance * |y(i,j)|, AbsoluteTolerance(i,j))$$

Thus at state values of larger magnitude, the accuracy is determined by `RelativeTolerance`. As the state values approach zero, the accuracy is controlled by `AbsoluteTolerance`.

The correct choice of values for `RelativeTolerance` and `AbsoluteTolerance` varies depending on the problem. The default values may work for first trials of the simulation. As you adjust the tolerances, consider that there are trade-offs between speed and accuracy:

- If the simulation takes too long, you can increase (or loosen) the values of `RelativeTolerance` and `AbsoluteTolerance` at the cost of some accuracy.
- If the results seem inaccurate, you can decrease (or tighten) the relative tolerance values by dividing with 10^N, where N is a real positive number. But this tends to slow down the solver.
- If the magnitude of the state values is high, you can decrease the relative tolerance to get more accurate results.

Absolute Tolerance Scaling

How SimBiology uses `AbsoluteTolerance` to determine the error depends on whether the AbsoluteToleranceScaling property is enabled. By default, `AbsoluteToleranceScaling` is enabled which means each state has its own absolute tolerance that may increase over the course of simulation:

$$AbsoluteTolerance(i,j) = CSAbsTol * Scale(i,j)$$

`CSAbsTol` is the `AbsoluteTolerance` property defined in `SolverOptions` of the active configuration set object.

For a state that has a nonzero initial value, the scale is the maximum magnitude over the state, as seen over the simulation thus far:

$$Scale(i, j) = \max(|y(1{:}i, j)|)$$

For a state that has an initial value of zero, the scale is estimated as the state value after taking a trial step of size AbsoluteToleranceStepSize using the Euler method. Let us call this value `ye(j)`. Then:

$$Scale(i, j) = \max(|[ye(j); y(2{:}i, j)]|)$$

If an initial state is zero and has no dynamic at time = 0, then:

$$AbsoluteTolerance(i, j) = CSAbsTol$$

Doses, events, and initial assignment rules at simulation time = 0 are not considered when calculating absolute tolerance scaling.

See Also

More About

- “Model Simulation” on page 4-2
- “Choosing a Simulation Solver” on page 4-6
- “Ordinary Differential Equations” (MATLAB)

Troubleshooting Simulation Problems

SimBiology uses ODE solvers for model simulation on page 4-2. Solver errors can cause simulation problems. Many solver errors relate to the stiffness of the model and the relative and absolute tolerances. As a result, the simulation can take long. You might also see one of the following error messages, indicating that the solver is not able to solve the problem within the tolerances.

- Integration tolerance not met
- CVODES returned -4 from module CVODES function CVode: At t = `...` and h = `...` the corrector convergence test failed repeatedly or with |h| = hmin.

You might also see one or more of the following warning messages, which are precursors to potential solver tolerance issues.

- The right-hand side of the system of SimBiology ODEs results in complex numbers. The imaginary part of the result will be ignored.
- The right-hand side of the system of SimBiology ODEs results in infinite or NaN values. This usually indicates a modeling error and can lead to solver integration errors.
- The repeated assignment rules of the SimBiology model result in complex numbers. The imaginary part of the result will be ignored.

Tips for Solving Simulation Problems

To fix the simulation problems that solver errors can cause, try the following troubleshooting tips.

Improve Performance by Acceleration

You can accelerate the simulation by converting the model to compiled C code, which executes faster. For details, see “Accelerating Model Simulations and Analyses” on page 4-85. If the simulation is still slow after acceleration, there might be solver tolerance issues. Try the remaining tips without accelerating the model.

Debug the Model Using MaximumNumberOfLogs and MaximumWallClock

MaximumNumberOfLogs and MaximumWallClock are some of the configuration options you can use to stop the simulation just before the error happens. Then you can check for unusual simulated values, such as negative species amounts.

For instance, set `MaximumNumberOfLogs` to `1` to get the values of the model immediately after applying initial and repeated assignment rules. If you set the value to `2`, and the simulation fails with the integration error, then it probably indicates an error with the assignment rules.

While varying `MaximumNumberOfLogs`, simulate the model repeatedly using the same conditions that produce the error. The model might simulate without error until you reach a certain value of `MaximumNumberOfLogs`. Then check the simulated values at the final simulation time. If you see negative values for certain states, such as negative species amounts, examine the expressions in your model that can affect those states. Update the expressions to account for possible issues, such as negative values or division-by-0, by either rearranging the equations and/or inserting some protections, such as `max(0,x)` or `max(eps,x)`, where *x* is the variable that is responsible for the error.

Alternatively, you can look at the equations view of the model to check the initial conditions, such as species amounts and parameter values at simulation time = 0, and mathematical expressions of the model. For details, see getequations or "Equations View" on page 1-21.

Check the ODE Solver

If your model is stiff, and you have selected an inappropriate solver, the step size taken by the solver might be forced down to an unreasonably small level compared to the interval of integration. Make sure that you have selected either `ode15s` or `sundials` as your solver for stiff ODEs.

Disable AbsoluteToleranceScaling

Turn off AbsoluteToleranceScaling.

SimBiology uses AbsoluteTolerance and RelativeTolerance to control the accuracy of integration during simulation. Specifically, `AbsoluteTolerance` controls the largest allowable absolute error at any step during simulation.

When `AbsoluteToleranceScaling` is enabled (by default), each state has its own absolute tolerance that can increase over the course of simulation. Sometimes the automatic scaling is inadequate for models that have kinetics at largely different scales. For example, the reaction rate of a reaction can be in the order of 10^{22}, while another is `0.1`. By turning off `AbsoluteToleranceScaling`, you might be able to simulate the model.

Loosen Tolerances

If the simulation tolerance error still exists after disabling AbsoluteToleranceScaling, try loosening the relative and absolute tolerances.

Set `RelativeTolerance` to 10^{-m+1}, where m is the number of significant digits desired in the solution X. If X has multiple scales, start with using the smaller X and increase from there if the tolerance is not met.

Set `AbsoluteTolerance` to a value of X that is negligibly small for your problem. Similarly, start from the smaller X and increase from there.

For details, see "Selecting Absolute Tolerance and Relative Tolerance for Simulation" on page 3-13.

Set MassUnits and AmountUnits

The MassUnits and AmountUnits properties define the appropriate mass or amount unit that SimBiology uses internally during model simulation when UnitConversion is on. It is recommended that you use the default unit (`<automatic>`) but in some edge cases, you may need to change it.

Change `MassUnits` or `AmountUnits` to a unit so that the simulated values are not too large (that is, greater than 10^{6}) or too small (that is, smaller than 10^{-6}).

Suppose that you have a model with a state that takes on values around 10^{-12} moles for the entire simulation. It might be appropriate to set `AmountUnits` to `picomole`. In this case, the internal simulation values would be around `1`, instead of around 10^{-12} as in the default case.

How to Change Solver Options and Simulation Options

Solver and simulation options are stored in the configuration set object (`configset object`) of the model. Solver options contain settings such as relative and absolute tolerances. Simulation options are settings such as `MaximumNumberOfLogs` and `MaximumWallClock`. Depending on whether you are using the command line or graphical interface, the way to access and change the options differs.

Using the Command line

To access and change the SolverOptions, use the following commands, where *m1* is a SimBiology model.

```
configset = getconfigset(m1);
solverOpts = configset.SolverOptions;
solverOpts.AbsoluteTolerance = 1e-5;
solverOpts.RelativeTolerance = 1e-5;
solverOpts.AbsoluteToleranceScaling = false;
```

Access simulation options directly from the `configset object`.

```
configset = getconfigset(m1);
configset.MaximumNumberOfLogs = 1;
configset.MaximumWallClock = 10;
```

Using the Graphical Interface

If you are using the **SimBiology Model Analyzer** app, you can access the options by clicking **Simulation Settings** from the **Home** tab.

See Also

`getequations` | AbsoluteTolerance | `Configset object` | MaximumNumberOfLogs | MaximumWallClock | RelativeTolerance | SolverOptions

More About

- “Model Simulation” on page 4-2
- “Selecting Absolute Tolerance and Relative Tolerance for Simulation” on page 3-13
- “Equations View” on page 1-21

Simulate Model of Glucose-Insulin Response with Different Initial Conditions

This example shows how to simulate the glucose-insulin responses for the normal and diabetic subjects.

Load the model of glucose-insulin response. For details about the model, see the **Background** section in “Simulating the Glucose-Insulin Response”.

```
sbioloadproject('insulindemo', 'm1')
```

The model contains different initial conditions stored in various variants.

```
variants = getvariant(m1);
```

Get the initial conditions for the type 2 diabetic patient.

```
type2 = variants(1)
```

```
type2 =
   SimBiology Variant - Type 2 diabetic (inactive)

   ContentIndex:     Type:        Name:             Property:           Value:
   1                 parameter    Plasma Volume ... Value               1.49
   2                 parameter    k1                Value               0.042
   3                 parameter    k2                Value               0.071
   4                 parameter    Plasma Volume ... Value               0.04
   5                 parameter    m1                Value               0.379
   6                 parameter    m2                Value               0.673
   7                 parameter    m4                Value               0.269
   8                 parameter    m5                Value               0.0526
   9                 parameter    m6                Value               0.8118
   10                parameter    Hepatic Extrac... Value               0.6
   11                parameter    kmax              Value               0.0465
   12                parameter    kmin              Value               0.0076
   13                parameter    kabs              Value               0.023
   14                parameter    kgri              Value               0.0465
   15                parameter    f                 Value               0.9
   16                parameter    a                 Value               6e-05
   17                parameter    b                 Value               0.68
   18                parameter    c                 Value               0.00023
   19                parameter    d                 Value               0.09
   20                parameter    Stomach Glu Af... Value               125
   21                parameter    kp1               Value               3.09
   22                parameter    kp2               Value               0.0007
   23                parameter    kp3               Value               0.005
   24                parameter    kp4               Value               0.0786
   25                parameter    ki                Value               0.0066
   26                parameter    [Ins Ind Glu U... Value               1
   27                parameter    Vm0               Value               4.65
   28                parameter    Vmx               Value               0.034
   29                parameter    Km                Value               466.21
   30                parameter    p2U               Value               0.084
   31                parameter    K                 Value               0.99
   32                parameter    alpha             Value               0.013
   33                parameter    beta              Value               0.05
   34                parameter    gamma             Value               0.5
   35              parameter    ke1               Value               0.0007
   36              parameter    ke2               Value               269
   37              parameter    Basal Plasma G... Value               164.18
   38              parameter    Basal Plasma I... Value               54.81
```

Suppress an informational warning that is issued during simulations.

```
warnSettings = warning('off','SimBiology:DimAnalysisNotDone_MatlabFcn_Dimensionless');
```

Create SimFunction objects to simulate the glucose-insulin response for the normal and diabetic subjects.

- Specify an empty array {} for the second input argument to denote that the model will be simulated using the base parameter values (that is, no parameter scanning will be performed).
- Specify the plasma glucose and insulin concentrations as responses (outputs of the function to be plotted).
- Specify the species Dose as the dosed species. This species represents the initial concentration of glucose at the start of the simulation.

```
normSim = createSimFunction(m1,{},...
             {'[Plasma Glu Conc]','[Plasma Ins Conc]'},'Dose')
```

```
normSim = 
SimFunction

Parameters:

Observables: 

            Name                 Type                 Units         
    ______________________    ___________    _______________________

    {'[Plasma Glu Conc]'}    {'species'}    {'milligram/deciliter'}
    {'[Plasma Ins Conc]'}    {'species'}    {'picomole/liter'     }

Dosed: 

    TargetName       TargetDimension     
    __________    _____________________

     {'Dose'}     {'Mass (e.g., gram)'}
```

For the diabetic patient, specify the initial conditions using the variant type2.

```
diabSim = createSimFunction(m1,{},...
             {'[Plasma Glu Conc]','[Plasma Ins Conc]'},'Dose',type2)
```

```
diabSim = 
SimFunction

Parameters:

Observables: 

            Name                 Type                 Units         
```

```
        _____________________     ___________     _______________________

        {'[Plasma Glu Conc]'}     {'species'}     {'milligram/deciliter'}
        {'[Plasma Ins Conc]'}     {'species'}     {'picomole/liter'     }

    Dosed:

        TargetName       TargetDimension
        __________    _____________________

         {'Dose'}     {'Mass (e.g., gram)'}
```

Select a dose that represents a single meal of 78 grams of glucose at the start of the simulation.

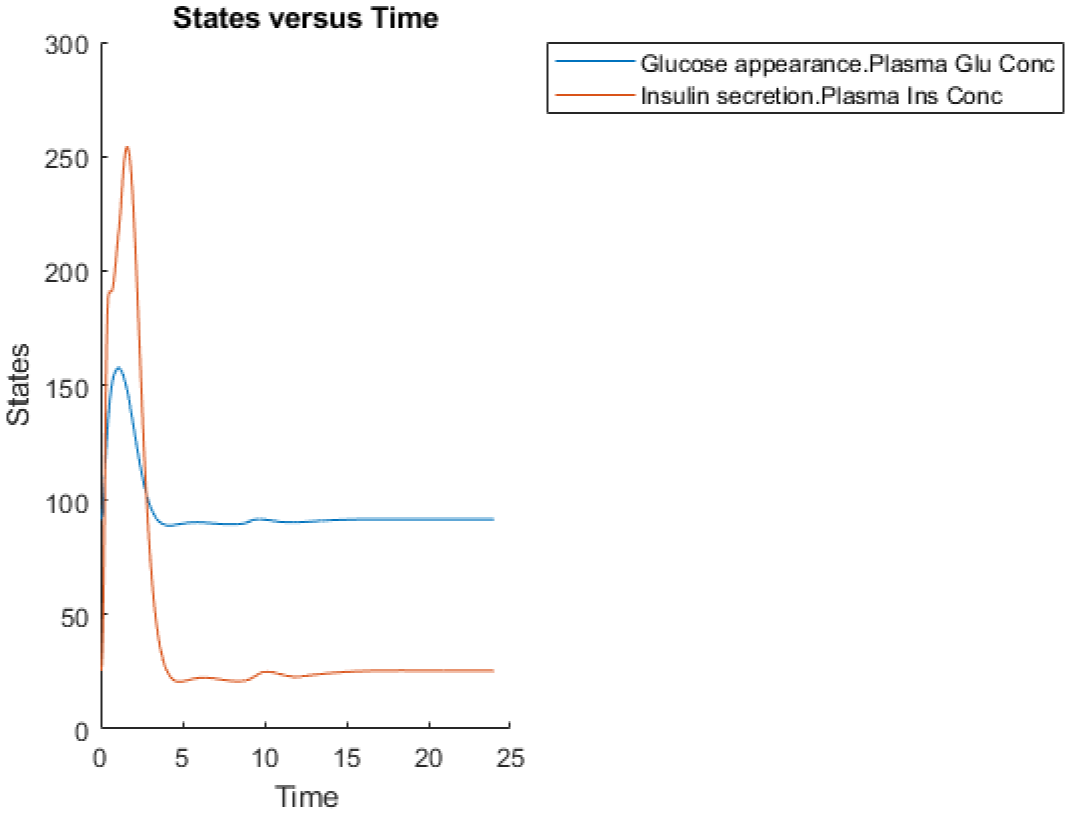
```
singleMeal = sbioselect(m1,'Name','Single Meal');
```

Convert the dosing information to the table format.

```
mealTable  = getTable(singleMeal);
```

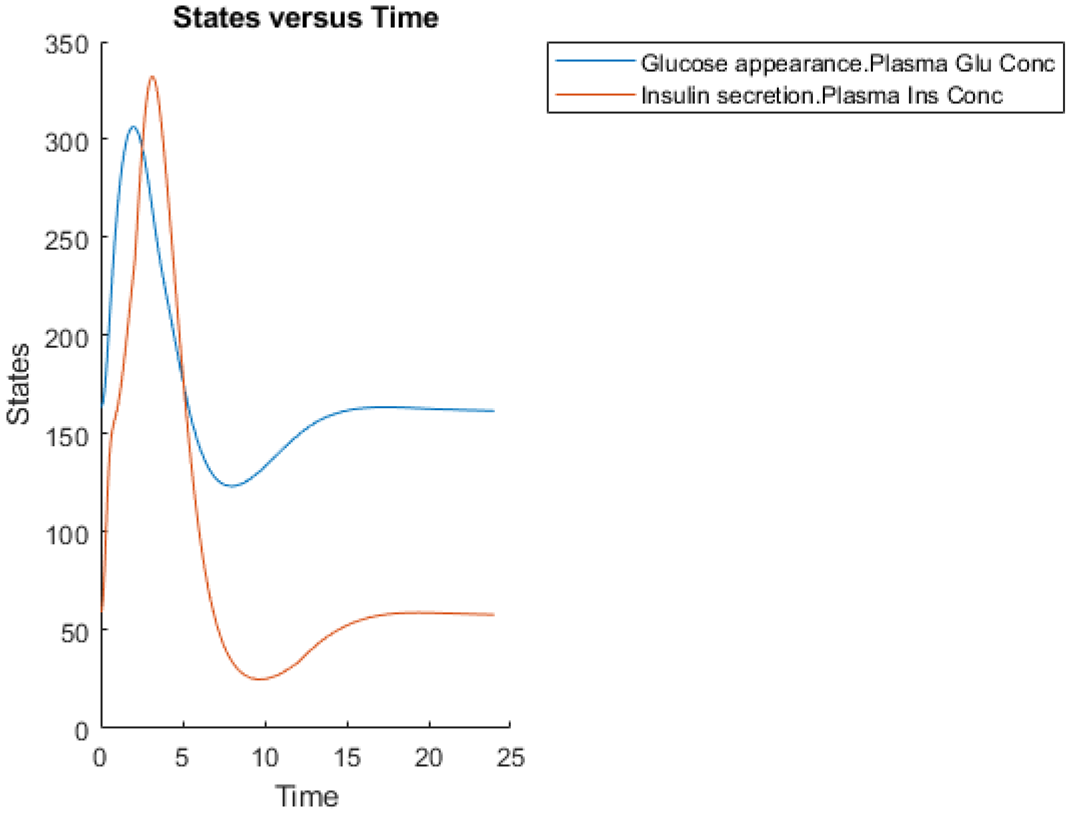
Simulate the glucose-insulin response for a normal subject for 24 hours.

```
sbioplot(normSim([],24,mealTable));
```



Simulate the glucose-insulin response for a diabetic subject for 24 hours.

```
sbioplot(diabSim([],24,mealTable));
```



Perform a scan using variants

Suppose you want to perform a parameter scan using an array of variants that contain different initial conditions for different insulin impairments. For example, the model `m1` has variants that correspond to the low insulin sensitivity and high insulin sensitivity. You can simulate the model for both conditions via a single call to the SimFunction object.

Select the variants to scan.

```
varToScan = sbioselect(m1,'Name',...
                    {'Low insulin sensitivity','High insulin sensitivity'});
```

Check which model parameters are being stored in each variant.

```
varToScan(1)
```

```
ans = 
   SimBiology Variant - Low insulin sensitivity (inactive)

   ContentIndex:     Type:         Name:             Property:           Value:
   1                 parameter     Vmx               Value               0.0235
   2                 parameter     kp3               Value               0.0045
```

```
varToScan(2)
```

```
ans = 
   SimBiology Variant - High insulin sensitivity (inactive)
```

```
ContentIndex:     Type:        Name:            Property:          Value:
1                 parameter    Vmx              Value              0.094
2                 parameter    kp3              Value              0.018
```

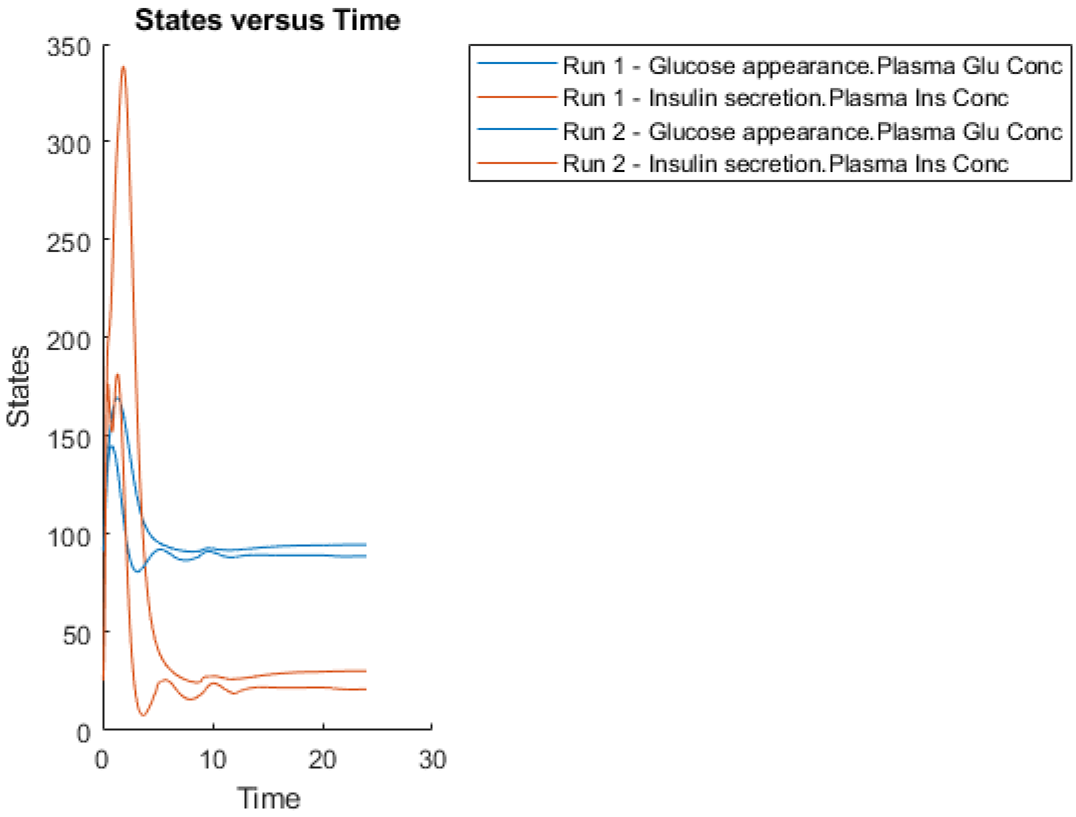
Both variants store alternate values for Vmx and kp3 parameters. You need to specify them as input parameters when you create a SimFunction object.

Create a SimFunction object to scan the variants.

```
variantScan = createSimFunction(m1,{'Vmx','kp3'},...
         {'[Plasma Glu Conc]','[Plasma Ins Conc]'},'Dose');
```

Simulate the model and plot the results. Run 1 include simulation results for the low insulin sensitivity and Run 2 for the high insulin sensitivity.

```
sbioplot(variantScan(varToScan,24,mealTable));
```



Low insulin sensitivity lead to increased and prolonged plasma glucose concentration.

Restore warning settings.

```
warning(warnSettings);
```

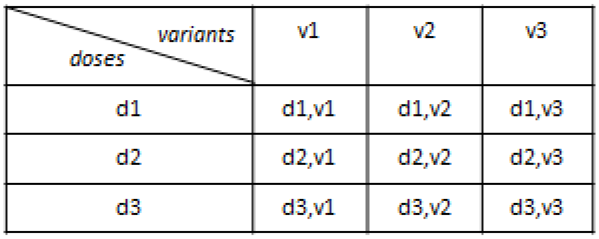
Combine Simulation Scenarios in SimBiology

There are two different methods to combine SimBiology simulation scenarios that have different doses, different variants, or different sample values for model quantities. After the combination, you get a final set of values (parameter set) that you can use to simulate and explore the model behavior. For instance, you can combine different dosing regimens with different categories of patients (variants) and simulate to explore the efficacy of the drug.

Cartesian Combination

This method is the Cartesian product of two sets. Suppose *doses* is a set of different `RepeaDose` objects `[d1,d2,d3]`, and *variants* is a set of variant objects: `[v1,v2,v3]`. The Cartesian combination of *doses* and *variants* is: *doses* x *variants* = `(d1,v1),(d2,v1),(d3,v1),(d1,v2),(d2,v2),(d3,v2),(d1,v3),(d2,v3),(d3,v3)`.

doses x variants =

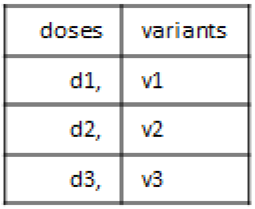


variants / *doses*	v1	v2	v3
d1	d1,v1	d1,v2	d1,v3
d2	d2,v1	d2,v2	d2,v3
d3	d3,v1	d3,v2	d3,v3

Elementwise Combination

This combination method combines the entries one to one, that is, elementwise. In other words, the first element of the first set (entry) is combined with the first element of the second set (entry) and so on. Using the same variables above, the *elementwise* combination of *doses* and *variants* is: *doses* + *variants* = `(d1,v1),(d2,v2),(d3,v3)`. Both sets must have the same number of elements (samples) for this combination method.

doses + variants =



doses	variants
d1,	v1
d2,	v2
d3,	v3

See Also

`SimBiology.Scenarios`

More About

- “SimBiology.Scenarios Terminology”

Simulation and Analysis

Model Simulation

SimBiology lets you simulate the dynamic behavior of a model. Before and during simulation, SimBiology performs a series of steps including converting the model reactions and rate rules into a set of ordinary differential equations (ODEs) that mathematically describe the model dynamics.

Specifically, before simulation begins, SimBiology:

1. Verifies the model. For details, see "Model Verification" on page 3-2.
2. Determines the initial conditions, that is, the quantity values at the beginning of simulation. In particular, SimBiology first initializes the quantity values based on the values specified in the model. Second, it updates the values by replacing them with the corresponding alternate values from variants if any. Then it updates the values based on the initial assignments and repeated assignments. SimBiology evaluates initial assignments and repeated assignments as a set of simultaneous constraints, and their order do not affect the final quantity values. For details, see "Evaluation Order of Rules" on page 2-15.
3. Constructs the ODEs based on model reactions and rate rules. Specifically, the left-hand-side (LHS) of each ODE represents the time-derivative of a model quantity. The right-hand-side (RHS) is defined using reaction fluxes that are derived from reaction rates. For details, see "Deriving ODEs from Reactions" on page 4-4.
4. Converts doses to state transitions that occur at specific simulation times.
5. Converts event functions to state transitions that depend on the conditions specified in the event triggers.

When the simulation begins, that is, at simulation time = 0, SimBiology:

1. Updates values based on initial assignments and repeated assignments.
2. Applies any state transitions due to dosing specified at simulation time = 0.
3. Logs the updated quantity values.

Note Events cannot cause transitions at time = 0 since events only apply when a trigger changes from false to true. If a trigger is true at simulation time = 0, then no transition has occurred and the event is not triggered.

During the simulation, SimBiology uses a solver to compute solutions for ODEs at different times. Specifically, the solver determines appropriate time steps and performs the following at each step.

1. Updates values for any repeated assignments.
2. Checks each event's trigger condition. If it switches from false to true at this time step, then it applies the state transitions according to the event functions, and updates values for any repeated assignments.
3. Logs the updated quantity values.

To see the system of ODEs of a model, use getequations in the command line or open the Equations View on page 1-21 in the SimBiology app.

Note If a model has algebraic equations, you must specify one of the following differential-algebraic-equation (DAE) solvers: sundials, ode15s, ode23t. SimBiology converts the algebraic equations to algebraic constraints and solves them along with the rest of ODEs. For details about available solvers, see “Choosing a Simulation Solver” on page 4-6.

See Also

Equations View on page 1-21 | getequations

Related Examples

- “Simulate the Yeast Heterotrimeric G Protein Cycle” on page 4-17
- “Create Model of Receptor-Ligand Kinetics” on page 4-13

More About

- “Simulation”
- “Choosing a Simulation Solver” on page 4-6
- “Configuring Simulation Settings” on page 4-12

Deriving ODEs from Reactions

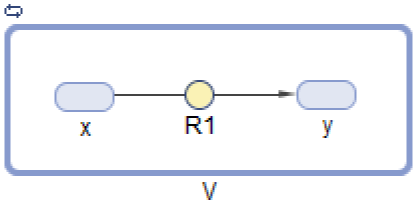
For model simulation on page 4-2, SimBiology derives ordinary differential equations (ODEs) from model reactions using mass-balance principles. The left-hand-side (LHS) of each ODE is the time-derivative of a model quantity and the right-hand-side (RHS) is defined using reaction fluxes that are derived from reaction rates and rate rules. In other words, SimBiology represents a system of ODEs as:

$$\dot{x} = S \cdot v$$

$\dot{x}$ is an M-by-`1` vector containing the rates of change for model quantities, S is an M-by-R stoichiometry matrix on page 3-11, v is an R-by-`1` flux vector. `M` equals the total number of species, and `R` equals the total number of reactions in the model

During the conversion of model reactions into ODEs, SimBiology performs a dimensional analysis to ensure each reaction flux has the dimension of `substance/time` such as `amount/time` or `mass/time`. If the reaction rate has the dimension of `concentration/time`, then SimBiology multiplies it by the compartment volume to get the reaction flux. If the reaction rate has the dimension of `substance/time`, then the flux is identical to the rate, and no volume-correction is performed. If there are no units specified with the model, the default dimension for a species (`DefaultSpeciesDimension`) is `concentration`, and that for a flux is `substance/time`. For such cases, the ODE is the flux divided by a compartment volume to make the dimension of LHS and RHS consistent. See the following figure for an illustration.

Suppose there is a reaction `x –> y`, with the reaction rate *R1*. The following figure explains the dimensional analysis performed by SimBiology to make the dimensions of LHS and RHS of an ODE consistent.



x, y = species
R1 = reaction rate
V = compartment volume

Flux

If the dimension of the reaction rate R1 is `substance/time`, then:

$$flux = R1$$

If the dimension of R1 is `concentration/time`, then:

$$flux = R1 \cdot V$$

ODE

If the dimension of species is `substance`, then:

$$\frac{dx}{dt} = -flux$$

If the dimension of species is `concentration`, then:

$$\frac{dx}{dt} = (-flux) \cdot {}^{1}/_{V}$$

See Also

`getstoichmatrix (model)`

More About

- "Model Simulation" on page 4-2
- "Determining the Stoichiometry Matrix for a Model" on page 3-11

Choosing a Simulation Solver

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

Available solvers are:

- **ODE Solvers** — These include Nonstiff Deterministic Solvers and Stiff Deterministic Solvers. The solver functions implement numerical integration methods for solving initial value problems for ordinary differential equations (ODEs). Beginning at the initial time with initial conditions, they step through the time interval, computing a solution at each time step. If the solution for a time step satisfies the solver's error tolerance criteria, it is a successful step. Otherwise, it is a failed attempt; the solver shrinks the step size and tries again. For more information, see ODE Solvers (MATLAB).
- **SUNDIALS Solvers** — At a fundamental level the core algorithms for the SUNDIALS solvers are similar to those for some of the solvers in the MATLAB ODE suite and work as described above in ODE Solvers. SimBiology always uses the SUNDIALS solver to perform sensitivity analysis on a model, regardless of what you have selected as the SolverType. For more information, see “SUNDIALS Solvers” on page 4-7.
- **Stochastic Solvers** — Use with models containing a small number of molecules. Stochastic solvers include stochastic simulation algorithm, explicit tau-leaping algorithm, and implicit tau-leaping algorithm. For more information, see “Stochastic Solvers” on page 4-8.

See Also

Related Examples

- “Simulate the Yeast Heterotrimeric G Protein Cycle” on page 4-17
- “Create Model of Receptor-Ligand Kinetics” on page 4-13

More About

- “Model Simulation” on page 4-2
- ODE Solvers (MATLAB)
- “SUNDIALS Solvers” on page 4-7
- “Stochastic Solvers” on page 4-8

SUNDIALS Solvers

SUNDIALS (Suite of Nonlinear and Differential/Algebraic Equation Solvers) are part of a freely available third-party package developed at Lawrence Livermore National Laboratory. All other ODE solvers used for simulation of SimBiology models, such as `ode45` and `ode15s`, are part of the MATLAB ODE suite. SimBiology currently (R2018b or later) uses SUNDIALS 3.1.0.

SimBiology always uses the SUNDIALS solver to perform sensitivity analysis on a model, regardless of what you have selected as the SolverType in the configuration set.

In addition, if you are estimating model parameters using `sbiofit` or the Fit Data program with one of these gradient-based estimation functions: `fmincon`, `fminunc`, `lsqnonlin`, or `lsqcurvefit`, SimBiology uses the SUNDIALS solver by default to calculate sensitivities and use them to improve fitting. If you are using `sbiofit`, you can turn off this sensitivity calculation feature by setting the “'SensitivityAnalysis'” name-value pair argument to `false`. However, if you are using the Fit Data program, you cannot turn off this feature. It is recommended that you keep the sensitivity analysis feature on whenever possible for more accurate gradient approximations and better parameter fits.

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

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

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

See Also

More About

- “Model Simulation” on page 4-2
- ODE Solvers (MATLAB)
- “Stochastic Solvers” on page 4-8

Stochastic Solvers

In this section...

When to Use Stochastic Solvers

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

Model Prerequisites for Simulating with a Stochastic Solver

Model prerequisites include:

- All reactions in the model must have their KineticLaw property set to `MassAction`.
- If your model contains events, you can simulate using the stochastic `ssa` solver. Other stochastic solvers do not support events.
- Your model must not contain doses. No stochastic solvers support doses.

Additionally, if you perform an individual or population fitting on a model whose `Configset object` specifies a stochastic solver and options, be aware that during the fitting SimBiology temporarily changes:

- `SolverType` property to the default solver of `ode15s`
- `SolverOptions` property to the options last configured for a deterministic solver

What Happens During a Stochastic Simulation?

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

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

Stochastic Simulation Algorithm (SSA)

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

Advantage

- This algorithm is exact.

Disadvantages

- Because this algorithm evaluates one reaction at a time, it might be too slow for models with a large number of reactions.
- If the number of molecules of any reactants is huge, it might take a long time to complete the simulation.

Explicit Tau-Leaping Algorithm

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

Advantages

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

Disadvantages

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

Implicit Tau-Leaping Algorithm

Like the explicit tau-leaping algorithm, the implicit tau-leaping algorithm is also an approximate method of simulation designed to speed up the simulation at the cost of some accuracy. It can handle numerically stiff problems better than the explicit tau-leaping algorithm. For deterministic systems, a problem is said to be numerically stiff if there are “fast” and “slow” time scales present in the system. For such problems, the explicit tau-leaping method performs well only if it continues to take small time steps that are of the order of the fastest time scale. The implicit tau-leaping method can potentially take much larger steps and still be stable. The algorithm treats each reaction as being independent of others. It automatically selects a time interval such that the relative change in the propensity function for each reaction is less than the user-specified error tolerance. After selecting a time interval, the algorithm computes the number of times each reaction occurs during the time interval and makes the appropriate changes to the concentration of various chemical species involved.

Advantages

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

Disadvantages

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

References

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

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

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

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

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

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

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

See Also

Related Examples

- "Analysis of Stochastic Ensemble Data in SimBiology®"

More About

- "Ensemble Runs of Stochastic Simulations" on page 4-11
- "Model Simulation" on page 4-2
- "Choosing a Simulation Solver" on page 4-6

Ensemble Runs of Stochastic Simulations

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

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

Running Ensemble Simulations

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

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

See Also

More About

- “Stochastic Solvers” on page 4-8
- “Model Simulation” on page 4-2
- “Choosing a Simulation Solver” on page 4-6

Configuring Simulation Settings

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

- Stop time (StopTime, MaximumNumberOfLogs, and MaximumWallClock properties)
- Time units (TimeUnits property)
- Solver and error tolerances (SolverType and SolverOptions properties)
- Maximum time step size (MaxStep property)
- Data to record (RuntimeOptions property)
- Frequency of data recording (OutputTimes and LogDecimation properties)
- Sensitivity analysis (SensitivityAnalysisOptions and SolverOptions properties)
- Dimensional analysis and unit conversion (CompileOptions property)

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

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

For more information on viewing and editing the stop time and other simulation settings, see “Simulate the Yeast Heterotrimeric G Protein Cycle” on page 4-17.

See Also

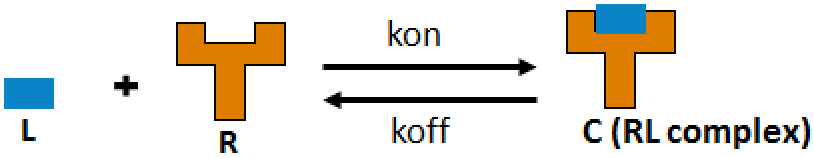
More About

- “Model Simulation” on page 4-2

Create Model of Receptor-Ligand Kinetics

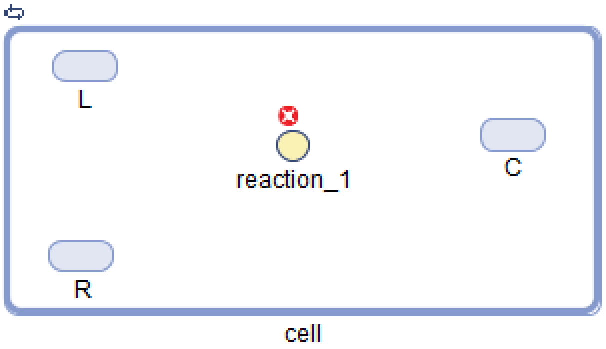
This example shows how to create and simulate a simple model of receptor-ligand kinetics using the **SimBiology** app and **SimBiology Model Analyzer** app.

Use the **SimBiology** app to create a model in which ligand L and receptor R species form receptor-ligand complexes through reversible binding reactions. These reactions are defined using mass action kinetics by $\frac{dC}{dt} = k_{on} \cdot L \cdot R - k_{off} \cdot C$, where k_{on} and k_{off} are forward and reverse rate constants. L, R, and C are the concentrations of ligand, receptor, and receptor-ligand complex respectively.

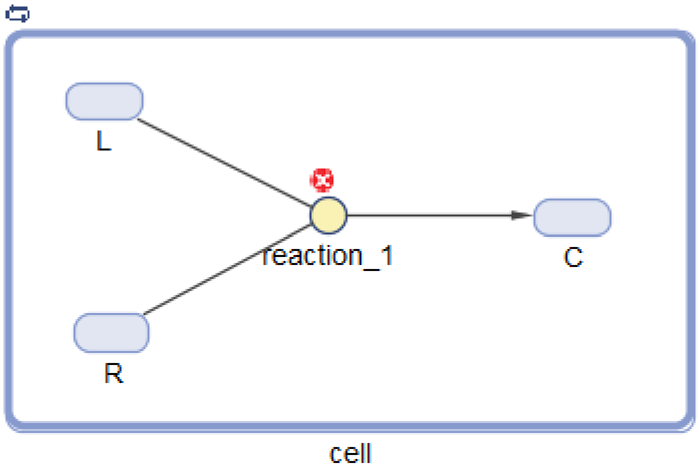


Open the app by typing `simbiology` at the command line or clicking the app icon on the **Apps** tab. Then on the **Home** tab of the app, select **Add Model** > **Create New Blank Model**. Enter a name for the model, such as `m1`. Open the Diagram view by selecting **Open** > **Diagram**.

Drag and drop three species blocks ⬭ and one reaction block ● into the compartment. Rename the species and compartment by double-clicking the default text. For instance, change `unnamed` to `cell`.

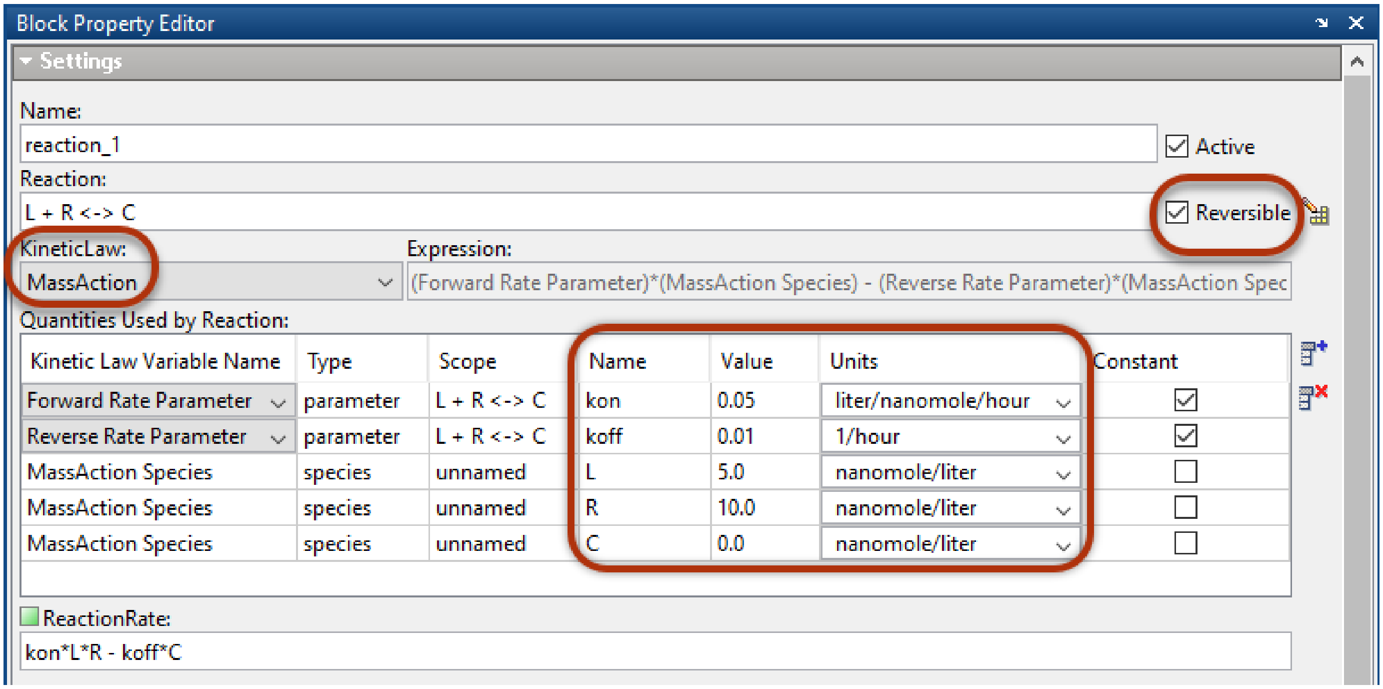


To connect the species to the reaction, press and hold the **Ctrl** key (onWindows® and Linux® systems) or the **Option** key (on Macintosh® systems), click the species block, and drag the line.



Define the reaction properties:

- Double-click the reaction block to open the Block Property Editor.
- In the **Settings** tab, select **Reversible**.
- Set **KineticLaw** to **MassAction**.
- For **Forward Rate Parameter**, set the name to `kon` and the value to `0.05`. Set the unit to `liter/nanomole/hour`. For **Reverse Rate Parameter**, set the name to `koff` and the value to `0.1` with the unit `1/hour`.
- Update the initial amounts of `L` to `5` and `R` to `10` with the unit `nanomole/liter`.


Block Property Editor

Settings

Name:
reaction_1 ☑ Active

Reaction:
L + R <-> C ☑ Reversible

KineticLaw:
MassAction

Expression:
(Forward Rate Parameter)*(MassAction Species) - (Reverse Rate Parameter)*(MassAction Spec

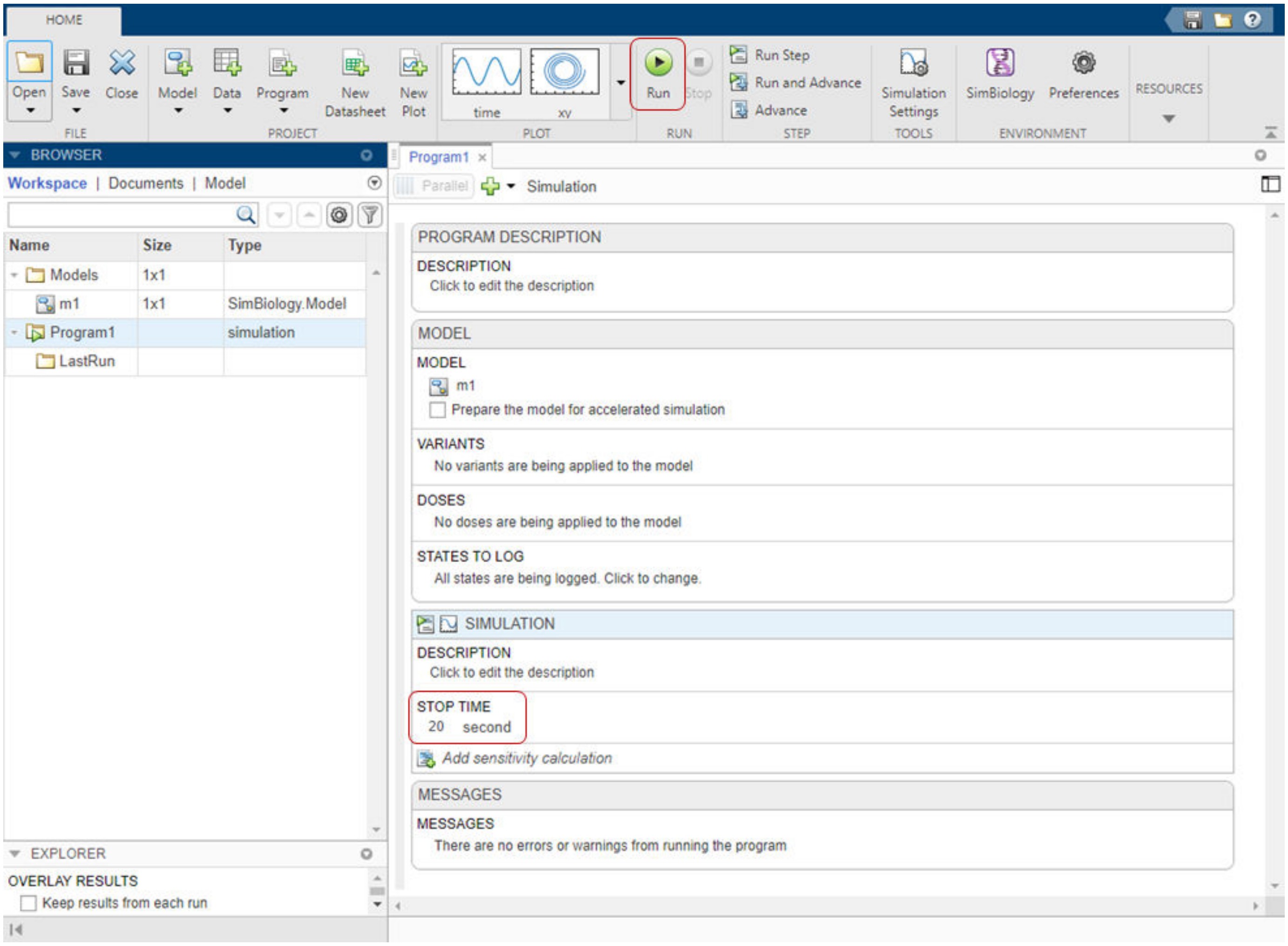
Quantities Used by Reaction:

Kinetic Law Variable Name	Type	Scope	Name	Value	Units	Constant
Forward Rate Parameter	parameter	L + R <-> C	kon	0.05	liter/nanomole/hour	☑
Reverse Rate Parameter	parameter	L + R <-> C	koff	0.01	1/hour	☑
MassAction Species	species	unnamed	L	5.0	nanomole/liter	☐
MassAction Species	species	unnamed	R	10.0	nanomole/liter	☐
MassAction Species	species	unnamed	C	0.0	nanomole/liter	☐

ReactionRate:
kon*L*R - koff*C

Open the **SimBiology Model Analyzer** app by typing `simBiologyModelAnalyzer` at the command line or clicking the app icon from the **Home** tab.

To simulate this reaction, select **Program > Simulate Model** in the **Project** section from the **Home** tab. Set the Stop Time to 20 seconds in the **Simulation** step since the model reaches a saturated state after that. Click **Run** from the **Home** tab.



Name	Size	Type
Models	1x1	
m1	1x1	SimBiology.Model
Program1		simulation
LastRun		

OVERLAY RESULTS

□ Keep results from each run

PROGRAM DESCRIPTION

DESCRIPTION
Click to edit the description

MODEL

MODEL
m1
□ Prepare the model for accelerated simulation

VARIANTS
No variants are being applied to the model

DOSES
No doses are being applied to the model

STATES TO LOG
All states are being logged. Click to change.

SIMULATION

DESCRIPTION
Click to edit the description

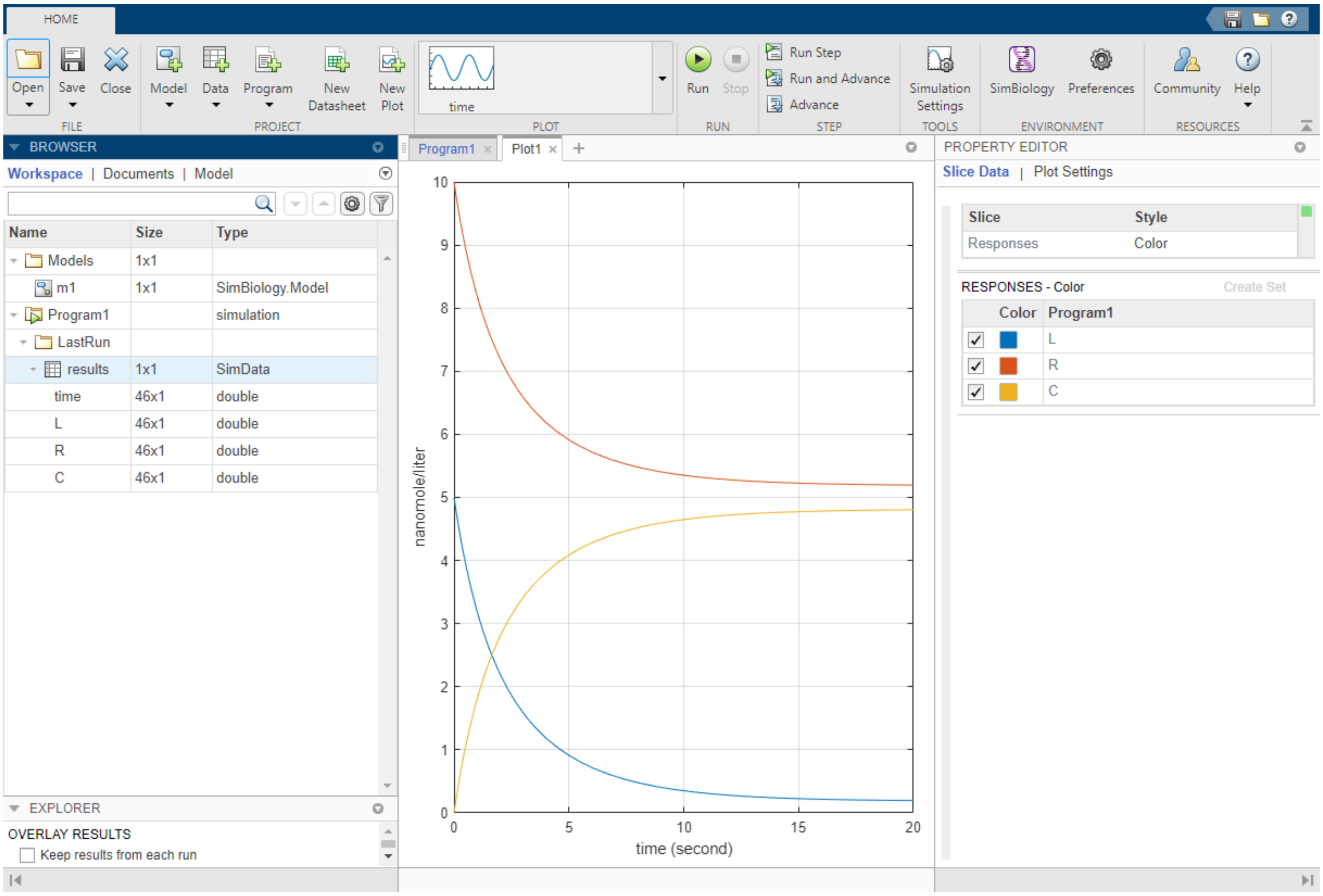
STOP TIME
20 second

Add sensitivity calculation

MESSAGES

MESSAGES
There are no errors or warnings from running the program

Running the program plots the results in the **Plot1** tab. The plot shows the simulated responses in different colors. The program stores the simulation **results** in the **LastRun** folder of the program.



Name	Size	Type
Models	1x1	
m1	1x1	SimBiology.Model
Program1		simulation
LastRun		
results	1x1	SimData
time	46x1	double
L	46x1	double
R	46x1	double
C	46x1	double

OVERLAY RESULTS

☐ Keep results from each run

Slice	Style
Responses	Color

RESPONSES - Color

	Color	Program1
☑	🟦	L
☑	🟧	R
☑	🟨	C

See Also

More About

- “SimBiology Apps”

Simulate the Yeast Heterotrimeric G Protein Cycle

This example shows how to configure simulation settings, add an event to the model to trigger a time-based change, save, and plot the simulation results. This example uses the model described in “Model of the Yeast Heterotrimeric G Protein Cycle” on page B-14 to illustrate model simulation.

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

```
sbioloadproject gprotein
```

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

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

Specify to log simulation results of all species.

```
csObj.RuntimeOptions.StatesToLog = 'all';
```

Suppose the amount of the ligand species `L` is 0 at the start of the simulation, but it increases to a particular amount at time = 100. Use `sbioselect` to select the species named `L` and set its initial amount to 0. Use `addevent` to set up the desired event.

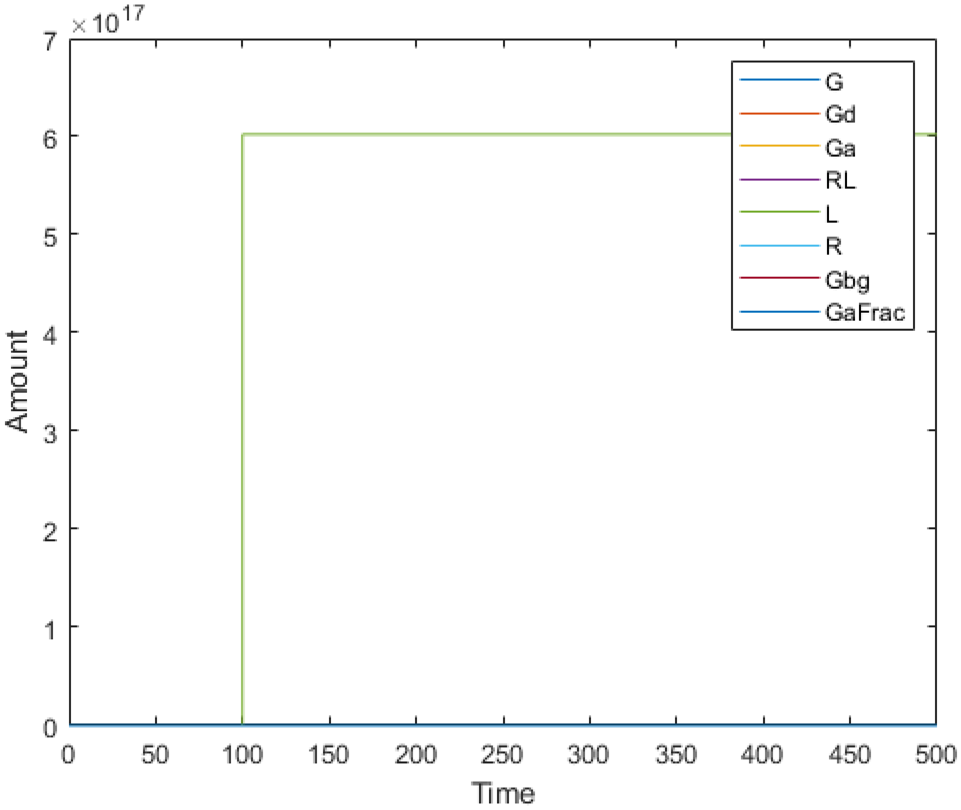
```
speciesObj = sbioselect(m1,'Type','species','Name','L');
speciesObj.InitialAmount = 0;
evt = addevent(m1,'time >= 100','L = 6.022E17');
```

Simulate the model.

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

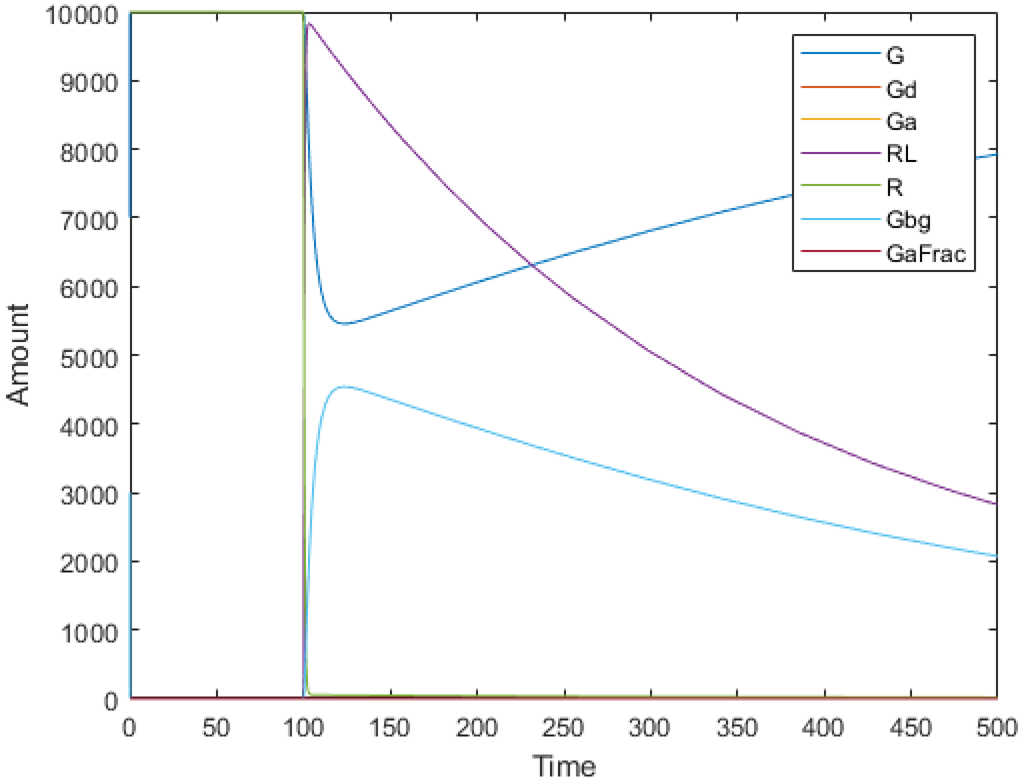
Simulate the simulation results. Notice that the species L amount increases when the event is triggered at simulation time 100. Changes in other species do not show up in the plot due to the wide range in species amounts.

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



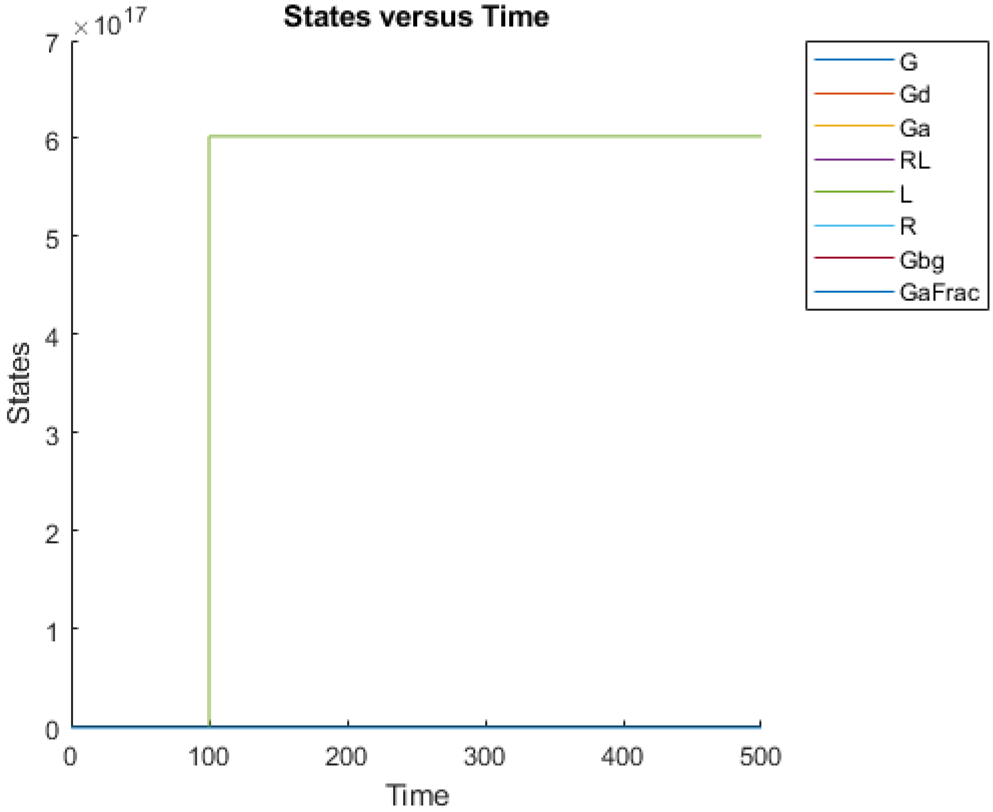
To see the changes of other species, plot without the species L (the 5th species) data.

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



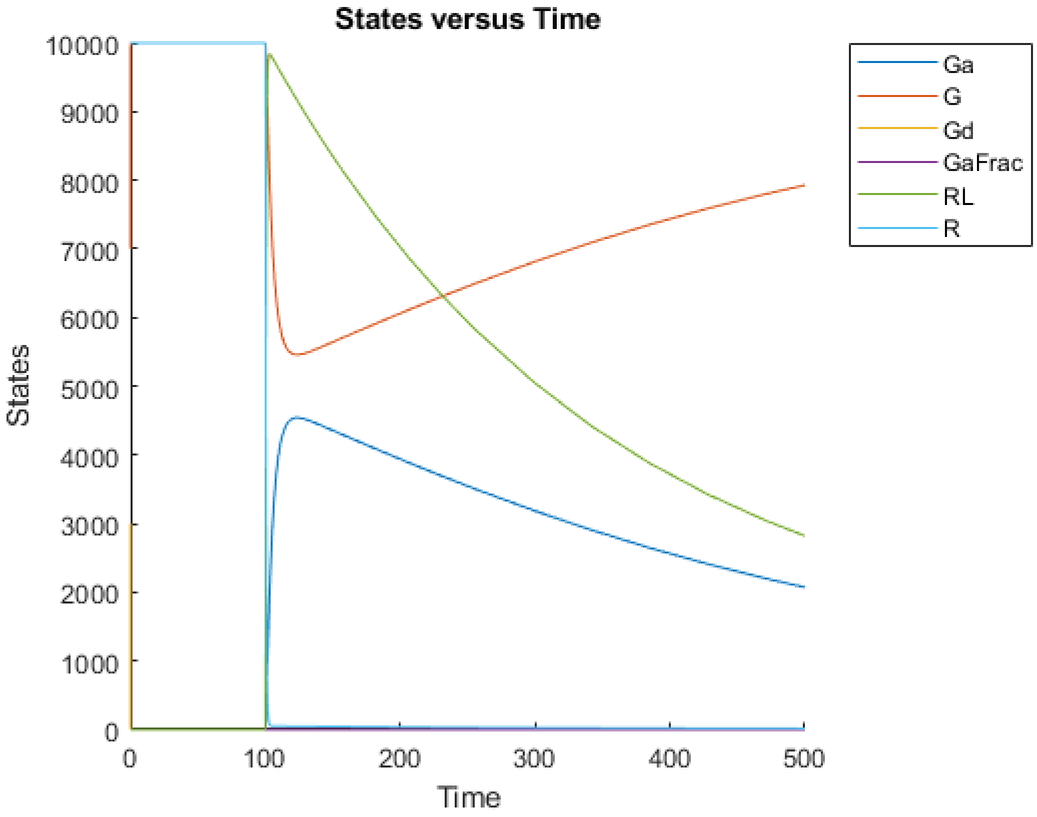
Alternative to storing simulation data in separate outputs, such as `t`, `x`, and `names` as above, you can store them all in a single `SimData object`. You can then use `selectbyname` to extract arrays containing the simulation data of your interest.

```
simdata     = sbiosimulate(m1);
sbioplot(simdata);
```



Expand **Run 1** to see the names of species and parameter that are plotted.

```
simdata_noL = selectbyname(simdata, {'Ga','G','Gd','GaFrac','RL','R'});
sbioplot(simdata_noL);
```


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

Sensitivity Analysis in SimBiology

In this section...

Sensitivity Analysis

Sensitivity analysis lets you explore the effects of variations in model quantities (species, compartments, and parameters) on a model response. You can use the analysis to validate preexisting knowledge or assumption about influential model quantities on a model response or to find such quantities. You can use the information from sensitivity analysis for decision making, designing experiments, and parameter estimation. SimBiology supports two types of sensitivity analyses: local sensitivity analysis and global sensitivity analysis.

Global sensitivity analysis uses Monte Carlo simulations, where a representative (*global*) set of parameter sample values are used to explore the effects of variations in model parameters of interest on the model response. GSA provides insights into relative contributions of individual parameters that contribute most to the overall model behavior.

On the other hand, local sensitivity analysis is derivative based. This technique analyzes the effect of one model parameter at a time, keeping the other parameters fixed. Local sensitivities are dependent on a specific choice of parameter values at a time point where the analysis is performed and do not capture how parameters interact with each other during simulation when they are varied jointly.

Global Sensitivity Analysis (GSA)

In GSA, model quantities are varied together to simultaneously evaluate the relative contributions of each quantity with respect to a model response. SimBiology provides two ways to perform GSA.

Sobol Indices

In this approach, SimBiology performs a decomposition of the model output (response) variance by calculating the first- and total-order Sobol indices [1]. The first-order Sobol indices give the fractions of the overall response variance that can be attributed to variations in an input parameter alone. The total-order Sobol index gives the fraction of the overall response variance that can be attributed to joint parameter variations. For details, see "Saltelli Method to Compute Sobol Indices".

Use `sbiosobol` to compute the Sobol indices. The function requires Statistics and Machine Learning Toolbox.

Multiparametric GSA (MPGSA)

MPGSA lets you study the relative importance of parameters with respect to a classifier defined by model responses. SimBiology implements the MPSA method proposed by Tiemann et. al. [2]. For details, see "Multiparametric Global Sensitivity Analysis (MPGSA)".

Use `sbiompgsa` to perform MPGSA. The function requires Statistics and Machine Learning Toolbox.

Local Sensitivity Analysis (LSA)

In this analysis, SimBiology calculates the time-dependent sensitivities of all the species states with respect to species initial conditions and parameter values in the model.

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

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

where, the numerator is the sensitivity output and the denominators are the sensitivity inputs to sensitivity analysis. For more information on the calculations performed, see [3][4][5].

Model Requirements for LSA

LSA is supported only by the ordinary differential equation (ODE) solvers. SimBiology 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 calculates derivatives by using a technique called complex-step approximation. This technique yields accurate results for the vast majority of typical reaction kinetics, which involve only simple mathematical operations and functions. However, this technique can produce inaccurate results when analyzing models that contain mathematical expressions that involve nonanalytic functions, such as `abs`. In this case, SimBiology either disables the sensitivity analysis or warns you that the computed sensitivities may be inaccurate. If sensitivity analysis gives questionable results for a model with reaction rates that contain unusual functions, you may be running into limitations of the complex-step technique. Contact MathWorks Technical Support for additional information.

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

- Nonconstant compartments
- Algebraic rules
- Events

Note You can perform sensitivity analysis on a model containing repeated assignment rules, but only if the repeated assignment rules do not determine species or parameters used as inputs or outputs in sensitivity analysis.

SUNDIALS as Default Solver

SimBiology always uses the SUNDIALS solver to perform sensitivity analysis on a model, regardless of what you have selected as the SolverType in the configuration set.

In addition, if you are estimating model parameters using `sbiofit` or the Fit Data program with one of these gradient-based estimation functions: `fmincon`, `fminunc`, `lsqnonlin`, or `lsqcurvefit`, SimBiology uses the SUNDIALS solver by default to calculate sensitivities and use them to improve fitting. If you are using `sbiofit`, you can turn off this sensitivity calculation feature by setting the “'SensitivityAnalysis'” name-value pair argument to `false`. However, if you are using the Fit Data program, you cannot turn off this feature. It is recommended that you keep the sensitivity analysis feature on whenever possible for more accurate gradient approximations and better parameter fits.

Calculate Local Sensitivities Using sbiosimulate

Set the following properties of the SolverOptions property of your `configset` object, before running the `sbiosimulate` function:

- SensitivityAnalysis — Set to `true` to calculate the time-dependent sensitivities of all the species states defined by the `Outputs` property with respect to the initial conditions of the species and the values of the parameters specified in `Inputs`.
- SensitivityAnalysisOptions — An object that holds the sensitivity analysis options in the configuration set object. Properties of `SensitivityAnalysisOptions` are:
  - Outputs — Specify the species and parameters for which you want to compute the sensitivities. This is the numerator as described in “Sensitivity Analysis” on page 4-22.
  - Inputs — Specify the species and parameters with respect to which you want to compute the sensitivities. Sensitivities are calculated with respect to the InitialAmount property of the specified species. This is the denominator, described in “Sensitivity Analysis” on page 4-22.
  - Normalization — Specify the normalization for the calculated sensitivities:
    - `'None'` — No normalization
    - `'Half'` — Normalization relative to the numerator (species output) only
    - `'Full'` — Full dedimensionalization

    For more information about normalization, see Normalization.

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

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

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

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

For illustrated examples, see:

- “Calculate Sensitivities Using sbiosimulate” on page 4-26
- “Parameter Scanning, Parameter Estimation, and Sensitivity Analysis in the Yeast Heterotrimeric G Protein Cycle”

Calculate Local Sensitivities Using SimFunctionSensitivity object

Create a `SimFunctionSensitivity object` using the `createSimFunction` specifying the `'SensitivityOutputs'` and `'SensitivityInputs'` name-value pair arguments. Then execute the object. For an illustrated example, see “Calculate Sensitivities Using SimFunctionSensitivity Object”.

Calculate Local Sensitivities Using SimBiology Model Analyzer App

For a workflow example using the app, see “Find Important Parameters with Sensitivity Analysis Using SimBiology Model Analyzer App” on page 1-30.

References

[1] Saltelli, Andrea, Paola Annoni, Ivano Azzini, Francesca Campolongo, Marco Ratto, and Stefano Tarantola. “Variance Based Sensitivity Analysis of Model Output. Design and Estimator for the Total Sensitivity Index.” *Computer Physics Communications* 181, no. 2 (February 2010): 259–70. https://doi.org/10.1016/j.cpc.2009.09.018.

[2] Tiemann, Christian A., Joep Vanlier, Maaike H. Oosterveer, Albert K. Groen, Peter A. J. Hilbers, and Natal A. W. van Riel. “Parameter Trajectory Analysis to Identify Treatment Effects of Pharmacological Interventions.” Edited by Scott Markel. *PLoS Computational Biology* 9, no. 8 (August 1, 2013): e1003166. https://doi.org/10.1371/journal.pcbi.1003166.

[3] Martins, Joaquim, Ilan Kroo, and Juan Alonso. “An Automated Method for Sensitivity Analysis Using Complex Variables.” In *38th Aerospace Sciences Meeting and Exhibit*. Reno,NV,U.S.A.: American Institute of Aeronautics and Astronautics, 2000. https://doi.org/10.2514/6.2000-689.

[4] Martins, J., Peter Sturdza, and Juan Alonso. “The Connection between the Complex-Step Derivative Approximation and Algorithmic Differentiation.” In *39th Aerospace Sciences Meeting and Exhibit*. Reno,NV,U.S.A.: American Institute of Aeronautics and Astronautics, 2001. https://doi.org/10.2514/6.2001-921.

[5] Ingalls, Brian P., and Herbert M. Sauro. “Sensitivity Analysis of Stoichiometric Networks: An Extension of Metabolic Control Analysis to Non-Steady State Trajectories.” *Journal of Theoretical Biology* 222, no. 1 (May 2003): 23–36. https://doi.org/10.1016/S0022-5193(03)00011-0.

Calculate Sensitivities Using sbiosimulate

In this section...

Overview

About the Example Model

This example uses the model described in “Model of the Yeast Heterotrimeric G Protein Cycle” on page B-14 to illustrate SimBiology sensitivity analysis options.

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

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

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

About the Example

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

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

Load and Configure the Model for Sensitivity Analysis

1 The `gprotein_norules.sbproj` project contains a model that represents the wild-type strain (stored in variable `m1`).

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

   ```
   Ga = sbioselect(m1,'Type','species','Where','Name','==','Ga');
   ```
4. Set the Outputs property of the SensitivityAnalysisOptions object to the Ga species.

   ```
   csObj.SensitivityAnalysisOptions.Outputs = Ga;
   ```
5. Use the sbioselect function, which lets you query by type, to retrieve all the parameters from the model and store the vector in a variable, pif.

   ```
   pif = sbioselect(m1,'Type','parameter');
   ```
6. Set the Inputs property of the SensitivityAnalysisOptions object to the pif variable containing the parameters.

   ```
   csObj.SensitivityAnalysisOptions.Inputs =  pif;
   ```
7. Enable sensitivity analysis in the configuration set object (csObj) by setting the SensitivityAnalysis option to true.

   ```
   csObj.SolverOptions.SensitivityAnalysis = true;
   ```
8. Set the Normalization property of the SensitivityAnalysisOptions object to perform 'Full' normalization.

   ```
   csObj.SensitivityAnalysisOptions.Normalization = 'Full';
   ```

Perform Sensitivity Analysis

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

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

Extract and Plot Sensitivity Data

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

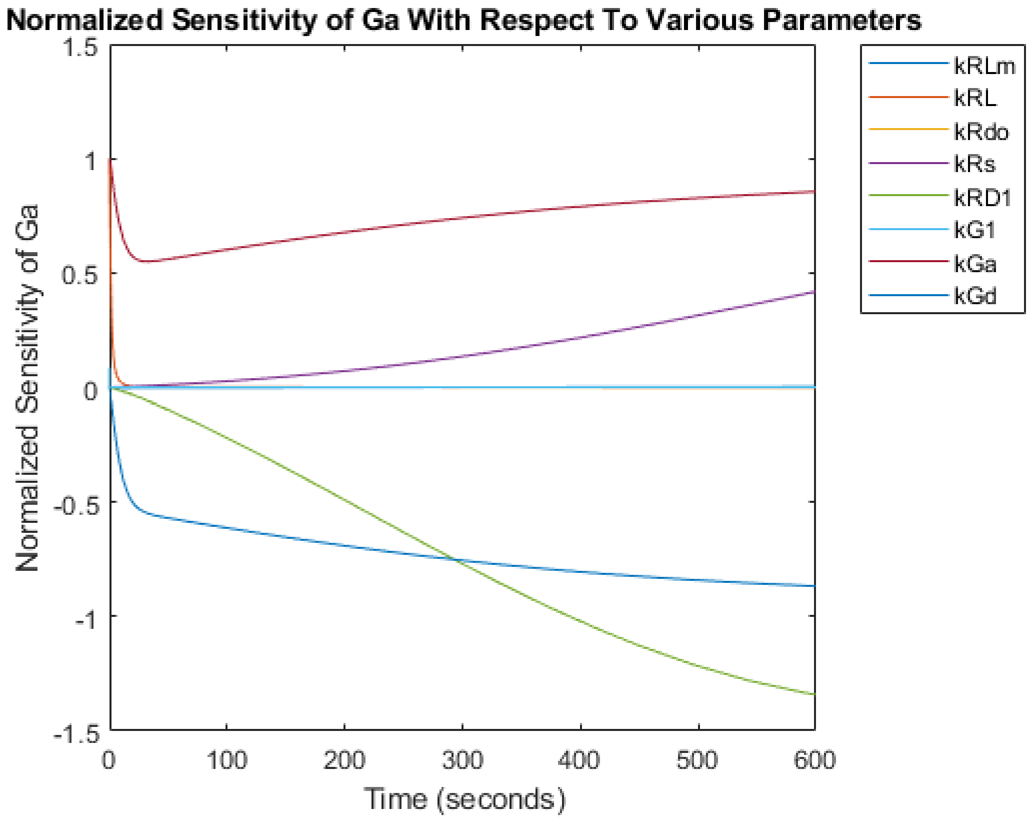
1. Extract sensitivity data in output variables T (time), R (sensitivity data for species Ga), snames (names of the states specified for sensitivity analysis), and ifacs (names of the input factors used for sensitivity analysis):

   ```
   [T, R, snames, ifacs] = getsensmatrix(simDataObj);
   ```
2. Because R is a 3-D array with dimensions corresponding to times, output factors, and input factors, reshape R into columns of input factors to facilitate visualization and plotting:

   ```
   R2 = squeeze(R);
   ```
3. After extracting the data and reshaping the matrix, plot the data:

   ```
   figure;
   plot(T,R2);
   title('Normalized Sensitivity of Ga With Respect To Various Parameters');
   ```

```
xlabel('Time (seconds)');
ylabel('Normalized Sensitivity of Ga');
leg = legend(ifacs, 'Location', 'NorthEastOutside');
set(leg, 'Interpreter', 'none');
```



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

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

Perform a Parameter Scan

This example shows how to perform a parameter scan by simulating a model multiple times, each time varying the value of a parameter.

In the model described in Model of the Yeast Heterotrimeric G Protein Cycle, the rate of G protein inactivation (`kGd`) is much lower in the mutant strain versus the wild-type strain (`kGd = 0.004` versus `kGd = 0.11`), which explains higher levels of activated G protein (`Ga`) in the mutant strain. For a detailed look at how varying the level of `kGd` affects the level of `Ga`, perform a parameter scan over different values of `kGd`.

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

```
sbioloadproject gprotein
```

Create a vector of five evenly spaced values for `kGd` ranging from `0.001` to `0.15`.

```
kGdValues = linspace(1e-3,0.15,5)';
```

Create a `SimFunction` object, where `kGd` is the input parameter to scan, and `Ga` is the observed species. Pass in an empty array [] as the last input argument to denote there are no dosed species.

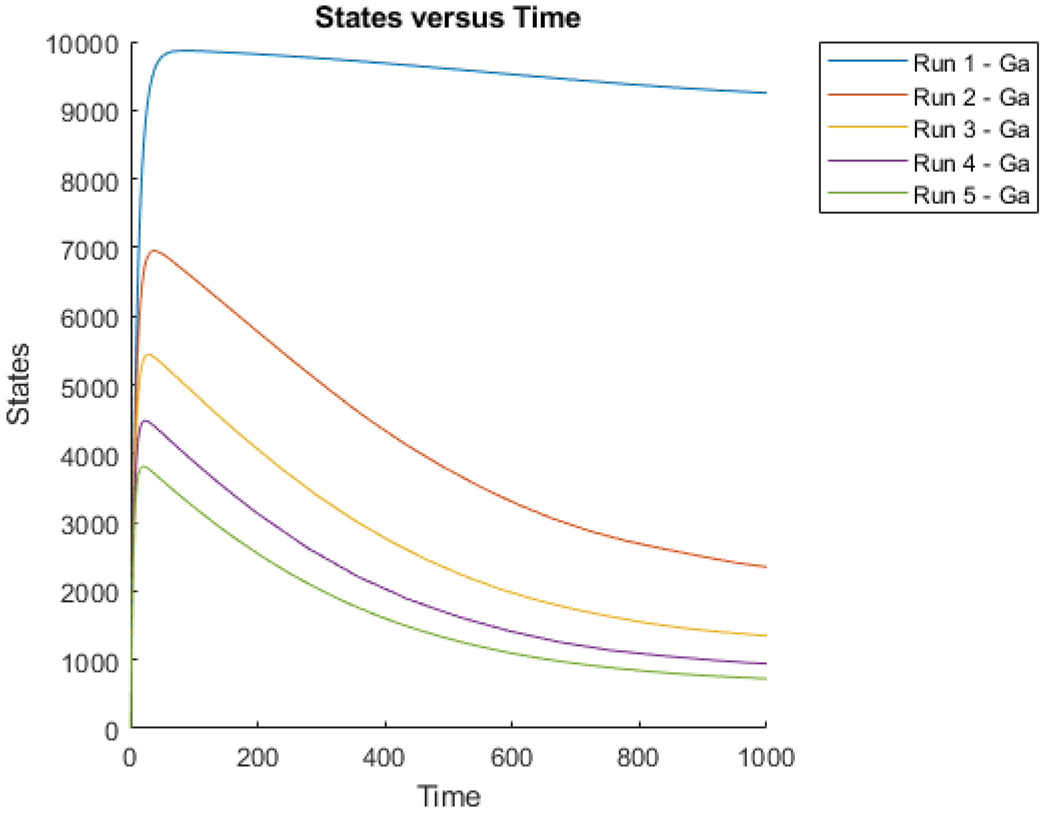
```
simfunc = createSimFunction(m1,{'kGd'},{'Ga'},[]);
```

Simulate the model multiple times with different kGd values. Set the stop time to 1000.

```
sd = simfunc(kGdValues,1000);
```

Plot the simulation results to see how varying the level of `kGd` affects the level of `Ga`.

```
sbioplot(sd);
```



See Also

`SimFunction object` | `createSimFunction`

More About

- "Model of the Yeast Heterotrimeric G Protein Cycle" on page B-14

Nonlinear Mixed-Effects Modeling

In this section...

What Is a Nonlinear Mixed-Effects Model?

A mixed-effects model is a statistical model that incorporates both *fixed effects* and *random effects*. Fixed effects are population parameters assumed to be the same each time data is collected, and random effects are random variables associated with each sample (individual) from a population. Mixed-effects models work with small sample sizes and sparse data sets, and are often used to make inferences on features underlying profiles of repeated measurements from a group of individuals from a population of interest.

As with all regression models, their purpose is to describe a response variable as a function of the predictor (independent) variables. Mixed-effects models, however, recognize correlations within sample subgroups, providing a reasonable compromise between ignoring data groups entirely, thereby losing valuable information, and fitting each group separately, which requires significantly more data points.

For instance, consider population pharmacokinetic data that involve the administration of a drug to several individuals and the subsequent observation of drug concentration for each individual, and the objective is to make a broader inference on population-wide parameters while considering individual variations. The nonlinear function often used for such data is an exponential function since many drugs once distributed in a patient are eliminated in an exponential fashion. Thus the measured drug concentration of an individual can be described as:

$$y_{ij} = \frac{D_i}{V}e^{-k_i t_{ij}} + a\varepsilon_{ij},$$

where y_{ij} is the jth response of the ith individual, D_i is the dose administered to the ith individual, V is the population mean volume of distribution, a is an error parameter, and $\varepsilon_{ij} \sim N(0, 1)$, representing some measurement error. The elimination rate parameter (k_i) depends on the clearance and volume of the central compartment $k_i = \frac{Cl_i}{V}$. Both k_i and Cl_i are for the ith patient, meaning they are patient-specific parameters.

To account for variations between individuals, assume that the clearance is a random variable depending on individuals, varying around the population mean. For the ith individual, $Cl_i = \theta_1 + \eta_i$, where θ_1 is the fixed effect (population parameter for the clearance) and η_i is the random effect, that is, the deviation of the ith individual from the mean clearance of the population $\eta_i \sim N(0, \sigma_\eta^2)$.

If you have any individual-specific covariates such as weight w that linearly relate to the clearance, you can try explaining some of the between-individual differences. For example, if w_i is the weight of the *i*th individual, then the model becomes $Cl_i = \theta_1 + \theta_2 * w_i + \eta_i$, where θ_2 is the fixed effect of weight on clearance.

A general nonlinear mixed-effects (NLME) model with constant variance is as follows:

$$y_{ij} = f(x_{ij}, p_i) + \varepsilon_{ij}$$

$$p_i = A_i\theta + B_i\eta_i$$

$$\varepsilon_{ij} \sim N(0, \sigma^2)$$

$$\eta_i \sim N(0, \Psi)$$

y_{ij}	Data vector of individual-specific response values
f	General, real-valued function of p_i and x_{ij}
x_{ij}	Data matrix of individual-specific predictor values
p_i	Vector of individual-specific model parameters
θ	Vector of fixed effects, modeling population parameters
η_i	Vector of multivariate normally distributed individual-specific random effects
A_i	Individual-specific design matrix for combining fixed effects
B_i	Individual-specific design matrix for combining random effects
ε_{ij}	Vector of group-specific errors, assumed to be independent, identically, normally distributed, and independent of η_i
Ψ	Covariance matrix for the random effects
σ^2	Error variance, assumed to be constant across observations

In addition to the constant error model, there are other error models such as proportional, exponential, and combined error models. For details, see “Error Models” on page 4-44.

Nonlinear Mixed-Effects Modeling Workflow

SimBiology lets you estimate fixed effects θs and random effects ηs as well as the covariance matrix of random effects Ψ. However, you cannot alter *A* and *B* design matrices since they are automatically determined from the covariate model you specify. Use the `sbiofitmixed` function to estimate nonlinear mixed-effects parameters. These steps show one of the workflows you can use at the command line.

1. Import data.
2. Convert the data to the `groupedData` format.
3. Define dosing data. For details, see “Doses in SimBiology Models” on page 2-30.
4. Create a structural model (one-, two-, or multicompartment model). For details, see “Create Pharmacokinetic Models” on page 5-13.
5. Create a covariate model to define parameter-covariate relationships if any. For details, see “Specify a Covariate Model” on page 4-33.
6. Map the response variable from data to the model component. For example, if you have the measured drug concentration data for the central compartment, then map it to the drug species in the central compartment (typically the `Drug_Central` species).

7. Specify parameters to estimate using the `EstimatedInfo object`. It lets you optionally specify parameter transformations, initial values, and parameter bounds. Supported transforms are `log`, `probit`, `logit`, and `none` (no transform).
8. (Optional) You can also specify an error model. The default model is the constant error model. For instance, you can change it to the proportional error model if you assume the measurement error is proportional to the response data. See “Specify an Error Model” on page 4-35.
9. Estimate parameters using `sbiofitmixed`, which performs “Maximum Likelihood Estimation” on page 4-35.
10. (Optional) If you have a large, complex model, the estimation might take longer. SimBiology lets you check the status of fitting as it progresses. See “Obtain the Fitting Status” on page 4-35.

For a complete workflow example, see “Modeling the Population Pharmacokinetics of Phenobarbital in Neonates”.

Specify a Covariate Model

When specifying a nonlinear mixed-effects model, you define parameter-covariate relationship using a covariate model (`CovariateModel object`). For example, suppose you have PK profile data for multiple individuals and are estimating three parameters (clearance *Cl*, compartment volume *V*, and elimination rate *k*) that have both fixed and random effects. Assume the clearance *Cl* has a correlation with a covariate variable weight (*w*) of each individual. Each parameter can be described as a linear combination of fixed and random effects.

$$Cl_i = \theta_1 + \theta_2 * w_i + \eta_{1i},$$

$$V_i = \theta_3 + \eta_{2i},$$

$$k_i = \theta_4 + \eta_{3i},$$

where θs represent fixed effects and ηs represent random effects, which are normally distributed $\begin{pmatrix} \eta_{1i} \\ \eta_{2i} \\ \eta_{3i} \end{pmatrix} \sim MVN(0, \Psi)$. By default, the random effects are uncorrelated. So $\Psi = \begin{pmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \sigma_2^2 & 0 \\ 0 & 0 & \sigma_3^2 \end{pmatrix}$.

1. Construct an empty `CovariateModel` object.

   ```
   covModel = CovariateModel;
   ```
2. Set the `Expression` property to define the relationships between parameters (*Cl*, *V*, and *k*) and covariate (*w*). You must use `theta` as a prefix for all fixed effects and `eta` for random effects.

   ```
   covModel.Expression = {'Cl = theta1 + theta2*w + eta1','V = theta3 + eta2','k = theta4 + eta3
   ```

   The `FixedEffectNames` property displays the fixed effects defined in the model.

   ```
   covModel.FixedEffectNames
   ```

   ```
   ans = 

       'theta1'
       'theta3'
       'theta4'
       'theta2'
   ```

The FixedEffectDescription property displays which fixed effects correspond to which parameter. For instance, *theta1* is the fixed effect for the *Cl* parameter, and *theta2* is the fixed effect for the weight covariate that has a correlation with *Cl* parameter, denoted as *Cl/w*.

```
covModel.FixedEffectDescription

ans = 

    'Cl'
    'V'
    'k'
    'Cl/w'
```

3 Specify initial estimates for the fixed effects. Create a structure containing initial estimates using the constructDefaultFixedEffectValues function.

```
initialEstimates = constructDefaultFixedEffectValues(covModel)

initialEstimates = 

    theta1: 0
    theta2: 0
    theta3: 0
    theta4: 0

initialEstimates.theta1 = 1.20;
initialEstimates.theta2 = 0.30;
initialEstimates.theta3 = 0.90;
initialEstimates.theta4 = 0.10;
```

4 Set the initial estimates to the FixedEffectValues property.

```
covModel.FixedEffectValues = initialEstimates;
```

Specify a Covariance Pattern Among Random Effects

By default, sbiofitmixed assumes no covariance among random effects, that is, a diagonal covariance matrix is used. Suppose you have η_1, η_2, and η_3, and there is a covariance σ_{12} between η_1 and η_2. You can indicate this using a pattern matrix where 1 indicates a variance or covariance parameter which is estimated by sbiofitmixed. For instance, a pattern matrix $\begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ represents $\begin{pmatrix} \sigma_1^2 & \sigma_{12} & 0 \\ \sigma_{21} & \sigma_2^2 & 0 \\ 0 & 0 & \sigma_3^2 \end{pmatrix}$.

Define such a pattern using an options struct.

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

Then you can use options as an input argument for sbiofitmixed. For a complete workflow, see “Nonlinear Mixed-Effects Modeling Workflow” on page 4-32.

Specify an Error Model

During the “Nonlinear Mixed-Effects Modeling Workflow” on page 4-32, you can optionally specify an error model using a structure.

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

Then you can use `options` as one of the input arguments when you run `sbiofitmixed`.

Supported error models are constant (default), proportional, combined, and exponential models. For details, see “Error Models” on page 4-44.

Maximum Likelihood Estimation

SimBiology estimates the parameters of a nonlinear mixed-effects model by maximizing a likelihood function. The function can be described as:

$$p(y \,\middle|\, \theta, \sigma^2, \Psi) = \int p(y \,\middle|\, \theta, \eta, \sigma^2) p(\eta \,\middle|\, \Psi)\, d\eta,$$

where y is the response data, θ is the vector of fixed effects, σ^2 is the error variance, Ψ is the covariance matrix for random effects, and η is the vector of unobserved random effects. $p(y \mid \theta, \sigma^2, \Psi)$ is the marginal density of y, $p(y \mid \theta, \eta, \sigma^2)$ is the conditional density of y given the random effects η, and the prior distribution of η is $p(\eta \mid \Psi)$.

This integral contains a nonlinear function of the fixed effects and variance parameters that you want to maximize. Typically for nonlinear models, the integral does not have a closed form, and needs to be solved numerically, which involves simulating the function at each time step of an optimization algorithm. Therefore, the estimation can take a long time for complex models, and initial values of parameters might play an important role for successful convergence. SimBiology provides these iterative algorithms to solve the integral and maximize the likelihood if you have Statistics and Machine Learning Toolbox.

- `LME` — Use the likelihood for the linear mixed-effects model at the current conditional estimates of θ and η. This is the default.
- `RELME` — Use the restricted likelihood for the linear mixed-effects model at the current conditional estimates of θ and η.
- `FO` — First-order (Laplacian) approximation without random effects.
- `FOCE` — First-order (Laplacian) approximation at the conditional estimates of θ.
- stochastic EM — Use the Expectation-Maximization (EM) algorithm in which the E step is replaced by a stochastic procedure.

For a complete workflow, see “Nonlinear Mixed-Effects Modeling Workflow” on page 4-32.

Obtain the Fitting Status

During the estimation of mixed-effects parameters of a large and complex model that may take a longer time, you may want to obtain the status of fitting as it progresses. Set `'ProgressPlot'` to `true` when you run `sbiofitmixed` to display the progress of the fitting. For details, see “Progress Plot” on page 4-45.

For a complete workflow, see “Nonlinear Mixed-Effects Modeling Workflow” on page 4-32.

See Also

`sbiofit` | `sbiofitmixed`

More About

- “Progress Plot” on page 4-45
- “Nonlinear Regression” on page 4-37
- “Supported Methods for Parameter Estimation in SimBiology” on page 4-42

Nonlinear Regression

In this section...

What is Nonlinear Regression?

The purpose of regression models is to describe a response variable as a function of independent variables. Multiple linear regression models describe the response as a linear combination of coefficients and functions of independent variables. Nonlinearities can be modeled using nonlinear functions of independent variables. However, the coefficients always enter the model in a linear fashion.

Nonlinear regression models are more mechanistic models of nonlinear relationships between the response and independent variables. The parameters can enter the model as exponential, trigonometric, power, or any other nonlinear function. The unknown parameters in the model are estimated by minimizing a statistical criterion such as the negative log likelihood or the sum of squared deviations between observed and predicted values.

In the case of pharmacokinetic (PK) studies, the response data usually represent some measured drug concentrations, and independent variables are often dose and time. The nonlinear function often used for such data is an exponential function since many drugs once distributed in a patient are eliminated in an exponential fashion. One PK parameter to estimate in this case is the rate at which the drug is eliminated from the body given the concentration-time data.

For instance, consider drug plasma concentration data from a single individual after an intravenous bolus dose measured at different time points with some errors. Assume the measured drug concentration follows a monoexponential decline: $C_t = C_0 e^{-k_e t} + a\varepsilon$.

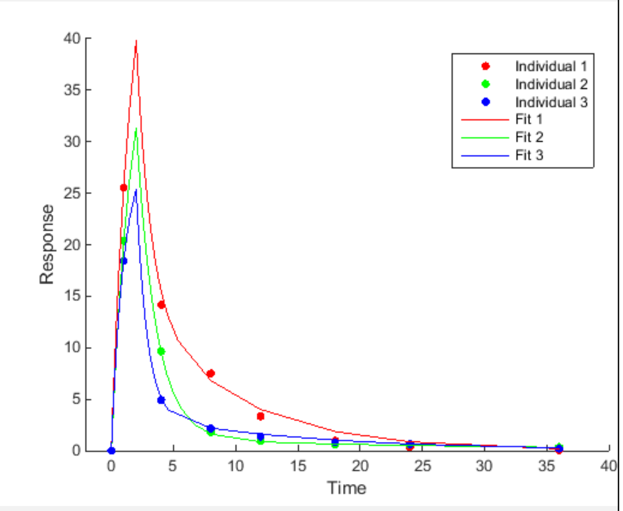
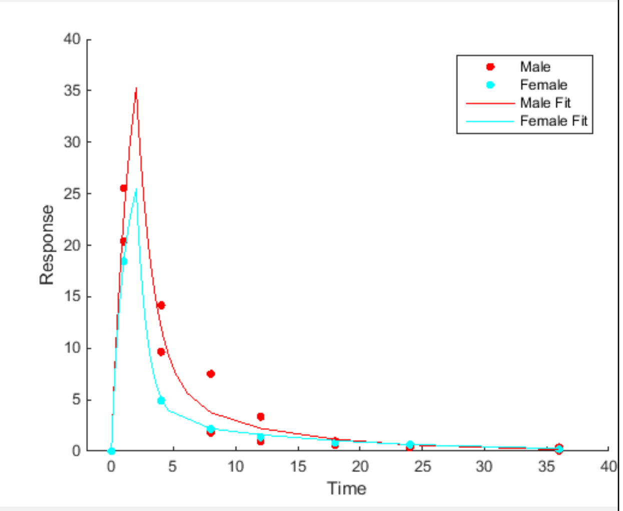
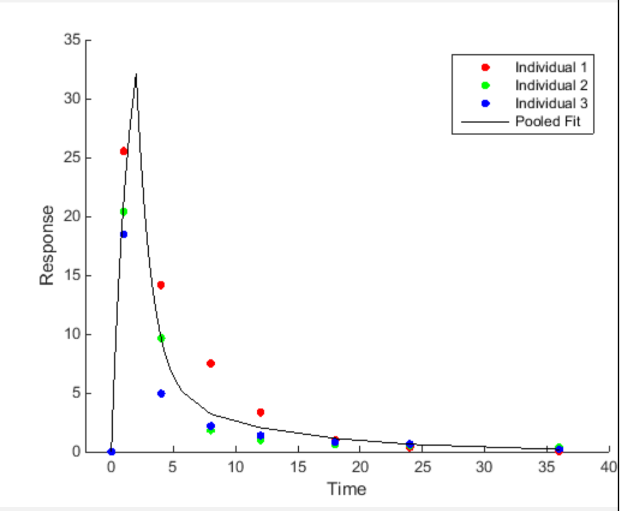
This model describes the time course of drug concentration in the body (C_t), as a function of the drug concentration after an intravenous bolus dose at t = 0 (C_0), time (t), and elimination rate parameter (k_e). ε is the mean-zero and unit-variance variable, that is, $\varepsilon \sim N(0, 1)$ representing the measurement error and a is the error model parameter (here, standard deviation).

More generically, you can write the model as $y_j = f(x_j; p) + g(\varepsilon_j)$, where y_j is the jth response of interest (such as C_t), f is a function of known quantities x (such as C_0 and time t), model parameters p (such as k_e), and an error model $g(\varepsilon_j)$.

If there are multiple observations on multiple individuals, the model becomes $y_{ij} = f(x_{ij}; p_j) + g(\varepsilon_{ij})$ where y_{ij} is the jth observation of the ith individual. Additionally, you can categorize your data into different groups based on different categories such as sex, age, or height.

Fitting Options in SimBiology

This table summarizes nonlinear regression options available in SimBiology.

Fitting Option	Example
Individual-specific parameter estimation (Unpooled fitting) Fit each individual separately, resulting in one set of parameter estimates for each individual.	
Category- or group-specific parameter estimation Fit each category or group separately, resulting in one set of parameter estimates for each category.	
Population-wide parameter estimation (Pooled fitting) Fit all of the data pooled together, resulting in just one set of parameter estimates.	

In addition, SimBiology supports four kinds of error models for measured or observed responses, namely, constant (default), proportional, combined, and exponential. For details, see “Error Models” on page 4-44. Depending on the optimization method, you can specify an error model for each response or all responses. For details, see “Supported Methods for Parameter Estimation in SimBiology” on page 4-42.

Maximum Likelihood Estimation

SimBiology estimates parameters by the method of maximum likelihood. Rather than directly maximizing the likelihood function, SimBiology constructs an equivalent minimization problem. Whenever possible, the estimation is formulated as a weighted least squares (**WLS**) optimization that minimizes the sum of the squares of weighted residuals. Otherwise, the estimation is formulated as the minimization of the negative of the logarithm of the likelihood (**NLL**). The **WLS** formulation often converges better than the **NLL** formulation, and SimBiology can take advantage of specialized **WLS** algorithms, such as the Levenberg-Marquardt algorithm implemented in lsqnonlin and lsqcurvefit. SimBiology uses **WLS** when there is a single error model that is constant, proportional, or exponential. SimBiology uses **NLL** if you have a combined error model or a multiple-error model, that is, a model having an error model for each response.

`sbiofit` supports different optimization methods, and passes in the formulated **WLS** or **NLL** expression to the optimization method that minimizes it.

	Expression that is being minimized
Weighted Least Squares (**WLS**)	For the constant error model, $\sum_i^N (y_i - f_i)^2$
	For the proportional error model, $\sum_i^N \frac{(y_i - f_i)^2}{f_i^2/f_{gm}^2}$
	For the exponential error model, $\sum_i^N (\ln y_i - \ln f_i)^2$
	For numeric weights, $\sum_i^N \frac{(y_i - f_i)^2}{w_{gm}/w_i}$
Negative Log-likelihood (**NLL**)	For the combined error model and multiple-error model, $\sum_i^N \frac{(y_i - f_i)^2}{2\sigma_i^2} + \sum_i^N \ln\sqrt{2\pi\sigma_i^2}$

The variables are defined as follows.

N	Number of experimental observations
y_i	The ith experimental observation
f_i	Predicted value of the ith observation

σ_i	Standard deviation of the ith observation. • For the constant error model, $\sigma_i = a$ • For the proportional error model, $\sigma_i = b\lvert f_i\rvert$ • For the combined error model, $\sigma_i = a + b\lvert f_i\rvert$
f_{gm}	$f_{gm} = \left(\prod_i^N \lvert f_i\rvert\right)^{1/N}$
w_i	The weight of the ith predicted value
w_{gm}	$w_{gm} = \left(\prod_i^N w_i\right)^{1/N}$

When you use numeric weights or the weight function, the weights are assumed to be inversely proportional to the variance of the error, i.e., $\sigma_i^2 = \frac{a^2}{w_i}$ where a is the constant error parameter. If you use weights, you cannot specify an error model except the constant error model.

Various optimization methods have different requirements on the function that is being minimized. For some methods, the estimation of model parameters is performed independently of the estimation of the error model parameters. The following table summarizes the error models and any separate formulas used for the estimation of error model parameters, where a and b are error model parameters and e is the standard mean-zero and unit-variance (Gaussian) variable.

Error Model	Error Parameter Estimation Function
`'constant'`: $y_i = f_i + ae$	$a^2 = \frac{1}{N}\sum_i^N (y_i - f_i)^2$
`'exponential'`: $y_i = f_i\exp(ae)$	$a^2 = \frac{1}{N}\sum_i^N (\ln y_i - \ln f_i)^2$
`'proportional'`: $y_i = f_i + b\lvert f_i\rvert e$	$b^2 = \frac{1}{N}\sum_i^N \left(\frac{y_i - f_i}{f_i}\right)^2$
`'combined'`: $y_i = f_i + (a + b\lvert f_i\rvert)e$	Error parameters are included in the minimization.
Weights	$a^2 = \frac{1}{N}\sum_i^N (y_i - f_i)^2 w_i$

Note `nlinfit` only support single error models, not multiple-error models, that is, response-specific error models. For a combined error model, it uses an iterative **WLS** algorithm. For other error models, it uses the **WLS** algorithm as described previously. For details, see `nlinfit`.

Fitting Workflow for sbiofit

The following steps show one of the workflows you can use at the command line to fit a PK model.

1. Import data.
2. Convert the data to the `groupedData` format.
3. Define dosing data. For details, see "Doses in SimBiology Models" on page 2-30.
4. Create a structural model (one-, two-, or a multicompartment model). For details, see "Create Pharmacokinetic Models" on page 5-13.
5. Map the response variable from data to the model component. For example, if you have the measured drug concentration data for the central compartment, then map it to the drug species in the central compartment (typically the `Drug_Central` species).
6. Specify parameters to estimate using an `EstimatedInfo object`. Optionally, you can specify parameter transformations, initial values, and parameter bounds.
7. Perform parameter estimation using `sbiofit`.

For illustrated examples, see the following.

- "Fit a One-Compartment Model to an Individual's PK Profile"
- "Fit a Two-Compartment Model to PK Profiles of Multiple Individuals"
- "Estimate Category-Specific PK Parameters for Multiple Individuals"

See Also

`EstimatedInfo object` | `groupedData` | `sbiofit` | `sbiofitmixed`

More About

- "Multiple Parameter Estimations in Parallel"
- "Parameter Estimation with Hybrid Solvers"
- "Progress Plot" on page 4-45
- "Supported Methods for Parameter Estimation in SimBiology" on page 4-42

Supported Methods for Parameter Estimation in SimBiology

SimBiology supports a variety of optimization methods for least-squares and mixed-effects estimation problems. Depending on the optimization method, you can specify parameter bounds for estimated parameters as well as response-specific error models, that is, an error model for each response variable. The following table summarizes the supported optimization methods in SimBiology, fitting options, and the corresponding toolboxes that are required in addition to MATLAB and SimBiology.

Method	Additional Toolbox Required	Supports Parameter Bounds	Uses Parameter Sensitivities†	Response-specific Error Models	Fixed or Mixed Effects	Supports Stochastic EM Algorithm	SimBiology Function to Use
`fminsearch`	—	Yes*	No	Yes	Fixed	No	`sbiofit`
scattersearch	—	Yes	Depends on the selected local solver.	Depends on the selected local solver.	Fixed	No	
`nlinfit`	Statistics and Machine Learning Toolbox	Yes*	No	No	Fixed	No	
`fminunc`	Optimization Toolbox	Yes*	Yes	Yes	Fixed	No	
`fmincon`	Optimization Toolbox	Yes	Yes	Yes	Fixed	No	
`lsqcurvefit`	Optimization Toolbox	Yes	Yes	Yes	Fixed	No	
`lsqnonlin`	Optimization Toolbox	Yes	Yes	Yes	Fixed	No	
`patternsearch`	Global Optimization Toolbox	Yes	No	Yes	Fixed	No	
`ga`	Global Optimization Toolbox	Yes	No	Yes	Fixed	No	
`particleswarm`	Global Optimization Toolbox	Yes	No	Yes	Fixed	No	
`nlmefit`	Statistics and Machine Learning Toolbox	No	No	No	Mixed	No	`sbiofitmixed`

Method	Additional Toolbox Required	Supports Parameter Bounds	Uses Parameter Sensitivities[†]	Response-specific Error Models	Fixed or Mixed Effects	Supports Stochastic EM Algorithm	SimBiology Function to Use
`nlmefitsa`	Statistics and Machine Learning Toolbox	No	No	No	Mixed	Yes	

[†] This column indicates whether the algorithm allows using parameter sensitivities to determine gradients of the objective function.

* When using `fminsearch`, `nlinfit`, or `fminunc` with bounds, the objective function returns `Inf` if bounds are exceeded. When you turn on options such as `FunValCheck`, the optimization may error if bounds are exceeded during estimation. If using `nlinfit`, it may report warnings about the Jacobian being ill-conditioned or not being able to estimate if the final result is too close to the bounds.

See Also

`sbiofit` | `sbiofitmixed`

More About

- "Nonlinear Regression"
- "Nonlinear Mixed-Effects Modeling"

Error Models

SimBiology supports the error models described in the following table. For instance, if you assume every observation has a constant amount of noise, use the constant error model, which is the default. Instead, if you assume the error is proportional to the response data, then the proportional error model might be more appropriate.

Error Model	Mathematical Representation	Standard Deviation of Error Model
constant (default)	$y = f + a\varepsilon$	a
proportional	$y = f + b\lvert f\rvert\varepsilon$	$b\lvert f\rvert$
combined	$y = f + (a + b\lvert f\rvert)\varepsilon$	$a+b\lvert f\rvert$
exponential	$y = f * \exp(a\varepsilon)$ or equivalently,	$\sqrt{e^{a^2} - 1} * e^{a}$
	$\log(y) = \log(f) + a\varepsilon$	a
Here, y is the response, f is the function value, ε is a standard mean-zero and unit-variance (Gaussian) variable, and a and b are error parameters. For instance, if you assume the error is approximately 5% of each observation, use the proportional error model with b = 0.05. In SimBiology, f typically represents the simulation result.		

See Also

More About

- “Nonlinear Regression”
- “Nonlinear Mixed-Effects Modeling”

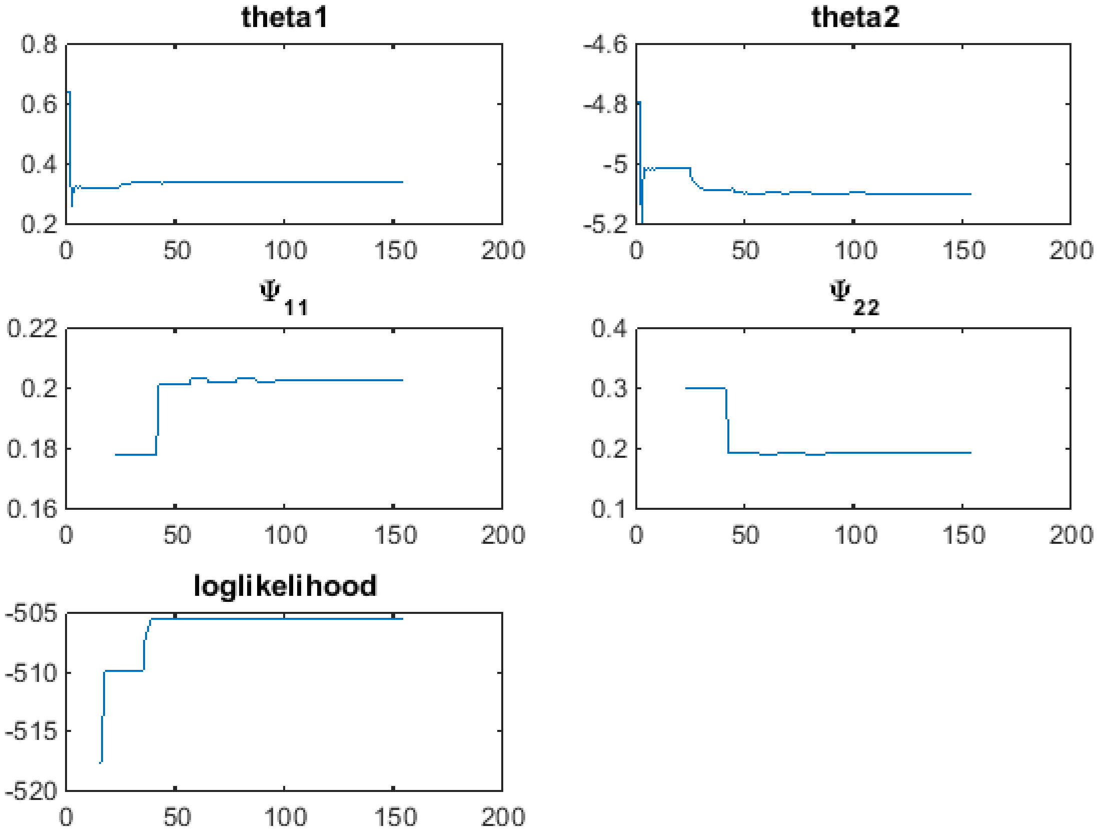
Progress Plot

The progress plot provides the live feedback on the status of parameter estimation while using `sbiofit`, `sbiofitmixed`, or the **Fit Data** program in the **SimBiology Model Analyzer** app. When you enable this feature, a new figure opens and shows the fitting quality measures such as log-likelihood and estimated parameter values for each function iteration. The plot monitors the progress whether you are running the fit on a local machine or in parallel using remote clusters.

When you estimate parameters, you can specify which estimation method on page 4-42 to use during the fitting. The progress plot is shown for all the estimation methods except for `nlinfit`. However, the progress plot differs depending on whether you are using a nonlinear mixed-effects method (`nlmefit` or `nlmefitsa`) or a nonlinear regression method, such as `lsqnonlin`.

Progress Plot for Nonlinear Mixed-Effects Methods

The progress plot figure contains a series of subplots. Specifically, the subplots show the values of fixed-effect parameters (theta), the estimates of the variance parameters, that is, the diagonal elements of the covariance matrix of the random effects (Ψ), and the log-likelihood.



Here are some tips for interpreting the plots.

- The fitting function tries to maximize the log-likelihood. When the plot begins to display a flat line, this might indicate that maximization is complete. Try setting the maximum iterations to a lower number to reduce the number of iterations you need and improve performance.
- Plots for the fixed effects (`theta`s) and the variance parameters (Ψs) should show convergence. If you see oscillations, or jumps without accompanying improvements in the log-likelihood, the model may be overparameterized. Try the following:

  - Reduce the number of fixed effects.
  - Reduce the number of random effects.
  - Simplify the covariance matrix pattern of random effects (if you have previously changed it from the default diagonal matrix).

Progress Plot for Nonlinear Regression Methods

The progress plot figure shows a series of subplots, and there are two categories of plots: quality measure plots on page 4-46 and estimated parameter plots on page 4-49. For a pooled fit, that is, estimating one set of parameter values for all groups (or individuals), there is only one line for each plot and the line is faded when the fit is finished. For an unpooled fit, that is, estimating one set of parameter values for each group (or individual), each line represents a single individual or group. You can select one or more lines by clicking and dragging the mouse cursor to create a rectangle on any plot. All lines that intersect the rectangle are selected and highlighted across all plots.

You can terminate the fitting at any time by selecting **Stop**, and partial results are returned. Specifically, for a pooled fit, the result up to the last function iteration is returned. For an unpooled fit, results for any groups that have finished running are returned. The groups currently running are interrupted and partial results from the last iteration are also returned.

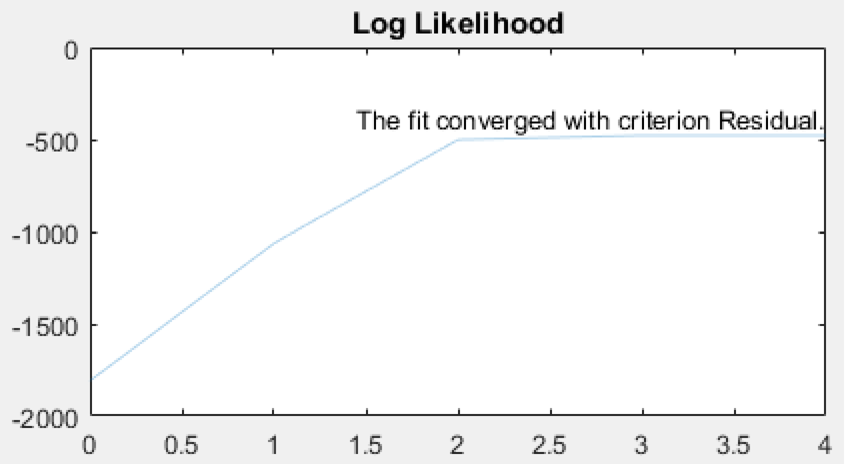
Quality Measure Plots

The quality measure plots include the log-likelihood, first-order optimality, and termination condition plots. They occupy the first row of the figure.

Log-likelihood

The estimation method tries to maximize the log-likelihood, and the plot shows the log-likelihood value for each function iteration. When the plot begins to display a flat line, it often indicates that maximization is complete. Try setting the maximum iterations to a lower number to reduce the number of iterations you need and improve performance.

For a pooled fit, there is only one line in the plot and the line is faded when the fit finishes. The log-likelihood plot shows whether the fit converges or fails along with the information on the estimation method termination condition. The next figure is an example of the log-likelihood plot of a pooled fit.



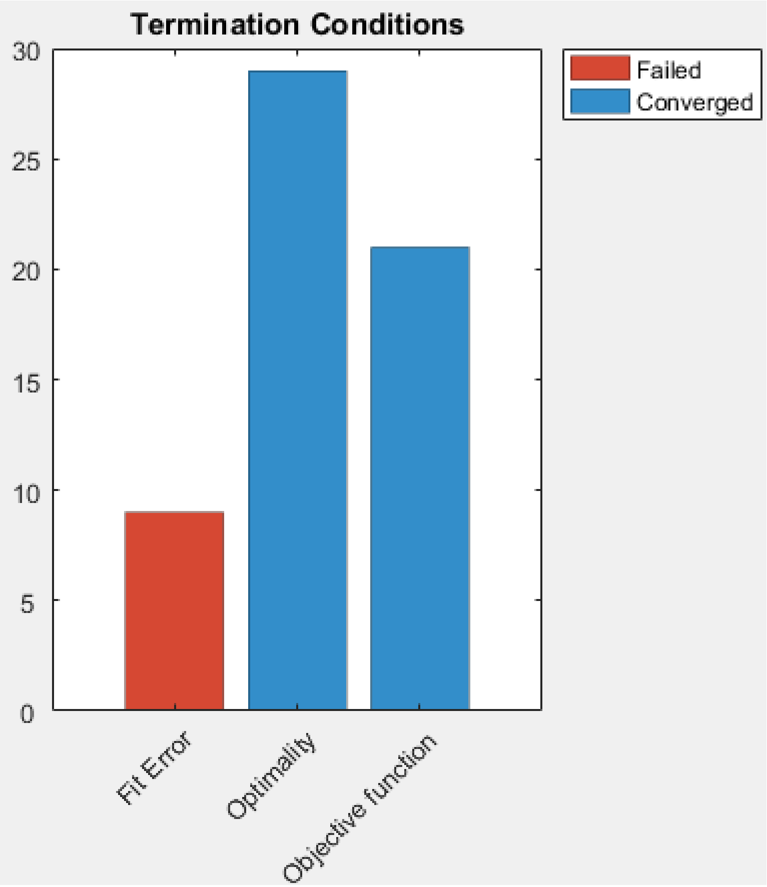
First-order Optimality

First-order optimality is a measure of how close a point x is to optimal, and the plot is shown when you are using the Optimization Toolbox methods (`lsqnonlin`, `lsqcurvefit`, `fminunc`, and `fmincon`). The first-order optimality measure must be zero at a minimum, but a point with first-order optimality equal to zero is not necessarily a minimum. For details, see First-order optimality (Optimization Toolbox).

Termination Condition

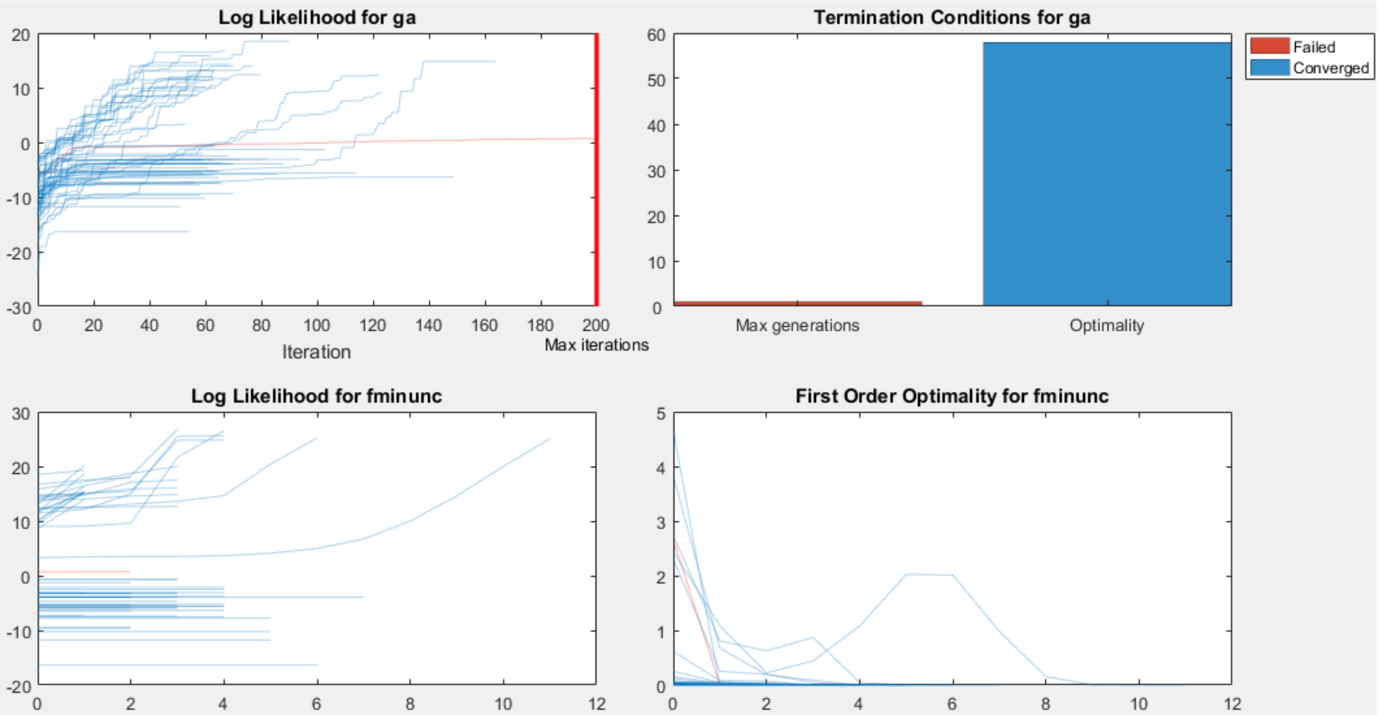
For a pooled fit, the termination condition is displayed together with the log-likelihood plot. For details about the termination condition, refer to the `exitflag` output argument description of the corresponding estimation method. Suppose that you are using the `lsqnonlin` method and see a message: `The fit converged with criterion Residual`. By checking the `exitflag` conditions of the `lsqnonlin` with the keyword `Residual`, this termination condition corresponds to the `exitflag` value of `3`, that is, change in the residual was less than the specified tolerance.

For an unpooled fit, the **Termination Conditions** plot contains the summary (histogram) of termination criteria for all groups (or individuals) as shown in the next figure. The *y*-axis represents the total number of fits for each termination condition, and the *x*-axis displays all the termination criteria.



Hybrid Functions

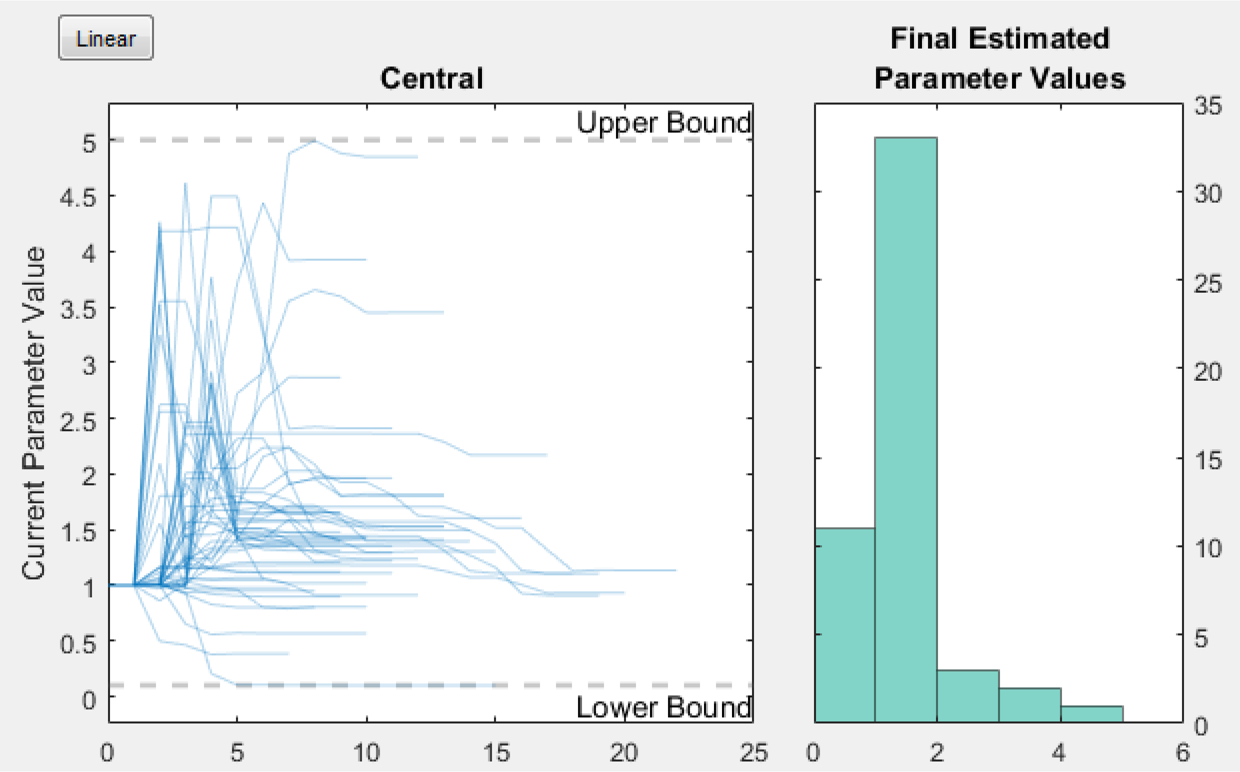
If you are performing a hybrid optimization by first running a global solver, such as `ga` or `particleswarm`, followed by a hybrid function, the `ProgressPlot` also shows the quality measure plots for the hybrid function in the second row. The following figure is an example where the global optimization algorithm is `ga` and the hybrid function is `fminunc`. For an illustrated example, see “Parameter Estimation with Hybrid Solvers”.



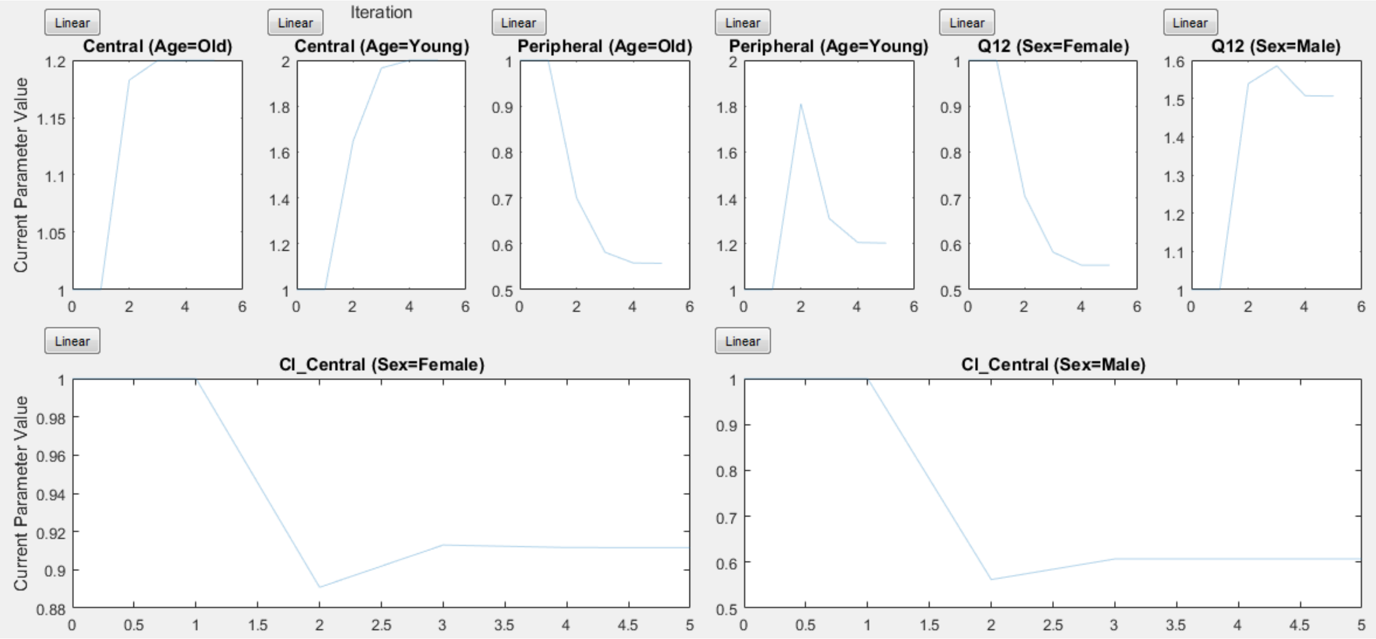
Estimated Parameter Plot

This plot displays the value of the estimated parameter versus iteration for each group. One estimated parameter plot is displayed for each parameter. The plots start on the second row of the figure and can span multiple rows. Each plot displays a horizontal dashed line for any lower or upper bound you specify for the estimated parameter. The bound lines show only if the range of the plot can include the lines.

For an unpooled fit, the Progress Plot also displays a histogram that shows the distribution of the parameter values for the completed runs. Use the toggle button over the *y*-axis for each plot to switch between the log and linear scale. The next figure shows an example of an estimated parameter plot with the bound information and distribution of estimated values.

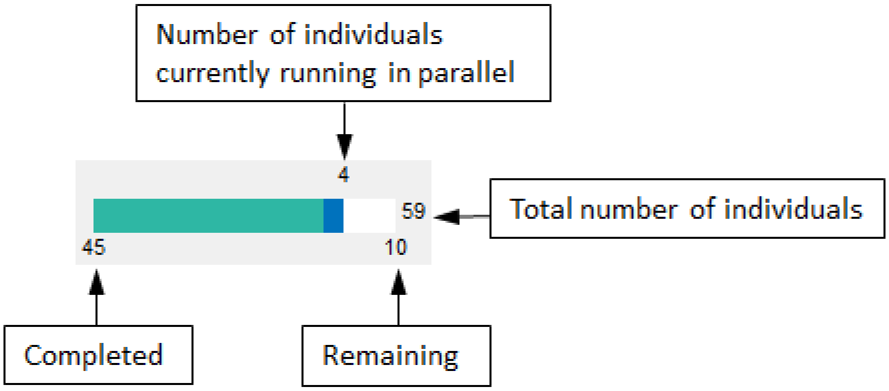


If you have a hierarchical model and are estimating parameters for each category such as estimating parameters for males versus females, the Progress Plot displays one plot per estimated parameter for each category. For example, in the next figure, the *Central* and *Peripheral* parameters are estimated for the age categories while *Q12* and *Cl_Central* are estimated for the sex categories.



Status Bar

For an unpooled fit running in parallel, the Progress Plot displays a status bar in the bottom right corner. The bar shows information about the remaining and completed number of individuals (or groups) throughout the fit.


Number of individuals currently running in parallel
4
59
Total number of individuals
45
10
Completed
Remaining

See Also

`sbiofit` | `sbiofitmixed` | `sbiofitstatusplot`

More About

- "Nonlinear Regression" on page 4-37
- "Nonlinear Mixed-Effects Modeling" on page 4-31
- "Supported Methods for Parameter Estimation in SimBiology" on page 4-42

Fit a One-Compartment Model to an Individual's PK Profile

Background

This example shows how to fit an individual's PK profile data to one-compartment model and estimate pharmacokinetic parameters.

Suppose you have drug plasma concentration data from an individual and want to estimate the volume of the central compartment and the clearance. Assume the drug concentration versus the time profile follows the monoexponential decline $C_t = C_0 e^{-k_e t}$, where C_t is the drug concentration at time t, C_0 is the initial concentration, and k_e is the elimination rate constant that depends on the clearance and volume of the central compartment $k_e = Cl/V$.

The synthetic data in this example was generated using the following model and parameters:

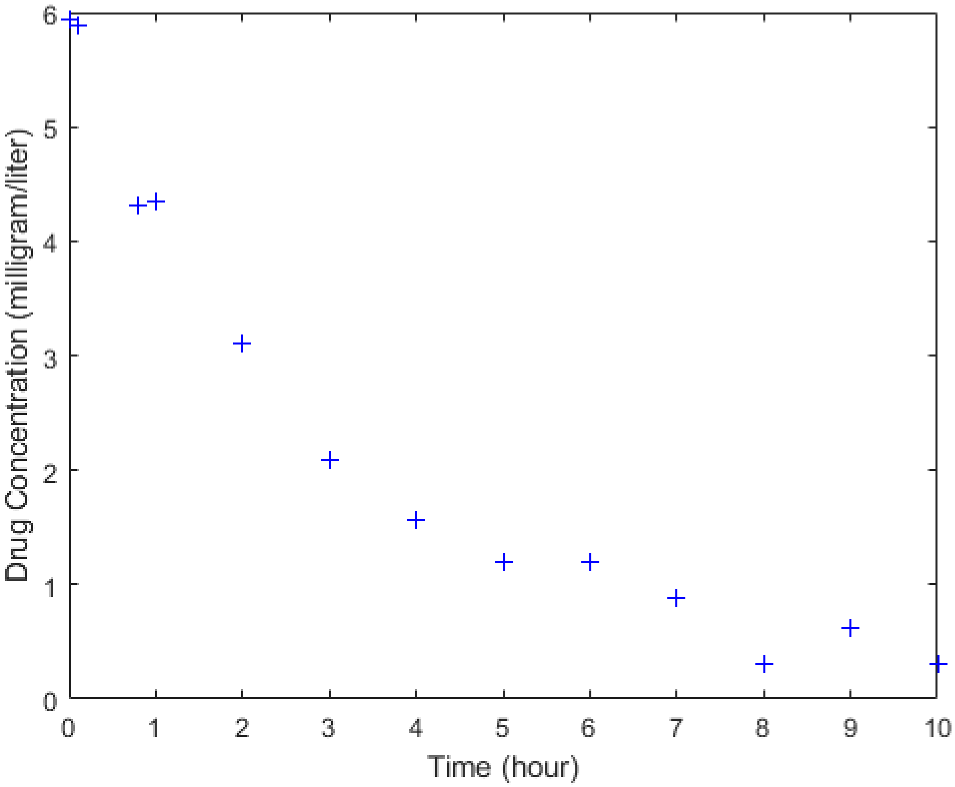
- One-compartment model with bolus dosing and first-order elimination
- Volume of the central compartment (`Central`) = 1.70 liter
- Clearance parameter (`Cl_Central`) = 0.55 liter/hour
- Constant error model

Load Data and Visualize

The data is stored as a table with variables `Time` and `Conc` that represent the time course of the plasma concentration of an individual after an intravenous bolus administration measured at 13 different time points. The variable units for `Time` and `Conc` are hour and milligram/liter, respectively.

```
clear all
load('data15.mat')
```

```
plot(data.Time,data.Conc,'b+')
xlabel('Time (hour)');
ylabel('Drug Concentration (milligram/liter)');
```



Convert to groupedData Format

Convert the data set to a `groupedData` object, which is the required data format for the fitting function `sbiofit` for later use. A `groupedData` object also lets you set independent variable and group variable names (if they exist). Set the units of the `Time` and `Conc` variables. The units are optional and only required for the `UnitConversion` feature, which automatically converts matching physical quantities to one consistent unit system.

```
gData = groupedData(data);
gData.Properties.VariableUnits = {'hour','milligram/liter'};
gData.Properties
```

```
ans = struct with fields:
                Description: ''
                   UserData: []
             DimensionNames: {'Row'  'Variables'}
              VariableNames: {'Time'  'Conc'}
       VariableDescriptions: {}
              VariableUnits: {'hour'  'milligram/liter'}
         VariableContinuity: []
                   RowNames: {}
           CustomProperties: [1x1 matlab.tabular.CustomProperties]
          GroupVariableName: ''
    IndependentVariableName: 'Time'
```

groupedData automatically set the name of the IndependentVariableName property to the Time variable of the data.

Construct a One-Compartment Model

Use the built-in PK library to construct a one-compartment model with bolus dosing and first-order elimination where the elimination rate depends on the clearance and volume of the central compartment. Use the configset object to turn on unit conversion.

```
pkmd                    = PKModelDesign;
pkc1                    = addCompartment(pkmd,'Central');
pkc1.DosingType         = 'Bolus';
pkc1.EliminationType    = 'linear-clearance';
pkc1.HasResponseVariable = true;
model                   = construct(pkmd);
configset               = getconfigset(model);
configset.CompileOptions.UnitConversion = true;
```

For details on creating compartmental PK models using the SimBiology® built-in library, see “Create Pharmacokinetic Models” on page 5-13.

Define Dosing

Define a single bolus dose of 10 milligram given at time = 0. For details on setting up different dosing schedules, see “Doses in SimBiology Models” on page 2-30.

```
dose                = sbiodose('dose');
dose.TargetName     = 'Drug_Central';
dose.StartTime      = 0;
dose.Amount         = 10;
dose.AmountUnits    = 'milligram';
dose.TimeUnits      = 'hour';
```

Map Response Data to the Corresponding Model Component

The data contains drug concentration data stored in the Conc variable. This data corresponds to the Drug_Central species in the model. Therefore, map the data to Drug_Central as follows.

```
responseMap = {'Drug_Central = Conc'};
```

Specify Parameters to Estimate

The parameters to fit in this model are the volume of the central compartment (Central) and the clearance rate (Cl_Central). In this case, specify log-transformation for these biological parameters since they are constrained to be positive. The estimatedInfo object lets you specify parameter transforms, initial values, and parameter bounds if needed.

```
paramsToEstimate    = {'log(Central)','log(Cl_Central)'};
estimatedParams     = estimatedInfo(paramsToEstimate,'InitialValue',[1 1],'Bounds',[1 5;0.5 2]);
```

Estimate Parameters

Now that you have defined one-compartment model, data to fit, mapped response data, parameters to estimate, and dosing, use sbiofit to estimate parameters. The default estimation function that sbiofit uses will change depending on which toolboxes are available. To see which function was used during fitting, check the EstimationFunction property of the corresponding results object.

```
fitConst = sbiofit(model,gData,responseMap,estimatedParams,dose);
```

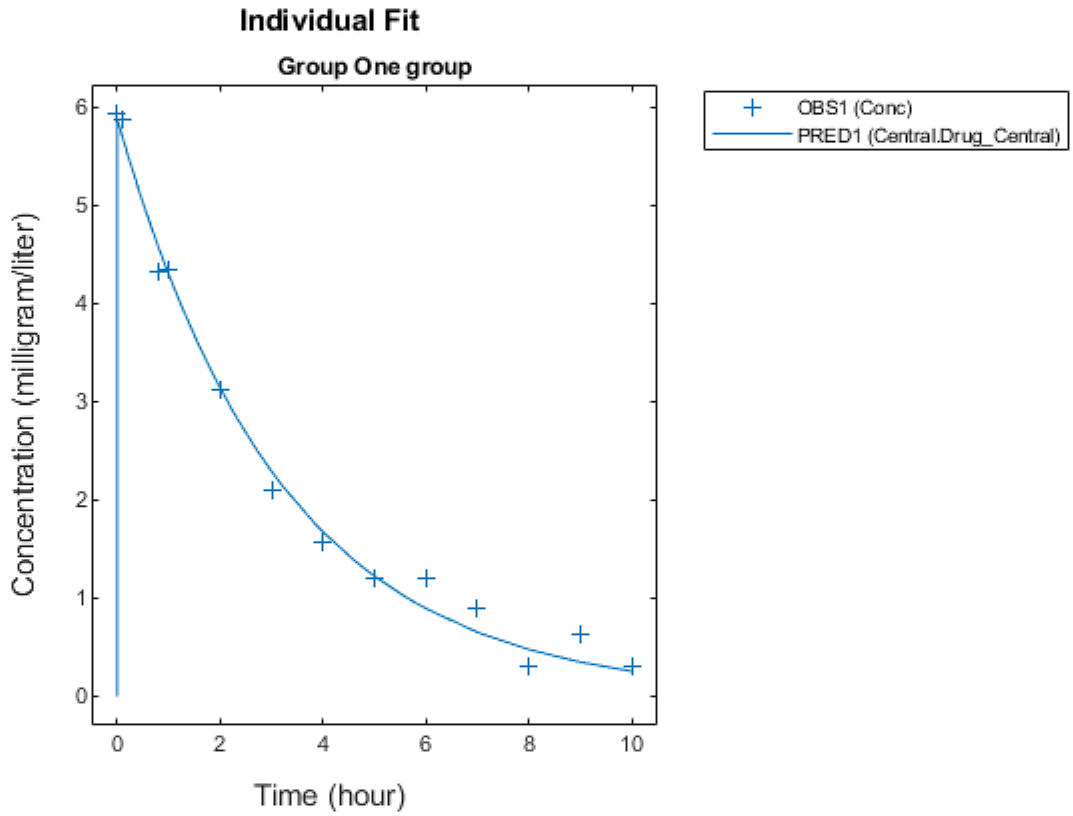
Display Estimated Parameters and Plot Results

Notice the parameter estimates were not far off from the true values (1.70 and 0.55) that were used to generate the data. You may also try different error models to see if they could further improve the parameter estimates.

```
fitConst.ParameterEstimates
```

```
ans=2×4 table
         Name          Estimate    StandardError      Bounds  
    _______________    ________    _____________    __________

    {'Central'    }     1.6993       0.034821         1      5
    {'Cl_Central'}     0.53358        0.01968       0.5      2
```

```
s.Labels.XLabel     = 'Time (hour)';
s.Labels.YLabel     = 'Concentration (milligram/liter)';
plot(fitConst,'AxesStyle',s);
```



Use Different Error Models

Try three other supported error models (proportional, combination of constant and proportional error models, and exponential).

```
fitProp = sbiofit(model,gData,responseMap,estimatedParams,dose,...
                      'ErrorModel','proportional');
fitExp  = sbiofit(model,gData,responseMap,estimatedParams,dose,...
                      'ErrorModel','exponential');
fitComb = sbiofit(model,gData,responseMap,estimatedParams,dose,...
                      'ErrorModel','combined');
```

Use Weights Instead of an Error Model

You can specify weights as a numeric matrix, where the number of columns corresponds to the number of responses. Setting all weights to 1 is equivalent to the constant error model.

```
weightsNumeric = ones(size(gData.Conc));
fitWeightsNumeric = sbiofit(model,gData,responseMap,estimatedParams,dose,'Weights',weightsNumeric
```

Alternatively, you can use a function handle that accepts a vector of predicted response values and returns a vector of weights. In this example, use a function handle that is equivalent to the proportional error model.

```
weightsFunction = @(y) 1./y.^2;
fitWeightsFunction = sbiofit(model,gData,responseMap,estimatedParams,dose,'Weights',weightsFuncti
```

Compare Information Criteria for Model Selection

Compare the loglikelihood, AIC, and BIC values of each model to see which error model best fits the data. A larger likelihood value indicates the corresponding model fits the model better. For AIC and BIC, the smaller values are better.

```
allResults = [fitConst,fitWeightsNumeric,fitWeightsFunction,fitProp,fitExp,fitComb];
errorModelNames = {'constant error model','equal weights','proportional weights', ...
                   'proportional error model','exponential error model',...
                   'combined error model'};
LogLikelihood = [allResults.LogLikelihood]';
AIC = [allResults.AIC]';
BIC = [allResults.BIC]';
t = table(LogLikelihood,AIC,BIC);
t.Properties.RowNames = errorModelNames;
t
```

```
t=6×3 table
                                LogLikelihood      AIC        BIC   
                                _____________    _______    _______

    constant error model            3.9866       -3.9732    -2.8433
    equal weights                   3.9866       -3.9732    -2.8433
    proportional weights           -3.8472        11.694     12.824
    proportional error model       -3.8257        11.651     12.781
    exponential error model         1.1984        1.6032     2.7331
    combined error model            3.9163       -3.8326    -2.7027
```

Based on the information criteria, the constant error model (or equal weights) fits the data best since it has the largest loglikelihood value and the smallest AIC and BIC.

Display Estimated Parameter Values

Show the estimated parameter values of each model.

```
Estimated_Central       = zeros(6,1);
Estimated_Cl_Central    = zeros(6,1);
t2 = table(Estimated_Central,Estimated_Cl_Central);
```

```
t2.Properties.RowNames = errorModelNames;
for i = 1:height(t2)
    t2{i,1} = allResults(i).ParameterEstimates.Estimate(1);
    t2{i,2} = allResults(i).ParameterEstimates.Estimate(2);
end
t2
```

```
t2=6×2 table
                                Estimated_Central    Estimated_Cl_Central
                                _________________    ____________________

    constant error model             1.6993                0.53358       
    equal weights                    1.6993                0.53358       
    proportional weights             1.9045                0.51734       
    proportional error model         1.8777                0.51147       
    exponential error model          1.7872                0.51701       
    combined error model             1.7008                0.53271       
```

Conclusion

This example showed how to estimate PK parameters, namely the volume of the central compartment and clearance parameter of an individual, by fitting the PK profile data to one-compartment model. You compared the information criteria of each model and estimated parameter values of different error models to see which model best explained the data. Final fitted results suggested both the constant and combined error models provided the closest estimates to the parameter values used to generate the data. However, the constant error model is a better model as indicated by the loglikelihood, AIC, and BIC information criteria.

See Also

`sbiofit`

More About

- “Fit a Two-Compartment Model to PK Profiles of Multiple Individuals” on page 4-71
- “Estimate Category-Specific PK Parameters for Multiple Individuals” on page 4-58
- “Nonlinear Regression” on page 4-37

Estimate Category-Specific PK Parameters for Multiple Individuals

This example shows how to estimate category-specific (such as young versus old, male versus female), individual-specific, and population-wide parameters using PK profile data from multiple individuals.

Background

Suppose you have drug plasma concentration data from 30 individuals and want to estimate pharmacokinetic parameters, namely the volumes of central and peripheral compartment, the clearance, and intercompartmental clearance. Assume the drug concentration versus the time profile follows the biexponential decline $C_t = Ae^{-at} + Be^{-bt}$, where C_t is the drug concentration at time t, and a and b are slopes for corresponding exponential declines.

Load Data

This synthetic data contains the time course of plasma concentrations of 30 individuals after a bolus dose (100 mg) measured at different times for both central and peripheral compartments. It also contains categorical variables, namely Sex and Age.

```
clear
load('sd5_302RAgeSex.mat')
```

Convert to groupedData Format

Convert the data set to a groupedData object, which is the required data format for the fitting function sbiofit. A groupedData object also allows you set independent variable and group variable names (if they exist). Set the units of the ID, Time, CentralConc, PeripheralConc, Age, and Sex variables. The units are optional and only required for the UnitConversion feature, which automatically converts matching physical quantities to one consistent unit system.

```
gData = groupedData(data);
gData.Properties.VariableUnits = {'','hour','milligram/liter','milligram/liter','',''};
gData.Properties
```

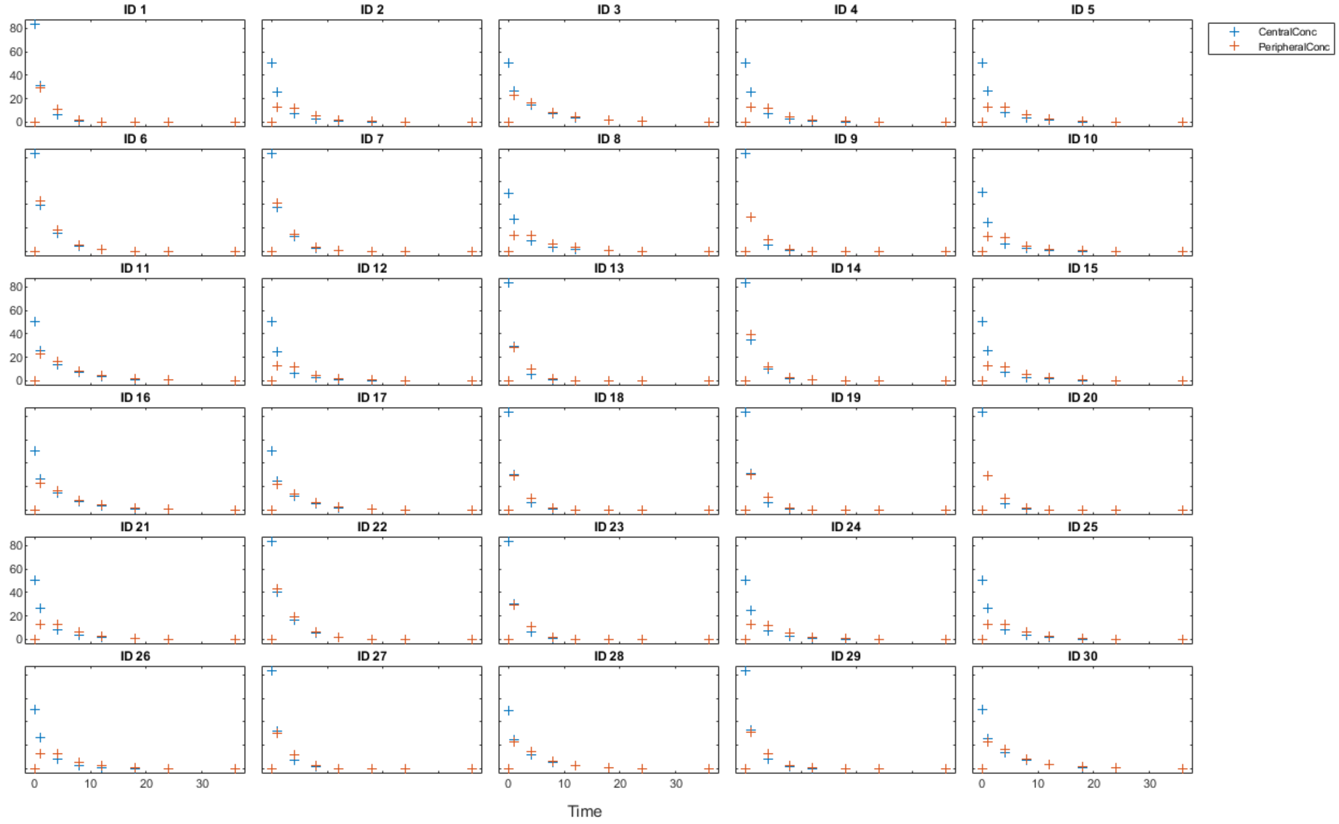
```
ans = struct with fields:
                Description: ''
                   UserData: []
             DimensionNames: {'Row'  'Variables'}
              VariableNames: {1x6 cell}
       VariableDescriptions: {}
              VariableUnits: {1x6 cell}
         VariableContinuity: []
                   RowNames: {}
           CustomProperties: [1x1 matlab.tabular.CustomProperties]
          GroupVariableName: 'ID'
    IndependentVariableName: 'Time'
```

The IndependentVariableName and GroupVariableName properties have been automatically set to the Time and ID variables of the data.

Visualize Data

Display the response data for each individual.

```
t = sbiotrellis(gData,'ID','Time',{'CentralConc','PeripheralConc'},...
                'Marker','+','LineStyle','none');
% Resize the figure.
t.hFig.Position(:) = [100 100 1280 800];
```



Set Up a Two-Compartment Model

Use the built-in PK library to construct a two-compartment model with infusion dosing and first-order elimination where the elimination rate depends on the clearance and volume of the central compartment. Use the `configset` object to turn on unit conversion.

```
pkmd                                    = PKModelDesign;
pkc1                                    = addCompartment(pkmd,'Central');
pkc1.DosingType                         = 'Bolus';
pkc1.EliminationType                    = 'linear-clearance';
pkc1.HasResponseVariable                = true;
pkc2                                    = addCompartment(pkmd,'Peripheral');
model                                   = construct(pkmd);
configset                               = getconfigset(model);
configset.CompileOptions.UnitConversion = true;
```

For details on creating compartmental PK models using the SimBiology® built-in library, see “Create Pharmacokinetic Models” on page 5-13.

Define Dosing

Assume every individual receives a bolus dose of 100 mg at time = 0. For details on setting up different dosing strategies, see “Doses in SimBiology Models” on page 2-30.

```
dose             = sbiodose('dose','TargetName','Drug_Central');
dose.StartTime   = 0;
dose.Amount      = 100;
dose.AmountUnits = 'milligram';
dose.TimeUnits   = 'hour';
```

Map the Response Data to Corresponding Model Components

The data contains measured plasma concentration in the central and peripheral compartments. Map these variables to the appropriate model components, which are `Drug_Central` and `Drug_Peripheral`.

```
responseMap = {'Drug_Central = CentralConc','Drug_Peripheral = PeripheralConc'};
```

Specify Parameters to Estimate

Specify the volumes of central and peripheral compartments `Central` and `Peripheral`, intercompartmental clearance `Q12`, and clearance `Cl_Central` as parameters to estimate. The `estimatedInfo` object lets you optionally specify parameter transforms, initial values, and parameter bounds. Since both `Central` and `Peripheral` are constrained to be positive, specify a log-transform for each parameter.

```
paramsToEstimate    = {'log(Central)', 'log(Peripheral)', 'Q12', 'Cl_Central'};
estimatedParam      = estimatedInfo(paramsToEstimate,'InitialValue',[1 1 1 1]);
```

Estimate Individual-Specific Parameters

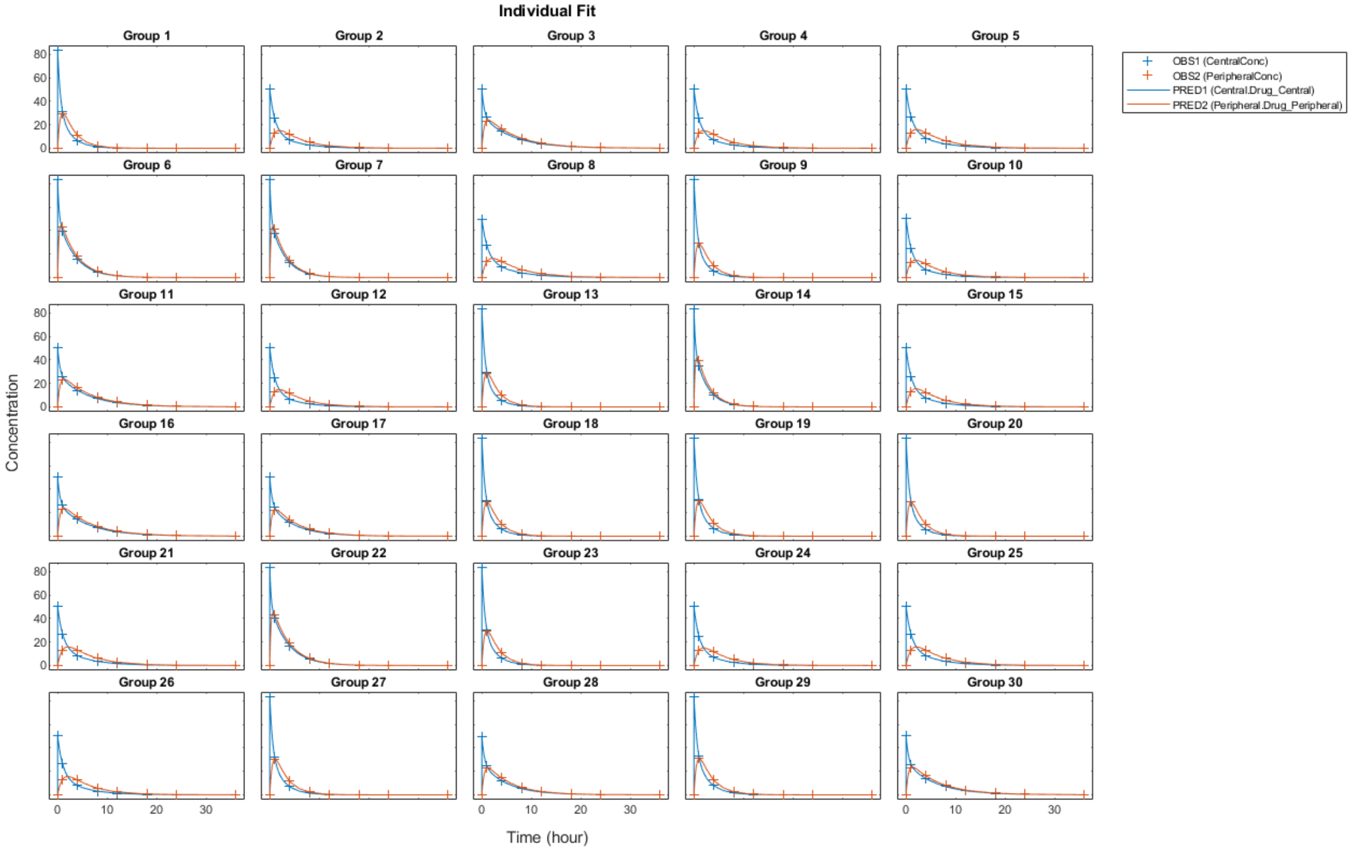
Estimate one set of parameters for each individual by setting the `'Pooled'` name-value pair argument to `false`.

```
unpooledFit =  sbiofit(model,gData,responseMap,estimatedParam,dose,'Pooled',false);
```

Display Results

Plot the fitted results versus the original data for each individual (group).

```
t = plot(unpooledFit);
% Resize the figure.
t.hFig.Position(:) = [100 100 1280 800];
```



For an unpooled fit, `sbiofit` always returns one results object for each individual.

Examine Parameter Estimates for Category Dependencies

Explore the unpooled estimates to see if there is any category-specific parameters, that is, if some parameters are related to one or more categories. If there are any category dependencies, it might be possible to reduce the number of degrees of freedom by estimating just category-specific values for those parameters.

First extract the ID and category values for each ID

```
catParamValues = unique(gData(:,{'ID','Sex','Age'}));
```

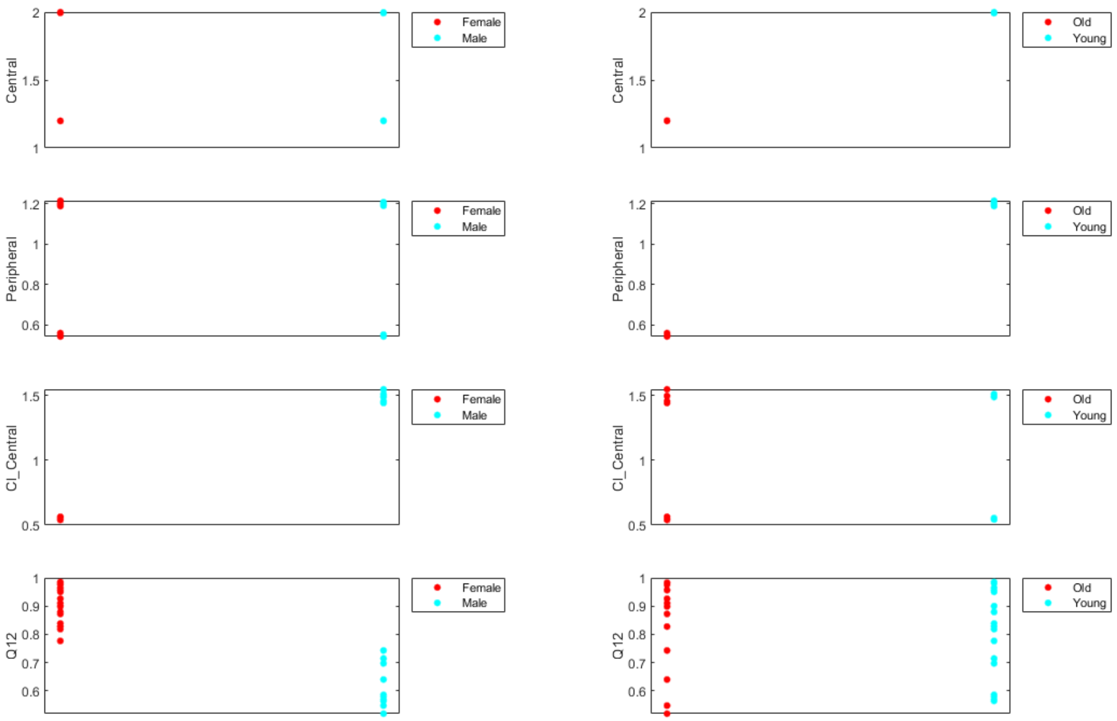
Add variables to the table containing each parameter's estimate.

```
allParamValues            = vertcat(unpooledFit.ParameterEstimates);
catParamValues.Central    = allParamValues.Estimate(strcmp(allParamValues.Name, 'Central'));
catParamValues.Peripheral = allParamValues.Estimate(strcmp(allParamValues.Name, 'Peripheral'));
catParamValues.Q12        = allParamValues.Estimate(strcmp(allParamValues.Name, 'Q12'));
catParamValues.Cl_Central = allParamValues.Estimate(strcmp(allParamValues.Name, 'Cl_Central'));
```

Plot estimates of each parameter for each category. `gscatter` requires Statistics and Machine Learning Toolbox™. If you do not have it, use other alternative plotting functions such as `plot`.

```
h           = figure;
ylabels     = {'Central','Peripheral','Cl\_Central','Q12'};
plotNumber  = 1;
```

```
for i = 1:4
    thisParam = estimatedParam(i).Name;

    % Plot for Sex category
    subplot(4,2,plotNumber);
    plotNumber  = plotNumber + 1;
    gscatter(double(catParamValues.Sex), catParamValues.(thisParam), catParamValues.Sex);
    ax          = gca;
    ax.XTick    = [];
    ylabel(ylabels(i));
    legend('Location','bestoutside')
    % Plot for Age category
    subplot(4,2,plotNumber);
    plotNumber  = plotNumber + 1;
    gscatter(double(catParamValues.Age), catParamValues.(thisParam), catParamValues.Age);
    ax          = gca;
    ax.XTick    = [];
    ylabel(ylabels(i));
    legend('Location','bestoutside')
end
% Resize the figure.
h.Position(:) = [100 100 1280 800];
```



Based on the plot, it seems that young individuals tend to have higher volumes of central and peripheral compartments (`Central`, `Peripheral`) than old individuals (that is, the volumes seem to be age-specific). In addition, males tend to have higher clearance rates (Cl_Central) than females but the opposite for the Q12 parameter (that is, the clearance and Q12 seem to be sex-specific).

Estimate Category-Specific Parameters

Use the 'CategoryVariableName' property of the estimatedInfo object to specify which category to use during fitting. Use 'Sex' as the group to fit for the clearance Cl_Central and Q12 parameters. Use 'Age' as the group to fit for the Central and Peripheral parameters.

```
estimatedParam(1).CategoryVariableName = 'Age';
estimatedParam(2).CategoryVariableName = 'Age';
estimatedParam(3).CategoryVariableName = 'Sex';
estimatedParam(4).CategoryVariableName = 'Sex';
categoryFit = sbiofit(model,gData,responseMap,estimatedParam,dose)
```

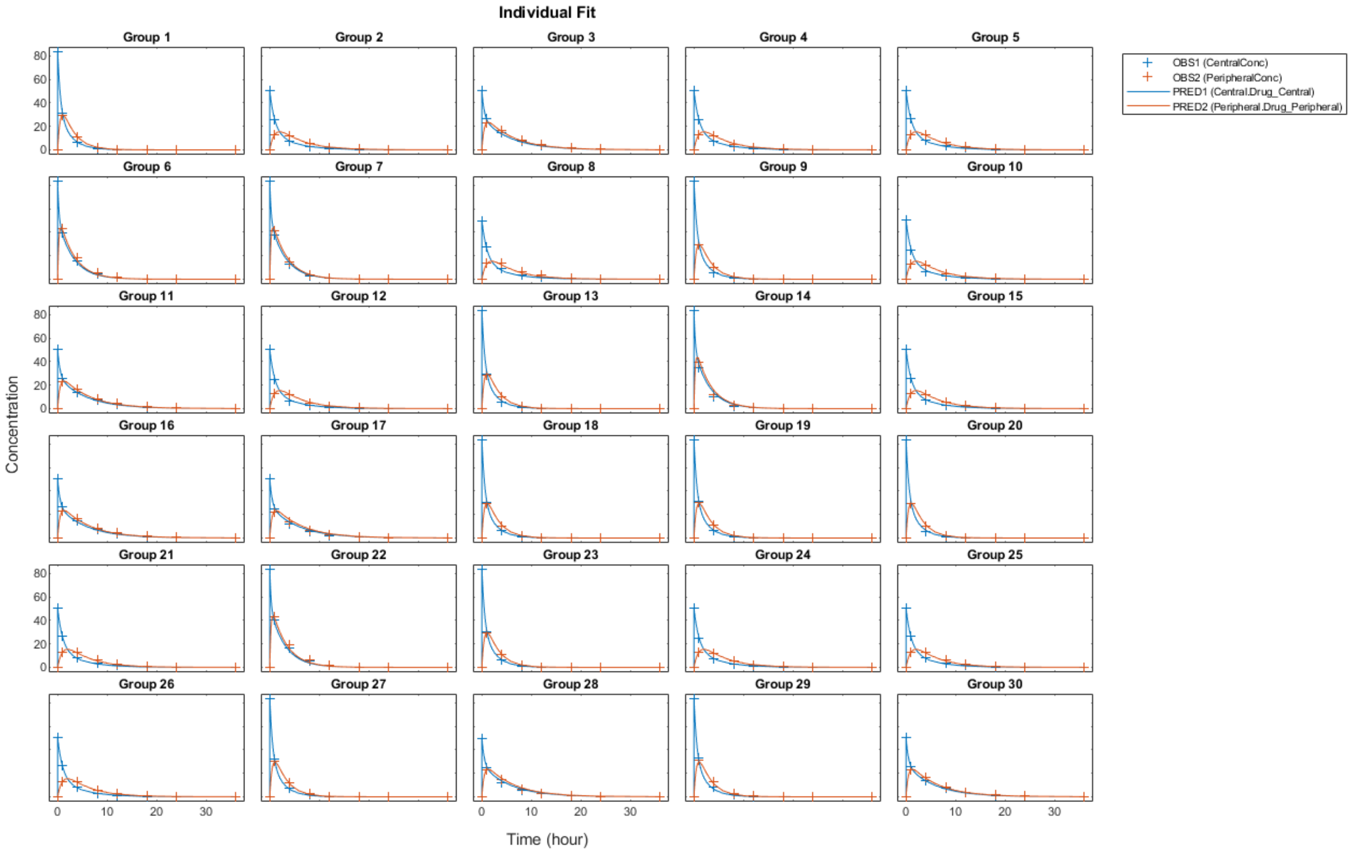
```
categoryFit = 
  OptimResults with properties:

                   ExitFlag: 3
                     Output: [1x1 struct]
                  GroupName: []
                       Beta: [8x5 table]
         ParameterEstimates: [120x6 table]
                          J: [240x8x2 double]
                       COVB: [8x8 double]
           CovarianceMatrix: [8x8 double]
                          R: [240x2 double]
                        MSE: 0.4362
                        SSE: 205.8690
                    Weights: []
              LogLikelihood: -477.9195
                        AIC: 971.8390
                        BIC: 1.0052e+03
                        DFE: 472
             DependentFiles: {1x3 cell}
    EstimatedParameterNames: {'Central'  'Peripheral'  'Q12'  'Cl_Central'}
             ErrorModelInfo: [1x3 table]
         EstimationFunction: 'lsqnonlin'
```

When fitting by category (or group), sbiofit always returns one results object, not one for each category level. This is because both male and female individuals are considered to be part of the same optimization using the same error model and error parameters, similarly for the young and old individuals.

Plot Results

Plot the category-specific estimated results.

```
t = plot(categoryFit);
% Resize the figure.
t.hFig.Position(:) = [100 100 1280 800];
```



For the `Cl_Central` and `Q12` parameters, all males had the same estimates, and similarly for the females. For the `Central` and `Peripheral` parameters, all young individuals had the same estimates, and similarly for the old individuals.

Estimate Population-Wide Parameters

To better compare the results, fit the model to all of the data pooled together, that is, estimate one set of parameters for all individuals by setting the `'Pooled'` name-value pair argument to `true`. The warning message tells you that this option will ignore any category-specific information (if they exist).

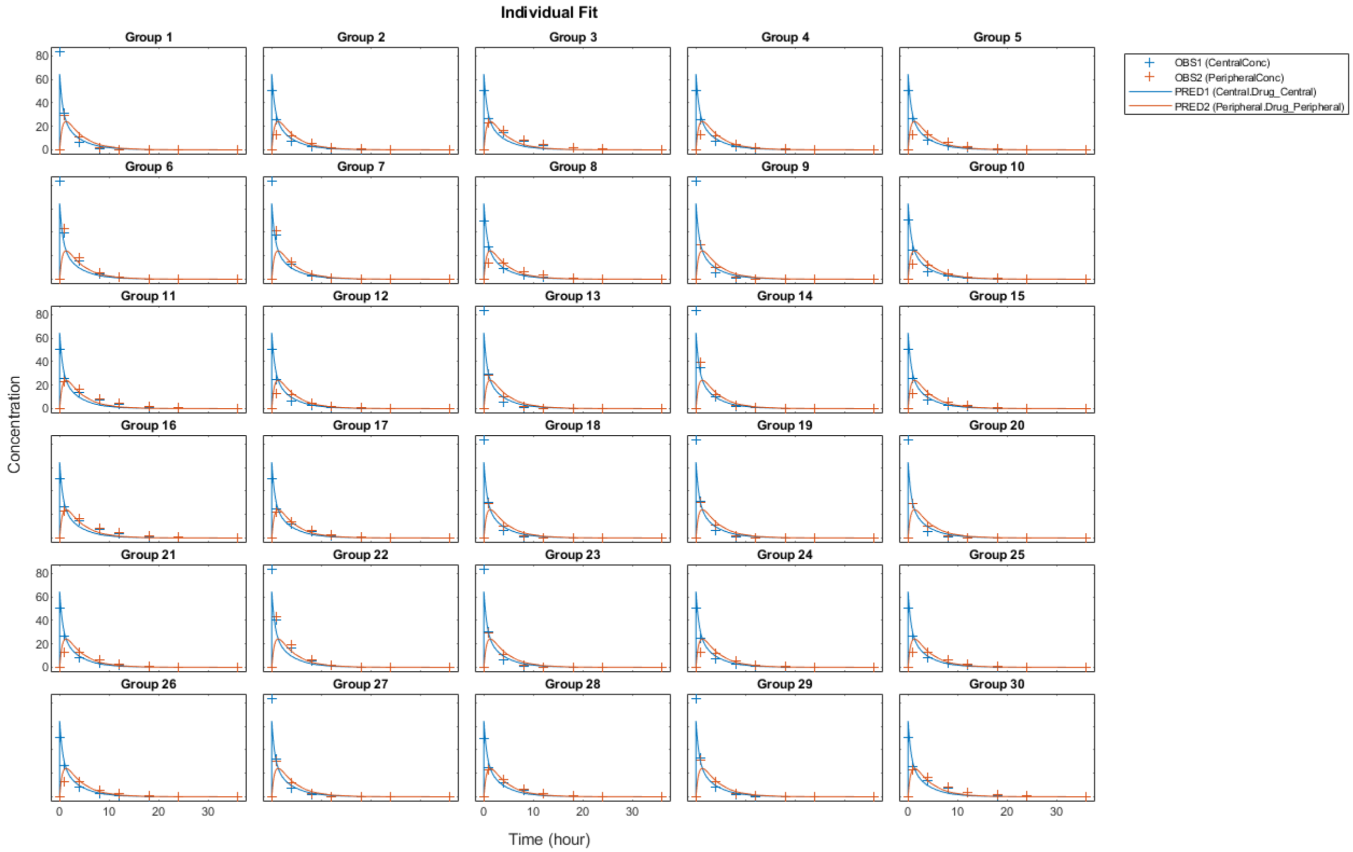
```
pooledFit = sbiofit(model,gData,responseMap,estimatedParam,dose,'Pooled',true);
```

```
Warning: CategoryVariableName property of the estimatedInfo object is ignored when using the 'Po
```

Plot Results

Plot the fitted results versus the original data. Although a separate plot was generated for each individual, the data was fitted using the same set of parameters (that is, all individuals had the same fitted line).

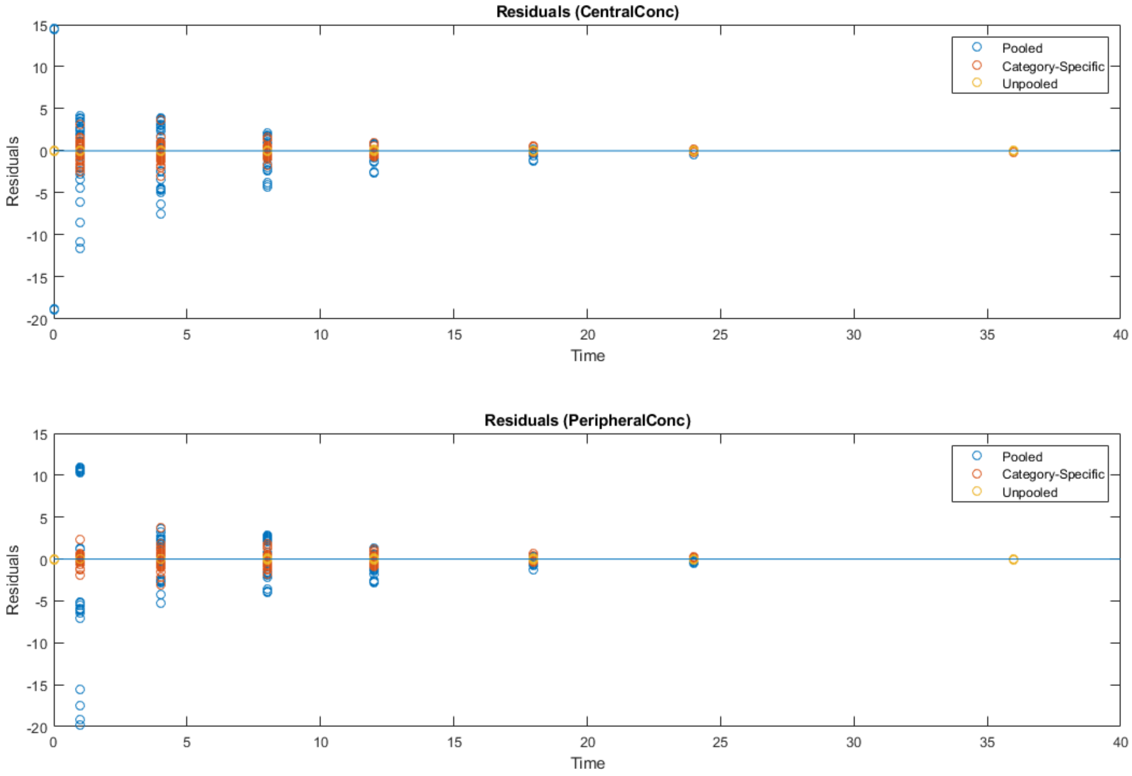
```
t = plot(pooledFit);
% Resize the figure.
t.hFig.Position(:) = [100 100 1280 800];
```



Compare Residuals

Compare residuals of `CentralConc` and `PeripheralConc` responses for each fit.

```
t = gData.Time;
allResid(:,:,1) = pooledFit.R;
allResid(:,:,2) = categoryFit.R;
allResid(:,:,3) = vertcat(unpooledFit.R);

h = figure;
responseList = {'CentralConc', 'PeripheralConc'};
for i = 1:2
    subplot(2,1,i);
    oneResid = squeeze(allResid(:,i,:));
    plot(t,oneResid,'o');
    refline(0,0); % A reference line representing a zero residual
    title(sprintf('Residuals (%s)', responseList{i}));
    xlabel('Time');
    ylabel('Residuals');
    legend({'Pooled','Category-Specific','Unpooled'});
end
% Resize the figure.
h.Position(:) = [100 100 1280 800];
```



As shown in the plot, the unpooled fit produced the best fit to the data as it fit the data to each individual. This was expected since it used the most number of degrees of freedom. The category-fit reduced the number of degrees of freedom by fitting the data to two categories (sex and age). As a result, the residuals were larger than the unpooled fit, but still smaller than the population-fit, which estimated just one set of parameters for all individuals. The category-fit might be a good compromise between the unpooled and pooled fitting provided that any hierarchical model exists within your data.

See Also

`sbiofit`

More About

- "Fit a One-Compartment Model to an Individual's PK Profile" on page 4-52
- "Fit a Two-Compartment Model to PK Profiles of Multiple Individuals" on page 4-71
- "Nonlinear Regression" on page 4-37

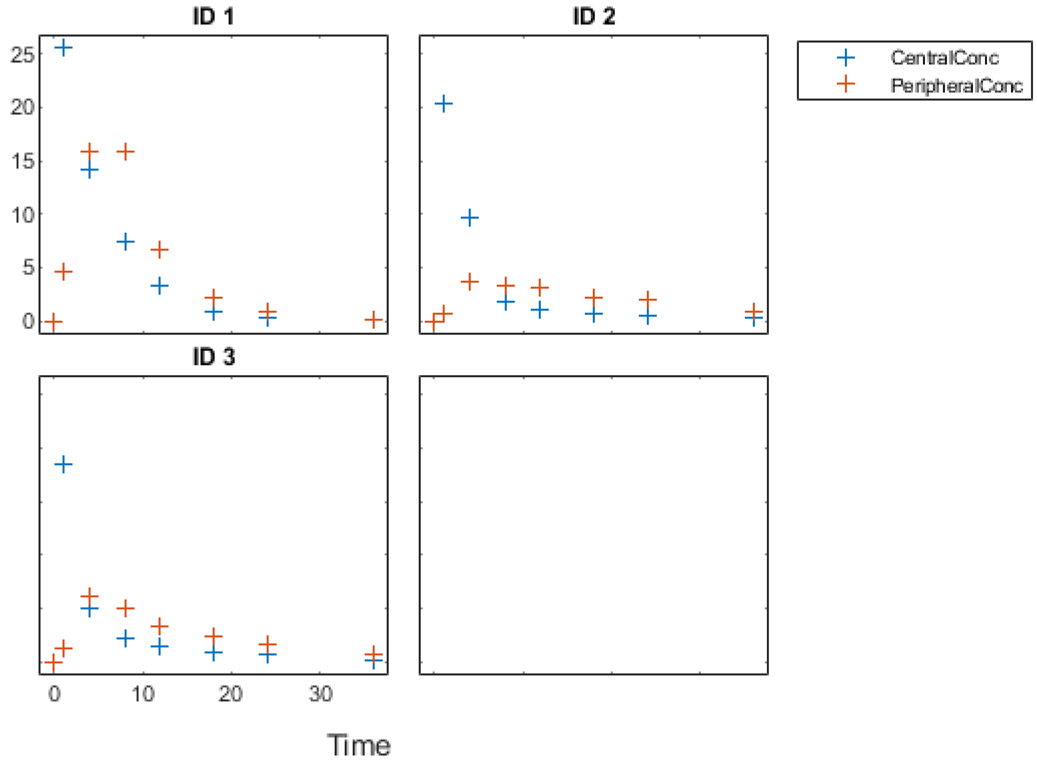
Perform Hybrid Optimization Using sbiofit

This example shows how to configure `sbiofit` to perform a hybrid optimization by first running the global solver `particleswarm`, followed by another minimization function, `fmincon`.

Load Data

Load the sample data to fit. The data is stored as a table with variables *ID*, *Time*, *CentralConc*, and *PeripheralConc*. This synthetic data represents the time course of plasma concentrations measured at eight different time points for both central and peripheral compartments after an infusion dose for three individuals.

```
clear all
load('data10_32R.mat')
gData = groupedData(data);
gData.Properties.VariableUnits = {'','hour','milligram/liter','milligram/liter'};
sbiotrellis(gData,'ID','Time',{'CentralConc','PeripheralConc'},'Marker','+',...
            'LineStyle','none');
```



Create Model

Create a two-compartment model with an infusion dose.

```
pkmd                 = PKModelDesign;
pkc1                 = addCompartment(pkmd,'Central');
pkc1.DosingType      = 'Infusion';
pkc1.EliminationType = 'linear-clearance';
```

```
pkc1.HasResponseVariable = true;
pkc2                     = addCompartment(pkmd,'Peripheral');
model                    = construct(pkmd);
configset                = getconfigset(model);
configset.CompileOptions.UnitConversion = true;
dose                 = sbiodose('dose','TargetName','Drug_Central');
dose.StartTime       = 0;
dose.Amount          = 100;
dose.Rate            = 50;
dose.AmountUnits     = 'milligram';
dose.TimeUnits       = 'hour';
dose.RateUnits       = 'milligram/hour';
responseMap = {'Drug_Central = CentralConc','Drug_Peripheral = PeripheralConc'};
```

Define Parameters to Estimate

Use the `estimatedInfo` object to define the estimated parameters.

```
paramsToEstimate   = {'log(Central)','log(Peripheral)','Q12','Cl_Central'};
estimatedParam     = estimatedInfo(paramsToEstimate,'InitialValue',[1 1 1 1],...
                                   'Bounds',[0 10]);
```

Define the Options for Hybrid Optimization

Define the options for the global solver and the hybrid solver. Because the parameters are bounded, make sure you use a compatible hybrid function for a constrained optimization, such as `fmincon`. Use `optimset` to define the options for `fminsearch`. Use `optimoptions` to define the options for `fminunc`, `patternsearch`, and `fmincon`.

```
rng('default');
globalMethod = 'particleswarm';
options = optimoptions(globalMethod);
hybridMethod = 'fmincon';
hybridopts = optimoptions(hybridMethod,'Display','none');
options = optimoptions(options,'HybridFcn',{hybridMethod,hybridopts});
```

Fit Data

Estimate model parameters. Turn on `ProgressPlot` to see the live feedback on the status of fitting. The first row of plots are the quality measure plots for the global solver. The second row plots are for the hybrid function. For details, see “Progress Plot” on page 4-45.

```
unpooledFit =  sbiofit(model,gData,responseMap,estimatedParam,dose,globalMethod,...
                       options,'Pooled',false,'ProgressPlot',true);
```

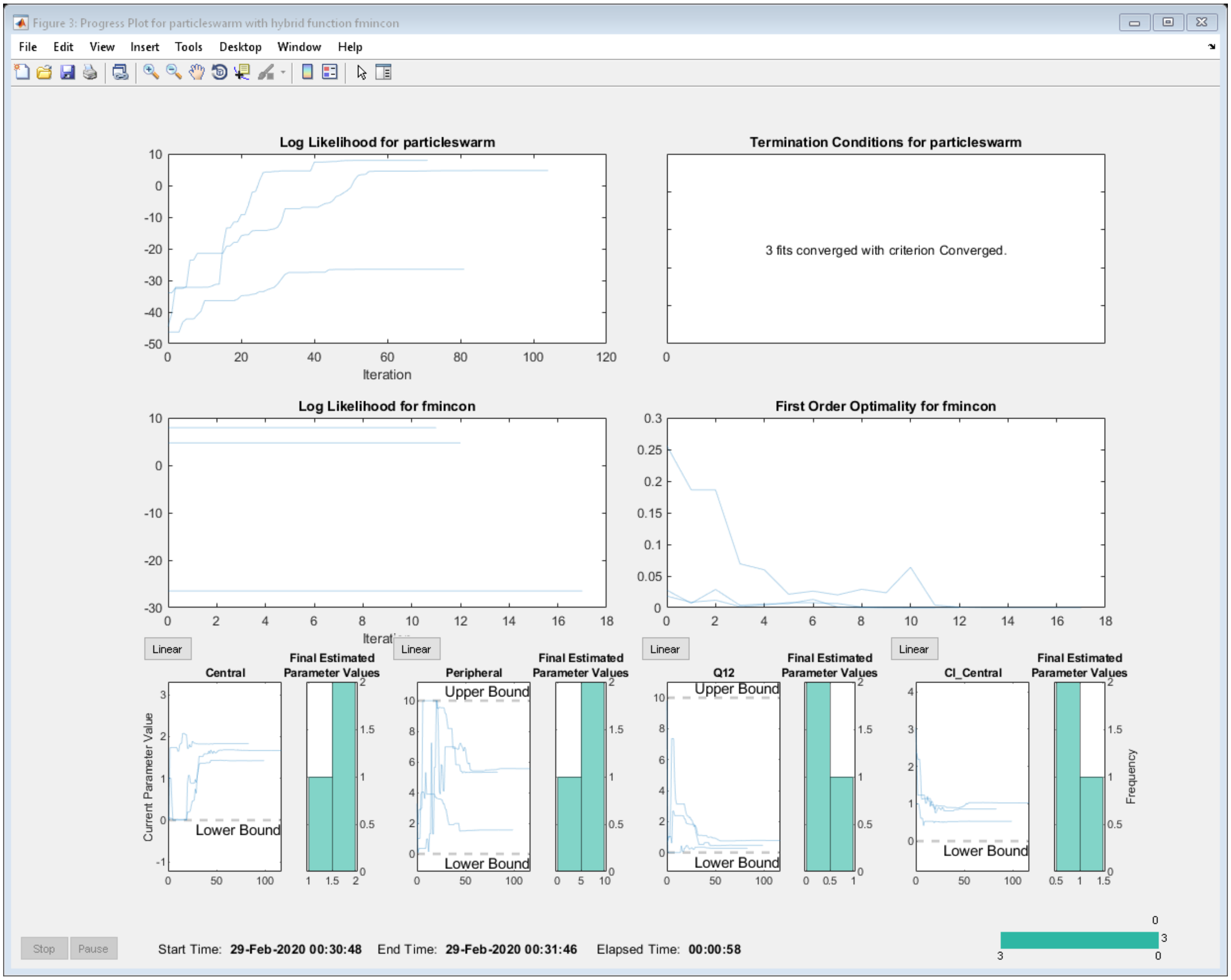


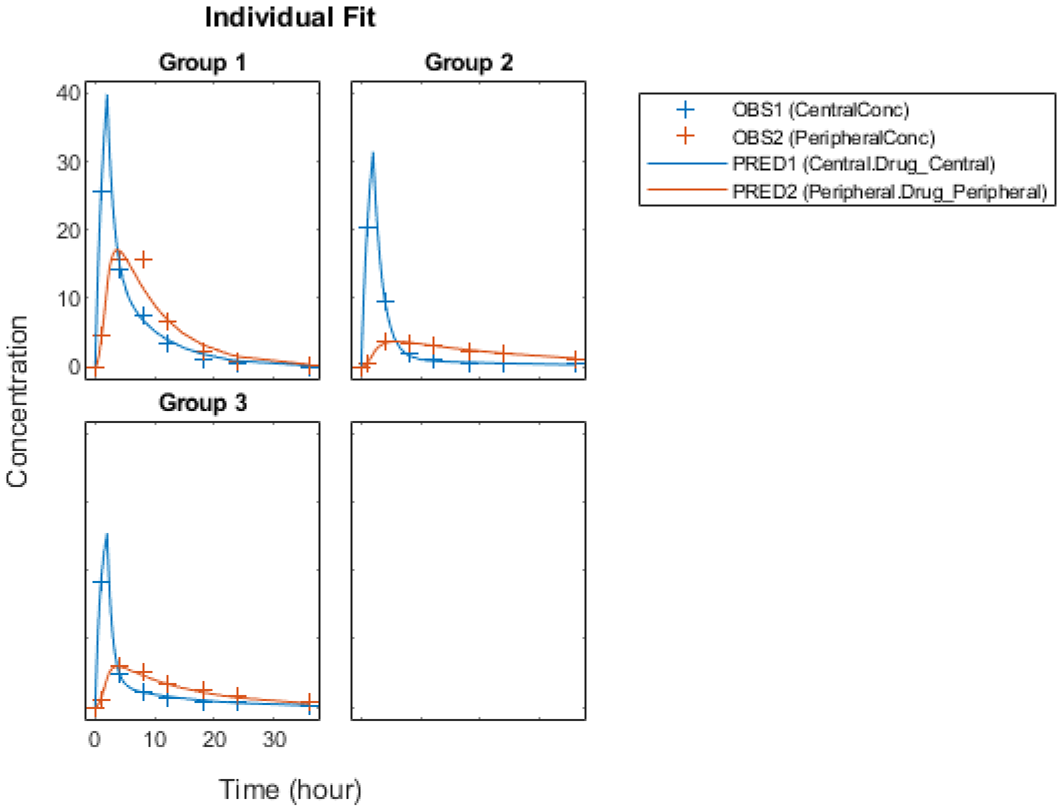
Figure 3: Progress Plot for particleswarm with hybrid function fmincon
File Edit View Insert Tools Desktop Window Help
Log Likelihood for particleswarm
Iteration
Termination Conditions for particleswarm
Log Likelihood for fmincon
Iteration
First Order Optimality for fmincon
Linear
Central
Final Estimated Parameter Values
Peripheral
Q12
Cl_Central
Current Parameter Value
Frequency
Upper Bound
Lower Bound
Stop
Pause

3 fits converged with criterion Converged.

Start Time: **29-Feb-2020 00:30:48** End Time: **29-Feb-2020 00:31:46** Elapsed Time: **00:00:58**

Plot Results

```
plot(unpooledFit);
```



See Also

`sbiofit`

More About

- “Fit a One-Compartment Model to an Individual's PK Profile” on page 4-52
- “Fit a Two-Compartment Model to PK Profiles of Multiple Individuals” on page 4-71
- “Nonlinear Regression” on page 4-37

Fit a Two-Compartment Model to PK Profiles of Multiple Individuals

This example shows how to estimate pharmacokinetic parameters of multiple individuals using a two-compartment model.

Suppose you have drug plasma concentration data from three individuals that you want to use to estimate corresponding pharmacokinetic parameters, namely the volume of central and peripheral compartment (`Central`, `Peripheral`), the clearance rate (`Cl_Central`), and intercompartmental clearance (`Q12`). Assume the drug concentration versus the time profile follows the biexponential decline $C_t = Ae^{-at} + Be^{-bt}$, where C_t is the drug concentration at time t, and a and b are slopes for corresponding exponential declines.

The synthetic data set contains drug plasma concentration data measured in both central and peripheral compartments. The data was generated using a two-compartment model with an infusion dose and first-order elimination. These parameters were used for each individual.

	Central	**Peripheral**	**Q12**	**Cl_Central**
Individual 1	1.90	0.68	0.24	0.57
Individual 2	2.10	6.05	0.36	0.95
Individual 3	1.70	4.21	0.46	0.95

The data is stored as a table with variables `ID`, `Time`, `CentralConc`, and `PeripheralConc`. It represents the time course of plasma concentrations measured at eight different time points for both central and peripheral compartments after an infusion dose.

```
clear all
load('data10_32R.mat')
```

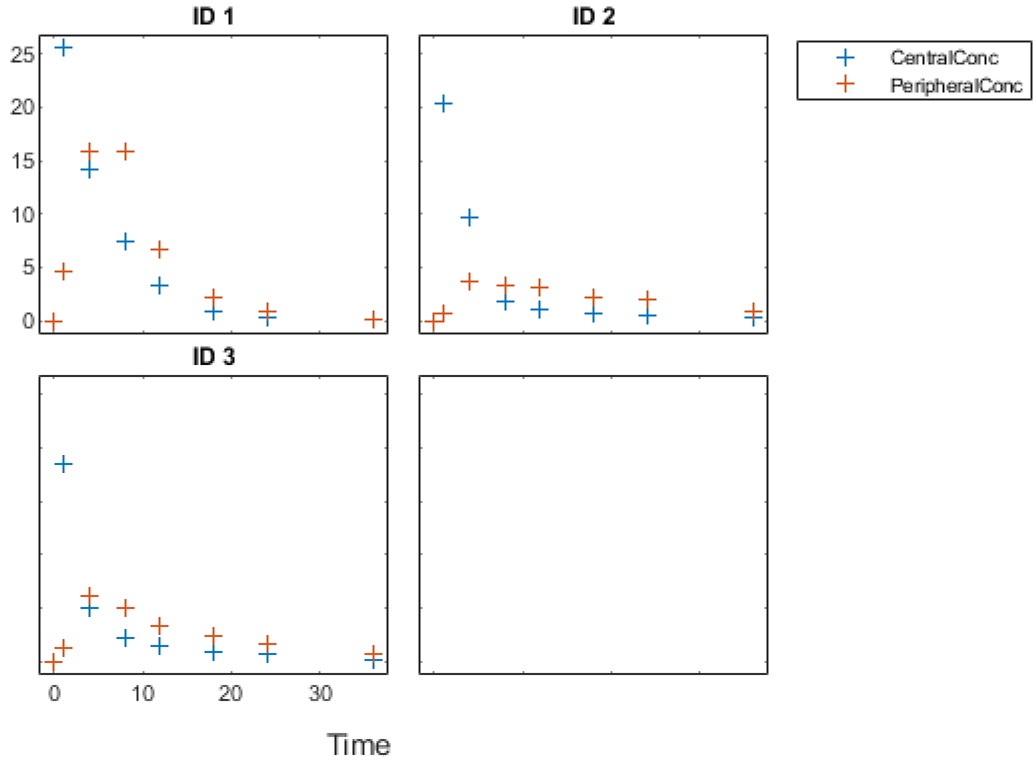
Convert the data set to a `groupedData` object which is the required data format for the fitting function `sbiofit` for later use. A `groupedData` object also lets you set independent variable and group variable names (if they exist). Set the units of the `ID`, `Time`, `CentralConc`, and `PeripheralConc` variables. The units are optional and only required for the `UnitConversion` feature, which automatically converts matching physical quantities to one consistent unit system.

```
gData = groupedData(data);
gData.Properties.VariableUnits = {'','hour','milligram/liter','milligram/liter'};
gData.Properties
```

```
ans = 

  struct with fields:

                Description: ''
                   UserData: []
             DimensionNames: {'Row'  'Variables'}
              VariableNames: {'ID'  'Time'  'CentralConc'  'PeripheralConc'}
       VariableDescriptions: {}
              VariableUnits: {1x4 cell}
         VariableContinuity: []
                   RowNames: {}
           CustomProperties: [1x1 matlab.tabular.CustomProperties]
          GroupVariableName: 'ID'
    IndependentVariableName: 'Time'
```

Create a trellis plot that shows the PK profiles of three individuals.

```
sbiotrellis(gData,'ID','Time',{'CentralConc','PeripheralConc'},...
            'Marker','+','LineStyle','none');
```



Use the built-in PK library to construct a two-compartment model with infusion dosing and first-order elimination where the elimination rate depends on the clearance and volume of the central compartment. Use the configset object to turn on unit conversion.

```
pkmd                    = PKModelDesign;
pkc1                    = addCompartment(pkmd,'Central');
pkc1.DosingType         = 'Infusion';
pkc1.EliminationType = 'linear-clearance';
pkc1.HasResponseVariable = true;
pkc2                    = addCompartment(pkmd,'Peripheral');
model                   = construct(pkmd);
configset               = getconfigset(model);
configset.CompileOptions.UnitConversion = true;
```

Assume every individual receives an infusion dose at time = 0, with a total infusion amount of 100 mg at a rate of 50 mg/hour. For details on setting up different dosing strategies, see “Doses in SimBiology Models” on page 2-30.

```
dose                = sbiodose('dose','TargetName','Drug_Central');
dose.StartTime      = 0;
dose.Amount         = 100;
dose.Rate           = 50;
dose.AmountUnits    = 'milligram';
dose.TimeUnits      = 'hour';
dose.RateUnits      = 'milligram/hour';
```

The data contains measured plasma concentrations in the central and peripheral compartments. Map these variables to the appropriate model species, which are `Drug_Central` and `Drug_Peripheral`.

```
responseMap = {'Drug_Central = CentralConc','Drug_Peripheral = PeripheralConc'};
```

The parameters to estimate in this model are the volumes of central and peripheral compartments (`Central` and `Peripheral`), intercompartmental clearance `Q12`, and clearance rate `Cl_Central`. In this case, specify log-transform for `Central` and `Peripheral` since they are constrained to be positive. The `estimatedInfo` object lets you specify parameter transforms, initial values, and parameter bounds (optional).

```
paramsToEstimate   = {'log(Central)','log(Peripheral)','Q12','Cl_Central'};
estimatedParam     = estimatedInfo(paramsToEstimate,'InitialValue',[1 1 1 1]);
```

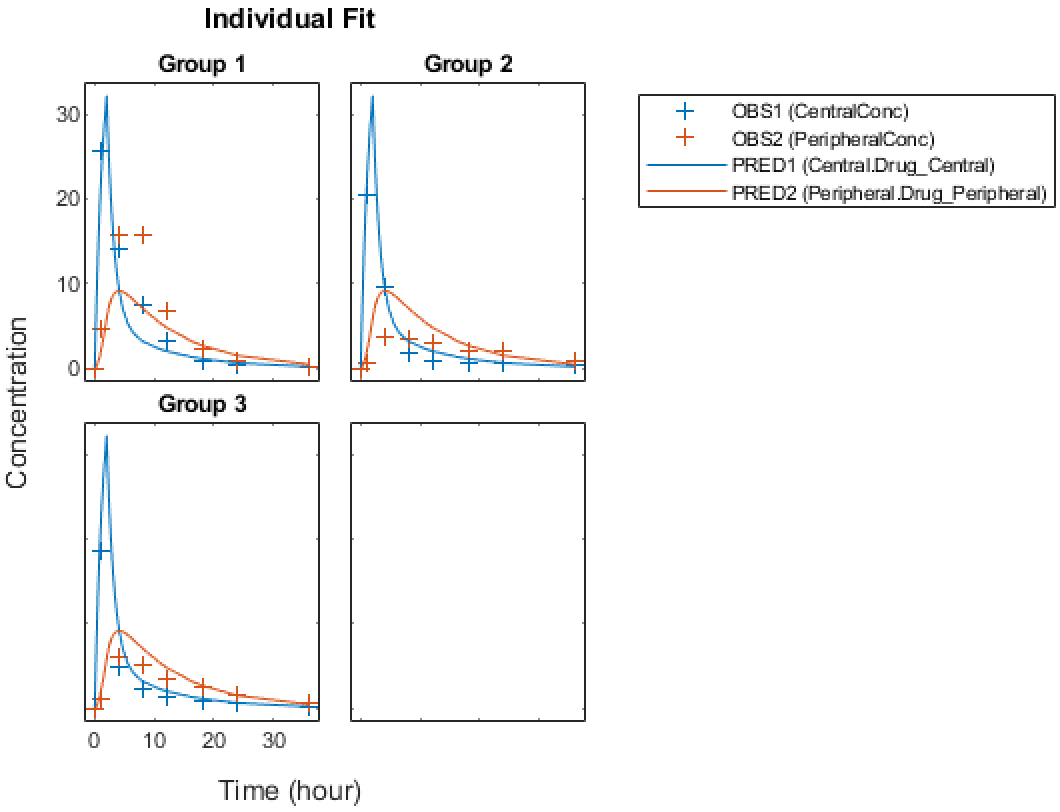
Fit the model to all of the data pooled together, that is, estimate one set of parameters for all individuals. The default estimation method that `sbiofit` uses will change depending on which toolboxes are available. To see which estimation function `sbiofit` used for the fitting, check the `EstimationFunction` property of the corresponding results object.

```
pooledFit = sbiofit(model,gData,responseMap,estimatedParam,dose,'Pooled',true)
```

```
pooledFit = 

  OptimResults with properties:

                   ExitFlag: 3
                     Output: [1x1 struct]
                  GroupName: []
                       Beta: [4x3 table]
         ParameterEstimates: [4x3 table]
                          J: [24x4x2 double]
                       COVB: [4x4 double]
           CovarianceMatrix: [4x4 double]
                          R: [24x2 double]
                        MSE: 6.6220
                        SSE: 291.3688
                    Weights: []
              LogLikelihood: -111.3904
                        AIC: 230.7808
                        BIC: 238.2656
                        DFE: 44
             DependentFiles: {1x3 cell}
    EstimatedParameterNames: {'Central'  'Peripheral'  'Q12'  'Cl_Central'}
             ErrorModelInfo: [1x3 table]
         EstimationFunction: 'lsqnonlin'
```

Plot the fitted results versus the original data. Although three separate plots were generated, the data was fitted using the same set of parameters (that is, all three individuals had the same fitted line).

```
plot(pooledFit);
```

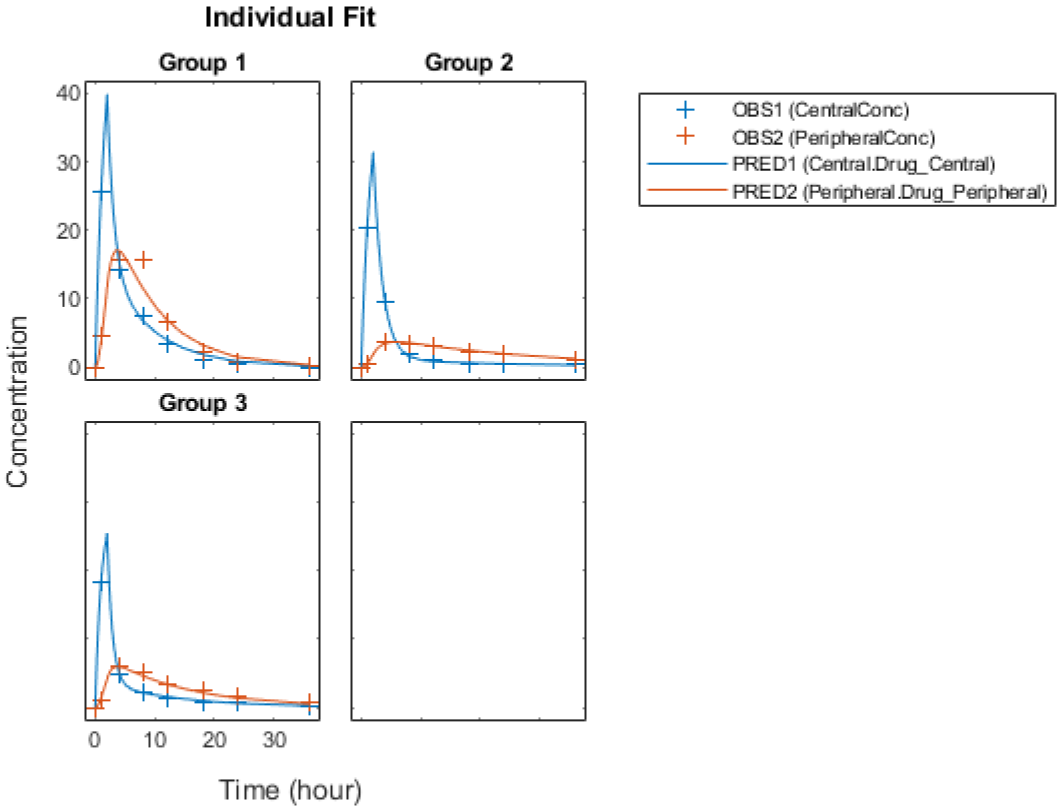


Estimate one set of parameters for each individual and see if there is any improvement in the parameter estimates. In this example, since there are three individuals, three sets of parameters are estimated.

```
unpooledFit =  sbiofit(model,gData,responseMap,estimatedParam,dose,'Pooled',false);
```

Plot the fitted results versus the original data. Each individual was fitted differently (that is, each fitted line is unique to each individual) and each line appeared to fit well to individual data.

```
plot(unpooledFit);
```



Display the fitted results of the first individual. The MSE was lower than that of the pooled fit. This is also true for the other two individuals.

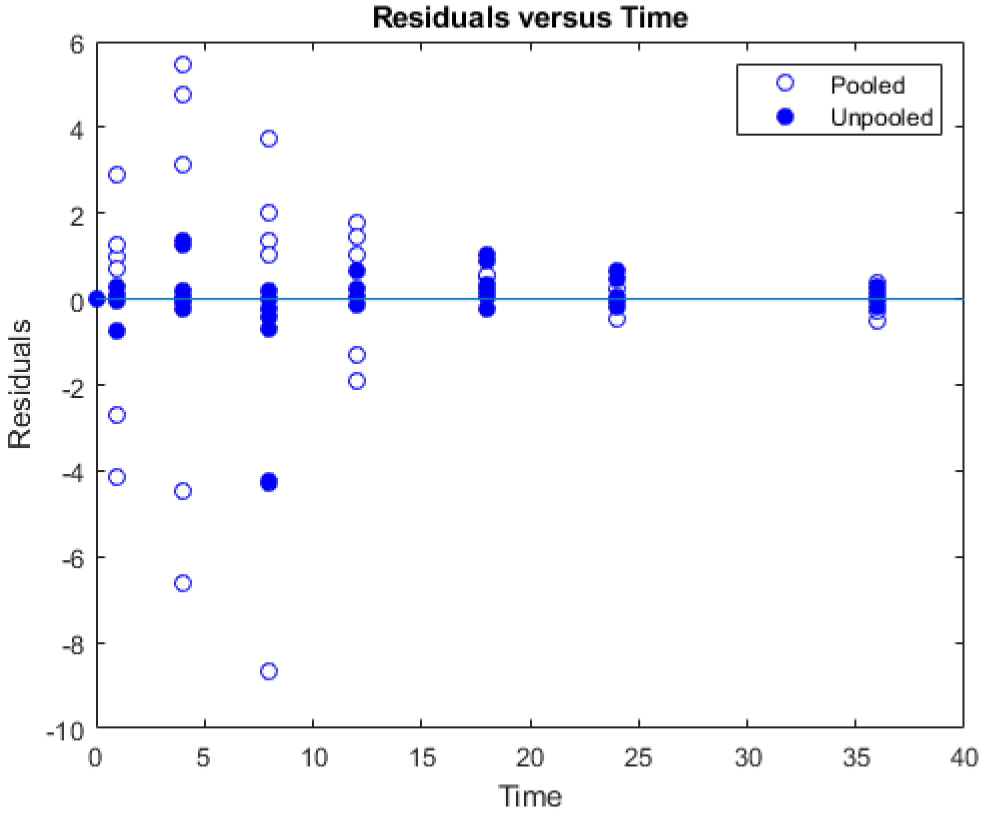
```
unpooledFit(1)
```

```
ans = 

  OptimResults with properties:

                   ExitFlag: 3
                     Output: [1x1 struct]
                  GroupName: 1
                       Beta: [4x3 table]
        ParameterEstimates: [4x3 table]
                          J: [8x4x2 double]
                       COVB: [4x4 double]
          CovarianceMatrix: [4x4 double]
                          R: [8x2 double]
                        MSE: 2.1380
                        SSE: 25.6559
                    Weights: []
             LogLikelihood: -26.4805
                        AIC: 60.9610
                        BIC: 64.0514
                        DFE: 12
            DependentFiles: {1x3 cell}
    EstimatedParameterNames: {'Central'  'Peripheral'  'Q12'  'Cl_Central'}
```

```
        ErrorModelInfo: [1x3 table]
    EstimationFunction: 'lsqnonlin'
```

Generate a plot of the residuals over time to compare the pooled and unpooled fit results. The figure indicates unpooled fit residuals are smaller than those of pooled fit as expected. In addition to comparing residuals, other rigorous criteria can be used to compare the fitted results.

```
t = [gData.Time;gData.Time];
res_pooled = vertcat(pooledFit.R);
res_pooled = res_pooled(:);
res_unpooled = vertcat(unpooledFit.R);
res_unpooled = res_unpooled(:);
plot(t,res_pooled,'o','MarkerFaceColor','w','markerEdgeColor','b')
hold on
plot(t,res_unpooled,'o','MarkerFaceColor','b','markerEdgeColor','b')
refl = refline(0,0); % A reference line representing a zero residual
title('Residuals versus Time');
xlabel('Time');
ylabel('Residuals');
legend({'Pooled','Unpooled'});
```



This example showed how to perform pooled and unpooled estimations using `sbiofit`. As illustrated, the unpooled fit accounts for variations due to the specific subjects in the study, and, in this case, the model fits better to the data. However, the pooled fit returns population-wide parameters. If you want to estimate population-wide parameters while considering individual variations, use `sbiofitmixed`.

See Also

`sbiofit`

More About

- “Fit a One-Compartment Model to an Individual's PK Profile” on page 4-52
- “Estimate Category-Specific PK Parameters for Multiple Individuals” on page 4-58
- “Nonlinear Regression” on page 4-37

Estimating the Bioavailability of a Drug

In this example, you will use the parameter estimation capabilities of SimBiology™ to calculate F, the bioavailability, of the drug ondansetron. You will calculate F by fitting a model of absorption and excretion of the drug to experimental data tracking drug concentration over time.

This example requires Optimization Toolbox™.

Background

Most drugs must be absorbed into the bloodstream in order to become active. An intravenous (IV) administration of a drug is one way to achieve this. However, it is impractical or impossible in many cases.

When a drug is not given by IV, it follows some other route into the bloodstream, such as absorption through the mucous membranes of the GI tract or mouth. Drugs administered through a route other than IV administration are generally not completely absorbed. Some portion of the drug is directly eliminated and never reaches the bloodstream.

The percentage of drug absorbed is the bioavailability of the drug. Bioavailability is one of the most important pharmacokinetic properties of a drug. It is useful when calculating safe dosages for non-IV routes of administration. Bioavailability is calculated relative to an IV administration. When administered intravenously, a drug has 100% bioavailability. Other routes of administration tend to reduce the amount of drug that reaches the blood stream.

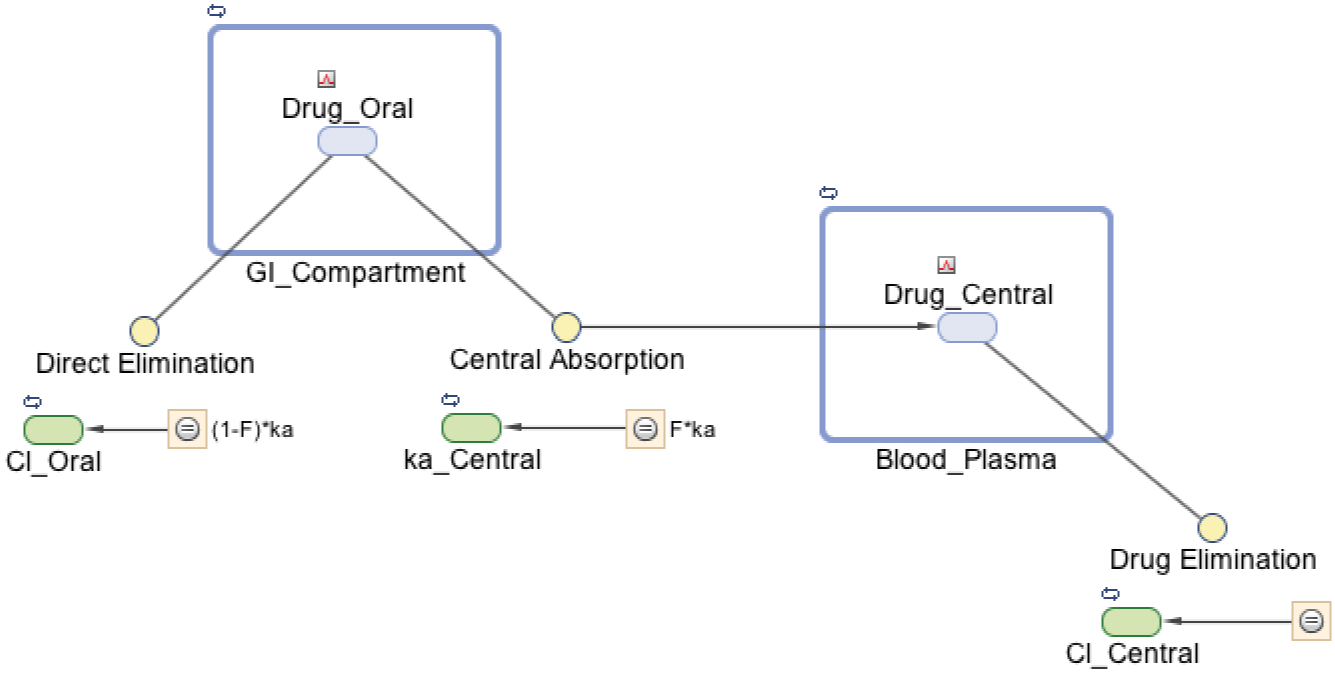
Modeling Bioavailability

Bioavailability can be modeled using one of several approaches. In this example, you use a model with a GI compartment and a blood plasma compartment. Oral administration is modeled by a dose event in the GI compartment. IV administration is modeled by a dose event in the blood plasma compartment.

The example models the drug leaving the GI compartment in two ways. The available fraction of the drug is absorbed into the bloodstream. The remainder is directly eliminated. The total rate of elimination, ka, is divided into absorption, ka_Central, and direct elimination, Cl_Oral. The bioavailability, F, connects total elimination with ka_Central and Cl_Oral via two initial assignment rules.

```
ka_Central = F*ka
Cl_Oral = (1-F)*ka
```

The drug is eliminated from the Blood_Plasma compartment through first-order kinetics, at a rate determined by the parameter Cl_Central.



Load the project that contains the model m1.

```
sbioloadproject('Bioavailability.sbproj','m1');
```

Format of the Data for Estimating Bioavailability

You can estimate bioavailability by comparing intrapatient measurements of drug concentration under different dosing conditions. For instance, a patient receives an IV dose on day 1, then receives an oral dose on day 2. On both days, we can measure the blood plasma concentration of the drug over some period of time.

Such data allow us to estimate the bioavailability, as well as other parameters of the model. Intrapatient time courses were generated for the drug ondansetron, reported in [2] and reproduced in [1].

Load the data, which is a table.

```
load('ondansetron_data.mat');
```

Convert the data to a groupedData object because the fitting function sbiofit requires it to be a groupedData object.

```
gd = groupedData(ondansetron_data);
```

Display the data.

```
gd
```

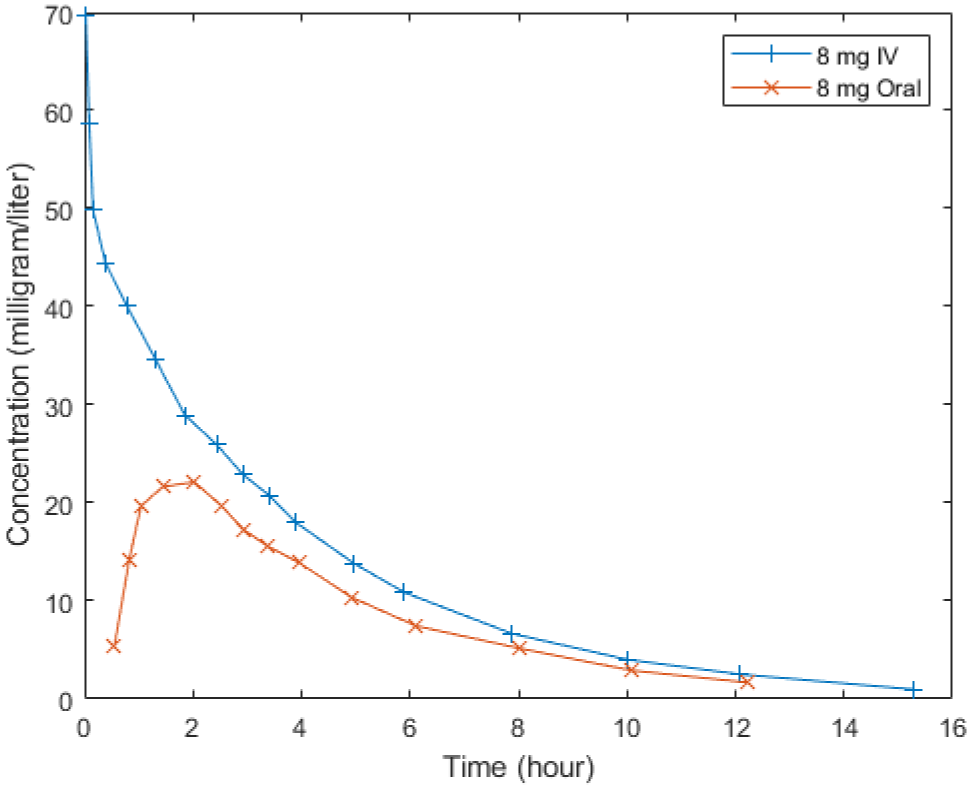
```
gd = 

  33x5 groupedData
```

```
      Time         Drug       Group    IV     Oral
    ________     _______     _____    ___    ____

           0         NaN       1        8     NaN
    0.024358      69.636       1      NaN     NaN
    0.087639      58.744       1      NaN     NaN
     0.15834      49.824       1      NaN     NaN
     0.38895      44.409       1      NaN     NaN
     0.78392      40.022       1      NaN     NaN
      1.3182      34.522       1      NaN     NaN
      1.8518      28.972       1      NaN     NaN
      2.4335       25.97       1      NaN     NaN
      2.9215      22.898       1      NaN     NaN
        3.41       20.75       1      NaN     NaN
      3.8744      18.095       1      NaN     NaN
      4.9668      13.839       1      NaN     NaN
      5.8962      10.876       1      NaN     NaN
      7.8717      6.6821       1      NaN     NaN
       10.01      4.0166       1      NaN     NaN
       12.08      2.5226       1      NaN     NaN
      15.284     0.97816       1      NaN     NaN
           0         NaN       2      NaN       8
     0.54951      5.3091       2      NaN     NaN
     0.82649      14.262       2      NaN     NaN
      1.0433       19.72       2      NaN     NaN
      1.4423      21.654       2      NaN     NaN
      2.0267      22.144       2      NaN     NaN
      2.5148      19.739       2      NaN     NaN
      2.9326      17.308       2      NaN     NaN
      3.3743      15.599       2      NaN     NaN
      3.9559      13.906       2      NaN     NaN
      4.9309      10.346       2      NaN     NaN
      6.1155      7.4489       2      NaN     NaN
      8.0002      5.1919       2      NaN     NaN
      10.091      2.9058       2      NaN     NaN
      12.228      1.6808       2      NaN     NaN
```

The data have variables for time, drug concentration, grouping information, IV, and oral dose amounts. Group 1 contains the data for the IV time course. Group 2 contains the data for the oral time course. NaN in the Drug column means no measurement was made at that time. NaN in one of the dosing columns means no dose was given through that route at that time.

Plot the pharmacokinetic profiles of the oral dose and IV administration.

```
plot(gd.Time(gd.Group==1),gd.Drug(gd.Group==1),'Marker','+')
hold on
plot(gd.Time(gd.Group==2),gd.Drug(gd.Group==2),'Marker','x')
legend({'8 mg IV','8 mg Oral'})
xlabel('Time (hour)')
ylabel('Concentration (milligram/liter)')
```



Notice there is a lag phase in the oral dose of about an hour while the drug is absorbed from the GI tract into the bloodstream.

Fitting the Data

Estimate the following four parameters of the model:

- Total forward rate out of the dose compartment, `ka`
- Clearance from the `Blood_Plasma` compartment, `clearance`
- Volume of the `Blood_Plasma` compartment
- Bioavailability of the orally administered drug, `F`

Set the initial values of these parameters and specify the log transform for all parameters using an `estimatedInfo` object.

```
init = [1 1 2 .8];
estimated_parameters = estimatedInfo({'log(ka)','log(clearance)',...
                      'log(Blood_Plasma)','logit(F)'},'InitialValue',init);
```

Because `ka`, `clearance`, and `Blood_Plasma` are positive physical quantities, log transforming reflects the underlying physical constraint and generally improves fitting. This example uses a logit transform on `F` because it is a quantity constrained between 0 and 1. The logit transform takes the interval of 0 to 1 and transforms it by taking the log-odds of `F` (treating `F` as a probability). For a few drugs, like theophyline, constraining `F` between 0 and 1 is inappropriate because oral bioavailability can be greater than 1 for drugs with unusual absorption or metabolism mechanisms.

Next, map the response data to the corresponding model component. In the model, the plasma drug concentration is represented by `Blood_Plasma.Drug_Central`. The corresponding concentration data is the `Drug` variable of the groupedData object `gd`.

```
responseMap = {'Blood_Plasma.Drug_Central = Drug'};
```

Create the dose objects required by `sbiofit` to handle the dosing information. First, create the IV dose targeting `Drug_Central` and the oral dose targeting `Dose_Central`.

```
iv_dose   = sbiodose('IV','TargetName','Drug_Central');
oral_dose = sbiodose('Oral','TargetName','Drug_Oral');
```

Use these dose objects as template doses to generate an array of dose objects from the dosing data variables `IV` and `Oral`.

```
doses_for_fit = createDoses(gd,{'IV','Oral'},'',[iv_dose, oral_dose]);
```

Estimate parameters using `sbiofit`.

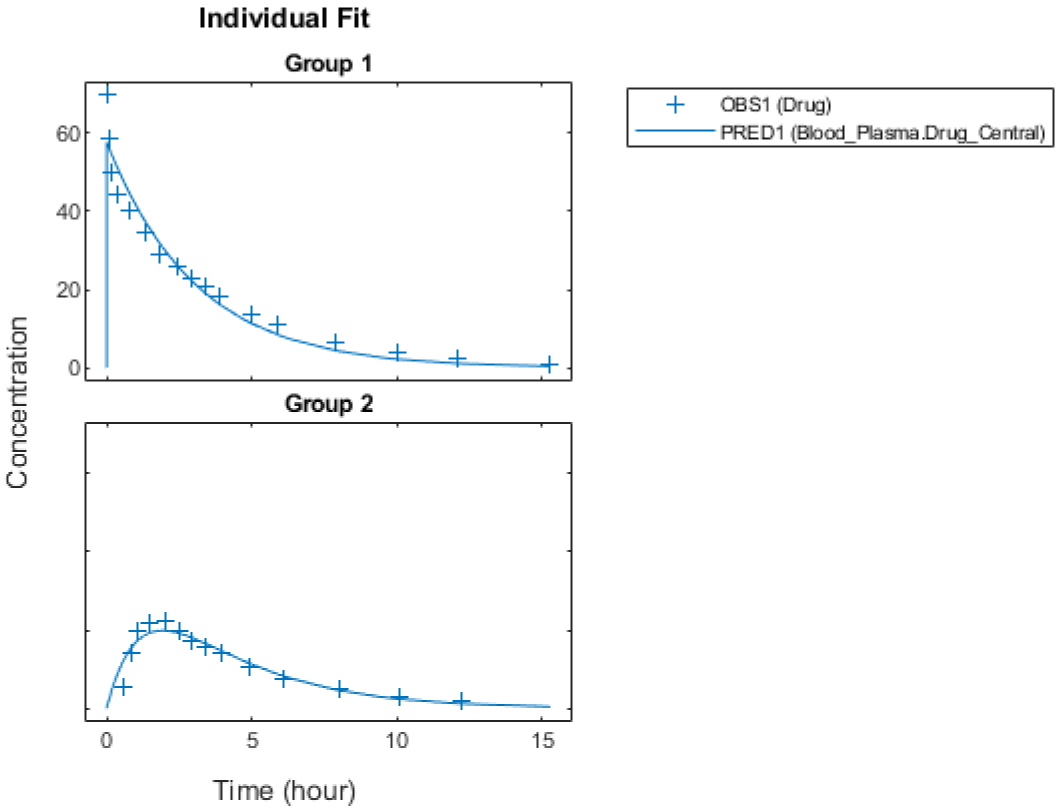
```
opts = optimoptions('lsqnonlin','Display','final');
results = sbiofit(m1, gd,responseMap,estimated_parameters,doses_for_fit,...
                  'lsqnonlin',opts,[],'pooled',true);
```

```
Local minimum possible.

lsqnonlin stopped because the final change in the sum of squares relative to
its initial value is less than the value of the function tolerance.
```

Interpreting Results

First, check if the fit is successful.

```
plot(results)
```



Overall, the results seem to be a good fit. However, they do not capture a distribution phase over the first hour. It might be possible to improve the fit by adding another compartment, but more data would be required to justify such an increase in model complexity.

When satisfied with the model fit, you can draw conclusions about the estimated parameters. Display the parameters stored in the results object.

```
results.ParameterEstimates
```

```
ans=4×3 table
          Name            Estimate    StandardError
    ________________      ________    _____________

    {'ka'          }      0.77947          0.1786
    {'clearance'   }        45.19          2.8674
    {'Blood_Plasma'}       138.73          4.5249
    {'F'           }      0.64455        0.066013
```

The parameter F is the bioavailability. The result indicates that ondansetron has approximately a 64% bioavailability. This estimate in line with the literature reports that oral administration of ondansetron in the 2-24 milligram range has a 60% bioavailability [1,2].

Blood_Plasma is the volume of distribution. This result is reasonably close to the 160 liter Vd reported for ondansetron [1]. The estimated clearance is 45.4 L/hr.

ka does not map directly onto a widely reported pharmacokinetic parameter. Consider it from two perspectives. We can say that 64% of the drug is available, and that the available drug has an absorption parameter of 0.4905/hr. Or, we can say that drug clearance from the GI compartment is 0.7402/hr, and 64% of the drug cleared from the GI tract is absorbed into the bloodstream.

Generalizing This Approach

`lsqnonlin`, as well as several other optimization algorithms supported by `sbiofit`, are local algorithms. Local algorithms are subject to the possibility of finding a result that is not the best result over all possible parameter choices. Because local algorithms do not guarantee convergence to the globally best fit, when fitting PK models, restarting the fit with different initial conditions multiple times is a good practice. Alternatively, `sbiofit` supports several global methods, such as particle swarm, or genetic algorithm optimization. Verifying that a fit is of sufficient quality is an important step before drawing inferences from the values of the parameters.

This example uses data that was the mean time course of several patients. When fitting a model with data from more patients, some parameters might be the same between patients, some not. Such requirements introduce the need for hierarchical modeling. You can perform hierarchical modeling can by configuring the `CategoryVariableName` flag of `EstimatedInfo` object.

References

1. Roila, Fausto, and Albano Del Favero. "Ondansetron clinical pharmacokinetics." Clinical Pharmacokinetics 29.2 (1995): 95-109.
2. Colthup, P. V., and J. L. Palmer. "The determination in plasma and pharmacokinetics of ondansetron." European Journal of Cancer & Clinical Oncology 25 (1988): S71-4.

See Also

`sbiofit`

More About

- “Fit a One-Compartment Model to an Individual's PK Profile” on page 4-52
- “Fit a Two-Compartment Model to PK Profiles of Multiple Individuals” on page 4-71
- “Nonlinear Regression” on page 4-37

Accelerating Model Simulations and Analyses

In this section...

What Is Acceleration?

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

When to Accelerate

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

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

Prerequisites for Accelerating Simulations and Analyses

To prepare your models for accelerated simulations, install and set up a compiler:

1. Install a C compiler (if one is not already installed on your system). For a current list of supported compilers, see Supported and Compatible Compilers.
2. Ensure that any user-defined functions in your model can be used for code generation from MATLAB, so they can convert to compiled C. For more information, see Language, Function, and Object support for C and C++ code generation (MATLAB Coder) or contact MathWorks Technical Support.

Note

- On 32-bit Windows platforms, the LCC compiler is automatically installed. However, for better performance of the acceleration functionality, you may want to install a supported compiler other than LCC, and it will be selected automatically.
- On 64-bit Windows platforms, if you have not installed another compiler, SimBiology uses the LCC64 compiler for model accelerations. If you have installed another supported compiler, it will be selected automatically.
- Microsoft® Visual Studio® 2010 run-time libraries must be available on any computer running accelerated models generated using Microsoft Windows SDK.

- If you plan to redistribute your accelerated models to other MATLAB users, be sure they have the same run-time libraries.

Accelerate Simulations Programmatically

Use `sbioaccelerate` if you are accelerating a SimBiology model. For a `SimFunction object` and an exported model (`SimBiology.export.Model`), use the corresponding `accelerate` method.

Using sbioaccelerate

Follow the two-step process for acceleration.

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

   ```
   sbioaccelerate(model,configset,doses);
   ```

   For a very large model, this step may take a minute or longer to complete.
2. Run `sbiosimulate` with the same input arguments that you used with `sbioaccelerate`. For example:

   ```
   simdata = sbiosimulate(model,configset,doses);
   ```

If you pass in an array of doses to `sbioaccelerate`, you can simulate the model using any subset of these doses and do not need to run acceleration again.

For illustrated examples, see the following.

- "Prepare a Model for Accelerated Simulation"
- "Accelerate Simulation With Array of Doses"

Using accelerate

A `SimFunction` object is automatically accelerated at the first function execution. Hence it is not necessary to accelerate the model before you create the object. However, manually accelerate using the `accelerate` method of the object if you want it accelerated in your deployment applications.

For exported model, see `accelerate`.

When to Rerun Acceleration

If you make any modifications to the model, such as changes to reactions or adding events, you need to rerun the acceleration, before running simulations.

However, there are exceptions. You do *not* need to accelerate again if you are making the changes to:

- Any variants
- InitialAmount property of species
- Capacity property of compartments
- Value property of parameters
- StopTime property of configset

- OutputTimes property of SolverOptions
- Active, Amount, and Rate properties of `ScheduleDose` and `RepeatDose`
- Time property of `ScheduleDose`
- Interval, RepeatCount, and StartTime properties of `RepeatDose`
- Notes, Tag, and UserData properties of any applicable objects

Accelerate Simulations using SimBiology Model Analyzer

You can enable the model acceleration in the **SimBiology Model Analyzer** app by checking the **Prepare the model for accelerated simulation** box in the **Model** step of the program.

Troubleshooting Accelerated Simulations

If you have custom functions, use persistent variables only for those (constant) variables that you do not want to recalculate or reload every function call. The reason is that during the acceleration process, SimBiology converts the model and custom functions to compiled C code. If you try to use a persistent variable to share data across generated (or compiled) C functions, you may have different results. For instance, if you use a persistent variable to count how many times a function is called, each compiled function will have a separate count. Those persistent variables in the corresponding compiled functions will be different from the one used in the MATLAB function that you defined.

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

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

where *Expression* is any of the following:

- Reaction rate/rule expression
- Initial assignment rule expression
- Repeated assignment rule expression
- Event trigger expression
- Event function expression

For more information, see Language, Function, and Object support for C and C++ code generation (MATLAB Coder) or contact MathWorks Technical Support.

See Also

`accelerate` | `SimBiology.export.Model` | `SimFunction object` | `accelerate` | `sbioaccelerate`

More About

- “SimBiology Apps”

Noncompartmental Analysis

Noncompartmental analysis (NCA) lets you compute pharmacokinetic (PK) parameters of a drug from the time course of measured drug concentrations. This approach does not require the assumption of a specific compartmental model. NCA is often used to determine the degree of exposure following administration of a drug, such as AUC, and other PK parameters, such as the clearance and the terminal half-life.

Data

SimBiology lets you calculate NCA parameters from concentration–time data. The data must contain a time column, a concentration column, and a dose column that defines dose amounts. Three types of drug administration routes are supported: IV bolus, IV infusion, and Extravascular. You can have a column for each type. For infusion doses, an infusion rate column is also needed.

If you have data containing multiple groups of observations, you can define a group column. If needed, you can use two levels of hierarchy to specify grouping. Specify the outer level of grouping using the group column, and specify the inner level (subgroups) in the ID column. Consider data that contains three groups, where each group contains four patients. The group column labels the three groups, and the ID column labels each patient.

Dosing

Single-dosing data contains a single dose amount for each individual. Multiple-dosing data has several doses at different times for each individual. There are common parameters calculated for either type of dosing data, and parameters that are specific to single or multiple dosing.

Common Parameters for Single and Multiple Dosing

SimBiology computes some common parameters for single- or multiple-dosing data. This figure represents the concentration-time profile after a single dose. For multiple dosing, the same principles apply, except that SimBiology uses the first period, that is, after the first dose and before the second.

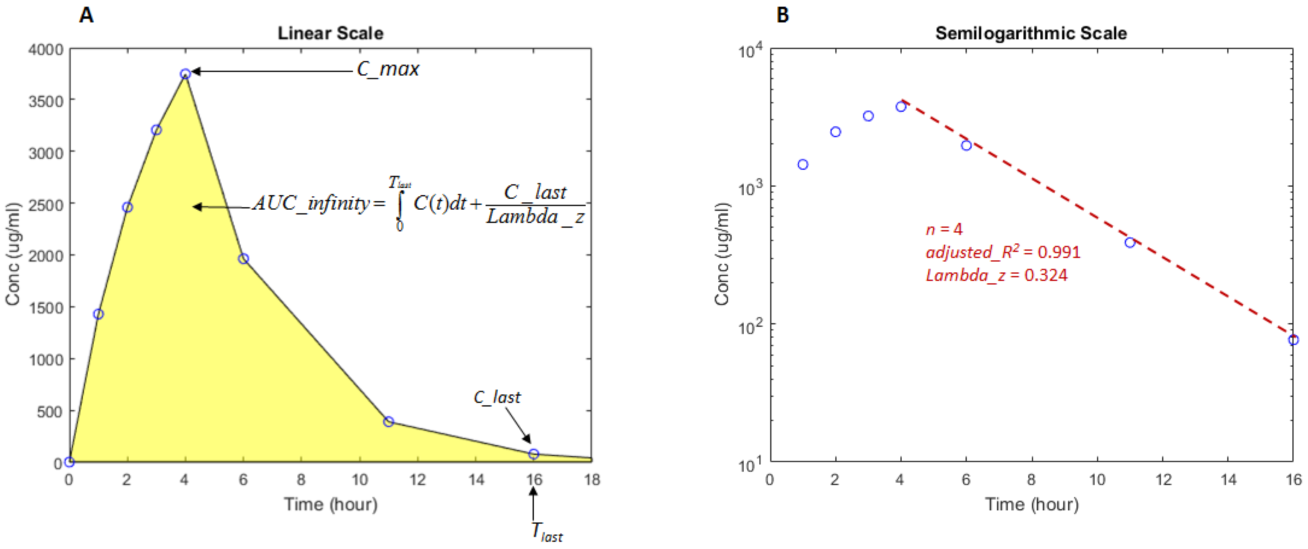


Figure **A** shows concentration–time data in a linear scale and illustrates how the AUC from time 0 to infinity is calculated. Figure **B** shows the same data in a semilogarithmic scale. To compute the terminal rate constant (*Lambda_z*), SimBiology performs a set of linear regressions of the log-transformed data using each of the last *n* points (*n* = `3, 4, 5, ...`) from the terminal portion of the curve. *Lambda_z* is chosen from the regression that uses the most points and has the maximum *adjusted_*R^2.

This table describes the common parameters for single and multiple dosing.

Parameter	Description
AUC_0_last	Area under the measured concentration–time curve from time = 0 to the last time point. $AUC\_0\_last = \int_{0}^{Tlast} C(t)dt,$ where *C(t)* is the plasma concentration at time *t*. SimBiology uses the linear trapezoidal method (MATLAB) to calculate the AUC.
AUC_infinity	Total area under the concentration–time curve extrapolating to *Inf* using the terminal rate constant *Lambda_z*. $AUC\_infinity = AUC\_0\_last + \frac{C\_last}{Lambda\_z},$ where *C_last* is the last observed concentration and *Lambda_z* is the terminal rate constant.
AUC_infinity_dose	$AUC\_infinity\_dose = \frac{AUC\_infinity}{DM}.$
AUCx_y	Partial AUC computed for a custom time range, where x and y are the start and end times, respectively.
AUC_extrap_percent	Fraction of total *AUC_infinity* obtained from extrapolation. $AUC\_extrap\_percent = \frac{AUC\_infinity - AUC\_0\_last}{AUC\_infinity} * 100.$
Lambda_z	To calculate the terminal rate constant (*Lambda_z*), SimBiology performs a set of linear regressions of the log(concentration)–time data using each of the last *n* points (`n = 3, 4, 5, ...`) from the terminal portion of the curve, that is, points satisfying the conditions: $(Time \geq T_{\max})\&(Conc \leq C_{\max})$. A minimum of three points is required to determine *Lambda_z*. *Lambda_z* is chosen from the regression that uses the most points and has the maximum *adjusted_*R^2 among all regressions. $adjusted\_R^2 = 1 - \frac{(1 - R^2) * (n - 1)}{n - 2}$
R2	Coefficient of determination for the `linear regressions` used in the *Lambda_z* calculation.
Num_points	Number of data points from the profile used in the determination of *Lambda_z*.

Parameter	Description
CL	Total drug clearance. $Cl = \frac{DM}{AUC\_infinity}$, where *DM* is the dose amount.
C_0	Extrapolated concentration at time = 0, computed using a regression of the first two data points in a profile. This parameter is for IV Bolus doses only.
C_max	Maximum observed concentration.
C_max_Dose	$C\_\mathrm{max}\_Dose = \frac{C\_max}{DM}$.
C_max_x_y	Maximum observed concentration within a given time range, specified by the start time *x* and the end time *y*. This parameter is computed when you specify a custom time range in the **Cmax Time Range** box in the SimBiology Model Analyzer app or set the `C_max_ranges` property of the options object created by `sbioncaoptions`.
MRT	Mean residence time. $MRT = \frac{AUMC}{AUC\_infinity}$.
Tlast	Time of the last observed concentration value above the lower limit of quantization (LOQ).
T_half	Terminal half-life of the drug. $T\_half = \frac{\ln(2)}{Lambda\_z}$.
T_max	*T_max* is the time point at which the maximum concentration (*C_max*) is observed.
T_max_x_y	Time point at which maximum concentration is observed within a given time range, specified by the start time *x* and the end time *y*. This parameter is computed when you specify a custom time range in the **Cmax Time Range** box in the SimBiology Model Analyzer or set the `C_max_ranges` property of the options object created by `sbioncaoptions`.
V_ss	Apparent volume of distribution at equilibrium. This parameter is for IV Bolus doses only. $V\_ss = \frac{DM * AUMC}{AUC\_infinity^2}$.
V_z	Volume of distribution during the terminal phase. $V\_z = \frac{DM}{AUC\_infinity * Lambda\_z}$.
DM	Administered dose amount. For multiple dosing, the last administered dose is reported.
doseSchedule	Single- or multiple-dosing data.
administrationRoute	Dose administration route. Supported routes are `IVBolus`, `IVInfusion`, `ExtraVascular`.

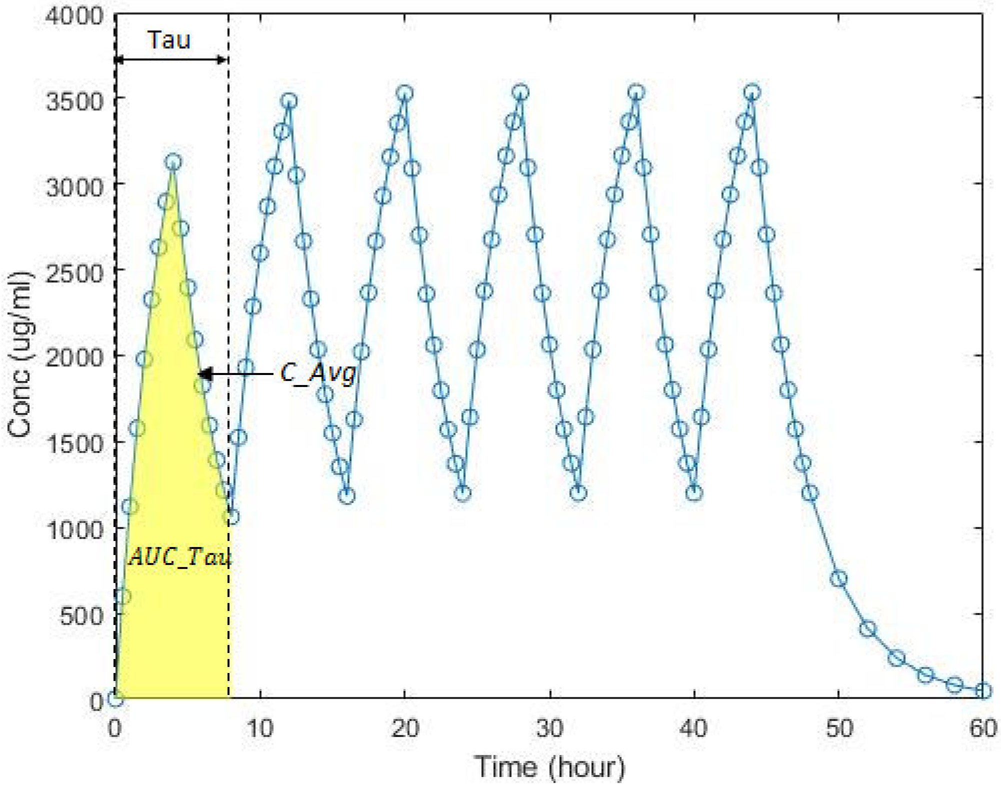
Parameters for Single Dosing

In addition to the common parameters, SimBiology reports parameters for single-dosing data.

Parameter	Description
AUMC_0_last	Area under the first moment of the concentration–time curve from time 0 to the last time point *Tlast*. $AUMC\_0\_last = \int_{0}^{Tlast} t*C(t)dt.$
AUMC	Total area under the first moment of the concentration–time curve extrapolating to *Inf* using *Lambda_z*. $AUMC = AUMC\_0\_last + \frac{C\_last}{Lambda\_z^2} + \frac{Tlast*C\_last}{Lambda\_z}.$
AUMC_extrap_percent	Fraction of total *AUMC* obtained from extrapolation. $AUMC\_extrap\_percent = \frac{AUMC - AUMC\_0\_last}{AUMC}*100.$

Parameters for Multiple Dosing

This figure shows the concentration-time profile after multiple doses. SimBiology uses the first period, that is, after the first dose and before the second, to compute the following NCA parameters for multiple-dosing data, in addition to the common parameters listed previously.



Parameter	Description
AUC_Tau	Area under the concentration–time curve during a dosing period of length *Tau*. SimBiology uses the first period, that is, after the first dose and before the second. $AUC\_Tau = \int_{T\_first\_period}^{T\_first\_period + Tau} C(t)dt.$
Tau	Dosing interval.
C_Avg	Average concentration over one period. $C\_Avg = \frac{AUC\_Tau}{Tau}.$
C_min	Minimum observed concentration during the first period, that is, `C_min = C(T_min)`.
PTF_percen t	Peak trough fluctuation over one dosing interval at steady state. $PTF\_Percent = \frac{C\_\max - C\_\min}{C\_Avg} * 100.$
Accumulati on_Index	Theoretical accumulation ratio. $Accumulation\_Index = \frac{1}{1 - e^{-Lambda\_z * Tau}}.$
T_min	Time at which the minimum concentration is reached in a dosing interval.

Sparse Sampling

To calculate PK parameters, measured concentrations at multiple time points for each individual is needed after the drug administration. Under certain circumstances, it is not feasible or not practical to obtain such longitudinal data on a single subject. In such cases, concentration data is collected from multiple individuals at each time point and then averaged to calculate NCA parameters for each group instead. SimBiology performs such sparse sampling by taking the average of the dependent variable for all individuals at the same time point. It then returns the values of NCA parameters for each group. Time values for each measurement across individuals (IDs) within a group must be identical.

Calculating NCA Parameters

You can calculate NCA parameters using the `sbionca` function in the command line or using the **SimBiology Model Analyzer** app.

Using sbionca

`sbionca` provides command line functionality to compute NCA parameters. Define the data classification options and parameter calculation options using an option object created by `sbioncaoptions`. For an example, see “Compute NCA Parameters from Concentration-Time Data”.

Using SimBiology Model Analyzer

After you import the data, select **Program** > **Non-Compartmental Analysis** on the **Home** tab. You can classify your data column in the **NCA** step of the program. If your data has a grouping column, specify it using **Group**. Use **ID** to specify the inner level of grouping. Specify the dosing data column (**IV Bolus Dose** or **Extravascular Dose**). **Lower limit of quantization (LOQ)** is a threshold below which the values of dependent variables are truncated to zero.

Lambda Time Range lets you specify a custom time range to compute the terminal rate constant (Lambda_z). The time range applies to all groups; you cannot specify a different time range for each group.

Cmax Time Range lets you specify a custom time range to report the maximum observed concentration within the range (*C_max*) and the time (*T_max*) when it is observed. You can specify a different time range for each group.

Partial AUC lets you specify a custom time range to compute the partial AUC bounded by the start and end times. You can specify a different time range for each group.

You can export the NCA results to MATLAB workspace. By default, the data is exported as a table (MATLAB). To convert it to a `dataset`, use `table2dataset`.

For a workflow example, see “Calculate NCA Parameters and Fit Model to PK/PD Data Using SimBiology Model Analyzer App” on page 1-64.

See Also

`sbionca` | `sbioncaoptions`

More About

- “SimBiology Apps”

Stochastic Simulation of Radioactive Decay

This example shows how to build and simulate a model using the SSA stochastic solver.

The following model will be constructed and stochastically simulated:

- Reaction 1: x -> z with a first-order reaction rate, c = 0.5.
- Initial conditions: x = 1000 molecules, z = 0.

This model can also be used to represent irreversible isomerization.

This example uses parameters and conditions as described in Daniel T. Gillespie, 1977, "Exact Stochastic Simulation of Coupled Chemical Reactions," The Journal of Physical Chemistry, vol. 81, no. 25, pp. 2340-2361.

Read the Radioactive Decay Model Saved in SBML Format

SBML = Systems Biology Markup Language, www.sbml.org

```
model  = sbmlimport('radiodecay.xml')
```

```
model = 
   SimBiology Model - RadioactiveDecay

   Model Components:
     Compartments:      1
     Events:            0
     Parameters:        1
     Reactions:         1
     Rules:             0
     Species:           2
     Observables:       0
```

View Species Objects of the Model

```
model.Species
```

```
ans = 
   SimBiology Species Array

   Index:    Compartment:    Name:    Value:    Units:
   1         unnamed         x        1000      molecule
   2         unnamed         z        0         molecule
```

View Reaction Objects of the Model

```
model.Reactions
```

```
ans = 
   SimBiology Reaction Array

   Index:    Reaction:
   1         x -> z
```

View Parameter Objects for the Kinetic Law

```
model.Reactions(1).KineticLaw(1).Parameters
```

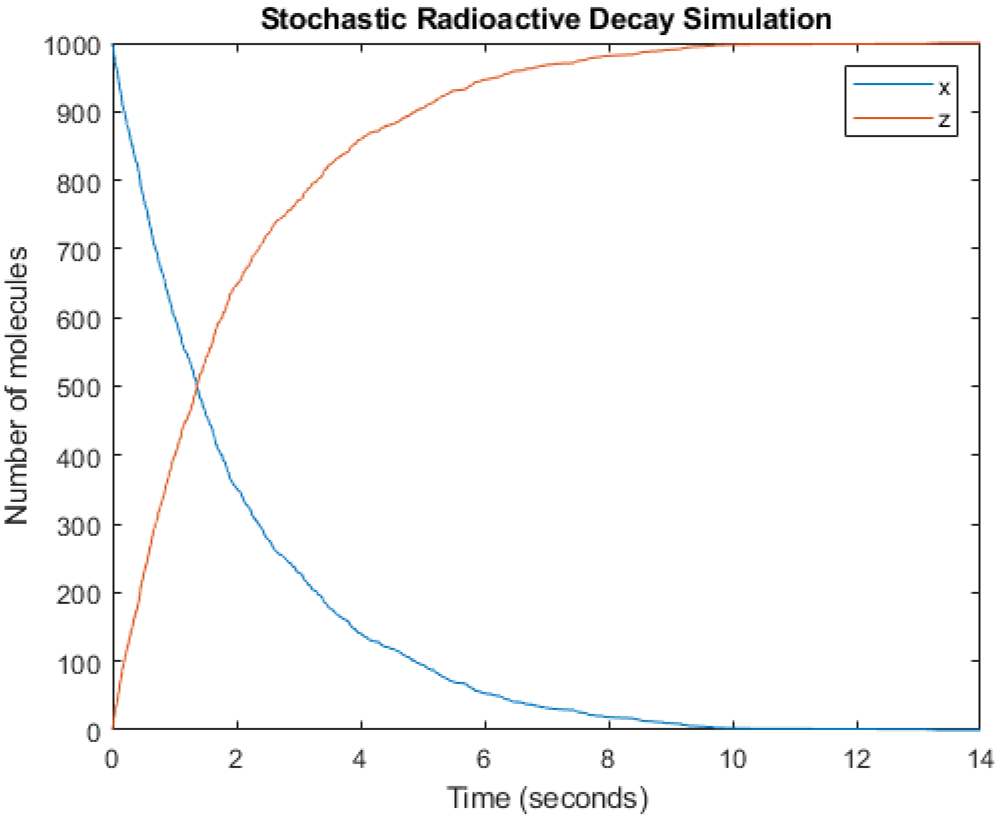
```
ans = 
   SimBiology Parameter Array

   Index:    Name:    Value:    Units:
   1         c        0.5       1/second
```

Update the Reaction to use MassAction Kinetic Law for Stochastic Solvers.

```
model.Reactions(1).KineticLaw(1).KineticLawName = 'MassAction';
model.Reactions(1).KineticLaw(1).ParameterVariableNames = {'c'};
```

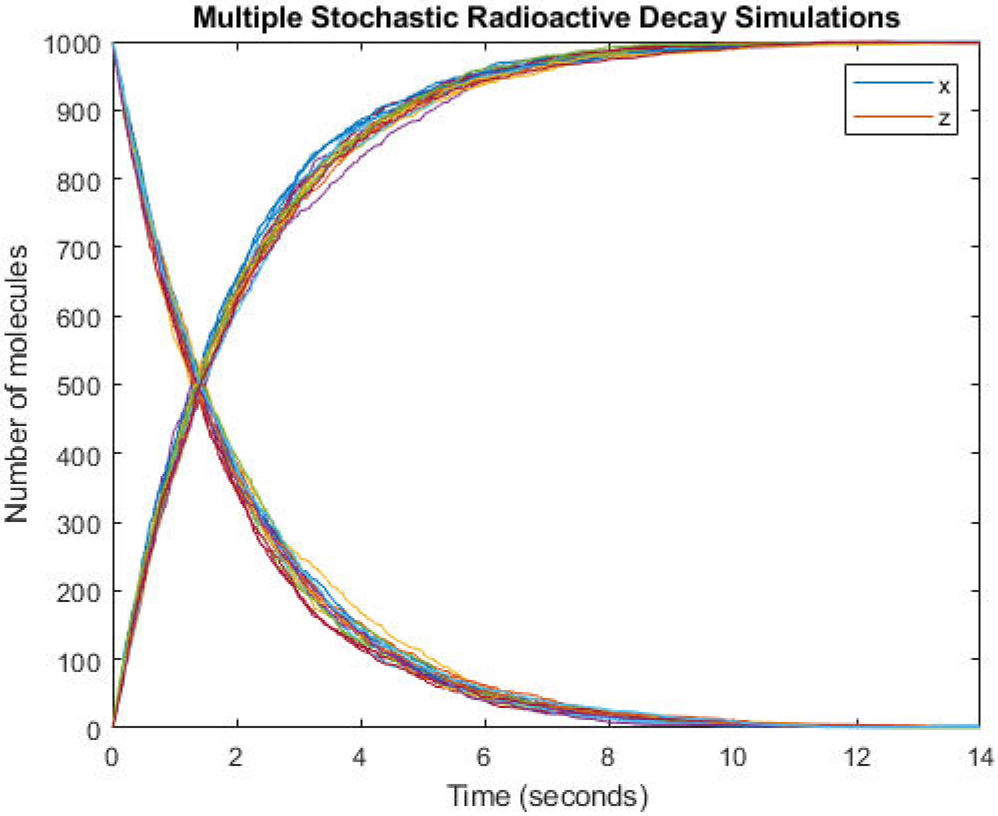
Simulate the Model Using the Stochastic (SSA) Solver & Plot

```
cs = getconfigset(model,'active');
cs.SolverType = 'ssa';
cs.StopTime = 14.0;
cs.CompileOptions.DimensionalAnalysis = false;
[t,X] = sbiosimulate(model);

plot(t,X);
legend('x', 'z', 'AutoUpdate', 'off');
title('Stochastic Radioactive Decay Simulation');
ylabel('Number of molecules');
xlabel('Time (seconds)');
```



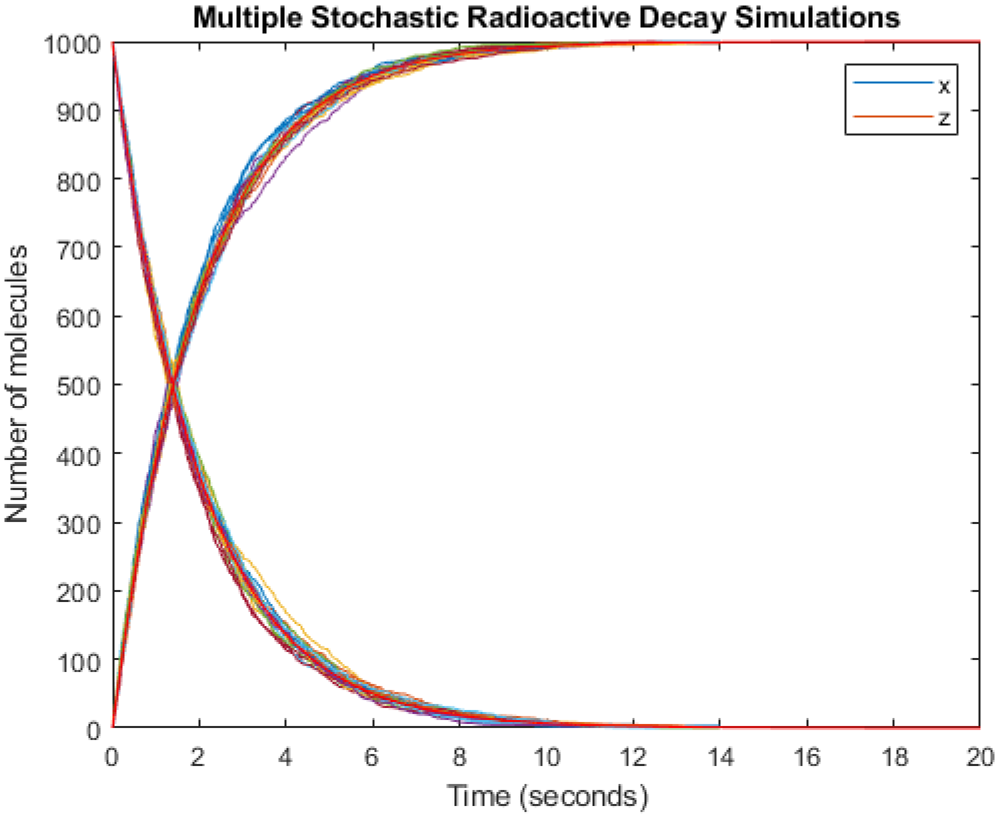
Repeat the Simulation to Show Run-to-Run Variability

```
title('Multiple Stochastic Radioactive Decay Simulations');
hold on;
for loop = 1:20
    [t,X] = sbiosimulate(model);
    plot(t,X);    % Just plot number of reactant molecules
    drawnow;
end
```



Overlay the Reaction's ODE Solution in Red

```
cs = getconfigset(model,'active');
cs.SolverType = 'sundials';
cs.StopTime = 20;
[t,X] = sbiosimulate(model);
plot(t,X,'red');
hold off;
```


Multiple Stochastic Radioactive Decay Simulations
Number of molecules
0
100
200
300
400
500
600
700
800
900
1000
Time (seconds)
0
2
4
6
8
10
12
14
16
18
20
x
z

Pharmacokinetic Modeling

Pharmacokinetic Modeling Functionality

Overview

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

- Create models — Use a model construction wizard. Alternatively, extend any model with pharmacodynamic (PD) model components, or build higher fidelity models. See "Model" on page 5-2 for more information.
- Fit data — Fit nonlinear, mixed-effects models to data, and estimate the fixed and random effects, or fit the data using nonlinear least squares. For more information, see "Analyze Data Using Models" on page 5-2.
- Generate diagnostic plots — For more information, see "Analyze Data Using Models" on page 5-2.

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

How SimBiology Supports Pharmacokinetic Modeling

Import and Work with Data

You can import tabular data into the SimBiology Model Analyzer or the MATLAB Workspace. The supported file types are `.xls`, `.csv`, and `.txt`. You can specify that the data is in a NONMEM® formatted file. The import process interprets the columns according to the NONMEM definitions. For details, see "Importing Data" on page 5-11.

Model

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

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

For more information, see "What is a SimBiology Model?" on page 2-2.

Analyze Data Using Models

Perform both individual and population fits to grouped longitudinal data:

- Individual fit — Fit data using nonlinear least-squares method, specify parameter transformations, estimate parameters, and calculate residuals and the estimated coefficient covariance matrix. For a command line workflow, see “Fitting Workflow for sbiofit” on page 4-41. For an app workflow, see “Calculate NCA Parameters and Fit Model to PK/PD Data Using SimBiology Model Analyzer App” on page 1-64.
- Population fit — Fit data, specify parameter transformations, and estimate the fixed effects and the random sources of variation on parameters using nonlinear mixed-effects models. For a command line workflow, see “Nonlinear Mixed-Effects Modeling Workflow” on page 4-32.
- Population fit using a stochastic algorithm — Fit data, specify parameter transformations, and estimate the fixed effects and the random sources of variation on parameters, using the Stochastic Approximation Expectation-Maximization (SAEM) algorithm. SAEM is more robust with respect to starting values. This functionality relaxes assumption of constant error variance. Specify `nlmefitsa` as the estimation function name when you run `sbiofitmixed`.

In addition, you can turn on the ProgressPlot on page 4-45 option to get the live feedback on the status of parameter estimation.

Pharmacokinetic Modeling Examples

The following examples show how to estimate pharmacokinetic parameters at the command line.

- “Modeling the Population Pharmacokinetics of Phenobarbital in Neonates”
- “Fit a One-Compartment Model to an Individual's PK Profile” on page 4-52
- “Fit a Two-Compartment Model to PK Profiles of Multiple Individuals” on page 4-71
- “Estimate Category-Specific PK Parameters for Multiple Individuals” on page 4-58
- “Perform Hybrid Optimization Using sbiofit” on page 4-67

For an app example, see “Calculate NCA Parameters and Fit Model to PK/PD Data Using SimBiology Model Analyzer App” on page 1-64.

Acknowledgements: Tobramycin Data Set

Acknowledgements for data in the `tobramycin.txt` file are located in the `/matlab/toolbox/simbio/simbiodemos` folder. Data set is provided by Dr. Leon Aarons (`laarons@fs1.pa.man.ac.uk`).

The data in the `tobramycin.txt` file were downloaded from the Web site of the Resource Facility for Population Kinetics `http://depts.washington.edu/rfpk/service/datasets/index.html` (no longer active). Funding source: NIH/NIBIB grant P41-EB01975.

The original data set was modified as follows:

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

References

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

See Also

`sbiofit` | `sbiofitmixed`

More About

- “Nonlinear Regression”
- “Nonlinear Mixed-Effects Modeling”

Importing Data — Supported Files and Data Types

In this section...

Supported Files and Data Types

You can import tabular data to the **SimBiology Model Analyzer** app or to the MATLAB Workspace. The supported file types are Excel files (.xls, .xlsx), text files (.csv, .txt), and SAS® XPORT files (.xpt). You can also specify that the data is in a NONMEM formatted file. The import process interprets the columns according to the NONMEM definitions. For more information see “Support for Importing NONMEM Formatted Files” on page 5-5.

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

Unit Conversion

Regardless of whether unit conversion functionality is on or off, dosing in the data file must be expressed in amounts (or as `amount/time` for infusion rate). By default **Unit Conversion** is off, so you must ensure that units for the data are consistent with each other. If you want to turn on unit conversion, see “Unit Conversion for Imported Data” on page 5-19 .

Support for Importing NONMEM Formatted Files

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

Column Header	Interpretation
`ID`	Text (character vector), numeric, or categorical values that identify the record or group. The import process assumes that contiguous data with the same value contains data from one individual. If the data contains non-contiguous references to the same value, the import process assigns the second ID encountered an indexed valued derived from the group first encountered. For example, if the ID columns contains `[1 1 1 2 2 2 1 1 1]`, the IDs assigned are `1, 2, 1_1`.

<table>
<tr><th>Column Header</th><th>Interpretation</th></tr>
<tr><td><code>TIME</code></td><td>Monotonically increasing positive values within each group, indicating time of observation or dose or text (character vector). The data file can specify clock (<code>2:30</code> as a character vector) or decimal values (<code>6.25</code>). The import process assigns a value of <code>0</code> to the first <code>TIME</code> value in the data file. The import process assigns subsequent values relative to the first value.

The following table is an example of how the import process interprets the clock values as decimal values.

<table>
<tr><th>Original Clock Values</th><th>Imported Values</th></tr>
<tr><td>10:00</td><td>0</td></tr>
<tr><td>10:30</td><td>0.5</td></tr>
<tr><td>11</td><td>1</td></tr>
<tr><td>12:30</td><td>2.5</td></tr>
</table>

If the data file also contains a <code>DATE</code> column, the import process uses it with the <code>TIME</code> column in calculating the relative <code>TIME</code> values. The column cannot contain <code>Inf</code>.</td></tr>
</table>

Column Header	Interpretation
`DATE`, `DAT1`, `DAT2`, or `DAT3`	Defines the day of the observation or the dose. The column can contain the month as a number (`9`) or a character vector (`Sep`). Specify date in the following formats: • `DATE` — The column can specify `month/day/year` or `month-day-year`. If you specify two numbers, the import process assumes they are month and day. You can use either `/` or `-` as a separator. • `DAT1` — The column can specify `day/month/year` or `day-month-year`. If you specify two numbers, the import process assumes they are day and month. • `DAT2` — The column can specify `year/month/day` or `year-month-day`. If you specify two numbers, the import process assumes they are month and day. • `DAT3` — The column can specify `year/day/month` or `year-day-month`. If you specify two numbers, the import process assumes they are day and month. **Note** • If you specify only one number, the import process assumes it is the day. • You can omit the year or specify 1, 2, 3, or 4 digits. If you specify two-digit years, it is assumed to be in the 1900s. • If the data has the `DAT1`, `DAT2`, or `DAT3` column, set the `DateLabel` property of a `NMFileDef` object accordingly using `sbionmfiledef`. Then specify the object as the second input argument when you run `sbionmimport`.
`DV`	Numeric value of an observation. Column cannot contain `Inf` or —`Inf`.
`MDV`	Defines whether a row describes an observation: • Row contains 0 — Observation event • Row contains 1 — Not an observation event

Column Header	Interpretation
`EVID`	Defines the type of event described for the row in the record: • 0 — Observation event; row contains an observed value. • 1 — Dose event; row describes a dose. • 2 — Other event; row describes some other event such as measurement of a covariate. If a column contains values for dose, but `EVID` is not `1`, the import process ignores the value. You see a warning and the value is ignored. If `EVID` is set to 2, then only those specified row data are imported as covariate data. However, if you have an `EVID` column as well as one or more covariate columns, but do not specify a value of `2` anywhere in the `EVID` column, then SimBiology imports all the row data as covariate values. The import process does not support values `3` and `4`. You see a warning and the value is ignored.
`CMT`	Indicates which compartment is used for observation value or for dose received. The interpretation also depends on `EVID`: • Observation event (`EVID = 0`) — `CMT` column indicates which compartment was used for observation value. • Dose Event (`EVID = 1`) — `CMT` column indicates which compartment received the dose. **Note** SimBiology numbers compartments starting with 1, while NONMEM numbers them starting with 0. For instance, if a NONMEM data file contains doses and measurements for `CMT = 0`, SimBiology generates data columns named `Dose1` and `Response1` respectively.
`AMT`	Positive number indicating dose. `0` or `NaN` specifies no dose administered. The column cannot contain `Inf`.
`RATE`	Positive number indicating rate of infusion. `0` specifies an infinite rate (equivalent to a bolus dose), and `NaN` specifies no rate. The column cannot contain `Inf`.
`II`	Positive number defining the time between doses.
`ADDL`	When the data specifies a number of identical serial doses at specific intervals (defined by `II`), `ADDL` specifies the number of doses in the series excluding the initial dose. If the data specifies `II` but not `ADDL`, then SimBiology assumes that the dosing occurs for the duration of that data record.

Unsupported NONMEM Definitions

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

- `EVID` values 3 and 4
- `SS` column for specifying steady state doses
- `PCMT` column to define whether to compute a prediction for the row
- `CALL` column for calling the ERROR or the PK subroutine
- If rate is specified as being less than zero, it is assumed to be zero

Creating a Data File with SimBiology Definitions

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

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

  If you set an `EVID` value of `2` for some rows, then SimBiology imports only those rows as covariate data. If you do not mention an `EVID` value of `2` anywhere and have one or more covariate columns, then SimBiology imports all the row data as covariate data.

Supported Table Column Types in SimBiology Model Analyzer

When you are importing data from a table using **SimBiology Model Analyzer**, the app supports the following column data types: `double`, `char`, cell array of character vectors, `categorical`, `duration`, and `datetime`.

Support for Importing Multidimensional SimData to SimBiology Model Analyzer

When you import a multidimensional `SimData` array to the app, the app flattens the `SimData` array and uses a single index (linear indexing) in the corresponding datasheet. For example, if you import a `2x2x2` SimData array *A*, the app creates a datasheet with 8 groups (one for each `SimData` object), indexing from 1 to 8. The app still displays the original size of the `SimData` array in the **Browser**.

See Also

SimBiology Model Analyzer | `sbiofit` | `sbiofitmixed`

Importing Data

In this section...

Import Data from Files

Use `table` to store tabular data that you can use later in fitting and analysis at the command line. Use `readtable` to import data without NONMEM interpretation of column headers. The `readtable` function lets you use name-value pair arguments in which you can specify options such as the type of delimiter and whether the first row contains header names. If you want to use the data for fitting using `sbiofit` or `sbiofitmixed`, convert it to a `groupedData` format using `groupedData`.

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

For example:

```
% Text files
data = readtable('tobramycin.txt');
% Text files with . in place of missing values
data = readtable('tobramycin.txt', 'TreatAsEmpty', '.');
```

Importing Data from NONMEM-Formatted Files

Use the `sbionmimport` function to import data from NONMEM formatted files. To import the data without NONMEM interpretation of column headers, see "Import Data from Files" on page 5-11.

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

- If the data file contains only the column header values shown in "Support for Importing NONMEM Formatted Files" on page 5-5, use the syntax shown in the following example:

  ```
  filename = 'C:\work\datafiles\dose.xls';
  ds = sbionmimport(filename);
  ```
- If the data file has column header labels different from the table shown in "Support for Importing NONMEM Formatted Files" on page 5-5 and you want to apply NONMEM interpretation of headers:

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

     To use the tobramycin data set [1] on page 5-3, create a NONMEM file definition object and define the following:

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

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

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

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

```
[data, pkDataObject] = sbionmimport('tobramycin.txt', def, ...
   'TreatAsEmpty', '.');
```

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

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

For information about creating a model to fit the data, see “Create a Pharmacokinetic Model Using the Command Line” on page 5-14.

Other Resources for Importing Data

For detailed information about supported data formats and the functions for importing data into the MATLAB Workspace, see the “Methods for Importing Data” (MATLAB).

You also can import data using the MATLAB Import Wizard (see “Import Images, Audio, and Video Interactively” (MATLAB). Use the Import Wizard, to import data as text files (such as `.txt` and `.dat`), MAT-files, and spreadsheet files, (such as `.xls`).

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

Create Pharmacokinetic Models

In this section...

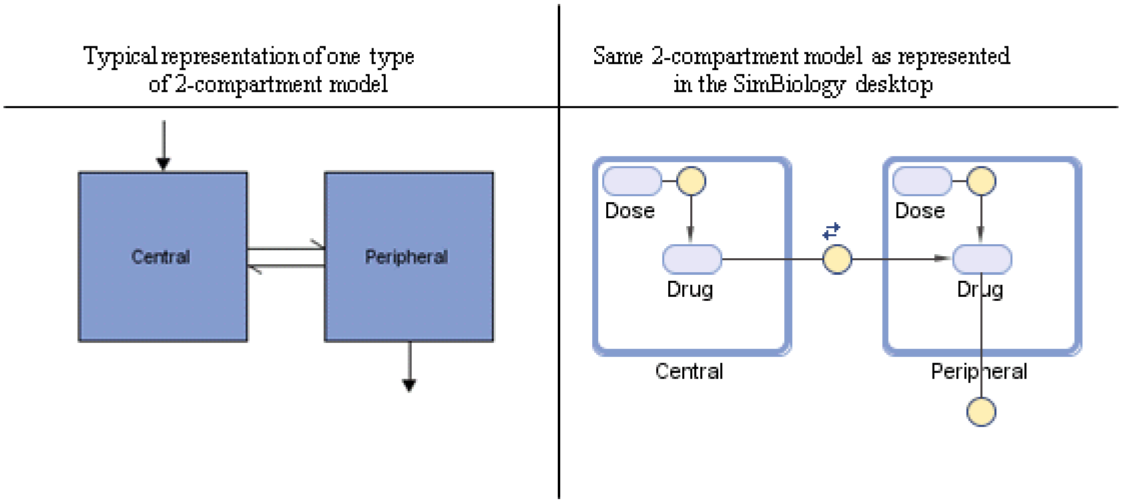
Ways to Create or Import Pharmacokinetic Model

To start modeling, you can:

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

How SimBiology Models Represent Pharmacokinetic Models

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



Typical representation of one type of 2-compartment model

Same 2-compartment model as represented in the SimBiology desktop

Comparison of views in PK

Note the following:

- SimBiology represents the concentration or amount of a drug in a given compartment or volume by a species *object* contained within the compartment.
- SimBiology represents the exchange or flow of the drug between compartments and the elimination of the drug by reactions.
- SimBiology represents intercompartmental clearance by a parameter (Q) which specifies the clearance between the compartments.
- SimBiology drives the dosing schedule with a combination of species (Drug and/or Dose) and reactions (Dose -> Drug), depending on whether the administration into the compartment follows bolus, zero-order, infusion, or first-order dosing kinetics. For more information on the components added and parameters estimated, see “Dosing Types” on page 5-15.

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

Create a Pharmacokinetic Model Using the Command Line

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

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

   ```
   pkm = PKModelDesign;
   ```
2. Add a compartment specifying the compartment name, and optionally, the type of dosing, and the method of elimination. Also specify whether the data contains a response variable measured in this compartment and whether the dose(s) have time lags. For example, if using the tobramycin data set [1] on page 5-3, specify a compartment named Central, with Bolus for the DosingType property, linear-clearance for the EliminationType property, and true for the HasResponseVariable property.

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

   For a description of other DosingType and EliminationType property values, see “Dosing Types” on page 5-15 and “Elimination Types” on page 5-17.

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

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

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

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

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

3 Optionally, add a second compartment named `Peripheral`, with no dosing, no elimination, and no time lag. Set the `HasResponseVariable` property to `true`. If you are using the tobramycin data set [1] on page 5-3, skip this step and use only one compartment.

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

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

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

4 Construct a SimBiology model object.

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

The `construct` method returns a SimBiology model object (`modelObj`) and a `PKModelMap` object (`PKModelMapObj`) that contains the mapping of the model components to the elements of the regression function.

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

5 Perform parameter fitting as shown in “Perform Data Fitting with PK/PD Models” on page 5-20.

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

For information on ...	See ...
Dosing types	“Dosing Types” on page 5-15
Elimination types	“Elimination Types” on page 5-17
Parameter fitting	“Perform Data Fitting with PK/PD Models” on page 5-20
Simulating the model and a description of configuration sets	“Model Simulation” on page 4-2

Dosing Types

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

- Two species (`Drug_CompartmentName` and `Dose_CompartmentName`) for each compartment.
- A reaction (`Dose_CompartmentName -> Drug_CompartmentName`) for each compartment, governed by mass action kinetics.
- A parameter (`ka_CompartmentName`) for each compartment, representing the absorption rate of the drug when absorption follows first-order kinetics. This is the forward rate parameter for the `Dose_CompartmentName -> Drug_CompartmentName` reaction.
- A parameter (`Tk0_CompartmentName`) for each compartment, representing the duration of drug absorption when absorption follows zero-order kinetics.

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

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

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

Dosing Type	Description	SimBiology Model Components Used	Default Parameters to Estimate
''(empty character vector)	No dose	The species (Drug_CompartmentName) in each compartment	None
SimBiology app — bolus Command line — Bolus	Assumes that the drug amount is increased instantly at the dose time. In the SimBiology model, the initial concentration of the drug is based on dose amount and volume of the compartment containing the drug.	The species (Drug_CompartmentName) in each compartment	None
SimBiology app — infusion Command line — Infusion	Assumes that the infused drug amount increases at a constant known absorption (or infusion) rate over a known duration. The imported data set must contain the rate and not an infusion duration. SimBiology uses this information to change the species concentration at the constant rate over the duration specified in the data set.	The species (Drug_CompartmentName) in each compartment	None
SimBiology app — zero-order Command line — ZeroOrder	Assumes that the drug is added at a constant rate over fixed, but unknown duration.	• The species Drug_CompartmentName in each compartment • The parameter (Tk0_CompartmentName) in each compartment that has zero-order dosing. This parameter represents the duration of drug absorption	Tk0_CompartmentName (absorption duration)

Dosing Type	Description	SimBiology Model Components Used	Default Parameters to Estimate
SimBiology app — `first-order` Command line — `FirstOrder`	Assumes that the rate at which the drug is absorbed is not constant. In the SimBiology model, absorption rate is assumed to be governed by mass-action kinetics.	• A species (`Dose_CompartmentName`) representing the dose amount before it is absorbed • A species (`Drug_CompartmentName`) for each compartment • A parameter (`ka_CompartmentName`) representing the absorption rate of the drug • A `MassAction reaction (Dose_CompartmentName —> Drug_CompartmentName)` with forward rate parameter (`ka_CompartmentName`)	`ka_CompartmentName` (absorption rate)

Elimination Types

Elimination Type	Description	SimBiology Model Components Created	Default Parameters to Estimate
SimBiology app — `Linear {Elimination Rate, Volume}` Command line — `'linear'`	Assumes simple mass-action kinetics in the elimination of the drug. In the SimBiology model, elimination is specified by mass-action kinetics with the elimination rate constant specified by the forward rate parameter (`ke`).	• A parameter representing the elimination rate (`ke_CompartmentName`) • A `MassAction reaction (drug —> null)` with forward rate parameter (`ke_CompartmentName`) specific to the compartment	• Compartment volume (`Capacity` property) • Elimination rate constant (`ke_CompartmentName`) • Inter-compartmental clearance (`Q`) when there is more than one compartment. See “Intercompartmental Clearance” on page 5-18.

Elimination Type	Description	SimBiology Model Components Created	Default Parameters to Estimate
SimBiology app — `Linear {Clearance, Volume}` Command line — `'linear-clearance'`	Assumes simple mass-action kinetics in the elimination of the drug. In the SimBiology model, similar to `Linear {Elimination Rate, Volume}`. But, in addition, this option lets you specify the model in terms of clearance (`Cl`) where, `Cl = ke * volume`).	• A parameter representing the clearance (`Cl_CompartmentName`) • A parameter representing the elimination rate constant (`ke_CompartmentName`) • An `InitialAssignment` rule that initializes `ke_CompartmentName` based on the initial values for `Cl_CompartmentName` and compartment volume • A `MassAction` `reaction (drug –> null)` with forward rate parameter (`ke_CompartmentName`)	• Compartment volume (`Capacity` property) • Clearance (`Cl_CompartmentName`) • Inter-compartmental clearance (`Q`) when there is more than one compartment. See “Intercompartmental Clearance” on page 5-18.
SimBiology app — `Enzymatic (Michaelis-Menten)` Command line — `'enzymatic'`	Assumes that elimination is governed by Michaelis-Menten kinetics.	• Parameter representing the Michaelis constant, (`Km_CompartmentName`) • A parameter for maximum velocity (`Vm_CompartmentName` • A reaction with Michaelis-Menten kinetics (`drug -> null`), with kinetic law parameters `Vm_CompartmentName` and `Km_CompartmentName`	• Compartment volume (`Capacity` property) • Parameter (`Km_CompartmentName`) • Parameter (`Vm_CompartmentName`) • Inter-compartmental clearance (`Q`) when there is more than one compartment. See “Intercompartmental Clearance” on page 5-18

Intercompartmental Clearance

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

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

- A parameter Q_{mn} representing the compartmental clearance between those two compartments. This parameter is added to the list of parameters to be estimated (`Estimated` property of `PKModelMap` object).
- A parameter (`kmn`) representing the rate of transfer of the drug from `Cm` to `Cn`, where $k_{mn} = Q_{mn}/V_m$.
- A parameter (`knm`) representing the rate of `Cn` to `Cm`, where $k_{nm} = Q_{mn}/V_n$.
- A reversible mass-action reaction between the two compartments, `Cm.Drug_Cm <-> Cn.Drug_Cn`, with forward rate parameter `kmn`, and reverse rate parameter `knm`.
- An initial assignment rule that initializes the value of the parameter `kmn`, based on the initial values for `Cm` and `Qmn`.
- An initial assignment rule that initializes the value of the parameter `knm`, based on the initial values for `Cn` and `Qmn`.

Unit Conversion for Imported Data

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

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

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

Physical Quantity or Model Parameter	Unit
Central or peripheral compartment volume (*Central* or *Peripheral*)	`liter`
First-order elimination rate (*ke*)	`1/second`
Michaelis constant (*Km*)	`milligram/liter`
Maximum reaction-velocity, Michaelis-Menten kinetics (*Vm*)	`milligram/second`
Clearance (*Cl*)	`liter/second`
Absorption duration (*Tk0*)	`second`
Absorption rate (*ka*)	`1/second`

Use the configuration settings options to turn unit conversion `on` or `off`. For details, see “Model Simulation” on page 4-2.

For details on dimensional analysis for reaction rates, see “How Reaction Rates Are Evaluated” on page 2-11.

Perform Data Fitting with PK/PD Models

SimBiology lets you estimate model parameters by fitting the model to experimental time-course data, using either nonlinear regression or mixed-effects (NLME) techniques. You can perform both individual and population fits to grouped data.

- Individual fit — Fit data using nonlinear regression (least-squares) methods, specify parameter transformations, estimate parameters, and calculate residuals and the estimated coefficient covariance matrix. For a command line workflow, see “Fitting Workflow for sbiofit” on page 4-41. For an app workflow, see “Calculate NCA Parameters and Fit Model to PK/PD Data Using SimBiology Model Analyzer App” on page 1-64.
- Population fit — Fit data, specify parameter transformations, and estimate the fixed effects and the random sources of variation on parameters using nonlinear mixed-effects models. For a command line workflow, see “Nonlinear Mixed-Effects Modeling Workflow” on page 4-32.
- Population fit using a stochastic algorithm — Fit data, specify parameter transformations, and estimate the fixed effects and the random sources of variation on parameters, using the Stochastic Approximation Expectation-Maximization (SAEM) algorithm. SAEM is more robust with respect to starting values. This functionality relaxes assumption of constant error variance. Specify `nlmefitsa` as the estimation function name when you run `sbiofitmixed` or select `mixed effects using stochastic solver` in the **Statistical Modeling** section of the Fit Data program in the **SimBiology Model Analyzer** app.

In addition, you can turn on the ProgressPlot on page 4-45 option to get the live feedback on the status of parameter estimation.

See Also

`sbiofit` | `sbiofitmixed`

More About

- “Nonlinear Regression” on page 4-37
- “Nonlinear Mixed-Effects Modeling” on page 4-31

Creating Reaction Rates

Define Reaction Rates with Mass Action Kinetics

Use mass action kinetics to define zero-order, first-order, second-order, and reversible reactions.

Definition of Mass Action Kinetics

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

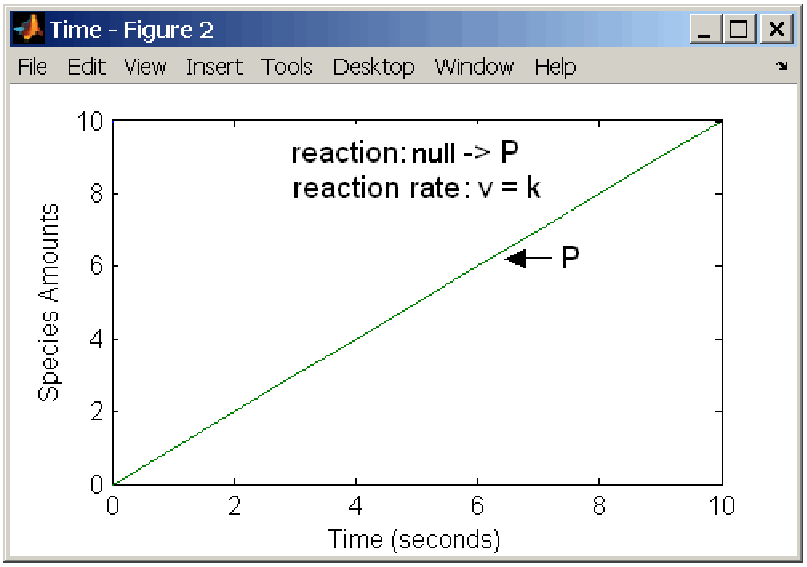
Zero-Order Reactions

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

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

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

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



Zero-Order Mass Action Kinetics

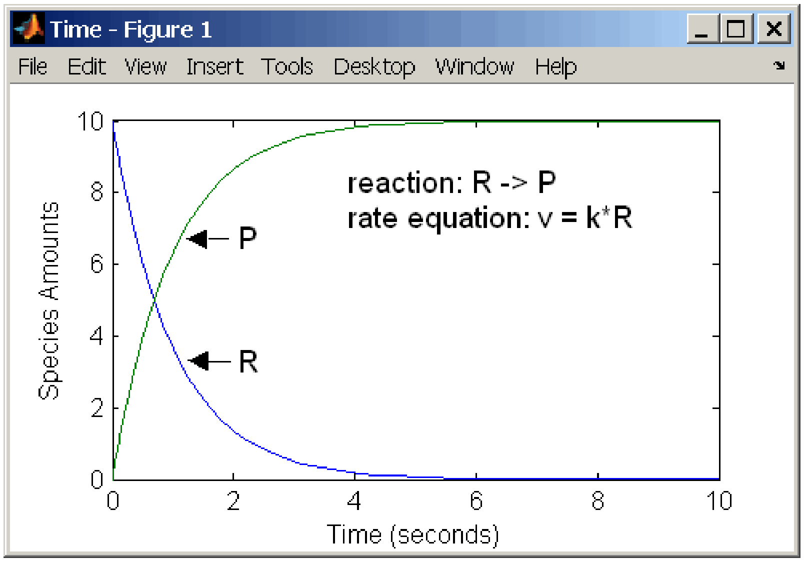
Note If the amount of a reactant with zero-order kinetics reaches zero before the end of a simulation, then the amount of reactant can go below zero regardless of the solver or tolerances you set.

First-Order Reactions

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

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

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



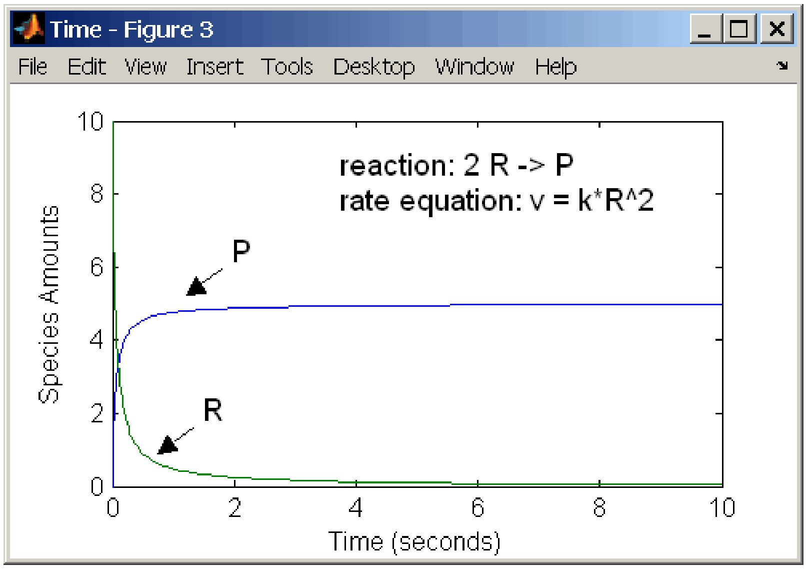
First-Order Mass Action Kinetics

Second-Order Reactions

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

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

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

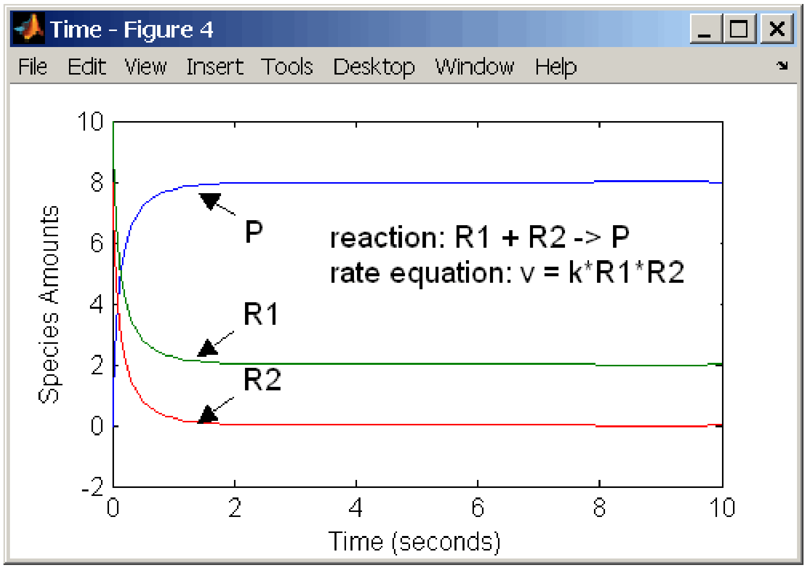


Second-Order Kinetics with Single Reactant

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

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

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



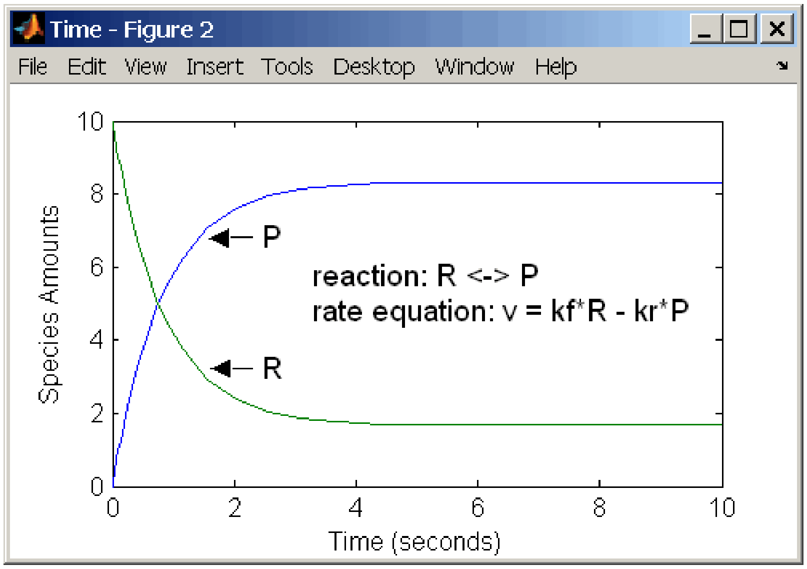
Second-Order Kinetics with Two Reactants

Reversible Mass Action

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

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

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


Time - Figure 2
File Edit View Insert Tools Desktop Window Help
Species Amounts
Time (seconds)
0
2
4
6
8
10
P
R
reaction: R <-> P
rate equation: v = kf*R - kr*P

Define Reaction Rates with Enzyme Kinetics

Use differential equations, mass action kinetics, or Michaelis-Menten kinetics to define enzyme reactions.

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.

$$S + E \underset{k1r}{\overset{k1}{\rightleftarrows}} ES \overset{k2}{\rightarrow} E + P$$

$$v1 = k1[S][E],\quad v1r = k1r[ES],\quad v2 = k2[ES]$$

This simple model can be defined with

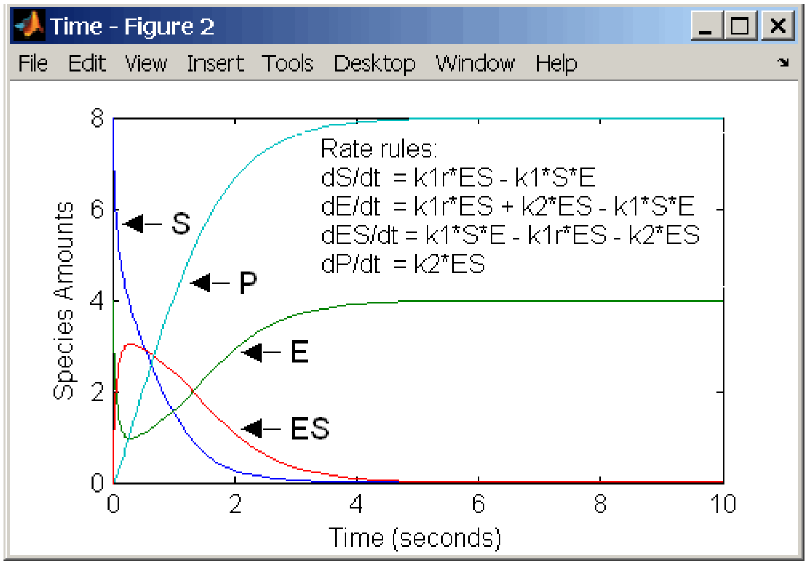
- Differential rate equations. See “Enzyme Reactions with Differential Rate Equations” on page A-6.
- Reactions with mass action kinetics. See “Enzyme Reactions with Mass Action Kinetics” on page A-7.
- Reactions with Henri-Michaelis-Menten kinetics. See “Enzyme Reactions with Irreversible Henri-Michaelis-Menten Kinetics” on page A-8.

Enzyme Reactions with Differential Rate Equations

The reactions for a single-substrate enzyme reaction mechanism (see “Simple Model for Single Substrate Catalyzed Reactions” on page A-6) can be described with differential rate equations. You can enter the differential rate equations into the software as rate rules.

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

Remember that the rate rule `dS/dt = f(x)` is written in a SimBiology rate rule expression as `S = f(x)`. For more information about rate rules see “Rate Rules” on page 2-14.



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

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

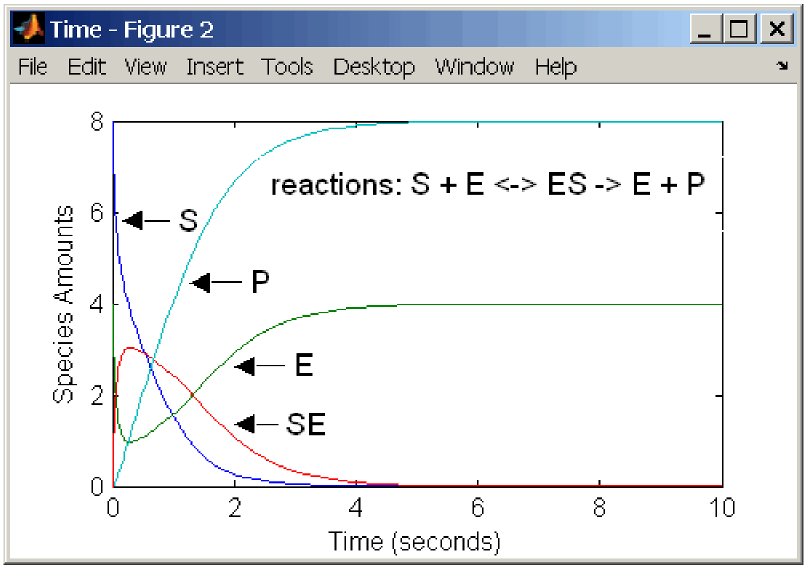
Enzyme Reactions with Mass Action Kinetics

Determining the differential rate equations for the reactions in a model is a time-consuming process. A better way is to enter the reactions for a single substrate enzyme reaction mechanism directly into the software. The following example using models an enzyme catalyzed reaction with mass action kinetics. For a description of the reaction model, see “Simple Model for Single Substrate Catalyzed Reactions” on page A-6.

```
     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.


Time - Figure 2
File Edit View Insert Tools Desktop Window Help
reactions: S + E <-> ES -> E + P
S
P
E
SE
Species Amounts
Time (seconds)
0
2
4
6
8
0
2
4
6
8
10

Enzyme Reactions with Irreversible Henri-Michaelis-Menten Kinetics

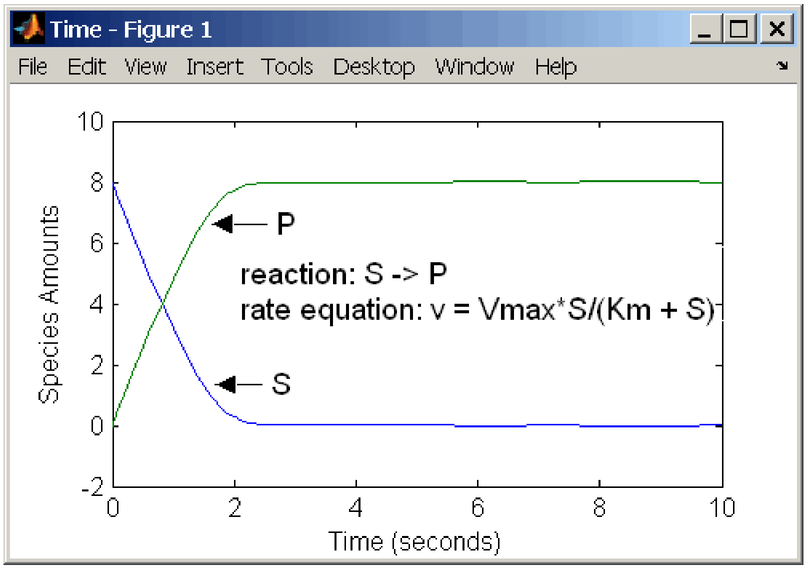
Representing an enzyme-catalyzed reaction with mass action kinetics requires you to know the rate constants `k1`, `k1r`, and `k2`. However, these rate constants are rarely reported in the literature. It is more common to give the rate constants for Henri-Michaelis-Menten kinetics with the maximum velocity `Vm=k2*E` and the constant `Km = (k1r + k2)/k1`. The reaction rate for a single substrate enzyme reaction using Henri-Michaelis-Menten kinetics is given below. For information about the model, see “Simple Model for Single Substrate Catalyzed Reactions” on page A-6.

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

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.


Time - Figure 1
File Edit View Insert Tools Desktop Window Help
Species Amounts
Time (seconds)
10
8
6
4
2
0
-2
0
2
4
6
8
10
P
S
reaction: S -> P
rate equation: v = Vmax*S/(Km + S)

B

Models Used in Examples

Minimal Cascade Model for a Mitotic Oscillator

Albert Goldbeter modified a model with enzyme cascades [Goldbeter and Koshland 1981 on page B-13] to fit cell cycle data from studies with embryonic cells [Goldbeter 1991 on page B-13]. He used this model to demonstrate thresholds with enzyme cascades and periodic behavior caused by negative feedback.

There are two SimBiology model variations using Goldbeter's model. The first model uses the differential rate equations directly from Goldbeter's paper. The second model is built with reactions using Henri-Michaelis-Menten kinetics.

In this section...

Goldbeter Model

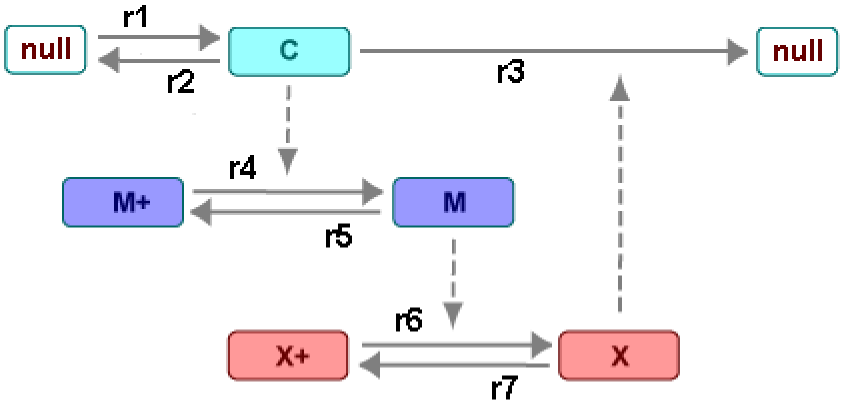
About the Goldbeter Model

Albert Goldbeter created a simple cell division model from studies with embryonic cells [Goldbeter 1991 on page B-13]. This model demonstrates thresholds with enzyme cascades and periodic behavior caused by negative feedback.

There are six species in Goldbeter's minimal mitotic oscillator model [Goldbeter 1991 on page B-13].

- C — Cyclin. The periodic behavior of cyclin activates and deactivates an enzyme cascade.
- M+, M — Inactive (phosphorylated) and active forms of cdc2 kinase. Kinases catalyze the addition of phosphate groups onto amino acid residues.
- X+, X — Inactive and active (phosphorylated) forms of a cyclin protease. Proteases degrade proteins by breaking peptide bonds.

The reactions are labeled `r1` to `r7` on the following diagram.



This model shows:

- How thresholds with cdc2 kinase activation (M+ -> M) and protease activation (X+ -> X) can occur as the result of covalent modification (for example, phosphorylation or dephosphorylation), but without the need for positive feedback.
- How periodic behavior with cdc2 kinase activation can occur with negative feedback and the time delay associated with activation/deactivation enzyme cascades.

Reaction Descriptions and Model Assumptions

The following list describes each of the reactions in Goldbeter's minimal mitotic oscillator with some of the simplifying assumptions. For a more detailed explanation of the model, see [Goldbeter 1991 on page B-13].

- Cyclin (C) is synthesized at a constant rate (r1) and degraded at a constant rate (r2).
- Cyclin (C) does not complex with cdc2 kinase (M).
- Cyclin (C) activates cdc2 kinase (M+ -> M) by increasing the velocity of the phosphatase that activates the kinase. Inactive cdc2 kinase (M+) is activated by removing inhibiting phosphate groups (r4).
- The amount of deactivating kinase (not modeled) for the cdc2 kinase (M) is constant. Active cdc2 kinase (M) is deactivated by adding inhibiting phosphate group (r5).
- The activation of cyclin protease (X+ -> X) by the active cdc2 kinase (M) is direct without other intervening cascades. Cyclin protease (X) is activated by adding phosphate groups (r6).
- The amount of deactivating phosphatase (not modeled) for the cyclin protease (X) is constant. Active cyclin protease (X) is deactivated by removing the activating phosphate groups (r7).
- The three species of interest are cyclin (C), active dephosphorylated cdc2 kinase (M), and active phosphorylated protease (X). The total amounts of (M + M+) and (X + X+) are constant.

Mathematical Model

Goldbeter's minimal mitotic oscillator model is defined with three differential rate equations and two algebraic equations that define changing parameters in the rate equations.

Differential Rate Equation 1, Cyclin (C)

The following differential rate equation is from [Goldbeter 1991 on page B-13] for cyclin (C).

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

Differential Rate Equation 2, Kinase (M)

The following differential rate equation is for cdc2 kinase (M). Notice that (1 - M) is the amount of inactive (phosphorylated) cdc2 kinase (M+).

$$\frac{dM}{dt} = V_1 \frac{(1 - M)}{K_1 + (1 - M)} - V_2 \frac{M}{K_2 + M}$$

$$V_1 = \frac{VM_1[C]}{K_c + [C]}$$

Differential Rate Equation 3, Protease (X)

Differential rate equations for cyclin protease (X). Notice that (1 - X) is the amount of inactive (unphosphorylated) cyclin protease (X+).

$$\frac{dX}{dt} = V_3 \frac{(1 - X)}{K_3 + (1 - X)} - V_4 \frac{X}{K_4 + X}$$

$$V_3 = VM_3[M]$$

SimBiology Model with Rate Rules

- “SimBiology Model with Rules” on page B-4
- “SimBiology Simulation with Rules” on page B-5

SimBiology Model with Rules

In the literature, many biological models are defined using differential rate and algebraic equations. With SimBiology software, you can enter the equations directly as SBML rules. The example in this section uses Goldbeter's mitotic oscillator to illustrate this point.

Writing differential rate equations in an unambiguous format that a software program can understand is a fairly simple process.

- Use an asterisk to indicate multiplication. For example, `k[a]` is written `k*a`.
- Remove square brackets that indicate concentration from around species. The units associated with the species will indicate concentration (`moles/liter`) or amount (`moles`, `molecules`).

  SimBiology software uses square brackets around species and parameter name to allow names that are not valid MATLAB variable names. For example, you could have a species named `glucose-6-phosphate dehydrogenase` but you need to add brackets around the name in reaction rate and rule equations.
- Use parentheses to clarify the order of evaluation for mathematical operations. For example, do not write a Henri-Michaelis-Menten rate as `Vm*C/Kd + C`, because `Vm*C` is divided by `Kd` before adding `C`, and then `C` is added to the result.

The following equation is the rate rule for “Differential Rate Equation 1, Cyclin (C)” on page B-3:

```
dC/dt = vi - (vd*X*C)/(Kd + C) - kd*C
```

The following equations are the rate and `repeatedAssignment` rules for “Differential Rate Equation 2, Kinase (M)” on page B-4:

```
dM/dt = (V1*Mplus)/(K1 + Mplus) - (V2*M)/(K2 + M)
V1 = (VM1*C)/(Kc + C)
Mplus = Mt - M
```

The following equations are the rate and `repeatedAssignment` rules for “Differential Rate Equation 3, Protease (X)” on page B-4:

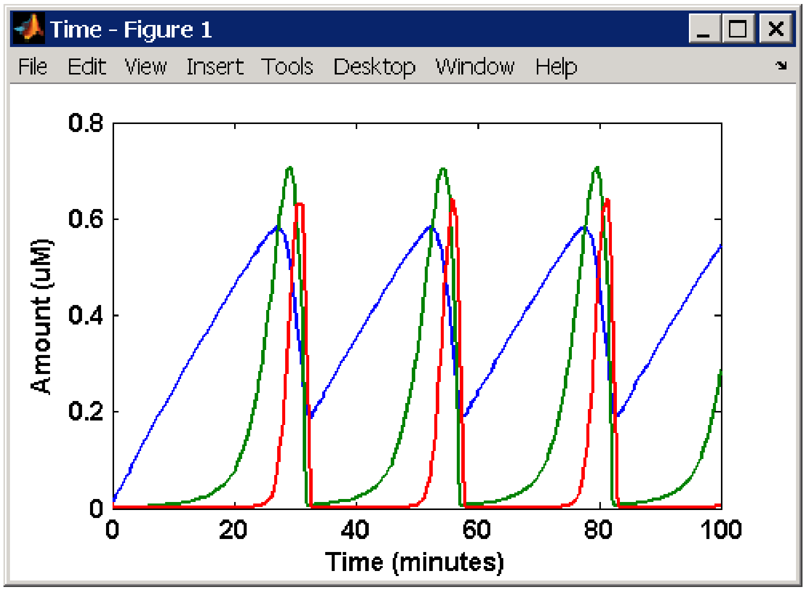
```
dX/dt = (V3*Xplus)/(K3 + Xplus) - (V4*X)/(K4 + X)
V3 = VM3*M
Xplus = Xt - X
```

Rules

The active (`M`) and inactive (`Mplus`) forms of the kinase are assumed to be part of a conserved cycle with the total concentration (Mt) remaining constant during the simulation. You need only one differential rate equation with a mass balance equation to define the amounts of both species. Similarly, the active (`X`) and inactive (`Xplus`) forms of the protease are part of a second conserved cycle.

SimBiology Simulation with Rules

This is a simulation of Goldbeter's minimal mitotic oscillator using differential rate and algebraic equations. Simulate with the `sundials` solver and plot species `C`, `M`, and `X`. For a description of the model, see “SimBiology Model with Rules” on page B-4.



SimBiology Model with Reactions

- “Converting Differential Rate Equations to Reactions” on page B-6
- “Calculating Initial Values for Reactions” on page B-7
- “SimBiology Simulation with Reactions” on page B-12

Converting Differential Rate Equations to Reactions

In the literature, many models are defined with differential rate equations. With SimBiology software, creating the differential equations from reactions is unnecessary; you can enter the reactions and let the software calculate the equations.

Some models are defined with differential rate equations, and you might need the reactions to be compatible with your model. Two rules you can use to convert differential rate equations to reactions are:

- **For a positive term** — The species described by the equation is placed on the right as a product, and the species in the term are placed on the left as reactants.
- **For a negative term** — The species described by the equation is placed on the left as a product, and the species in the term are also placed on the left as reactants.

  You need to determine the products using additional information, for example, a reaction diagram, a description of the model, or an understanding of a reaction. If a reaction is catalyzed by a kinase, then you can conclude that the product has one or more additional phosphate groups.

A simple first-order reaction has differential rate equation `dR/dt = +kr[P] - kf[R]`. The negative term implies that the reaction is `R -> ?` with an unknown product. The positive term identifies the product and completes the reaction, `R <-> P`.

Reactions R1 to R3 from Equation E1

The differential rate equation 1 is repeated here for comparison with the reactions. See “Differential Rate Equation 1, Cyclin (C)” on page B-3.

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

The reaction and reaction rate equations from the differential rate equation E1 are given below:

```
r1      reaction: null -> C
   reaction rate: vi

r2      reaction: C -> null
   reaction rate: kd*C

r3      reaction: C -> null
   reaction rate: (vd*X*C)/(Kd + C)
```

Reactions R4 and R5 from Equation E2

The differential rate equation 2 and algebraic equation 2 are repeated here for comparison with the reactions. See “Differential Rate Equation 2, Kinase (M)” on page B-4.

$$\frac{dM}{dt} = V_1 \frac{(1-M)}{K_1 + (1-M)} - V_2 \frac{M}{K_2 + M}$$

$$V_1 = \frac{VM_1[C]}{K_c + [C]}$$

The reaction and reaction rate equations from the differential rate equation E2 are given below:

```
r4              reaction: Mplus -> M
          reaction rate: V1*Mplus/(K1 + Mplus)
```

```
 repeatedAssignment rule: V1 = VM1*C/(Kc + C)

r5      reaction: M -> Mplus
   reaction rate: V2*M/(K2 + M)
```

Reactions R6 and R7 from Equation E3

The differential rate equation for equation 3 and algebraic equation 3 is repeated here for comparison with the reactions.

$$\frac{dX}{dt} = V_3 \frac{(1 - X)}{K_3 + (1 - X)} - V_4 \frac{X}{K_4 + X}$$

```
V3 = VM3*[M]
```

The reaction and reaction rate equations from the differential rate equation E3 are given below:

```
r6               reaction: Xplus -> X
            reaction rate: V3*Xplus]/(K3 + Xplus)
  repeatedAssignment rule: V3 = VM3*M

r7      reaction: X -> Xplus
   reaction rate: V4*X/(K4 + X)
```

Calculating Initial Values for Reactions

After you converted the differential rate equations to the reactions and reaction rate equations, you can start to fill in initial values for the species (reactants and products) and parameters.

The initial values for parameters and amounts for species are listed with four different units in the same dimension:

- A — Original units in the Goldbeter 1991 paper.
- B — Units of concentration with time converted to second. When converting a to b, use `1 minute = 60 second` for parameters.

$$\frac{\text{X uM}}{\text{minute}} \text{ x } \frac{\text{1e-6 mole/liter}}{\text{1 uM}} \text{ x } \frac{\text{1 minute}}{\text{60 second}} = \frac{\text{Y mole}}{\text{liter*second}}$$

- C — Units of amount as moles. When converting concentration to moles, use a cell volume of `1e-12` liter and assume that volume does not change.

$$\frac{\text{Y mole}}{\text{liter*second}} \text{ x } \frac{\text{1e-12 liter}}{} = \frac{\text{Z mole}}{\text{second}}$$

- D — Units of amount as molecules. When converting amount as moles to molecules, use `6.022e23 molecules = 1 mole`.

$$\frac{\text{Z mole}}{\text{second}} \text{ x } \frac{\text{6.022e23 molecule}}{\text{1 mole}} = \frac{\text{N molecules}}{\text{second}}$$

With dimensional analysis on and unit conversion off, select all of the units for one letter. For example, select all of the As. If dimensional analysis and unit conversion are on, you can mix and match letters and get the same answer.

Reaction 1 Cyclin Synthesis

R1		Value	Units
reaction	null -> C	----	----
reaction rate	vi	----	A. uM/minute
		----	B. mole/(liter*second)
		----	C. mole/second
		----	D. molecule/second
parameters	vi	0.025	A. uM/minute
		4.167e-10	B. mole/(liter*second)
		4.167e-22	C. mole/second
		205	D. molecule/second
species	C	0.01	A. uM
		1e-8	B. mole/liter
		1.0e-20	C. mole
		6.022e+3	D. molecule

Reaction 2 Cyclin Undifferentiated Degradation

R2		Value	Units
reaction	`C -> null`	----	----
reaction rate	`kd*C`	----	A. uM/minute
		----	B. mole/(liter*second)
		----	C. mole/second
		----	D. molecule/second
parameters	`kd`	0.010	A. 1/minute
		1.6667e-4	B, C, D. 1/second
species	`C`	0.01	A. uM
		1e-8	B. mole/liter
		1.0e-20	C. mole
		6.022e+3	D. molecule

Reaction 3 Cyclin Protease Degradation

R3		Value	Units
reaction	`C -> null`	----	----
reaction rate	`(vd*X*C)/(Kd + C)`	----	A. uM/minute
		----	B. mole/(liter*second)
		----	C. mole/second
		----	D. molecule/second
parameter	`vd`	0.25	A. 1/minute
		0.0042	B, C, D. 1/second

R3		Value	Units
parameter	Kd	0.02	A. uM
		2.0e-8	B. mole/liter
		2.0e-020	C. mole
		12044	D. molecule
species	C (substrate)	0.01	A. uM
		1e-8	B. mole/liter
		1.0e-20	C. mole
		6.022e+3	D. molecule
species	X (enzyme)	0.01	A. uM
		1e-8	B. mole/liter
		1.0e-20	C. mole
		6.022e+3	D. molecule

Reaction 4 Cdc2 Kinase Activation

R4		Value	Units
reaction	`Mplus -> M`	----	----
reaction rate	`(V1*Mplus)/(K1 + Mplus)`	----	A. uM/minute
		----	B. mole/(liter*second)
		----	C. mole/second
		----	D. molecule/second
`repeatedAssignment` rule	`V1 = (VM1*C)/(Kc + C)`	----	
parameter	V1 (variable by rule)	0.00	A. uM/minute
			B. mole/(liter*second)
			C. mole/second
			D. molecule/second
parameter	VM1	3.0	A. uM/minute
		5.0e-8	B. mole/(liter*second)
		5.0000e-020	C. mole/second
		30110	D. molecule/second
parameter	Kc	0.5	A. uM
		5.0000e-7	B. mole/liter
		5.0e-19	C. mole
		3.011e+5	D. molecule
parameter	K1	0.005	A. uM
		5e-9	B. mole/liter

R4		Value	Units
		5e-21	C. mole
		3.011e+3	D. molecule
species	`Mplus` (inactive substrate)	0.99	A. uM
		9.9e-7	B. mole/liter
		9.9e-19	C. mole
		5.962e+5	D. molecule
species	`M` (active product)	0.01	A. uM
		1e-8	B. mole/liter
		1.0e-20	C. mole
		6.022e+3	D. molecule
species	`C`	0.01	A. uM
		1e-8	B. mole/liter
		1.0e-20	C. mole
		6.022e+3	D. molecule

Reaction 5 Cdc2 Kinase Deactivation

R5		Value	Units
reaction	`M -> M_plus`	----	----
reaction rate	`(V2*M)/(K2 + M)`	----	A. uM/minute
		----	B. (mole/liter-second)
		----	C. mole/second
		----	D. molecule/second
parameter	`V2`	1.5	A. uM/minute
		2.5000e-008	B. mole/liter-second
		2.5000e-020	C. mole/second
		15055	D. molecule/second
parameter	`K2`	0.005	A. uM
		5.0000e-009	B. mole/liter
		5.0000e-021	C. mole
		3011	D. molecule
		1.0e-20	C. mole
species	`Mplus` (inactive)	0.99	A. uM
		9.9e-7	B. mole/liter
		9.9e-19	C. mole
		5.962e+5	D. molecule
species	`M` (active)	0.01	A. uM
		1e-8	B. mole/liter

R5	Value	Units
	1.0e-20	C. mole
	6.022e+3	D. molecule

Reaction 6 Protease Activation

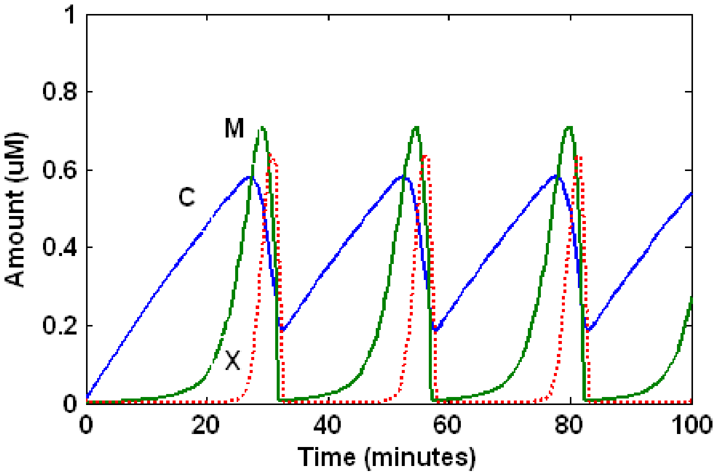
R6		Value	Units
reaction	Xplus -> X	----	----
reaction rate	(V3*Xplus)/(K3 + Xplus)	----	A. uM/minute
		----	B. mole/(liter*second)
		----	C. mole/second
		----	D. molecule/second
repeatedAssignment rule	V3 = VM3*M	----	
parameter	V3 (variable by rule)		A. uM/minute
			B. mole/liter-second
			C. mole/second
			D. molecule/second
parameter	VM3	1.0	A. 1/minute
		0.0167	B, C, D. 1/second
parameter	K3	0.005	A. uM
		5e-9	B. mole/liter
		5e-21	C. mole
		3.011e+3	D. molecule
species	Xplus (inactive substrate)	0.99	A. uM
		9.9e-7	B. mole/liter
		9.9e-19	C. mole
		5.962e+5	D. molecule
species	X (active product)	0.01	A. uM
		1e-8	B. mole/liter
		1.0e-20	C. mole
		6.022e+3	D. molecule
species	M (enzyme)	0.01	A. uM
		1e-8	B. mole/liter
		1.0e-20	C. mole
		6.022e+3	D. molecule

Reaction 7 Protease Deactivation

R7		Value	Units
reaction	`X -> X_plus`	----	----
reaction rate	`(V4*X)/(K4 + X)`	----	A. uM/minute
		----	B. mole/(liter*second)
		----	C. mole/second
		----	D. molecule/second
parameter	`V4`	0.5	A. uM/minute
		8.3333e-009	B. mole/(liter*second)
		8.3333e-021	C. mole/second
		5.0183e+003	D. molecule/second
parameter	`K4`	0.005	A. uM
		5e-9	B. mole/liter
		5e-21	C. mole
		3011	D. molecule
species	`Xplus` (inactive)	0.99	A. uM
		9.9e-7	B. mole/liter
		9.9e-19	C. mole
		5.962e+5	D. molecule
species	`X` (active)	0.01	A. uM
		1e-8	B. mole/liter
		1.0e-20	C. mole
		6.022e+3	D. molecule

SimBiology Simulation with Reactions

This is a simulation of Goldbeter's minimal mitotic oscillator with rate and algebraic equations. Simulate with the `sundials` solver and plot species `C`, `M`, and `X`. For a description of the model, see “SimBiology Model with Reactions” on page B-5.



References

[1] Goldbeter A. (1991), "A minimal cascade model for the mitotic oscillator involving cyclin and cdc2 kinase," Proceedings of the National Academy of Sciences USA, 88:9107-9111.

[2] Goldbeter A., Koshland D. (1981), "An amplified sensitivity arising from covalent modification in biological systems," Proceedings of the National Academy of Sciences USA, 78:6840-6844.

[3] Goldbeter A., Koshland D. (1984), "Ultrasensitivity in biochemical systems controlled by covalent modification," The Journal of Biological Chemistry, 259:14441-14447.

[4] Goldbeter A., home page on the Web, http://www.ulb.ac.be/sciences/utc/GOLDBETER/agoldbet.html.

[5] Murray A.W., Kirschner M.W. (1989), "Cyclin synthesis drives the early embryonic cell cycle," Nature, 339:275-280.

Model of the Yeast Heterotrimeric G Protein Cycle

In this section...

Background on G Protein Cycles

G Proteins

Cells rely on signal transduction systems to communicate with each other and to regulate cellular processes. G proteins are GTP-binding proteins that are involved in the regulation of many cellular processes. There are two known classes of G proteins: the monomeric G proteins (one GTPase), and the heterotrimeric G proteins (three different monomers). The G proteins usually facilitate a step requiring energy. This energy is supplied by the hydrolysis of GTP by a GTPase activating protein (GAP). The exchange of GDP for GTP is catalyzed by a guanine nucleotide releasing protein (GNRP) [Alberts et al. 1994 on page B-17].

$$Gprotein + GTP \underset{GNRP}{\overset{GAP}{\rightleftarrows}} Gprotein + GDP$$

G protein-coupled receptors (GPCRs) are the targets of many pharmaceutical agents. Some estimates suggest that 40 to 50% of currently marketed drugs target GPCRs and that 40% of current drug discovery focus is on GPCR targets. Some examples include those for reducing stomach acid (ranitidine which targets histamine H2 receptor), migraine (sumatriptan, which targets a serotonin receptor subtype), schizophrenia (olanzapine, which targets serotonin and dopamine receptors), allergies (desloratadine, which targets histamine receptors). One approach in pharmaceutical research is to model signaling pathways to analyze and predict both downstream effects and effects in related pathways. This tutorial examines model building and analysis of the G protein cycle in the yeast pheromone response pathway using the SimBiology desktop.

G Proteins and Pheromone Response

In the yeast *Saccharomyces cerevisiae*, G protein signaling in pheromone response is a well characterized signal transduction pathway. The pheromone secreted by *alpha* cells activates the G protein-coupled α-factor receptor (Ste2p) in *a* cells which results in a variety of cell responses including cell-cycle arrest and synthesis of new proteins. The authors of the study performed a quantitative analysis of this cycle, compared the regulation of G protein activation in wild-type yeast haploid *a* cells with cells containing mutations that confer supersensitivity to α-factor. They analyzed the data in the context of cell-cycle arrest and pheromone-induced transcriptional activation and developed a mathematical model of the G protein cycle that they used to estimate rates of activation and deactivation of active G protein in the cell.

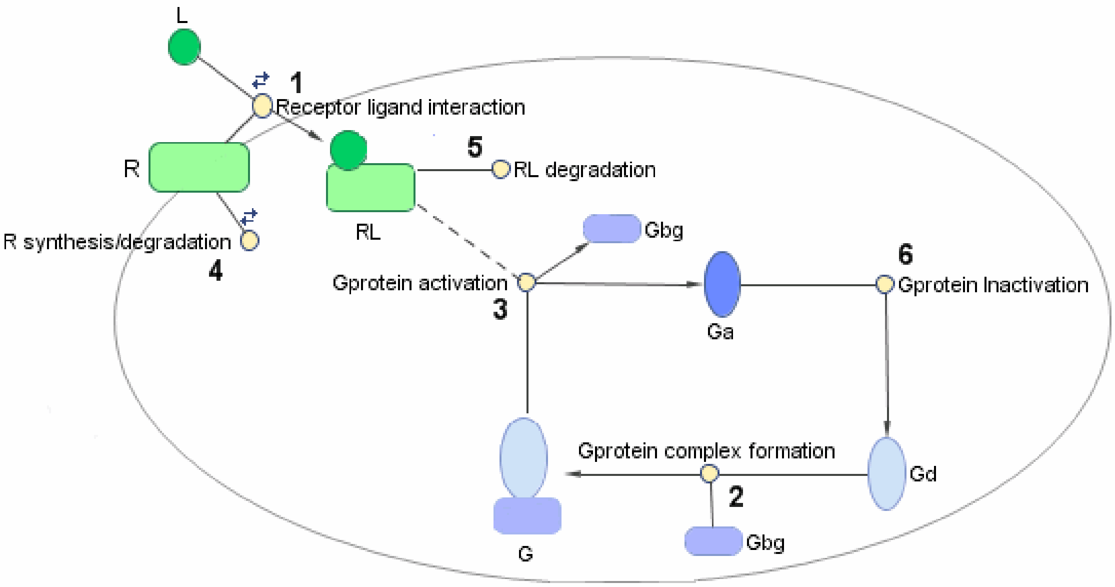
Modeling a G Protein Cycle

- “Reactions Overview” on page B-15
- “Assumptions, Experimental Data, and Units in the G Protein Model” on page B-16

Reactions Overview

Systems biologists represent biological pathways and processes as reactions with reaction rates, and treat the components of these pathways as individual species.

The G protein cycle in the yeast pheromone-response pathway can be condensed into a set of biochemical reactions. These reactions are complex formation, transformation, or disassociation reactions that Yi and colleagues [Yi et al. 2003 on page B-17] use to simplify and describe the system. In this example, α-factor, α-factor receptor, and the G protein subunits are all treated as species participating in reactions. The system can be graphically represented as follows.



Graphical Representation of the G protein cycle in yeast pheromone response. The numbers represent reaction numbers referenced in the text. L = Ligand (alpha factor), R = alpha-factor receptor, Gbg = free levels of G-beta:G-gamma complex, Ga = active G-alpha-GTP, Gd = inactive G-alpha-GDP, G = inactive Gbg:Gd complex.

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

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

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`

Note that in reaction 3 (G protein activation), `RL` appears on both sides of the reaction. This is because `RL` is treated as a modifier or catalyst, and the model assumes that there is no synthesis or consumption of `RL` in this reaction.

The authors use a set of ordinary differential equations (ODEs) to describe the system. In the software, you can represent the biological pathway as a system of biochemical reactions and the software creates the ODEs for you. Alternatively, if you have a set of ODEs that describe your system you can enter these as rate rules. For an example of modeling using rate rules, see “SimBiology Model with Rate Rules” on page B-4.

Assumptions, Experimental Data, and Units in the G Protein Model

The authors have obtained experimental data either through their own measurements or through published literature. As with any other model, the G protein cycle model simplifies the biological process while also trying to reconcile the experimental data. Consider these points:

- Reaction 2 — Binding and formation of the heterotrimeric G protein complex is treated as a single-step reaction.
- Reaction 3 — Activation of G protein is modeled as a single-step. Guanine nucleotide exchange factors (GEFs) are not modeled.
- Reactions 3 and 6 — The parameters for the rate of G protein activation and deactivation (`kGa` and `kGd`) have been estimated based on the dose response curves in the reference paper. The SimBiology model being built in this tutorial directly uses those values.
- Reactions 4 and 5 — Receptor synthesis and degradation are handled purely as two simple reaction steps.
- Reaction 6 — Deactivation of G protein by the regulator of G protein signaling (RGS) protein Sst2p is modeled as a single step. Sst2p is not modeled.

  The reaction is modeled with an estimated reaction rate of $0.11\ s^{-1}$) in the Sst2p containing wild-type strain. The uncatalyzed reaction rate is estimated to be $0.004\ s^{-1}$ in a strain with a deletion of SST2 (*sst2Δ*, mutant strain).
- Free GDP, GTP, and Pi are not included in the model.

This tutorial shows you how to plot the experimental data over the simulation plot of the active G protein fraction. You can estimate the values of the experimental data of interest for this example from the coordinates of the plots found in Figure 5 of the reference paper [Yi et al. 2003 on page B-17]. The following values were obtained by comparing the coordinates of the standards with those of the unknowns in the figure.

Time	Fraction of Active Ga (Experimental)
0	0.00

Time	Fraction of Active Ga (Experimental)
10	0.35
30	0.40
60	0.36
110	0.39
210	0.33
300	0.24
450	0.17
600	0.20

Note The SimBiology **Dimensional Analysis** feature is not used in this tutorial. For this tutorial, the values of all species are converted to have the unit `molecule`, and all rate parameters are converted to have either the unit `1/second` or the units `1/(molecule*second)`, depending on whether the reaction is first or second order. You should leave the **InitialAmountUnits** box for species and the **ValueUnits** box for rate parameters empty for the models in this tutorial.

References

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

[2] Alberts, B., Bray, D., Lewis, J., Raff, M., Roberts, K., and Watson, J.D. Molecular Biology of the Cell, 3rd edition, Garland Publishing, 1994.

See Also

Related Examples

- “Parameter Scanning, Parameter Estimation, and Sensitivity Analysis in the Yeast Heterotrimeric G Protein Cycle”

Model of M-Phase Control in Xenopus Oocyte Extracts

John Tyson's Computational Cell Biology Lab created a mathematical model for M-phase control in *Xenopus* oocyte (frog egg) extracts [Marlovits et al. 1998 on page B-41]. The M-phase control model shows principles by which you can apply phosphorylation and regulatory loops in your own models. Publications typically list systems of ordinary differential equations (ODEs) that represent a model system. This example shows you how to interpret these ODEs in the form of reaction pathways that are easier to represent and visualize in SimBiology software.

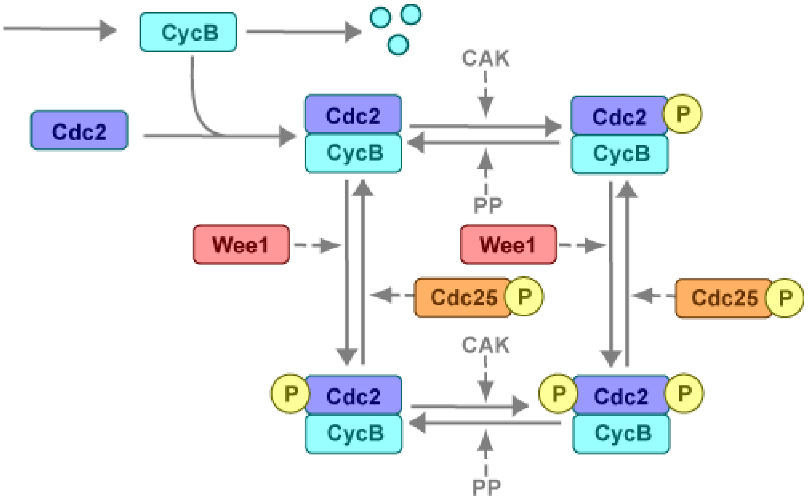
The model is centered around M-phase promoting factor (MPF). There are two positive feedback loops where MPF increases its synthesis and a negative feedback loop where MPF decreases its amount by increasing its degradation.

In this section...

M-Phase Control Model

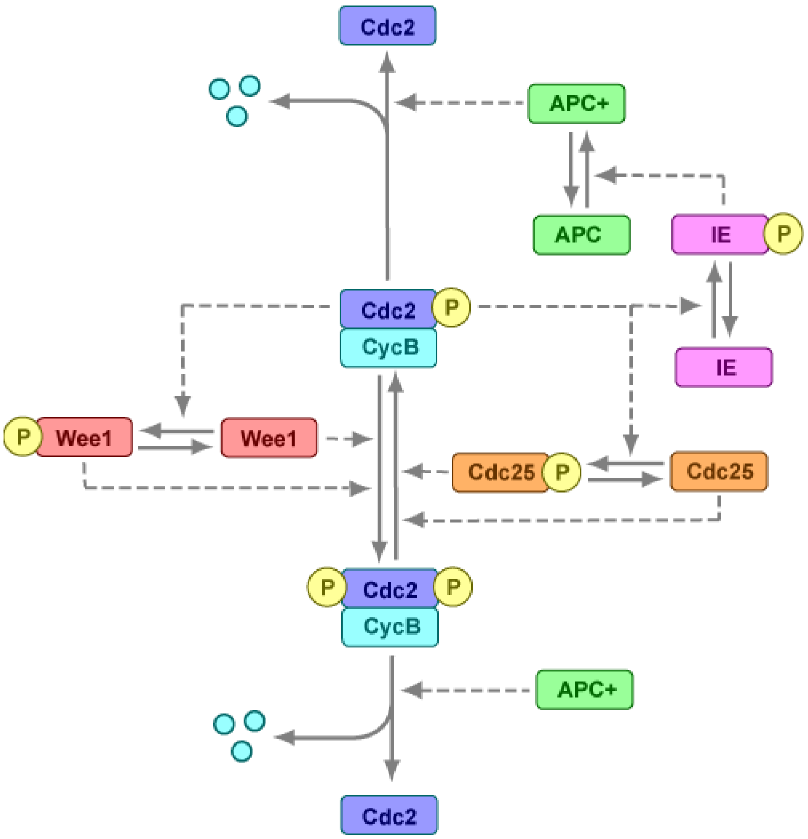
Synthesis Reactions

Cyclin B (CycB) dimerizes with Cdc2 kinase (Cdc2) to form M-phase promoting factor (MPF).



Regulation Reactions with Active MPF

Positive feedback loops with M-phase promoting factor (MPF) activate the Cdc25 phosphatase and deactivate the Wee1 kinase. A negative feedback loop with MPF activates anaphase-promoting complex (APC) that regulates the degradation of the Cyclin B subunit.



M-Phase Control Equations

- “About the Rate Equations in This Example” on page B-19
- “Converting Differential Equations to Reactions” on page B-20
- “Equation 1, Cyclin B” on page B-20
- “Equation 2, M-Phase Promoting Factor” on page B-21
- “Equation 3, Inhibited M-Phase Promoting Factor” on page B-21
- “Equation 4, Inhibited and Activated M-Phase Promoting Factor” on page B-22
- “Equation 5, Activated M-Phase Promoting Factor” on page B-22
- “Equation 11, Cell Division Control 25” on page B-23
- “Equation 12, Wee1 Activation/Deactivation” on page B-23
- “Equation 13, Intermediate Enzyme Activation/Deactivation” on page B-23
- “Equation 14, APC Activation/Deactivation” on page B-24
- “Equation 17, Rate Parameter K2” on page B-24
- “Equation 18, Rate Parameter Kcdc25” on page B-24
- “Equation 19, Rate Parameter Kwee1” on page B-25

About the Rate Equations in This Example

Models in systems biology are commonly described in the literature with differential rate equations. However, SimBiology software defines a model using reactions. This section shows you how to convert models published in the literature to a SimBiology format. The equation numbers match the published paper for this model [Marlovits et al. 1998 on page B-41]. Equations that are missing in the sequence involve the Cdk inhibitor (CKI) protein, which is not currently modeled in the SimBiology version.

Converting Differential Equations to Reactions

The rules for writing reaction and reaction rate equations from differential rate equations include not only the equations but also an understanding of the reactions. *dx/dt* refers to the species the differential rate equation is defining. *kinetics* refers to the species in the reaction rate.

- Positive terms: Rate species are placed on right side of the reactions; reaction rate equation species are placed on the left.

$$kinetics \rightarrow \frac{dx}{dt}$$

- Negative terms: Rate species are placed on the left side of the reaction because the species are being used up in some way; reaction rate equation species are also placed on left. You need to deduce the products from additional information about the model.

$$\text{kinetics or } (\frac{\text{dx}}{\text{dt}}) \rightarrow \text{products?}$$

The following table will help you deduce the products for a reaction. In this example, by convention, phosphate groups on the right side of a species name are activating while phosphate groups on left are inhibiting.

Enzyme	Description	Reaction
wee1	Kinase, add inhibiting phosphate group	MPF —> P-MPF
cdc25	Phosphatase, remove inhibiting phosphate group	P-MPF —> MPF + P
kcak	Kinase, add activating phosphate group	MPF —> MPFp
kpp	Phosphatase, remove activating phosphate group	MPF-P —> MPF + P
MPF	Kinase, add activating or inhibiting phosphate group	Wee1/Cdc25/IE —> X-P or P-X
ki	Add inhibiting Cki	Cki + MPF —> Cki:MPF
kir	Remove inhibiting Cki	Cki:MPF —> Cki + MPF

Equation 1, Cyclin B

Differential rate equation for cyclin B [Marlovits et al. 1998 on page B-41].

$$\frac{d[\text{CycB}]}{dt} = +\text{k1 -k2[CycB] -k3[Cdc2][CycB]}$$

Rate rule using SimBiology format for the differential rate equation 1. For a model using this rule, see “SimBiology Model with Rate and Algebraic Rules” on page B-25.

```
Rule 1 on page B-25  [CycB] = k1 - K2*[CycB] - k3*[Cdc2]*[CycB]
```

Reaction and reaction rate equations derived from the differential rate equation. For a model using these reactions, see “SimBiology Model with Reactions and Algebraic Rules” on page B-29.

```
Reaction 1 on page B-30   AA -> CycB           v = k1
Reaction 2 on page B-30   CycB -> AA           v = K2*[CycB]
Reaction 3 on page B-31   Cdc2 + CycB -> MPF   v = k3*[Cdc2]*[CycB]
```

Equation 2, M-Phase Promoting Factor

Differential rate equation for M-phase promoting factor (MPF) [Marlovits 1998 on page B-41]. Note that the parameter name `kcakr` [Marlovits et al. 1998 on page B-41] is changed to `kpp` [Borisuk 1998 on page B-41] in the following reaction equations. MPF is a heterodimer of cdc2 kinase and cyclin B.

$$\frac{d[\mathrm{MPF}]}{dt} = +\mathrm{k3[Cdc2][CycB]}\ \mathrm{-K2[MPF]}$$
$$+\mathrm{kpp[MPFp]}\ \mathrm{-kcak[MPF]}$$
$$+\mathrm{Kcdc25[pMPF]}\ \mathrm{-Kwee1[MPF]}$$
$$+\mathrm{kir[Cki{:}MPF]}\ \mathrm{-ki[MPF][Cki]}$$

Rate rule using SimBiology format for the differential rate equation 1. For a model using this rule, see “SimBiology Model with Rate and Algebraic Rules” on page B-25.

```
Rule 2 on page B-26 MPF = kpp*MPFp - (Kwee1 + kcak + K2)*MPF + Kcdc25*pMPF + k3*Cdc2*CycB
```

Reaction and reaction rate equations derived from the differential rate equation. For a model using these reactions, see “SimBiology Model with Reactions and Algebraic Rules” on page B-29. A reaction name in parentheses denotes a reaction repeated in another differential rate equation.

```
(Reaction 3 on page B-31) Cdc2 + CycB -> MPF     v = k3*[Cdc2]*[CycB]
Reaction 4 on page B-32   MPF -> Cdc2 + AA       v = K2*[MPF]
Reaction 5 on page B-33   MPFp -> MPF            v = kpp*[MPFp]
Reaction 6 on page B-34   MPF -> MPFp            v = kcak*[MPF]
Reaction 7 on page B-34   pMPF -> MPF            v = Kcdc25*[pMPF]
Reaction 8 on page B-35   MPF -> pMPF            v = Kwee1*[MPF]
```

Equation 3, Inhibited M-Phase Promoting Factor

Differential rate equation for inhibited M-phase promoting factor (pMPF) [Marlovits 1998 on page B-41].

$$\frac{d[\mathrm{pMPF}]}{dt} = -\mathrm{K2[pMPF]}$$
$$+\mathrm{kpp[pMPFp]}\ \mathrm{-kcak[pMPF]}$$
$$+\mathrm{Kwee1[MPF]}\ \mathrm{-Kcdc25[pMPF]}$$
$$+\mathrm{kd[Cki{:}pMPF]}$$

Rate rule using SimBiology format for the differential rate equation 3. For a model using this rule, see “SimBiology Model with Rate and Algebraic Rules” on page B-25.

```
Rule 3 on page B-26 pMPF = Kwee1*MPF - (Kcdc25 + kcak + K2)*pMPF + kpp*pMPFp
```

Reaction and reaction rate equations derived from the differential rate equation. For a model using these reactions, see “SimBiology Model with Reactions and Algebraic Rules” on page B-29.

```
Reaction 11 on page B-36  pMPF -> Cdc2 + AA      v = K2*[pMPF]
Reaction 12 on page B-36  pMPFp -> pMPF          v = kpp*[pMPFp]
Reaction 13 on page B-36  pMPF -> pMPFp          v = kcak*[pMPF]
(Reaction 8 on page B-35) MPF -> pMPF            v = Kwee1*[MPF]
(Reaction 7 on page B-34) pMPF -> MPF            v = Kcdc25*[pMPF]
```

Equation 4, Inhibited and Activated M-Phase Promoting Factor

Differential rate equation for inhibited and activated M-phase promoting factor (pMPFp) [Marlovits 1998 on page B-41].

$$\begin{aligned}\frac{d[\mathrm{pMPFp}]}{dt} = & -\mathrm{K2}[\mathrm{pMPFp}] \\ & +\mathrm{kcak}[\mathrm{pMPF}] -\mathrm{kpp}[\mathrm{pMPFp}] \\ & +\mathrm{Kwee1}[\mathrm{MPFp}] -\mathrm{Kcdc25}[\mathrm{pMPFp}] \\ & +\mathrm{kd}[\mathrm{Cki{:}pMPFp}]\end{aligned}$$

Rate rule using SimBiology format for the differential rate equation. For a model using this rule, see “SimBiology Model with Rate and Algebraic Rules” on page B-25.

```
Rule 4 on page B-28 pMPFp = Kwee1*MPFp - (kpp + Kcdc25 + K2)*pMPFp + kcak*pMPF
```

Reaction and reaction rate equations derived from the differential rate equation. For a model using these reactions, see “SimBiology Model with Reactions and Algebraic Rules” on page B-29.

```
Reaction 15 on page B-37   pMPFp -> Cdc2 + AA       v = K2*[pMPFp]
(Reaction 13 on page B-36) pMPF -> pMPFp            v = kcak*[pMPF]
(Reaction 12 on page B-36) pMPFp -> pMPF            v = kpp*[pMPFp]
Reaction 16 on page B-37   MPFp -> pMPFp            v = Kwee1*[MPFp]
Reaction 17 on page B-37   pMPFp -> MPFp            v = Kcdc25*[pMPFp]
```

Equation 5, Activated M-Phase Promoting Factor

Differential rate equation for activated M-phase promoting factor (MPFp) [Marlovits 1998 on page B-41].

$$\begin{aligned}\frac{d[\mathrm{MPFp}]}{dt} = & -\mathrm{K2}[\mathrm{MPFp}] \\ & +\mathrm{kcak}[\mathrm{MPF}] -\mathrm{kpp}[\mathrm{MPFp}] \\ & +\mathrm{Kcdc25}[\mathrm{pMPFp}] -\mathrm{Kwee1}[\mathrm{MPFp}] \\ & +\mathrm{kir}[\mathrm{CKI{:}MPFp}] -\mathrm{ki}[\mathrm{CKI}][\mathrm{MPFp}]\end{aligned}$$

Rate rule using SimBiology format for the differential rate equation 1. For a model using this rule, see “SimBiology Model with Rate and Algebraic Rules” on page B-25.

```
Rule 5 on page B-28  MPFp = kcak*MPF - (kpp + Kwee1 + K2)*MPFp + Kcdc25*pMPFp
```

Reaction and reaction rate equations derived from the differential rate equation. For a model using these reactions, see “SimBiology Model with Reactions and Algebraic Rules” on page B-29.

```
Reaction 19 on page B-38    MPFp -> MPF + AA        v = K2*[MPFp]
(Reaction 6 on page B-34)  MPF -> MPFp             v = kcak*[MPF]
(Reaction 5 on page B-33)  MPFp -> MPF             v = kpp*[MPFp]
(Reaction 17 on page B-37) pMPFp -> MPFp           v = Kcdc25*[pMPFp]
(Reaction 16 on page B-37) MPFp -> pMPFp           v = Kwee1*[MPFp]
```

Equation 11, Cell Division Control 25

Differential rate equation for activating and deactivating Cdc25 [Marlovits 1998 on page B-41].

$$\frac{d[\mathrm{Cdc25p}]}{dt} = +\frac{\mathrm{k25[MPFp][Cdc25]}}{\mathrm{Km25+[Cdc25]}} - \frac{\mathrm{k25r[Cdc25p]}}{\mathrm{Km25r+[Cdc25p]}}$$

Rate rule in SimBiology format for the differential rate equation 1. For a model using this rule, see “SimBiology Model with Rate and Algebraic Rules” on page B-25. Note that since there isn't a rate rule for `Cdc25`, its amount is written as `(TotalCdc25 - Cdc25p)`.

```
Rule 11 on page B-28 Cdc25p = (k25*MPFp*(TotalCdc25 - Cdc25p))/(Km25 + (TotalCdc25 - Cdc25p)) -
```

Reaction and reaction rate equations derived from the differential rate equation. For a model using these reactions, see “SimBiology Model with Reactions and Algebraic Rules” on page B-29.

```
Reaction 36 Cdc25 -> Cdc25p,  v = k25*[MPFp]*[Cdc25]/(Km25 + [Cdc25])
Reaction 37 Cdc25p -> Cdc25,  v = k25r*[Cdc25p]/(Km25r + [Cdc25p])
```

Equation 12, Wee1 Activation/Deactivation

Differential rate equation for activating and deactivating Wee1 kinase [Marlovits 1998 on page B-41]. The kinase (MPFp) phosphorylates active Wee1 (Wee1) to its inactive form (Wee1p). The dephosphorylation of inactive Wee1 (Wee1p) is by an unknown phosphatase.

$$\frac{d[Wee1]}{dt} = -\frac{kw[MPFp][Wee1]}{Kmw + [Wee1]} + \frac{kwr[Wee1P]}{Kmwr + [Wee1P]}$$

Rate rule in SimBiology format for the differential rate equation 1. For a model using this rule, see “SimBiology Model with Rate and Algebraic Rules” on page B-25.

```
Rule 12  Wee1p = (kw*MPFp*(TotalWee1 - Wee1p))/(Kmw + (TotalWee1 - Wee1p))
                                     - (kwr*Wee1p)/(Kmwr + Wee1p)
```

Reaction and reaction rate equations derived from the differential rate equation. For a model using these reactions, see “SimBiology Model with Reactions and Algebraic Rules” on page B-29.

```
reaction 38 Wee1 -> Wee1p, v = (kw*[MPFp]*[Wee1])/(Kmw + [Wee1])
reaction 39 Wee1p -> Wee1, v = (kwr*[Wee1p])/(Kmwr + [Wee1p])
```

Equation 13, Intermediate Enzyme Activation/Deactivation

Differential rate equation for activating and deactivating the intermediate enzyme (IE) [Marlovits 1998 on page B-41]. The active kinase (MPFp) phosphorylates the inactive intermediate enzyme (IE) to its active form (IEp).

$$\frac{d[IEp]}{dt} = +\frac{kie[MPFp][IE]}{Kmie + [IE]} - \frac{kier[IEp]}{Kmier + [IEp]}$$

Rate rule in SimBiology format for the differential rate equation 1. For a model using this rule, see “SimBiology Model with Rate and Algebraic Rules” on page B-25.

```
Rule 13 IEp = (kie*MPFp*(TotalIE - IEp))/(Kmie + (TotalIE - IEp))
                 - (kier*IEp)/(Kmier + IEp)
```

Reaction and reaction rate equations derived from the differential rate equation. For a model using these reactions, see “SimBiology Model with Reactions and Algebraic Rules” on page B-29.

```
reaction 40 IE -> IEp, v = (kie*[MPFp]*[IE])/(Kmie + [IE])
reaction 41 IEp -> IE, v = (kier*[IEp])/(Kmier + [IEp])
```

Equation 14, APC Activation/Deactivation

Differential rate equation for [Marlovits 1998 on page B-41].

$$\frac{d[APCa]}{dt} = +\frac{kap[IEP][APCi]}{Kmap + [APCi]} - \frac{kapr[APCa]}{Kmapr + [APCa]}$$

Rate rule in SimBiology format for the differential rate equation 1. For a model using this rule, see “SimBiology Model with Rate and Algebraic Rules” on page B-25.

```
Rule 14 APCa = (kap*IEp*(TotalAPC - APCa))/(Kmap + (TotalAPC - APCa))
                    - (kapr*APCa)/(Kmapr + APCa)
```

Reaction and reaction rate equations derived from the differential rate equation. For a model using these reactions, see “SimBiology Model with Reactions and Algebraic Rules” on page B-29.

```
Reaction 42 APCi -> APCa, v = (kap*[IEp]*[APCi])/(Kmap + [APCi])
Reaction 43 APCa -> APCi,  v = (kapr*[APCa])/(Kmapr + [APCa])
```

Equation 17, Rate Parameter K2

Algebraic equation to define the rate parameter `K2` [Marlovits 1998 on page B-41]. Inactive APC (APCi) is catalyzed by IE (intermediate enzyme) to active APC (APCa).

k2 = V2'[APC] + V2''[APC']

Algebraic rule in SimBiology format for the algebraic equation 17. For a model using this rule, see “SimBiology Model with Rate and Algebraic Rules” on page B-25.

```
Algebraic Rule 17  V2i*(TotalAPC - APCa) + V2a*APCa - K2
```

Algebraic rule when simulating with reactions. For a model using this rule with reactions, see “SimBiology Model with Reactions and Algebraic Rules” on page B-29. `V2'` is renamed to `V2i` and `V2"`is renamed to `V2a`. APCi (APC) is the inactive form of the enzyme while APCa (APC') is the active form. `K2` is the independent variable.

```
Algebraic Rule 1 (V2i*APCi) + (V2a*APCa) - K2
```

Equation 18, Rate Parameter Kcdc25

Algebraic equation to define the rate parameter `Kcdc25` [Marlovits 1998 on page B-41]. Inactive Cdc25 (`Cdc25`) is phosphorylated by MPF to active Cdc25 (Cdc25p).

kcdc25 = V25'[Cdc25] + V25''[Cdc25p]

Algebraic rule in SimBiology format for the algebraic equation 18. For a model using this rule, see “SimBiology Model with Rate and Algebraic Rules” on page B-25.

```
Algebraic Rule 18  V25i*(TotalCdc25 - Cdc25p) + V25a*Cdc25p - Kcdc25
```

Algebraic rule when simulating with reactions. `Kcdc25` is the independent variable. For a model using this rule with reactions, see “SimBiology Model with Reactions and Algebraic Rules” on page B-29.

```
Algebraic Rule 2 (V25i*Cdc25) + (V25a*Cdc25p) - Kcdc25
```

Equation 19, Rate Parameter Kwee1

Algebraic equation to define the rate parameter [Marlovits 1998 on page B-41]. Active Wee1 (`Wee1`) is phosphorylated by MPF to inactive Wee1 (`Wee1p`).

$$kwee1 = Vwee1'[Wee1p] + Vwee1''[Wee1]$$

Algebraic rule in SimBiology format for rate parameter equation 19. For a model using this rule, see "SimBiology Model with Rate and Algebraic Rules" on page B-25.

```
Algebraic Rule 19  Vwee1i*Wee1p + Vwee1a*(TotalWee1 - Wee1p) - Kwee1
```

Algebraic rule when simulating with reactions. `Kwee1` is the independent variable. For a model using this rule with reactions, see "SimBiology Model with Reactions and Algebraic Rules" on page B-29.

```
Algebraic Rule 3 (Vwee1i*Wee1p) + (Vwee1a*Wee1) - Kwee1
```

SimBiology Model with Rate and Algebraic Rules

- "Overview" on page B-25
- "Writing Differential Rate Equations as Rate Rules" on page B-25
- "Species" on page B-26
- "Parameters" on page B-26
- "Rate Rule 1, Cyclin B (CycB)" on page B-27
- "Rate Rule 2, M-Phase Promoting Factor (MPF)" on page B-27
- "Rate Rule 3, Inhibited M-Phase Promoting Factor (pMPF)" on page B-28
- "Rate Rule 4, Activated but Inhibited M-Phase Promoting Factor (pMPFp)" on page B-28
- "Rate Rule 5, Activated M-Phase Promoting Factor (MPFp)" on page B-28
- "Rate Rule 11, Activated Cdc25 (Cdc25p)" on page B-28
- "Rate Rule 12, Inhibited Wee1 (Wee1p)" on page B-28
- "Rate Rule 13, Activated Intermediate Enzyme (IEp)" on page B-28
- "Rate Rule 14, Activated APC (APCa)" on page B-28
- "Algebraic Rule 17, Rate Parameter K2" on page B-29
- "Algebraic Rule 18, Rate Parameter Kcdc25" on page B-29
- "Algebraic Rule 19, Rate Parameter Kwee1" on page B-29

Overview

There is one rate rule for each equation defining a species and one algebraic rule for each variable parameter in the M-phase control model [Marlovits 1998 on page B-41]. For a list and description of the equations, see "M-Phase Control Equations" on page B-19.

A basic model includes rate rules 1 to 5 and 11 to 14 with algebraic rules 17, 18, and 19.

Writing Differential Rate Equations as Rate Rules

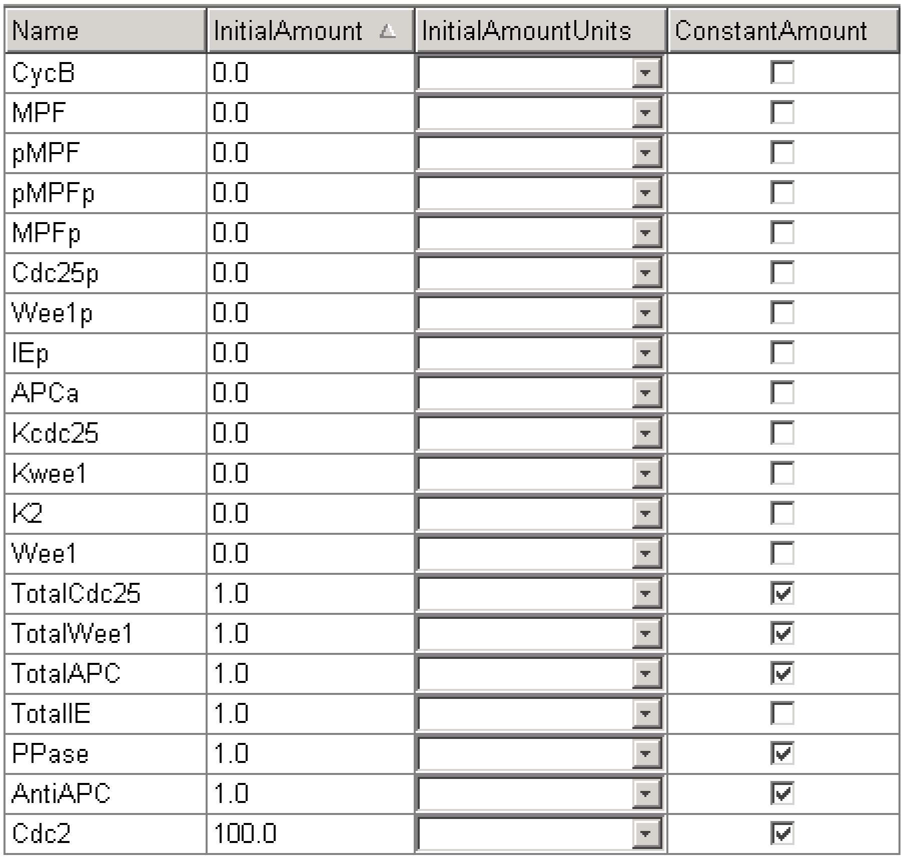
Writing differential rate equations in an unambiguous format that a software program can understand is a simple process when you follow the syntax rules for programming languages.

- Use an asterisk to indicate multiplication. For example, `k[A]` is written `k*A` or `k*[A]`. The brackets around the species `A` do not indicate concentration.
- SimBiology uses square brackets around species and parameter name to allow names that are not valid MATLAB variable names. For example, you could have a species named `glucose-6-phosphate dehydrogenase` but you need to add brackets around the name in reaction rate and rule equations.

  ```
  [glucose-6-phosphate dehydrogenase]
  ```
- Use parentheses to clarify the order of evaluation for mathematical operations. For example, do not write Henri-Michaelis-Menten reaction rates as `Vm*C/Kd + C`, because `Vm*C` is divided by `Kd` before adding `C` to the result. Instead, write this reaction rate as `(Vm*C)/(Kd + C)`.

Species

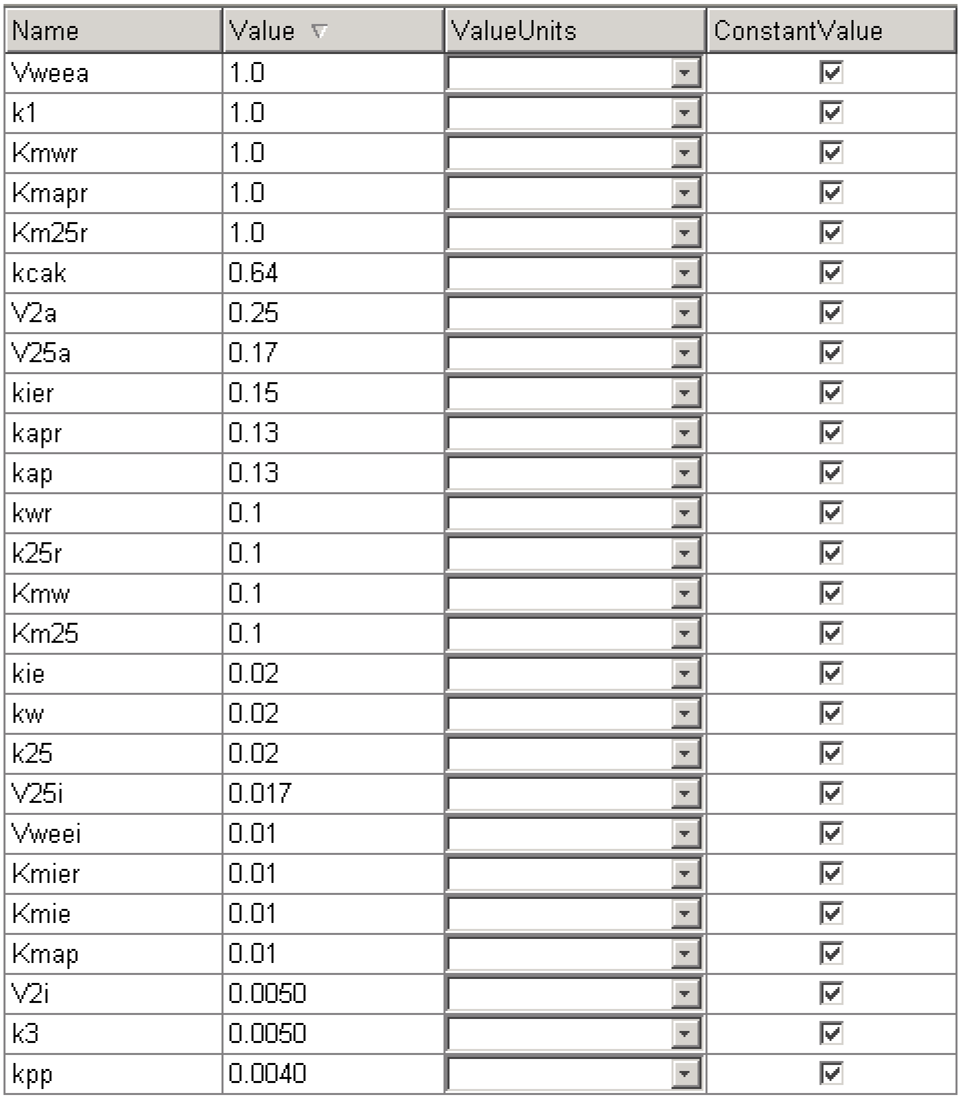
The following table lists species in the model with their initial amounts. There are three variable parameters modeled as species (`K2`, `Kcdc25`, and `KWee1`). You could also model the variable parameters as parameters with the property **ConstantAmount** cleared.



Name	InitialAmount	InitialAmountUnits	ConstantAmount
CycB	0.0		□
MPF	0.0		□
pMPF	0.0		□
pMPFp	0.0		□
MPFp	0.0		□
Cdc25p	0.0		□
Wee1p	0.0		□
IEp	0.0		□
APCa	0.0		□
Kcdc25	0.0		□
Kwee1	0.0		□
K2	0.0		□
Wee1	0.0		□
TotalCdc25	1.0		☑
TotalWee1	1.0		☑
TotalAPC	1.0		☑
TotalIE	1.0		□
PPase	1.0		☑
AntiAPC	1.0		☑
Cdc2	100.0		☑

Parameters

The following table lists parameters in the model with their initial values. The property **ConstantValue** is selected for all of the parameters.



Name	Value ▽	ValueUnits	ConstantValue
Vweea	1.0		☑
k1	1.0		☑
Kmwr	1.0		☑
Kmapr	1.0		☑
Km25r	1.0		☑
kcak	0.64		☑
V2a	0.25		☑
V25a	0.17		☑
kier	0.15		☑
kapr	0.13		☑
kap	0.13		☑
kwr	0.1		☑
k25r	0.1		☑
Kmw	0.1		☑
Km25	0.1		☑
kie	0.02		☑
kw	0.02		☑
k25	0.02		☑
V25i	0.017		☑
Vweei	0.01		☑
Kmier	0.01		☑
Kmie	0.01		☑
Kmap	0.01		☑
V2i	0.0050		☑
k3	0.0050		☑
kpp	0.0040		☑

Rate Rule 1, Cyclin B (CycB)

The rate rule is from "Equation 1, Cyclin B" on page B-20.

```
 rate rule: CycB = k1 - K2*CycB - k3*Cdc2*CycB
   species: CycB = 0 nM
            Cdc2 = 100 nM, [x]constant
parameters: k1 = 1 nM/minute
            K2 = 0 1/minute, []constant
            k3 = 0.005 1/(nM*minute)
```

K2 is a variable rate parameter whose value is defined by an algebraic rule. See "Algebraic Rule 17, Rate Parameter K2" on page B-29. Its value varies from `0.005` to `0.25 1/minute`.

Rate Rule 2, M-Phase Promoting Factor (MPF)

The rate rule is from "Equation 2, M-Phase Promoting Factor" on page B-21.

```
 rate rule: MPF = kpp*MPFp - (Kwee1 + kcak + K2)*MPF + Kcdc25*pMPF
                  + k3*Cdc2*CycB
```

```
   species: MPF = 0 nM
            MPFp = 0 nM
            pMPF = 0 nM
parameters: kpp = 0.004 1/minute
            kcak = 0.64 1/minute
            k3 = 0.005 1/(nM*minute)
            K2 = 0 1/minute
            Kcdc25 = 0 1/minute
            Kwee1 = 0 1/minute
```

K2, Kcdc25, and Kwee1 are variable rate parameters whose values are defined by algebraic rules. See “Algebraic Rule 17, Rate Parameter K2” on page B-29, “Algebraic Rule 18, Rate Parameter Kcdc25” on page B-29, and “Algebraic Rule 19, Rate Parameter Kwee1” on page B-29.

Rate Rule 3, Inhibited M-Phase Promoting Factor (pMPF)

The rate rule is from “Equation 3, Inhibited M-Phase Promoting Factor” on page B-21.

```
rate rule: pMPF = Kwee1*MPF - (Kcdc25 + kcak + K2)*pMPF + kpp*pMPFp
```

Rate Rule 4, Activated but Inhibited M-Phase Promoting Factor (pMPFp)

The rate rule is from “Equation 4, Inhibited and Activated M-Phase Promoting Factor” on page B-22.

```
rate rule: pMPFp = Kwee1*MPFp - (kpp + Kcdc25 + K2)*pMPFp + kcak*pMPF
```

Rate Rule 5, Activated M-Phase Promoting Factor (MPFp)

The rate rule is from “Equation 5, Activated M-Phase Promoting Factor” on page B-22.

```
rate rule: MPFp = kcak*MPF - (kpp + Kwee1 + K2)*MPFp + Kcdc25*pMPFp
```

Rate Rule 11, Activated Cdc25 (Cdc25p)

The rate rule is from “Equation 11, Cell Division Control 25” on page B-23.

```
rate rule: Cdc25p = (k25*MPFp*(TotalCdc25 - Cdc25p))/(Km25 + (TotalCdc25 - Cdc25p))
                      - (k25r*PPase*Cdc25p)/(Km25r + Cdc25p)
```

Rate Rule 12, Inhibited Wee1 (Wee1p)

The rate rule is from “Equation 12, Wee1 Activation/Deactivation” on page B-23.

```
rate rule: Wee1p = (kw*MPFp*(TotalWee1 - Wee1p))/(Kmw + (TotalWee1 - Wee1p))
                     - (kwr*PPase*Wee1p)/(Kmwr + Wee1p)
```

Rate Rule 13, Activated Intermediate Enzyme (IEp)

The rate rule is from “Equation 13, Intermediate Enzyme Activation/Deactivation” on page B-23.

```
rate rule: IEp = (kie*MPFp*(TotalIE - IEp))/(Kmie + (TotalIE - IEp))
                        - (kier*PPase*IEp)/(Kmier + IEp)
```

Rate Rule 14, Activated APC (APCa)

The rate rule is from “Equation 14, APC Activation/Deactivation” on page B-24.

```
rate rule: APCa = (kap*IEp*(TotalAPC - APCa))/(Kmap + (TotalAPC - APCa))
                     - (kapr*AntiAPC*APCa)/(Kmapr + APCa)
```

Algebraic Rule 17, Rate Parameter K2

K2 is a variable rate parameter whose value is determined by the amount of active and inactive APC. The algebraic rule is from “Equation 17, Rate Parameter K2” on page B-24.

```
algebraic rule: V2i*(TotalAPC - APCa) + V2a*APCa - K2
       species: APCi = 1 nM
                APCa = 0 nM
                TotalAPC = 1 nM [x]constant
    parameters: K2  = 0 or 0.25 1/minute, []constant
                V2i = 0.005 1/(nM*minute)
                V2a = 0.25  1/(nM*minute)
```

Algebraic Rule 18, Rate Parameter Kcdc25

Kcdc25 is a variable rate parameter whose value is determined by the amount of active and inactive Cdc25. The algebraic rule is from “Algebraic Rule 18, Rate Parameter Kcdc25” on page B-29.

```
algebraic rule: V25i*(TotalCdc25 - Cdc25p) + V25a*Cdc25p - Kcdc25
```

Algebraic Rule 19, Rate Parameter Kwee1

Kwee1 is a variable rate parameter whose value is determined by the amount of active and inactive Wee1. The algebraic rule is from “Equation 19, Rate Parameter Kwee1” on page B-25.

```
algebraic rule: Vweei*Wee1p + Vweea*(TotalWee1 - Wee1p) - Kwee1
```

SimBiology Model with Reactions and Algebraic Rules

- "Reaction 39, Activation of Wee1" on page B-38
- "Reaction 40, Activation of Intermediate Enzyme by Active MPF" on page B-39
- "Reaction 41, Deactivation of IE" on page B-39
- "Reaction 42, APC Activation by IEp" on page B-39
- "Reaction 43, APC Deactivation" on page B-39
- "Block Diagram of the M-Phase Control Model with Reactions" on page B-39

Overview

There can be one or more reactions for an equation defining a species and one algebraic rule for each variable parameter in the M-phase control model [Marlovits 1998 on page B-41]. For a list and description of the equations, see "M-Phase Control Equations" on page B-19.

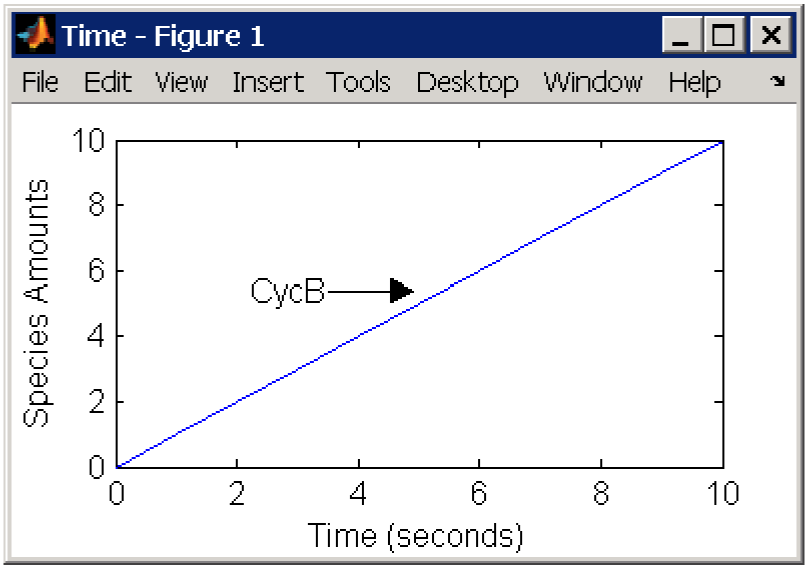
A basic model includes reactions 1 to 8, 11 to 13, 15 to 17, 19, and 36 to 43 with algebraic rules from equations 17, 18, and 19.

Reaction 1, Synthesis of Cyclin B

Cyclin B is synthesized at a constant rate.

```
     reaction: AA -> CycB
reaction rate: k1 nM/minute
    parameter: k1 = 1 nM/minute
      species: CycB = 0 nM
               AA = 100 nM [x]constant [x]boundary
```

Simulate reaction 1 with the `sundials` solver.



Reaction 2, Degradation of Cyclin B

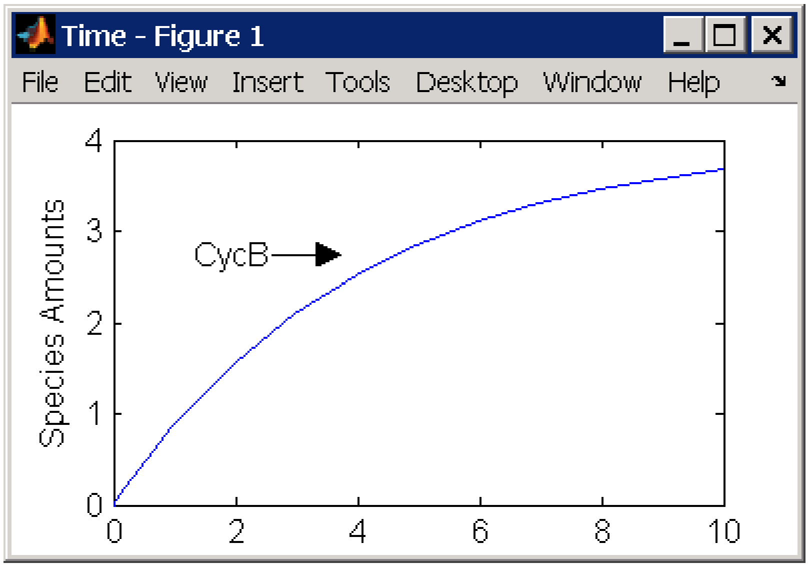
Cyclin B is degraded at the end of the M-phase.

```
      reaction: CycB -> AA
 reaction rate: K2*CycB nM/minute
```

```
    parameters: K2  = 0 1/minute, []constant, variable by rule
                V2i = 0.005 1/nM*minute
                V2a = 0.25  1/nM*minute
       species: CycB = 0 nM
                APCi = 1 nM
                APCa = 0 nM
                AA = 100 nM [x]constant [x]boundary
algebraic rule: (V2i*APCi) + (V2a*APCa) - K2
```

Initially, Cyclin B degradation is low. This implies the amount of active APC (APCa) = 0 and inactive APC (APCi) = APCtotal = 1 nM.

Test the algebraic rule by simulating reactions 1 and 2 with `APCi = 0` and `APCa = 1`.

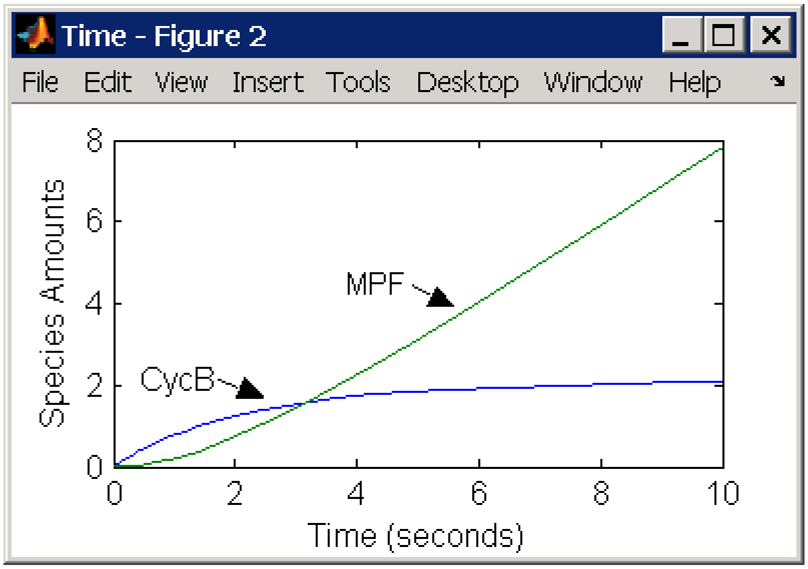


Reaction 3, Dimerization of Cyclin B with Cdc2 Kinase

Cyclin B dimerizes with Cdc2 kinase to form M-phase promoting factor (MPF).

```
     reaction: Cdc2 + CycB -> MPF
reaction rate: k3*Cdc2*CycB nM/minute
   parameters: k3 =   0.005 1/(nM*minute)
      species: Cdc2 = 100     nM
               CycB =   0     nM
               MPF  =   0     nM
```

Test the model by simulating with `K2 = 0.25`.

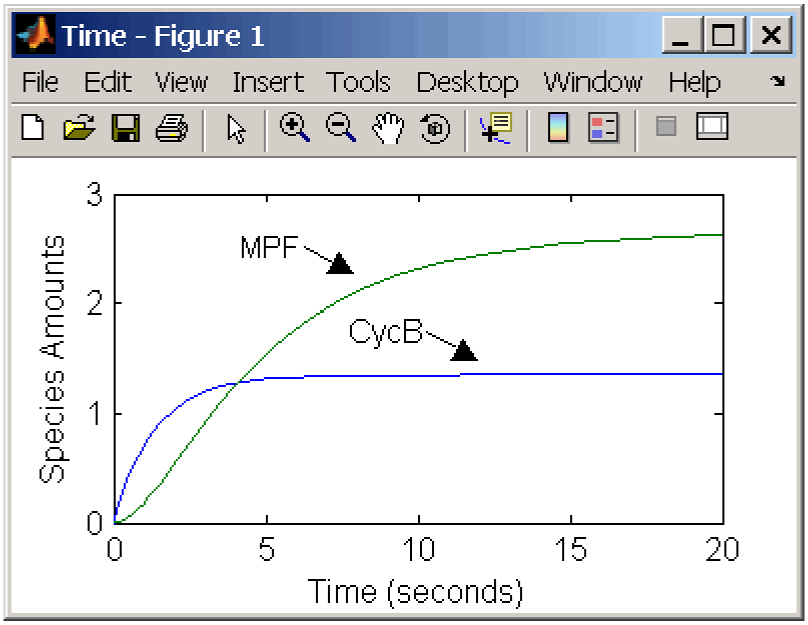


Reaction 4, Degradation of Cyclin B on MPF

Cyclin B is tagged with ubiquitin groups and degrades while bound to Cdc2.

```
      reaction: MPF -> Cdc2 + AA
 reaction rate: K2*[MPF]
    parameters: K2 = 0 or 0.25 1/minute, variable by rule
                v2i = 0.005 1/(nM*minute)
                v2a = 0.25 1/(nM*minute)
       species: MPF = 0 nM
                APCi = 1 nM
                APCa = 0 nM
                AA = 100 nM [x]constant [x]boundary
algebraic rule: (v2i*APCi) + (v2a*APCa) - K2
```

Test the simulation with `APCa = 1 and APCi = 0`. Because the amount of APCa (active) is high, K2 increases and the degradation starts to balance the synthesis of MPF.



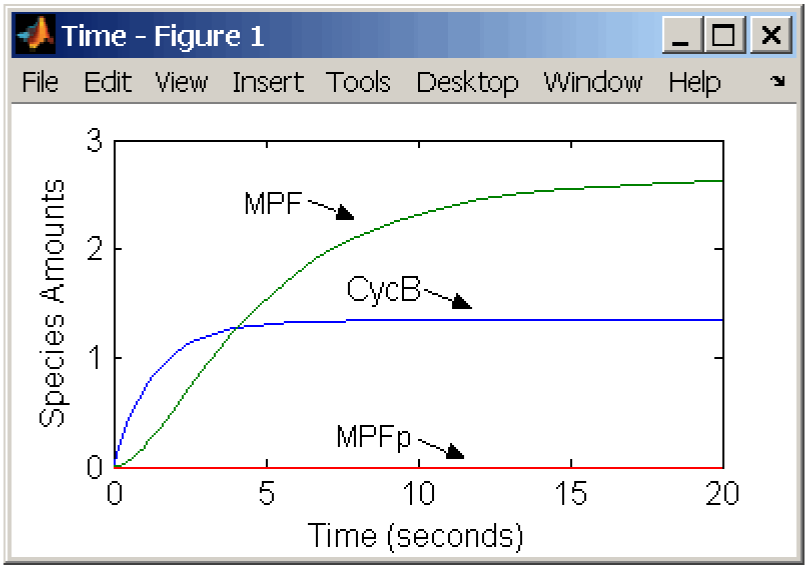
Reaction 5, Deactivation of Active MPF

Active MPF (MPFp) is dephosphorylated on Thr-161 by an unknown phosphatase (PP) to inactive MPF (MPF).

```
     reaction: MPFp -> MPF
reaction rate: kpp*[MPFp]
   parameters: kpp = 0.004 1/minute
      species: MPFp = 0 nM
               MPF = 0 nM
```

`kcakr = 0.004` 1/minute [Marlovits 1998, p. 175], but is renamed to `kpp` [Borisuk 1998].

Test simulation with APCa = 1 and APCi = 0. MPF increases without reaching steady state.



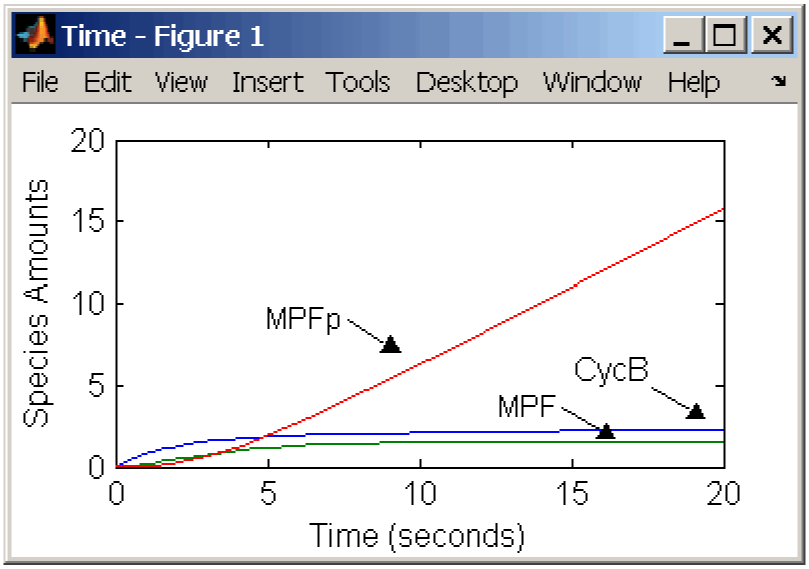
Reaction 6, Activation of MPF

Inactive MPF (MPF) is phosphorylated on Thr-161 by an unknown cyclin activating kinase (CAK).

```
     reaction: MPF -> MPFp
reaction rate: kcak*[MPF]
   parameters: kcak = 0.64 1/minute
      species: MPF = 0 nM
               MPFp = 0 nM
```

The kinase reaction that phosphorylates MPF to the active form is 160 times faster than the phosphatase reaction that dephosphorylates active MPF.

Simulate the model with reactions 1 to 6. Notice that after adding reaction 6, most of the product goes to active MPF (MPFp).



Reaction 7, Remove Inhibiting Phosphate from Inhibited MPF

Cdc25 phosphatase removes the inhibiting phosphate groups at the threonine 14 and tyrosine 15 residues on Cdc2 kinase.

```
     reaction: pMPF -> MPF
reaction rate: Kcdc25*[pMPF]
   parameters: Kcdc25 = 0.0 1/minute or 0.017 1/minute, variable by
                                                  algebraic rule
               V25i = 0.017 1/(mM*minute)
               V25a = 0.17 1/mM*minute
      species: pMPF = 0 nM
               MPF = 0 nM
               Cdc25 = 1 nM (inactive)
               Cdc25p = 0 nM (active)
algebraic rule: (V25i*Cdc25) + (V25a*Cdc25p) - Kcdc25
```

Initially, all of the Cdc25 phosphatase is in the inactive form (Cdc25).

Enter the initial value for Kcdc25 as 0.0 and let the first time step calculate the value from the rule, or enter an initial value using the rule.

Initially, set **ConstantAmount** for Cdc25 and Cdc25p to test reactions 1 through 7. Then after you can add the reactions to regulate the Cdc25 phosphatase by clearing the **ConstantAmount** property.

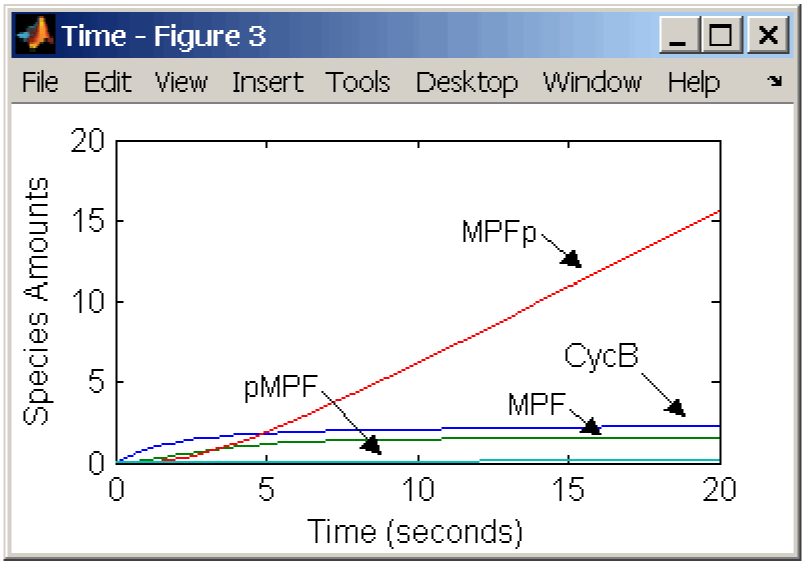
Reaction 8, Inhibition of MPF by Phosphorylation

Addition of inhibiting phosphate groups by Wee1 kinase to inhibit active M-phase promoting factor (MPF). Myt1 kinase is also involved with the phosphorylation, but its contribution is grouped with Wee1.

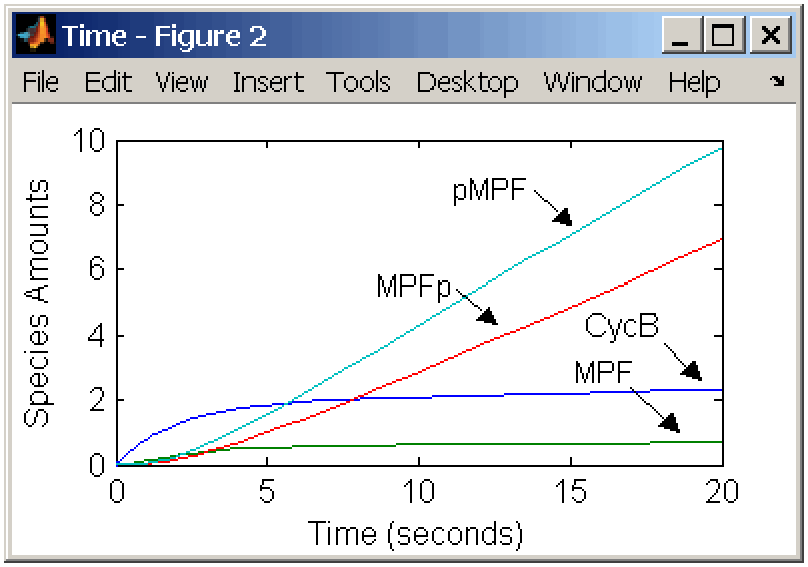
```
     reaction: MPF -> pMPF
reaction rate: Kwee1*[MPF]
   parameters: Kwee1 = 0.0 1/minute or 0.01 1/minute, variable by
                                 algebraic rule
               Vwee1i = 0.01 1/(nM*minute)
               Vwee1a = 1.0 1/(nM*minute)
      species: MPF = 0 nM
               pMPF = 0 nM
               Wee1p = 1 nM (inactive)
               Wee1 = 0 nM  (active)
algebraic rule: (Vwee1i*Wee1p) + (Vwee1a*Wee1) - Kwee1
```

The initial capitalization for the parameter Kwee1 is a convention to indicate that this value changes during the simulation.

Test the simulation for reactions 1 through 8 with `Wee1p` (inactive) = `1` and `Wee1` (active) = `0`.



Test the simulation with `Wee1p` (inactive) = `0` and `Wee1` (active) = `1`.



Reaction 11, Degradation of Cyclin B on Inhibited MPF

Degradation of cyclin B (CycB) on inhibited MPF (pMPF). Cyclin B is tagged with ubiquitin groups and degrades while bound to Cdc2.

```
      reaction: pMPF -> Cdc2 + AA
 reaction rate: K2*[pMPF] nM/minute
    parameters: K2 = 0 or 0.25 1/minute, variable by rule
                V2i = 0.005 1/nM*minute
                V2a = 0.25 1/nM*minute
       species: MPF = 0 nM
                APCi = 1 nM
                APCa = 0 nM
                AA = 100 nM [x]constant [x]boundary
                Cdc2 = 100 nm
algebraic rule: (V2i*APCi) + (V2a*APCa) - K2
```

Test the simulation with Wee1 active (Wee1 = 1) and APC active (APCi = 1).

Reaction 12, Deactivation of MPF to Inhibited MPF

Inhibited/active MPF (pMPFp) is dephosphorylated on Thr-161 by an unknown phosphatase (PP) to inhibited MPF (pMPF). Compare reaction 12 with reaction 5 on page B-33.

```
     reaction: pMPFp -> pMPF
reaction rate: kpp*[pMPFp]
   parameters: kpp = 0.004 1/minute
      species: pMPFp = 0 nM
               pMPF  = 0 nM
```

Reaction 13, Activation of Inhibited MPF

Inhibited MPF (pMPF) is phosphorylated on Thr-161 by an unknown cyclin-activating kinase (CAK). Compare reaction 13 with reaction 6 on page B-34.

```
     reaction: pMPF -> pMPFp
reaction rate: kcak*[pMPF] nM/minute
```

```
   parameters: kcak = 0.64 1/minute
      species: pMPF = 0 nM
               pMPFp = 0 nM
```

Test the simulation with Wee1p = 1 (inactive)/ Wee1 = 0 and then test with Wee1p = 0 (inactive)/ Wee1 = 1.

Reaction 15, Degradation of Cyclin B on Active but Inhibited MPF

Degradation of cyclin B (CycB) on inhibited MPF (pMPF). Cyclin B is tagged with ubiquitin groups and degrades while bound to cdc2 kinase.

```
      reaction: pMPFp -> Cdc2 + AA
 reaction rate: K2*[pMPFp] nM/minute
    parameters: K2 = 0 or 0.25 1/minute, variable by rule
                v2i = 0.005 1/nM*minute
                v2a = 0.25 1/nM*minute
       species: MPF = 0 nM
                APCi = 1 nM
                APCa = 0 nM
                AA = 100 nM [x]constant [x]boundary
                Cdc2 = 100 nm
algebraic rule: (V2i*APCi) + (V2a*APCa) - K2
```

Reaction 16, Inhibit MPF by Phosphorylation

Addition of inhibiting phosphate groups by Wee1 kinase to inhibit active M-phase promoting factor (MPF). Myt1 kinase is also involved with the phosphorylation, but its contribution is grouped with Wee1.

```
      reaction: MPFp -> pMPFp
 reaction rate: Kwee1*[MPFp] nM/minute
    parameters: Kwee1 = 1/minute []constant, variable by rule
                Vweei = 0.01 1/nM*minute
                Vweea = 1 1/nM*minute
       species: MPFp = 0 nM
                pMPFp = 0 nM
                Wee1p = 1 nM (inactive)
                Wee1 = 0 nM (active)
algebraic rule: (Vwee1i*Wee1p) + (Vwee1a*Wee1) - Kwee1
```

Reaction 17, Remove Inhibiting Phosphate from Activated MPF

Remove the inhibiting phosphate group from pMPFp with cdc25 phosphatase.

```
     reaction: pMPFp -> MPFp
reaction rate: Kcdc25*[pMPFp]
   parameters: Kcdc25 = 0 1/minue, []constant, variable by rule
               V25i = 0.017 1/nM*minute
               V25a = 0.17 1/nM*minute
      species: pMPFp = 0 nM
               MPFp = 0 nM
algebraic rule: (V25i*Cdc25) + (V25a*Cdc25p) - Kcdc25
```

Reaction 19, Degradation of Cyclin B on Activated MPF

Degradation of cyclin B (CycB) on inhibited MPF (pMPF). Cyclin B is tagged with ubiquitin groups and degrades while bound to cdc2 kinase.

```
      reaction: MPFp -> MPF + AA
 reaction rate: K2*[MPFp] nM/minute
    parameters: K2 = 0 or 0.25 1/minute, variable by rule
                V2i = 0.005 1/nM*minute
                V2a = 0.25 1/nM*minute
       species: MPF = 0 nM
                MPFp = 0 nM
                APCi = 1 nM
                APCa = 0 nM
                AA = 100 nM [x]constant [x]boundary
                Cdc2 = 100 nm
algebraic rule: (V2i*APCi) + (V2a*APCa) - K2
```

Reaction 36, Activation of Cdc25 by Activated MPF

Activation of cdc25 phosphatase by phosphorylation with active M-phase promoting factor (MPFp).

```
     reaction: Cdc25 + (MPFp) -> Cdc25p + (MPFp)
reaction rate: (k25*[MPFp]*[Cdc25])/(Km25 + [Cdc25])
   parameters: k25 = 0.02 1/minute
               Km25 = 0.1 nM
      species: Cdc25 = 1 nM (inactive)
               Cdc25p = 0 nM (active)
```

Initially MPF is inhibited (MPF* reacts to pMPF*).

Reaction 37, Deactivation of Cdc25

Deactivation of cdc25 phosphatase by dephosphorylation with an unknown phosphatase.

```
     reaction: Cdc25p -> Cdc25
reaction rate: (k25r*[Cdc25p])/(Km25r + [Cdc25p])
   parameters: k25r = 0.1 nM/minute
               Km25r = 1 nM
      species: Cdc25 = 1 nM (inactive)
               Cdc25p = 0 nM (active)
```

Reaction 38, Deactivation of Wee1 by Active MPF

Deactivation of Wee1 kinase by phosphorylation with active M-phase promoting factor (MPFp).

```
     reaction: Wee1 + (MPFp) -> Wee1p + (MPFp)
reaction rate: (kw*[MPFp]*[Wee1])/(Kmw + [Wee1]) nM/minute
   parameters: kw = 0.02 1/minute
               Kmw = 0.1 nM
      species: Wee1p = 1 nM (inactive)
               Wee1 = 0 nM (active)
```

Initially MPF is inhibited (MPF* reacts to pMPF*).

Reaction 39, Activation of Wee1

Activation of Wee1 kinase by dephosphorylation with an unknown kinase.

```
     reaction: Wee1p -> Wee1
reaction rate: (kwr*[Wee1p])/(Kmwr + [Wee1p]) nM/minute
   parameters: kwr = 0.1 nM/minute
               Kmwr = 1 nM
      species: Wee1p = 1 nM (inactive)
               Wee1 = 0 nM (active)
```

Reaction 40, Activation of Intermediate Enzyme by Active MPF

The inactive intermediate enzyme (IE) is activated by phosphorylation with active M-phase promoting factor (MPFp).

```
     reaction: IE + (MPFp) -> IEp + (MPFp)
reaction rate: (kie*[MPFp]*[IE])/(Kmie + [IE])
   parameters: kie = 0.02 1/minute
               Kmie = 0.01nM
      species: IE = 1 nM (inactive)
               IEp = 0 nM (active)
```

Reaction 41, Deactivation of IE

The active intermediate enzyme (IE) is deactivated by dephosphorylation.

```
     reaction: IEp -> IE
reaction rate: (kier*[IEp])/(Kmier + [IEp])
   parameters: kier = 0.15 nM/minute
               Kmier = 0.01 nM
      species: IE = 1 nM (inactive)
               IEp = 0 nM (active)
```

Reaction 42, APC Activation by IEp

Anaphase-promoting complex (APC) is activated by an active intermediate enzyme (IEp).

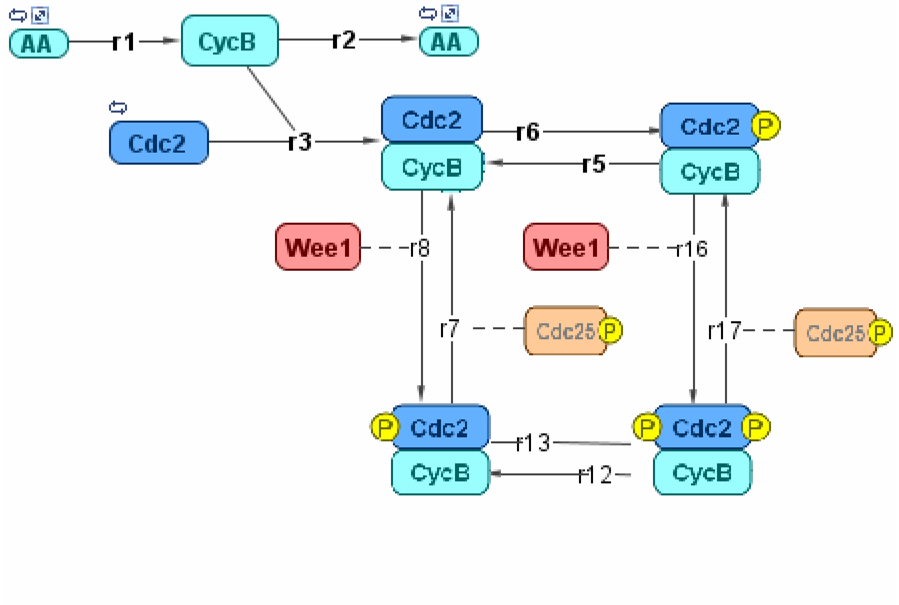
```
     reaction: APCi + IEp -> APCa + IEp
reaction rate: (kap*[IEp]*[APCi])/(Kmap + [APCi])
   parameters: kap = 0.13 1/minute
               Kmap = 0.01 nM
     species : APCi = 1 nM
               APCa = 0 nM
```

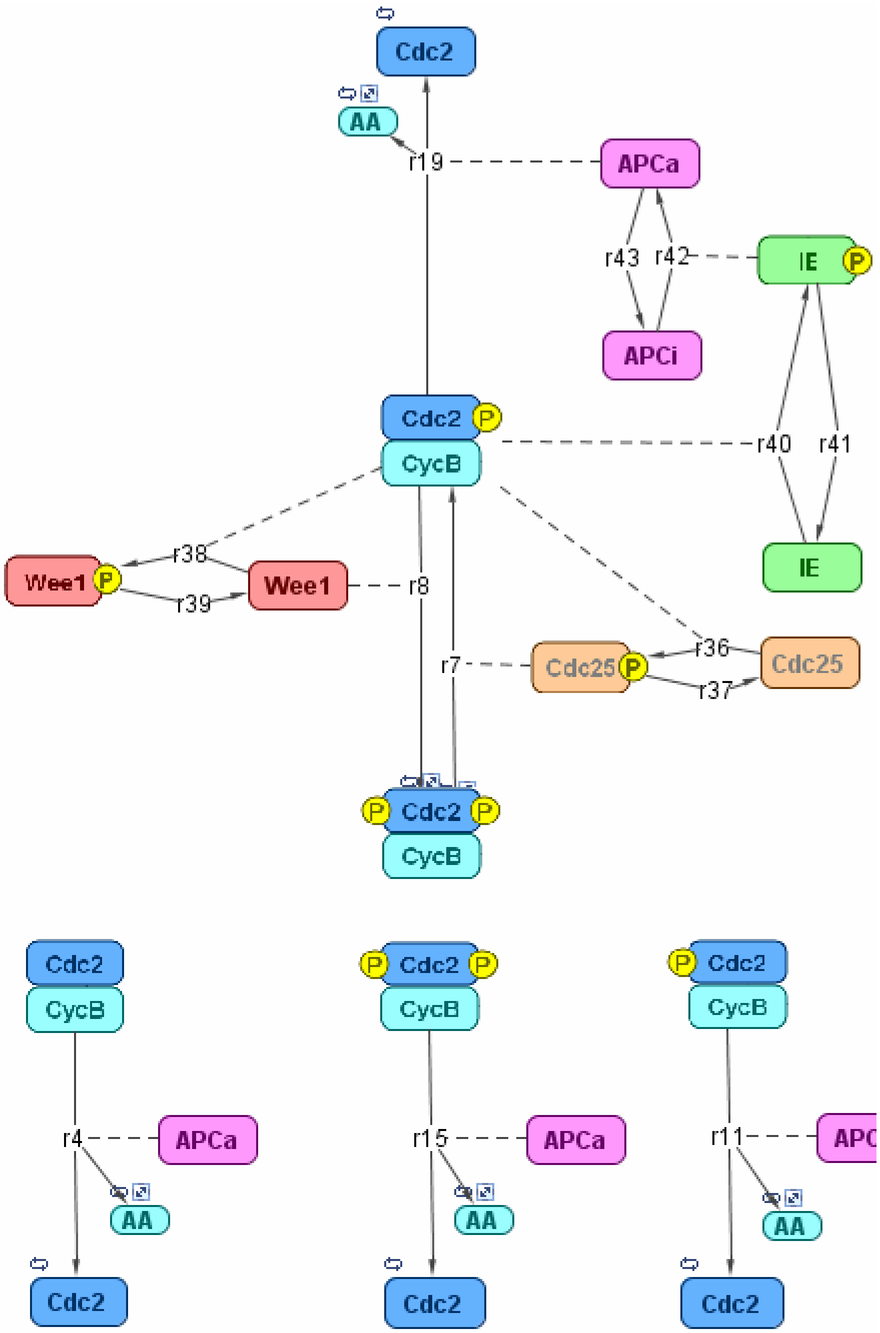
Reaction 43, APC Deactivation

Anaphase-promoting complex (APC) is deactivated.

```
     reaction: APCa -> APCi
reaction rate: (kapr*[APCa])/(Kmapr + [APCa])
   parameters: kapr = 0.13 nM/minute
               Kmapr = 1 nM
     species : APCi = 1 nM
               APCa = 0 nM
```

Block Diagram of the M-Phase Control Model with Reactions


AA
r1
CycB
r2
AA
Cdc2
r3
Cdc2
CycB
r6
Cdc2 P
CycB
r5
Wee1
r8
Wee1
r16
r7
Cdc25 P
r17
Cdc25 P
P Cdc2
CycB
r13
P Cdc2 P
CycB
r12



References

[1] Borisuk M, Tyson J (1998), “Bifurcation analysis of a model of mitotic control in frog eggs,” Journal of Theoretical Biology, 195(1):69–85, PubMed 9802951.

[2] Marlovits G, Tyson C, Novak B, Tyson J (1998), “Modeling M-phase control in Xenopus oocyte extracts: the surveillance mechanism for unreplicated DNA,” Biophysical Chemistry, 72(1-2):169–184, PubMed 9652093.

[3] Novák B, Tyson J (1993), “Numerical analysis of a comprehensive model of M-phase control in Xenopus oocyte extracts and intact embryos,” Journal of Cell Science, 106(4):1153–1168, PubMed 8126097.