SBMLToolbox
4 MATLAB

User's manual

Sarah M Keating

Science and Technology Research Institute
University of Hertfordshire
Hatfield, AL10 9AB
United Kingdom

http://www.sbml.org
mailto:sbml-team@caltech.edu

Contents

1. Introduction

The SBMLToolbox provides a set of functions that allow an SBML model to be imported into MATLAB and stored as a structure within the MATLAB environment. At present the toolbox includes functions to translate a sbml document into a MATLAB_SBML structure, save and load these structures to/from a MATLAB data file, validate each structure (e.g. reaction structure), view the structures using a set of GUIs and to convert elements of the MATLAB_SBML structure into symbolic form and thus allow access to MATLAB's Symbolic Toolbox.

The toolbox is not intended to be a complete Systems Biology toolbox for MATLAB but a platform which facilitates the import/export of SBML and from which a user can develop their own functionality.

2. Installation

2.1 Downloads

There are two downloads available

1) SBMLToolbox-setup.exe - windows setup program that will install the SBMLToolbox with prebuilt executables and all necessary library files

2) SBMLToolbox_src.zip – a zip file containing all the code for the SBMLToolbox; suitable for use with any operating system

2.2 Windows

At the command prompt change to directory 'SBMLToolbox/toolbox' and type 'make'

This will start Matlab and run a script that performs the following

1) Adds this folder (SBMLToolbox/toolbox) and all its subdirectories to the Matlab path

2) Checks whether the appropriate libraries are on the system PATH and if not adds these libraries to the MATLABROOT\bin\win32 directory which is on the PATH

3) Prompts for whether to exit Matlab

The installation process described above can also be performed from within the MATLAB environment by changing to directory SBMLToolbox/toolbox and typing 'install'.

2.3 Linux

In order to use the SBMLToolbox on linux you must have downloaded and installed libSBML (see http://sbml.org/software/libsbml/) prior to the installation.

Assuming libSBML is installed; to build SBMLToolbox:

1) Change to the directory 'SBMLToolbox/toolbox.

2) Ensure that Matlab's mex compiler is in your PATH.

   You can verify this by typing 'mex' or 'which mex' at the command-prompt (The mex executable is located in Matlab's bin directory).

3) Ensure the CFLAGS and LDFLAGS point to the directories containing the libsbml header and library files.

   For example, if you installed libsbml in /usr/local:

   In sh or Bash:

```
export CFLAGS=-I/usr/local/include
export LDLAGS=-L/usr/local/lib
```

   In csh or tcsh:

```
setenv CFLAGS -I/usr/local/include
setenv LDLAGS -L/usr/local/lib
```

4) Type 'make'

This should build TranslateSBML.mexglx
OutputSBML.mexglx.
ReadAndValidateSBML.mexglx.

To run:

Ensure the directory containing these files and the all the toolbox subdirectories are in your Matlab path. For example, at the Matlab prompt:

```
>> addpath('SBMLToolbox/toolbox);
>> addpath('SBMLToolbox/toolbox/StoreModels');
```

etc…

You may wish to add these commands to your Matlab startup script in ${HOME}/matlab/startup.m

3. Importing and exporting SBML

The functions to import and export SBML use MATLAB's mexFunction and therefore must be compiled prior to use. The windows-setup download of the toolbox provides the necessary dlls and therefore no compilation is necessary.

In order to import a sbml model into MATLAB type

```
   >> Model = TranslateSBML
or >> Model = TranslateSBML('../path/filename.xml')
```

If no filename is supplied this will open a browse window. If a filename is supplied the file to be opened must be in the MATLAB's current directory or the full pathname must be supplied as the argument.

Alternatively if the version of libSBML being used has been built with the Xerces-c XML library (as is the case with the prebuilt windows installation) the alternative function ReadAndValidateSBML can be used. This performs validation and consistency checking on the sbml document being imported and reports possible problems prior to import.

Both these functions return a MATLAB_SBML structure named Model within the MATLAB environment (Figure 1). The MATLAB_SBML structure is defined in full in the document MATLAB_SBML_Structure.pdf which is also part of the SBMLToolbox download.

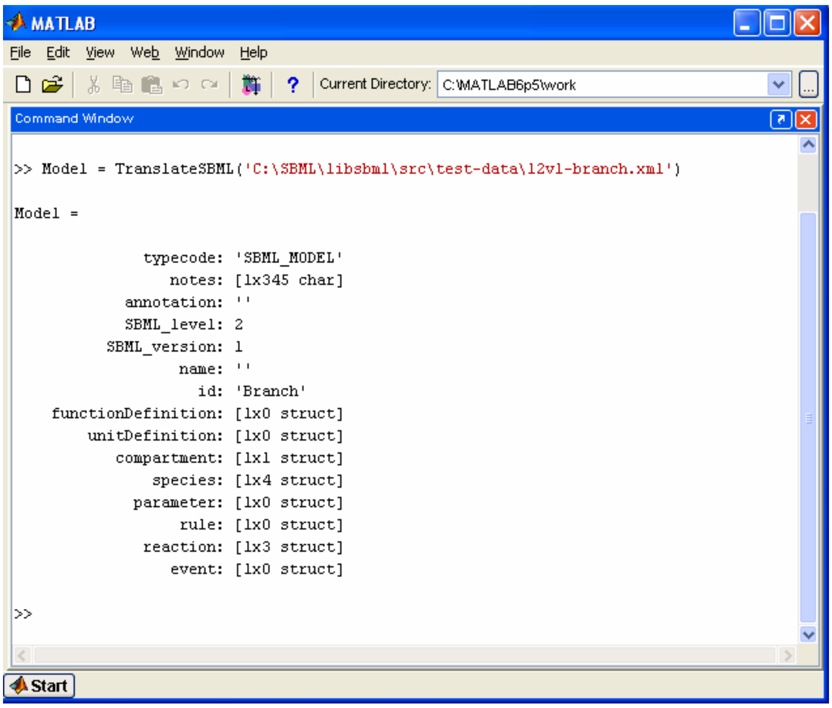


Figure 1*: Screenshot of the command 'Model = TranslateSBML('../branch.xml')' and the resulting MATLAB_SBML structure returned*

The structure returned can then be passed as an argument to other functions within the SBMLToolbox or functions developed by the user.

To export SBML from MATLAB type

>> OutputSBML(Model)

where ‘Model’ is the MATLAB_SBML structure.

A browse window is opened to allow the user to specify the name and location of the output file which will be saved as an .xml document.

4. Access model

The AccessModel folder contains a number of functions that derive information from the MATLAB_SBML structure.

The functions in the AccessModel folder are listed in Table 1.

Table 1: Functions and their type in folder AccessModel

Type of function	Function name
MATLAB help	Contents.m
Getting information from model	GetAllParameters.m GetAllParametersUnique.m GetCompartments.m GetGlobalParameters.m GetSpecies.m
Getting information from reaction	GetParameterFromReaction.m GetParameterFromReactionUnique.m
Deriving information	DetermineSpeciesRoleInReaction.m GetRateLawsFromReactions.m GetRateLawsFromRules.m GetSpeciesAlgebraicRules.m GetSpeciesAssignmentRules.m GetStoichiometryMatrix.m
Overview of model	CheckValues.fig CheckValues.m

4.1 Getting information from model functions

All the functions in this category have the same format.

Format	>> [names, values] = GetAllParameters(model)	
Argument(s)	model	MATLAB_SBML_Model structure
Returns	names	array of the character string representation of the names[1] of elements
	values	array of the values of each element

NOTE: the function GetAllParametersUnique appends the reaction name to the names of any parameter local to that reaction (Figure 2).

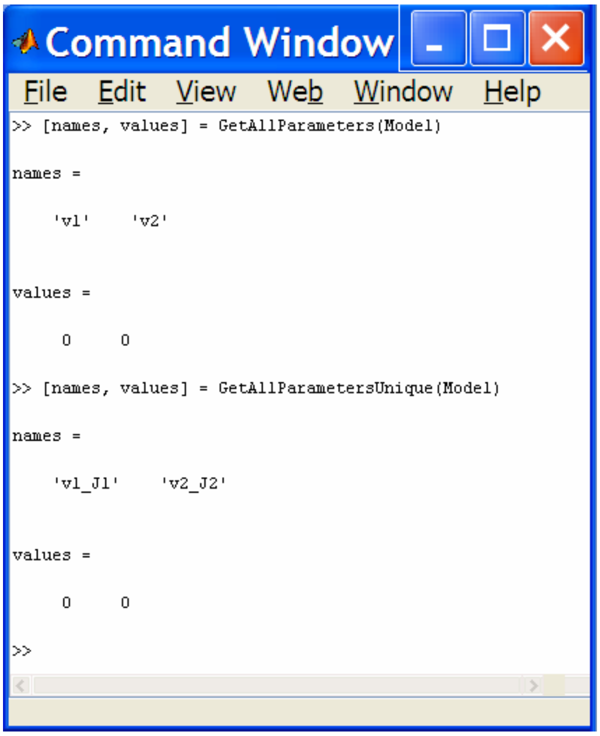


Figure 2: *Using the GetAllParameters and the GetAllParametersUnique functions*

4.2 Getting information from reaction functions

All the functions in this category have the same format.

Format	>> [names, values] = GetParameterFromReaction(reaction)	
Argument(s)	reaction	MATLAB_SBML_Reaction structure
Returns	names	array of the character string representation of the names[1] of elements
	values	array of the values of each element

[1]When the name of an element is returned this will refer to the ‘name’ field in SBML level 1 models and the ‘id’ field in SBML level 2 models.

4.3 Deriving information functions

4.3.1 DetermineSpeciesRoleInReaction

Format	>> y = DetermineSpeciesRoleInReaction(species, reaction)	
Argument(s)	species	MATLAB_SBML_Species structure
	reaction	MATLAB_SBML_Reaction structure
Returns	y = 0	If species is NOT part of reaction
	y = [isProduct, isReactant, isModifier, positionInProductList, posInReactantList] indicating whether the species is a product/reactant/modifier and its position in the relevant List within the reaction	

EXAMPLE:	y	=	DetermineSpeciesRoleInReaction(s, r)	
		=	0	s is not in r
		=	[1, 0, 0, 2, 0]	s is product no 2 in r
		=	[0, 1, 0, 0, 1]	s is reactant no 1 in r
		=	[0, 0, 1, 0, 0]	s is a modifier in r

4.3.2 GetRateLawsFrom…

Format	>> [species, rateLaws] = GetRateLawsFromReactions(model)	
Argument(s)	model	MATLAB_SBML_Model structure
Returns	species	array of the character string representation of all species
	rateLaws	array of the character representation of the rate laws from reactions (for each species in order of species array)

Format	>> [species, rateLaws] = GetRateLawsFromRules (model)	
Argument(s)	model	MATLAB_SBML_Model structure
Returns	species	array of the character string representation of all species
	rateLaws	array of the character representation of the rate laws from rules (for each species in order of species array)

4.3.3 GetSpecies…Rules

Format	>> [species, rules] = GetSpeciesAlgebraicRules (model)	
Argument(s)	model	MATLAB_SBML_Model structure
Returns	species	array of the character string representation of all species
	rules	an array of the character representation of each algebraic rule the species appears in

Format	>> [species, rules] = GetSpeciesAssignmentRules (model)	
Argument(s)	model	MATLAB_SBML_Model structure
Returns	species	array of the character string representation of all species
	rules	an array of the character representation of the assignment rule used to assign value to each species

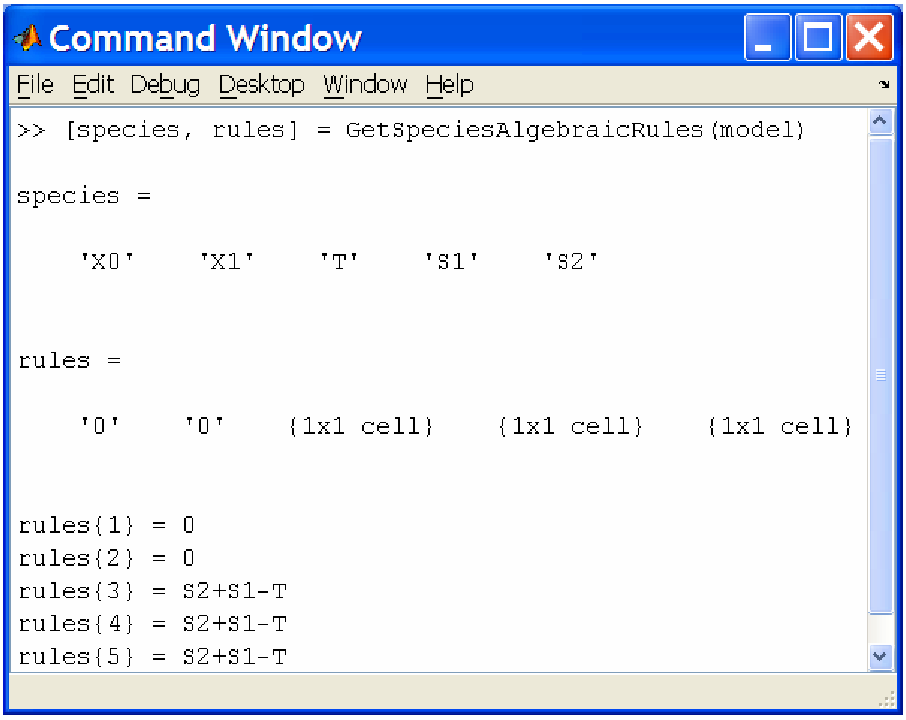


***Figure 3**: Typical output from GetSpeciesAlgebraicRule*

4.3.4 GetStoichiometryMatrix

Format	>> [matrix, species] = GetStoichiometryMatrix (model)	
Argument(s)	model	MATLAB_SBML_Model structure
Returns	matrix	stoichiometry matrix for the species and reactions in the model
	species	array of the character string representation of all species in the order in which the stoichiometry matrix deals with them

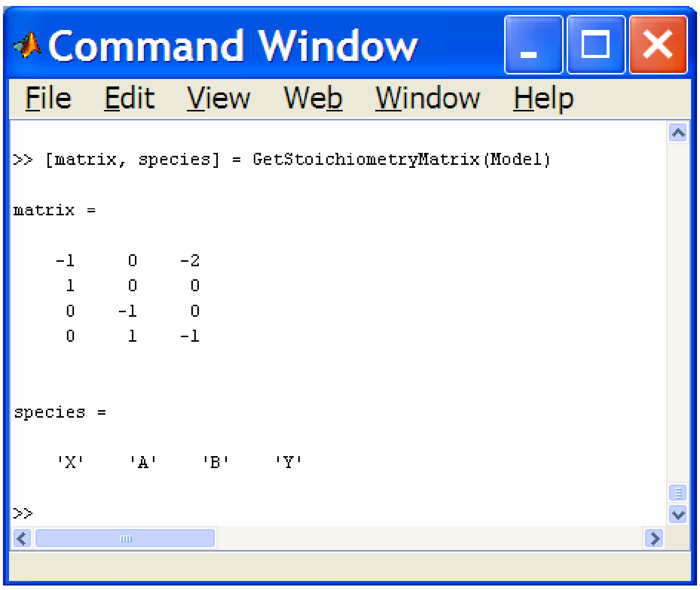


***Figure 4**: Typical output from GetStoichiometryMatrix*

4.4 Overview of model functions

Format	>> [speciesValues, parameterValues] = CheckValues (model)	
Argument(s)	model	MATLAB_SBML_Model structure
Returns	speciesValues	array of values for the initial amount/concentration of the species
	parameterValues	array of values for the parameters
Displays	a GUI that allows the user to check that the values for the parameters and the initial amounts/concentrations of the species are as expected and edit as appropriate.	

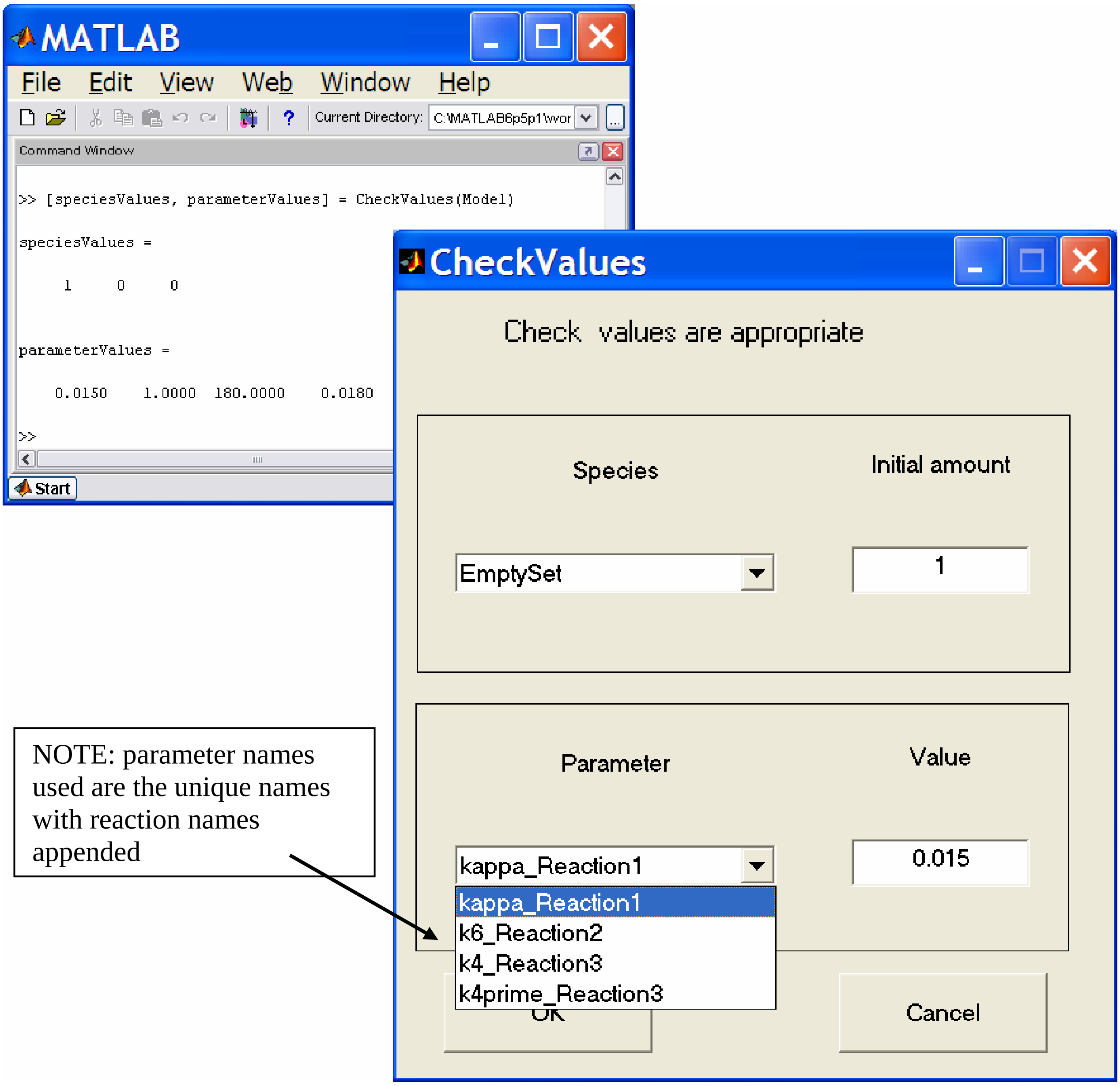


***Figure 5**: Typical output from CheckValues function*

5. Access to symbols

The AccessToSymbols folder contains a number of functions that take elements of the MATLAB_SBML model and convert them to a symbolic form for use with the MATLAB Symbolic Toolbox.

The functions in the AccessToSymbols folder are listed in Table 2.

Table 2: Functions and their type in folder AccessToSymbols

Type of function	Function name
MATLAB help	Contents.m
Getting symbols	GetAllParameterSymbols.m GetAllParameterSymbolsUnique.m GetCompartmentSymbols.m GetGlobalParameterSymbols.m GetParameterSymbolsFromReaction.m GetParameterSymbolsFromReactionUnique.m GetSpeciesSymbols.m
Deriving information	AnalyseSpeciesSymbolic.m GetEquilibrium.m GetStoichiometryMatrixSyms.m GetSymbolicRateLawsFromReactions.m GetSymbolicRateLawsFromRules.m GetSymbolicSpeciesAlgebraicRules.m GetSymbolicSpeciesAssignmentRules.m
Overview of model	PlotTimeCourse.m PlotSelectedTimeCourse.m
General	charFormula2sym.m CreateSymArray.m GetDegree.m

NOTE: The majority of the functions in the AccessToSymbols folder mimic functions explained elsewhere in this manual. Thus explanation will be kept to a minimum.

5.1 Getting symbols functions

All the functions in this category have the same format.

Format	[symbols, values, names] = GetAllParametersSymbols (model)	
Argument(s)	model	MATLAB_SBML_Model structure
Returns	symbols	array of symbols representing of the names[1] of elements
	values	array of the values of each element
	names	array of the character string representation of the names[1] of elements

[1]When the name of an element is returned this will refer to the 'name' field in SBML level 1 models and the 'id' field in SBML level 2 models.

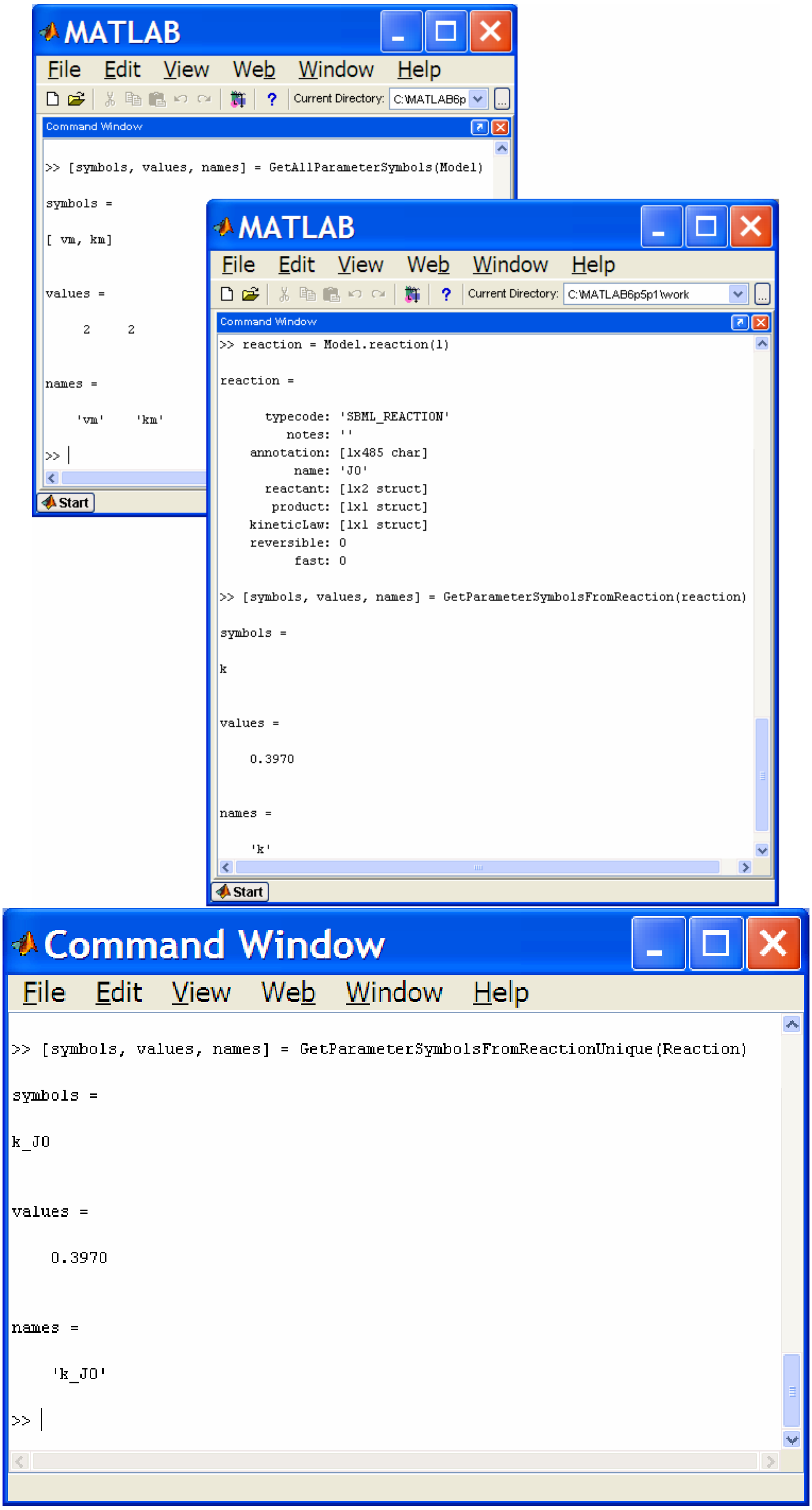
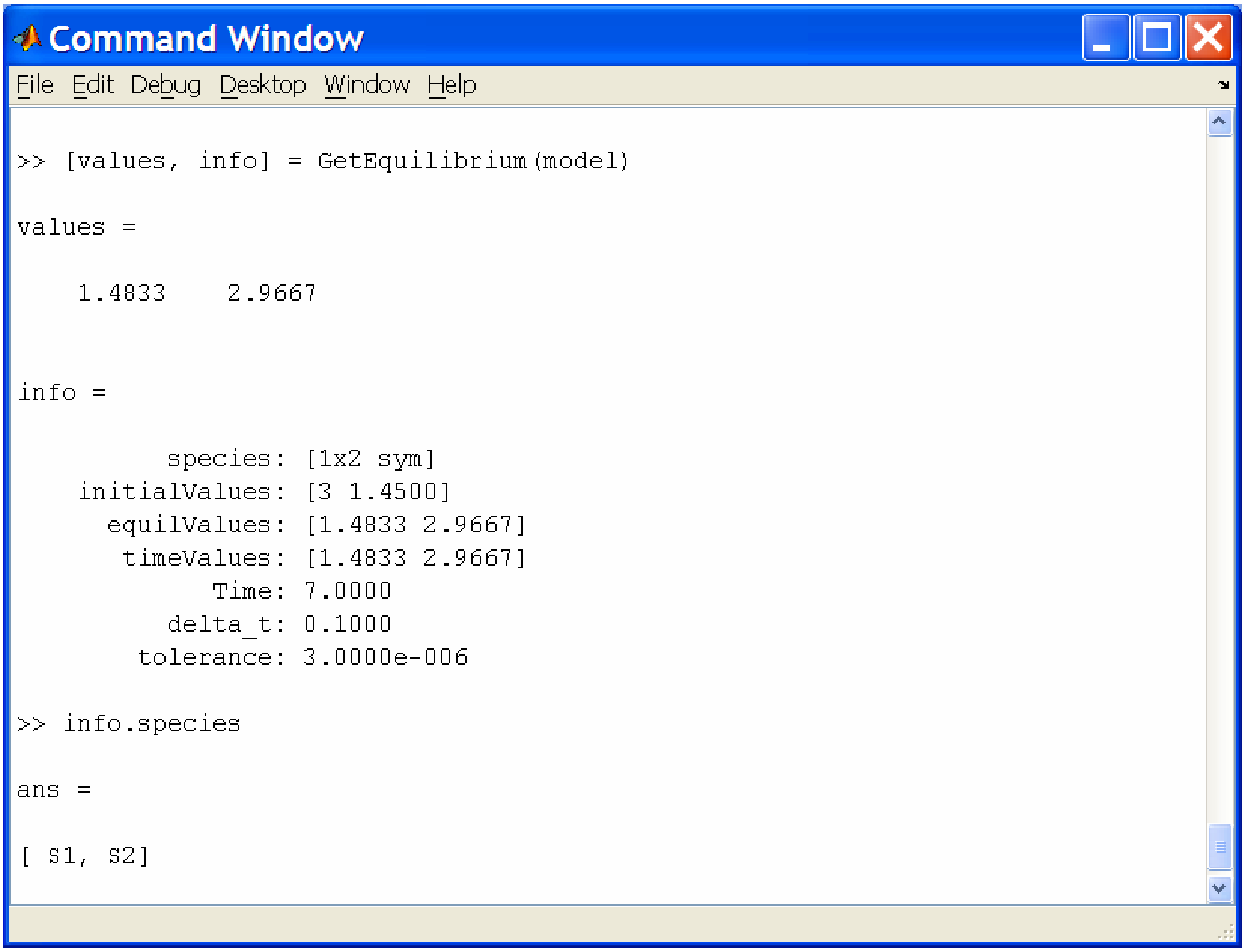


***Figure 6**: Examples of output from Getting Symbols functions*

5.2 Deriving information functions

5.2.1 GetEquilibrium

Format >> [values, info] = GetEquilibrium (model)
Argument(s) model MATLAB_SBML_Model structure
Returns values array of the equilibrium values of each species
info structure detailing the equilibrium
- .species array of symbolic representation of the species
- .initialValues array of the initial amounts used
- .equilValues array of the equilibrium values (= 0 if equilibrium not reached)
- .timeValues array of the amount of each species at the time shown (equal to equilValues if equilibrium was reached)
- .Time elapsed time
- .delta_t time step used in calculations
- .tolerance difference value at which equilibrium was considered to be reached




```
>> [values, info] = GetEquilibrium(model)

values =

    1.4833    2.9667


info =

          species: [1x2 sym]
    initialValues: [3 1.4500]
      equilValues: [1.4833 2.9667]
       timeValues: [1.4833 2.9667]
             Time: 7.0000
          delta_t: 0.1000
        tolerance: 3.0000e-006

>> info.species

ans =

[ S1, S2]
```

***Figure 7**: Typical output from the GetEquilibrium function.*

The algorithm used to calculate the equilibrium involves using the rate equations to produce a set of functions for the change in the amount of each species for a corresponding change in time.

Example:

Reaction A -> B with kinetic law formula k * B

The rate equations are

$$\frac{dA}{dt} = -kB$$

$$\frac{dB}{dt} = kB$$

Rewriting these, the change in amount of A and B for each change in time becomes

$$\Delta A = -kB\Delta t$$

$$\Delta B = kB\Delta t$$

An appropriate time step, time limit and tolerance are calculated from the initial values of the species amounts and parameters involved. The procedure then iterative calculates the new species amounts using the derived functions until either the required tolerance (difference between newly calculated figure and previously calculated figure) has been achieved or the time limit has been reached. If the time limit is reached it is assumed that equilibrium is unlikely to be achieved and the function terminates and reports the values calculated within the info structure returned.

5.3 Overview of model functions

5.3.1 PlotTimeCourse

Format	>> [values] = PlotTimeCourse (model, variableArgs)	
Argument(s)	model	MATLAB_SBML_Model structure
optional	limit	time limit for calculations
	steps	number of time steps to consider
	flag	indicate whether to output data as a comma separated variable file
Returns	values	array of species amounts at the end of the plot time (either at equilibrium or time limit if this has been specified)
Displays	plot of the time course for each of the species within the model as separate graphs	

5.3.2 PlotSelectedTimeCourse

Format	>> [values] = PlotSelectedTimeCourse (model, variableArgs)	
Argument(s)	model	MATLAB_SBML_Model structure
optional	limit	time limit for calculations
	steps	number of time steps to consider
Returns	values	array of species amounts at the end of the plot time (either at equilibrium or time limit if this has been specified)
Displays	plot of the time course for each of the species selected on a single graph	

NOTE: PlotTimeCourse/PlotSelectedTimeCourse uses the same algorithm as GetEquilibrium.

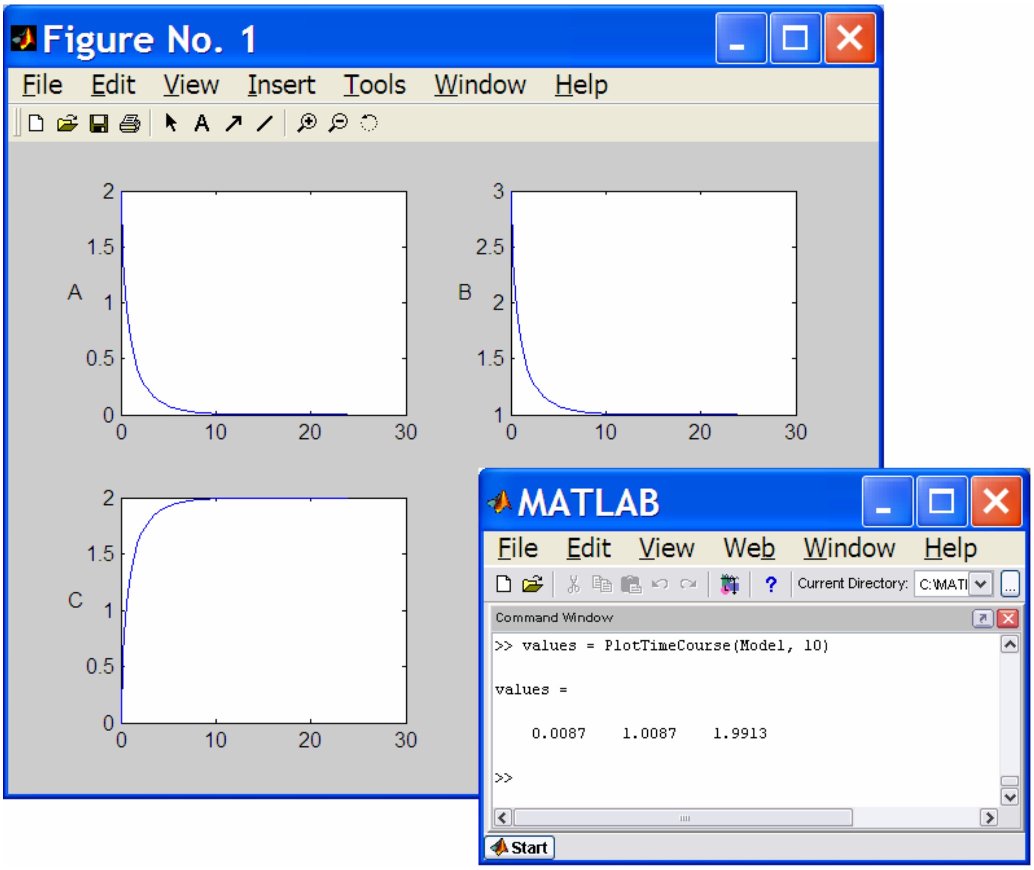


Figure 8: *Examples of the output from PlotTimeCourse function*

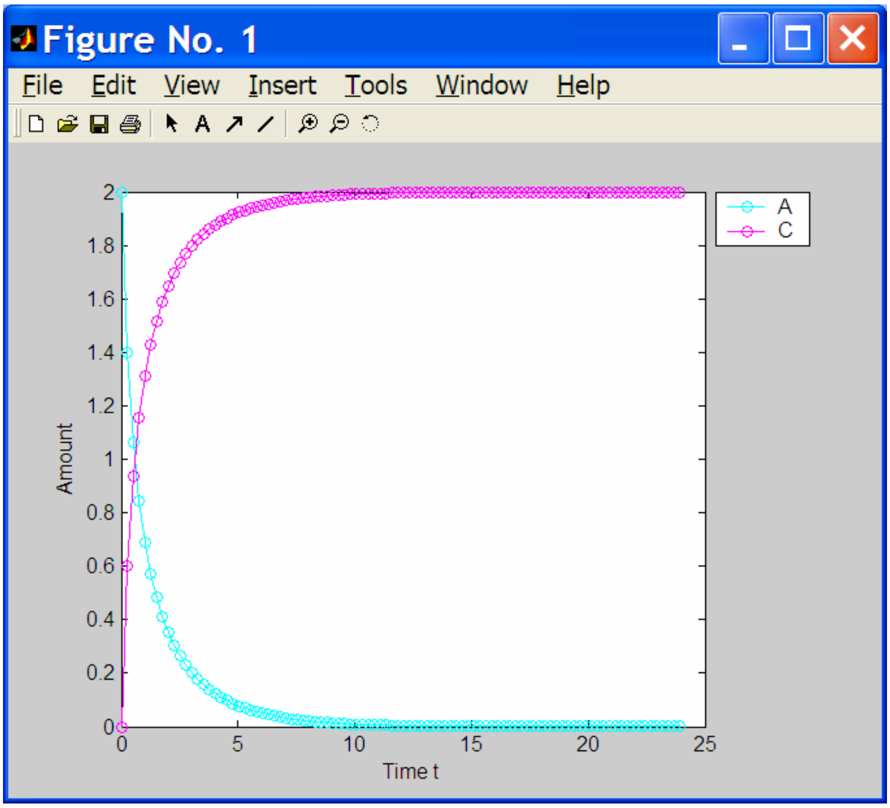


Figure 9:*Output from PlotSelectedTimeCourse function*

5.4 General functions

5.4.1 charFormula2sym

Format	>> [symFormula, symbols] = charFormula2sym(charFormula)	
Argument(s)	charFormula	character respresentation of a mathematical formula
Returns	symFormula	symbolic representation of charFormula
	symbols	array of the symbols used in the formula

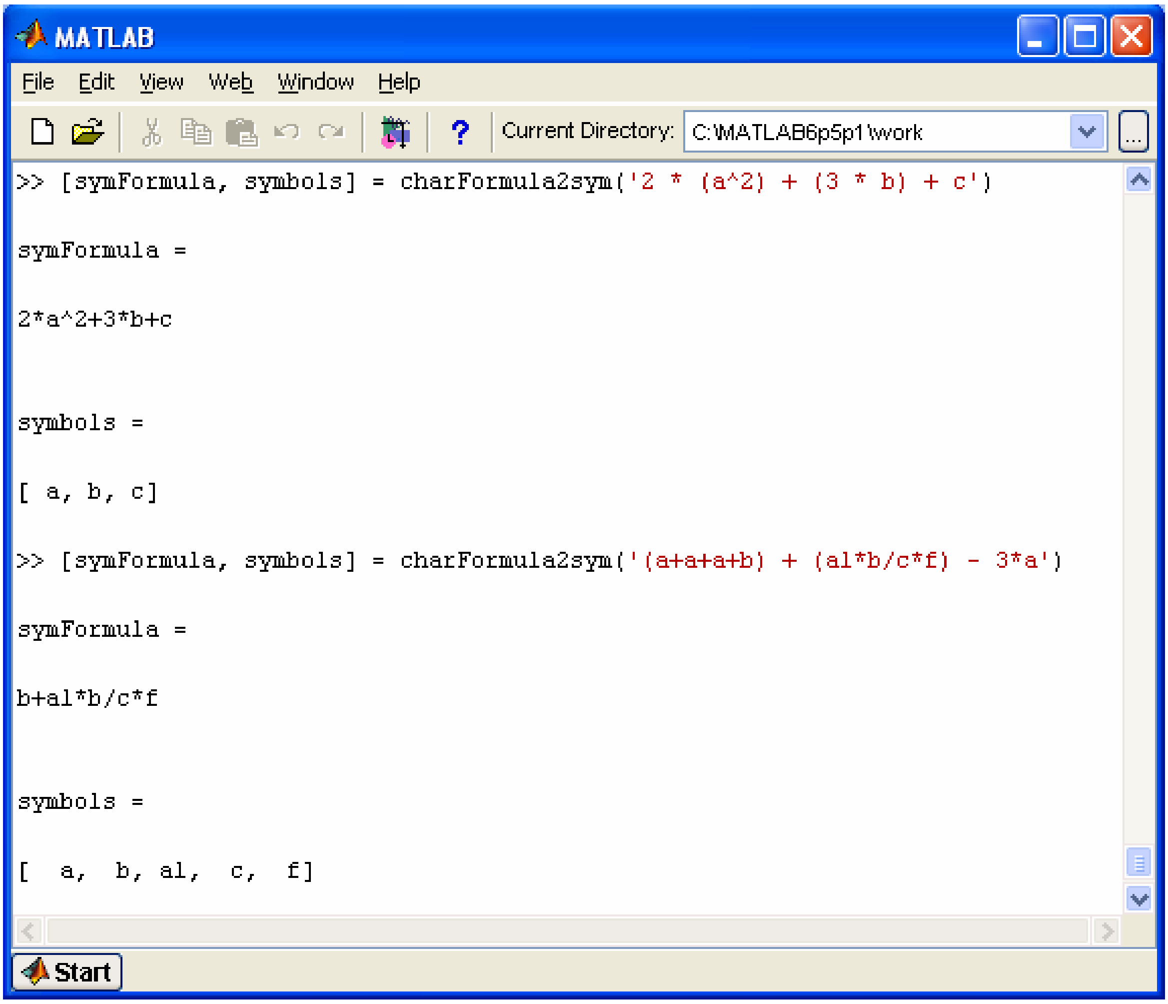


***Figure 10**: Sample outputs from charFormula2sym function*

5.4.2 CreateSymArray

Format	>> [symbols] = CreateSymArray (symFormula)	
Argument(s)	symFormula	symbolic respresentation of a mathematical formula
Returns	symbols	array of the symbols used in the formula

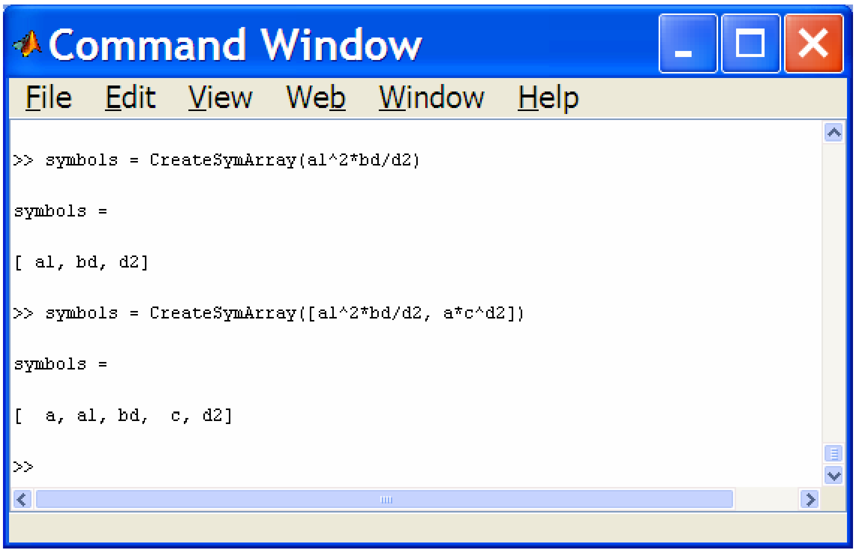


Figure 11*: Output from CreateSymArray function*

5.4.3 GetDegree

Format	>> degree = GetDegree (symPolynomial, symVariable)	
Argument(s)	symPolynomial	symbolic respresentation of a polynomial
	symVariable	single symbol
Returns	degree	the degree of the single symbol in the polynomial

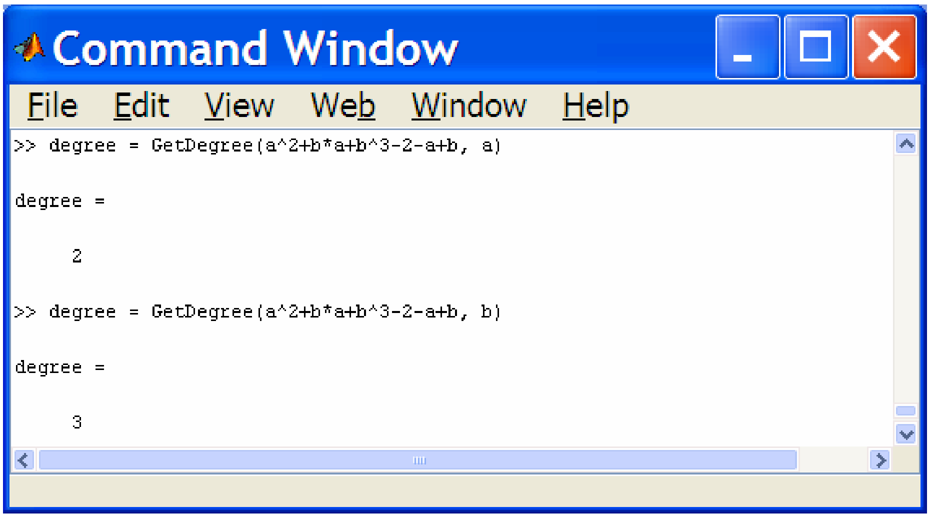


Figure 12*: Output from GetDegree function*

6. Convenience functions

The Convenience folder contains a number of convenience functions.

The functions in the AccessModel folder are listed in Table 3.

Table 3: Functions and their type in folder Convenience

Type of function	Function name
MATLAB help	Contents.m
Checking information	isIntegralNumber.m isValidUnitKind.m
Other	LoseWhiteSpace.m PairBrackets.m RemoveDuplicates.m SubstituteFunction.m

6.1 Checking information functions

6.1.1 isIntegralNumber

Format	>> y = isIntegralNumber(number)	
Argument(s)	number	any number
Returns	y = 1	if number is an integrer
	y = 0	otherwise

NOTE: MATLAB's 'isinteger' function only returns true if the number has been declared as an int; whereas the default type for numbers in MATLAB is double. Thus isIntegralNumber will return true for a number of type double that is can be represented as an integer.

6.1.2 isValidUnitKind

Format	>> y = isValidUnitKind(kind)	
Argument(s)	kind	a string representation of a unit kind
Returns	y = 1	if kind is a valid SBML unit kind
	y = 0	otherwise

6.2 Other functions

6.2.1 LoseWhiteSpace

Format	>> array = LoseWhiteSpace(charArray)	
Argument(s)	charArray	an array of characters
Returns	array	the array of characters with any white space removed

6.2.2 PairBrackets

Format	>> pairs = PairBrackets(charArray)	
Argument(s)	charArray	an array of characters
Returns	pairs	an array of the indices of matching pairs of brackets (ordered using the opening bracket index)

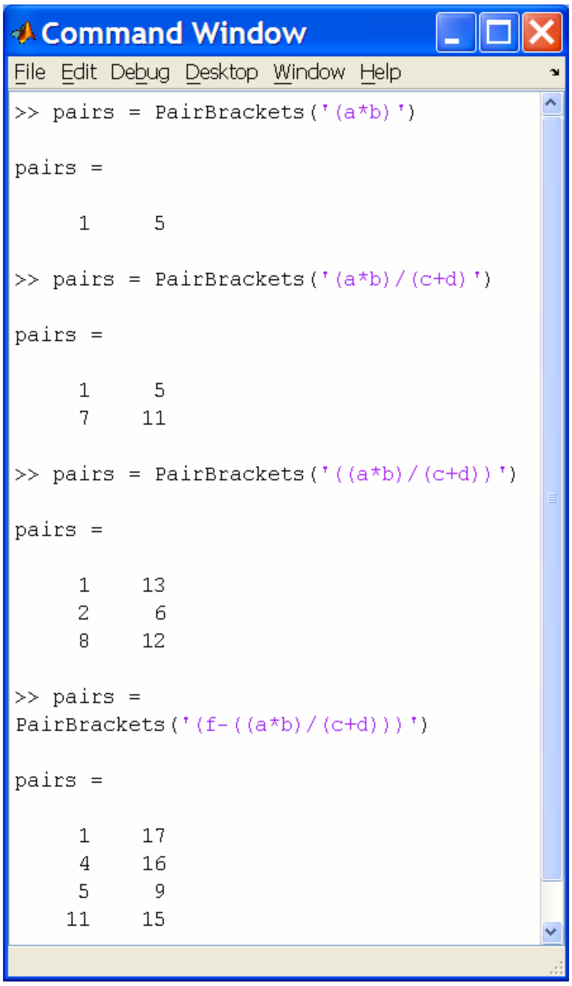


***Figure 13**: Output from PairBrackets function*

6.2.3 RemoveDuplicates

Format	>> array = RemoveDuplicates(anyArray)	
Argument(s)	anyArray	any array
Returns	array	the array with any duplicates removed

EXAMPLE: array = RemoveDuplicates('abcacsdab')
array = 'abcsd'

array = RemoveDuplicates([1,3,2,1,4,3,2,5,1,2])
array = [1,3,2,4,5]

6.2.4 SubstituteFunction

Format	>> formula = SubstituteFunction(charFormula, functionDefinition)	
Argument(s)	charFormula	character respresentation of a mathematical formula
	functionDefinition	MATLAB_SBML_FunctionDefinition structure
Returns	formula	charFormula with the functionDefinition substituted

NOTE: charFormula must contain the ‘id’ of the functionDefinition

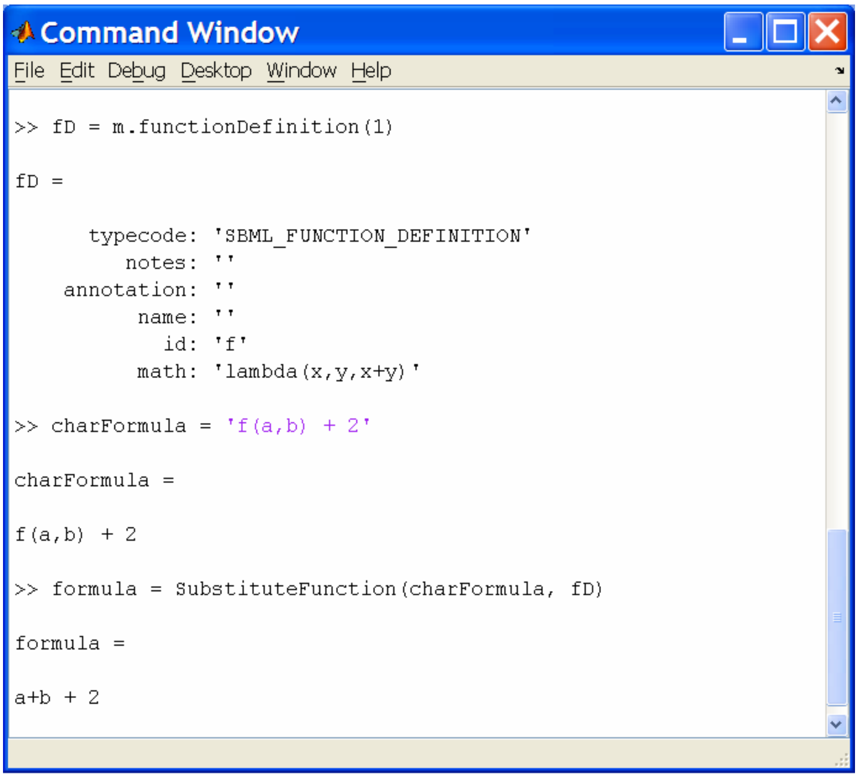


***Figure 14**: Output from SubstituteFunction function*

7. MATLAB_SBML Structure functions

The MATLAB_SBML_Structure_functions folder contains a number of functions that mimic the functions contained in the libSBML C API.

The folder contains subfolders named after the elements of an SBML model e.g. Model, Species, Parameter etc. Each of these subfolders then contains a create function, query functions, get functions and set/unset functions as appropriate to the element.

Full details are not given here as the formats of the functions are similar. However the contents of the parameter folder are used as an example.

7.1 Parameter subfolder

The functions in the parameter subfolder are listed in Table 4.

Table 4: Functions and their type in folder MATLAB_SBML_Structure_functions/Parameter

Type of function	Function name
MATLAB help	Contents.m
create function	Parameter_create.m
query functions	Parameter_isSetId.m Parameter_isSetName.m Parameter_isSetUnits.m Parameter_isSetValue.m
get functions	Parameter_getConstant.m Parameter_getId.m Parameter_getName.m Parameter_getUnits.m Parameter_getValue.m
set functions	Parameter_setConstant.m Parameter_setId.m Parameter_setName.m Parameter_setUnits.m Parameter_setValue.m
unset functions	Parameter_unsetName.m Parameter_unsetUnits.m Parameter_unsetValue.m
Other	Parameter_moveIdToName.m Parameter_moveNameToId.m

7.1.1 create function

Format	>> parameter = Parameter_create(variableArgs)	
Argument(s)		
optional	SBML_level	SBML_level of parameter structure to create (default = 2)
Returns	parameter	MATLAB_SBML_Parameter structure

7.1.2 query functions

Format	>> y = Parameter_isSetId(parameter)	
Argument(s)	parameter	MATLAB_SBML_Parameter structure
Returns	y = 1	if id field is set
	y = 0	if id field is empty

7.1.3 get functions

Format	>> id = Parameter_getId(parameter)	
Argument(s)	parameter	MATLAB_SBML_Parameter structure
Returns	id	id field of the parameter as a string

7.1.4 set functions

Format	>> parameter = Parameter_setId(parameter, id)	
Argument(s)	parameter	MATLAB_SBML_Parameter structure
	id	string that is to be set as the parameter id
Returns	parameter	the parameter structure with the id set

7.1.5 unset functions

Format	>> parameter = Parameter_unsetName(parameter)	
Argument(s)	parameter	MATLAB_SBML_Parameter structure
Returns	parameter	the parameter structure with the name field empty

7.1.5 other functions

Format	>> parameter = Parameter_moveIdToName(parameter)	
Argument(s)	parameter	MATLAB_SBML_Parameter structure
Returns	parameter	the parameter structure with the name field set to the original id – unless the name field was already set

8. Simulation

The Simulation folder contains a number of functions that take a MATLAB_SBML model and convert them files that can be used to simulate the model with MATLABs ode functions.

The functions in the Simulation folder are listed in Table 8.

Table 5: Functions and their type in folder Simulation

Type of function	Function name
MATLAB help	Contents.m
Simulation	AnalyseSpecies.m DisplayODEFunction.m OutputODEFunction.m WriteODEFunction.m
Event handling (called as necessary by WriteODEFunction)	WriteEventAssignmentFunction.m WriteEventHandlerFunction.m
MathML	DealWithPiecewise.m GetArgumentsFromLambdaFunction.m
Other	SelectSpecies.m SelectSpecies.fig

8.1 Simulation functions

8.1.1 AnalyseSpecies

Format	>> [info] = AnalyseSpecies (model)		
Argument(s)	model	MATLAB_SBML_Model structure	
Returns	info	structure detailing the species and how they are affected by the model	
	.Name		character representation of the name of the species
	.constant		flag (1 if constant)
	.boundaryCondition		flag (1 if boundaryCondition)
	.initialValue		initial amount/concentration
	.isConcentration		flag (1 if initialValue is concentration)
	.compartment		compartment containing the species
	.ChangedByReaction		flag (1 if species is in reaction)
	.KineticLaw		KineticLaw formula in which species appears
	.ChangedByRateRule		flag (1 if species is changed by rate rule)
	.RateRule		RateRule formula in which species appears
	.ChangedByAssignmentRule		flag (1 if species is assigned by rule)
	.AssignmentRule		assignment formula for species
	.InAlgebraicRule		flag (1 if species is in an algebraicRule)
	.ConvertedToAssignRule		flag (1 if species is assigned by the algebraic rule)
	.ConvertedRule		algebraicRule converted to assignment for species

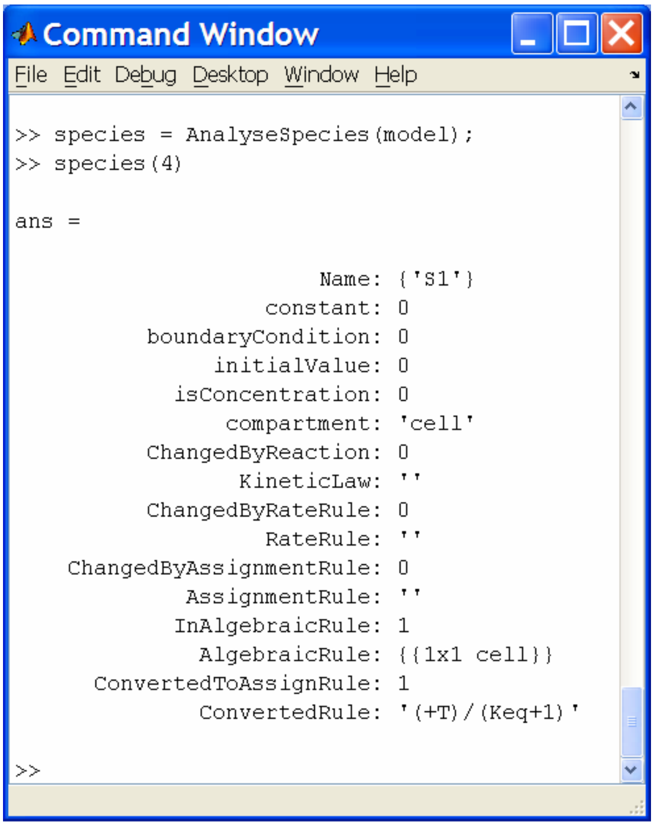


***Figure 15**: Output from AnalyseSpecies function*

8.1.2 WriteODEFunction function

Format	>> WriteODEFunction(model, variableArgs)	
Argument(s)	model	MATLAB_SBML_Model structure
optional	filename	name to give to the .m file to use with the ode solvers[1]
Outputs		file for use with ode solvers

[1] if no name is given the model id/name is used

8.1.3 DisplayODEFunction function

Format	>> DisplayODEFunction(model, variableArgs)	
Argument(s)	model	MATLAB_SBML_Model structure
optional	limit	time limit to use in simulation
	steps	number of steps tp use in the simulation
	filename	name of the .m file to use with the ode solvers[2]
Outputs		plot of the result of the ode solvers

[2] if a filename was used with WriteODEFilename this must be supplied

8.1.4 OutputODEFunction function

Format	>> OutputODEFunction(model, variableArgs)	
Argument(s)	model	MATLAB_SBML_Model structure
optional	flag	indicate whether to plot output
	limit	time limit to use in simulation
	steps	number of steps tp use in the simulation
	flag	indicate whether to output a csv file
	filename	name of the .m file to use with the ode solvers[2]
Outputs		plot of the result of the ode solvers

[2] if a filename was used with WriteODEFilename this must be supplied

8.2 MathML functions

8.2.1 DealWithPiecewise

Format	>> elements = DealWithPiecewise(formula)	
Argument(s)	formula	character representation of a formula containing the MathML function ‘piecewise’
Returns	elements	the elements of the piecewise function

8.2.2 GetArgumentsFromLambdaFunction

Format	>> elements = GetArgumentsFromLambdaFunction(formula)	
Argument(s)	formula	character representation of a formula containing the MathML function ‘lambda’
Returns	elements	the elements of the lambda function

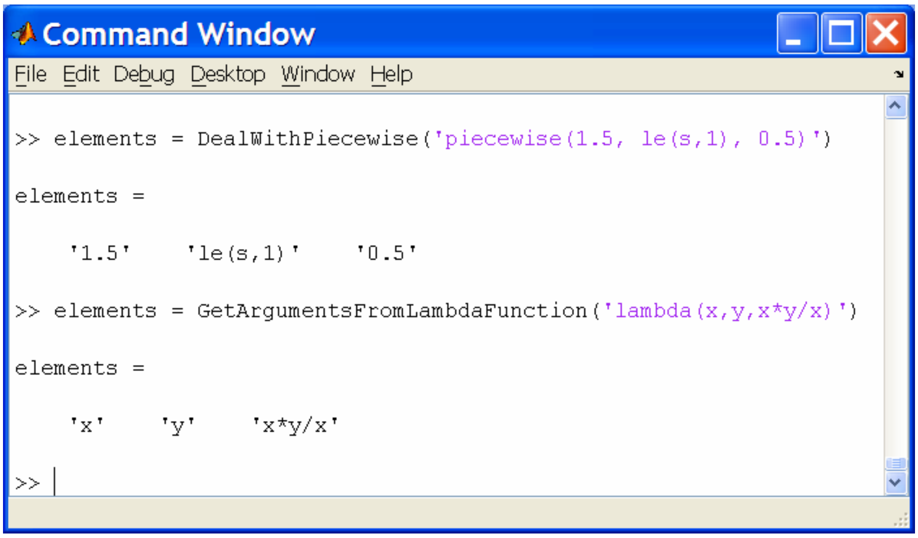


Figure 16: *Output from the MathML functions*

8.3 Other functions

8.3.1 SelectSpecies

Format	>> [species] = SelectSpecies (model)	
Argument(s)	model	MATLAB_SBML_Model structure
Returns	species	array of species selected by users
Displays	a GUI that allows the user to select species from the model	

NOTE: this function is called by DisplayODESolver and PlotSelectedTimeCourse to allow the user to output data relating to the selected species only.

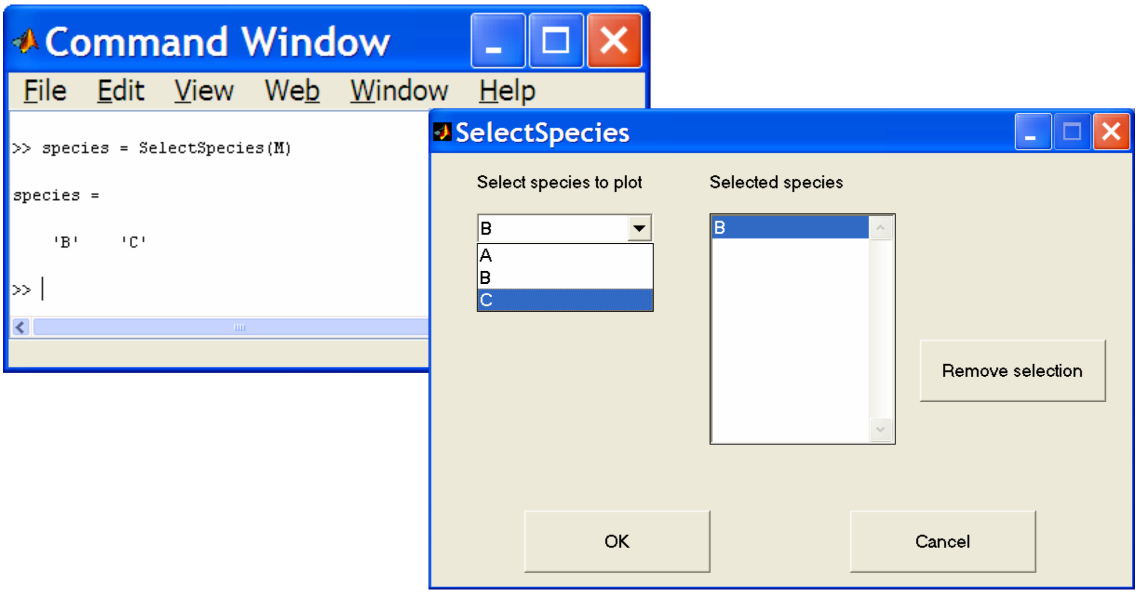


Figure 17: *Output from SelectSpecies function*

9. Storing models in MATLAB

Once a model has been imported into the MATLAB environment it is convenient to be able to store it there. MATLAB uses data files to store workspace variables and thus the MATLAB_SBML structures can be stored in such a data file. This facilitates the fast retrieval of any imported models.

The first time a model is saved the function creates a data file 'SBML_Models.mat'. Models are stored within the data file in two arrays; one containing level 1 sbml models, the other level 2 sbml models. Models are added to the appropriate array sequentially.

Functions in the StoreModels folder are listed in Table 6.

Table 6: Functions and their type in folder StoreModels

Type of function	Function name
MATLAB help	Contents.m
Save/Load functions	LoadSBMLModel.m SaveSBMLModel.m
Data file functions	ListSBMLModels.m DeleteSBMLModel.m
Graphical user functions	BrowseSBML_Models.m ViewModel.fig ViewModel.m
Sub-functions	AlreadyExists.fig AlreadyExists.m BrowseModels.fig BrowseModels.m

9.1 Saving and loading functions

9.1.1 SaveSBMLModel

Format >> SaveSBMLModel(model)
Argument(s) model MATLAB_SBML_Model structure

Saves model to the data file SBMLModels.mat

performing the following:

- validates the input structure SBMLModel
- checks whether SBMLModels.mat exists and creates it if not
- checks whether a model with same name/id is already saved and prompts user for permission to add this model as well
- adds the model as the next element of the level 1 or level 2 array
- saves SBMLModels.mat

9.1.2 LoadSBMLModel

Format	>> model = LoadSBMLModel(inputArg, SBMLlevel)	
Argument(s)	inputArg	a number representing the index of the model in the data file OR a string representing the name/id of the model
	SBMLlevel	SBML level of model to be retrieved
Returns	model	MATLAB_SBML_Model structure of SBMLlevel from data file

Note: if more than one model of the same name exists LoadSBMLModel(name, level) returns the first model that matches the name.

9.2 Data file functions

9.2.1 ListSBMLModel

Format >> ListSBMLModels

prints a list of the elements in SBMLModels.mat detailing the index number, the sbml level and the name of each model stored in the data file.

Example:

NUMBER	LEVEL	NAME
1	1	Branch
2	1	ODE
1	2	Branch
2	2	Oscillator

Obviously as the number of models stored increases this is not the most productive method for keeping track of the contents of the data file. Thus a graphical user interface that will browse the data file is also available (see BrowseSBML_Models below).

9.2.2 DeleteSBMLModel

Format	>> DeleteSBMLModel(inputArg, SBMLlevel)	
Argument(s)	inputArg	a number representing the index of the model in the data file OR a string representing the name/id of the model
	SBMLlevel	SBML level of model to be retrieved

deletes a MATLAB_SBMLModel of **SBMLlevel** from the data file SBMLModels.mat

Note: if more than one model of the same name exists DeleteSBMLModel(name, level) deletes the first model that matches the name.

9.3 Graphical user functions

9.3.1 BrowseSBML_Models

Format	>> optionalOutput = BrowseSBML_Models	
Returns	model	MATLAB_SBML_Model structure
Displays	a GUI that details the contents of the SBMLModels data file	

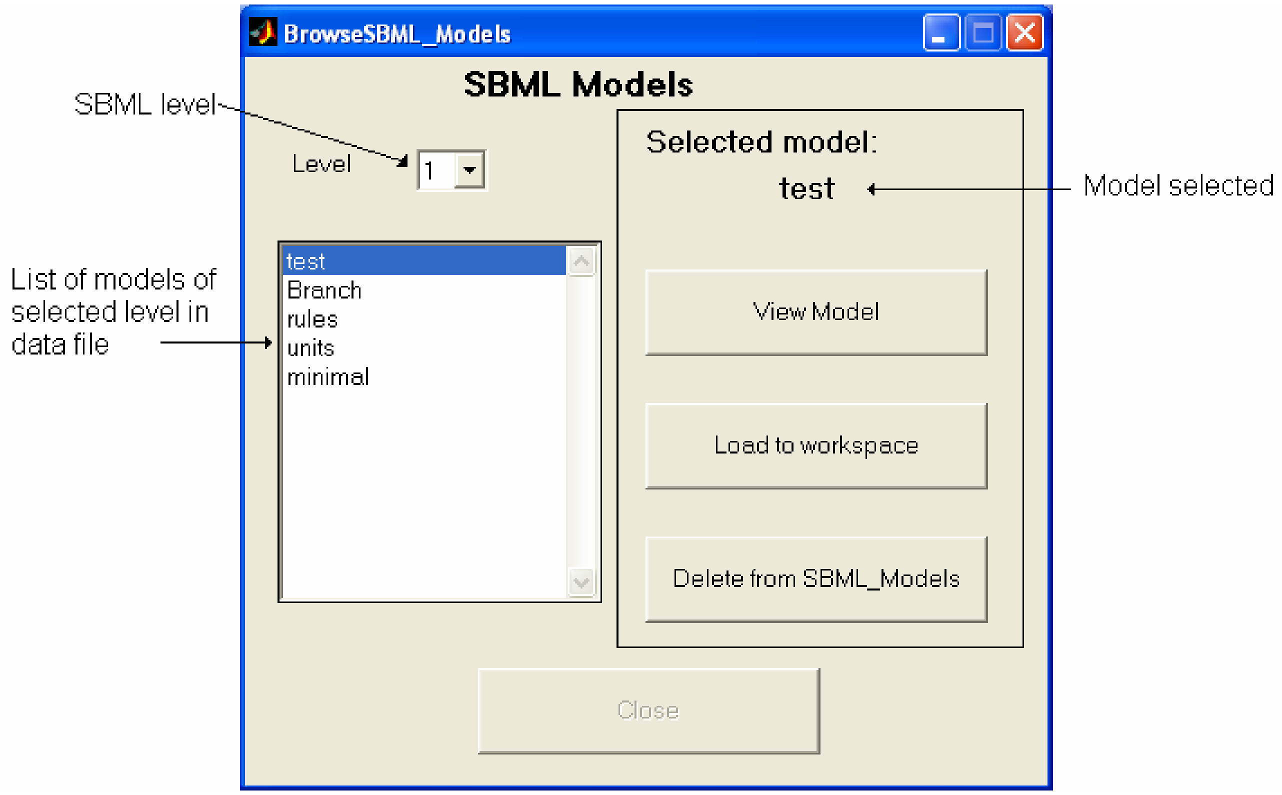


***Figure 18**: Screenshot of the BrowseSBML_Models GUI.*

View Model button activates a further GUI to view details of the model (see ViewModel below).

Load to workspace button is only active if the function has been called with an output argument, otherwise it is greyed out. Once pressed this button loads the selected model to the output argument, becomes inactive and the *Close* button becomes active.

Delete from SBML_Models button deletes the selected model from the data file.

Close button closes the window and if a model has been loaded returns the model to the workspace as the output argument.

9.3.2 ViewModel

Format >> ViewModel(model)
Argument(s) model MATLAB_SBML_Model structure
Displays a GUI that details the model

NOTE: This function is located in this directory as it also provides an alternative means of saving the displayed model to the SBMLModels data file.

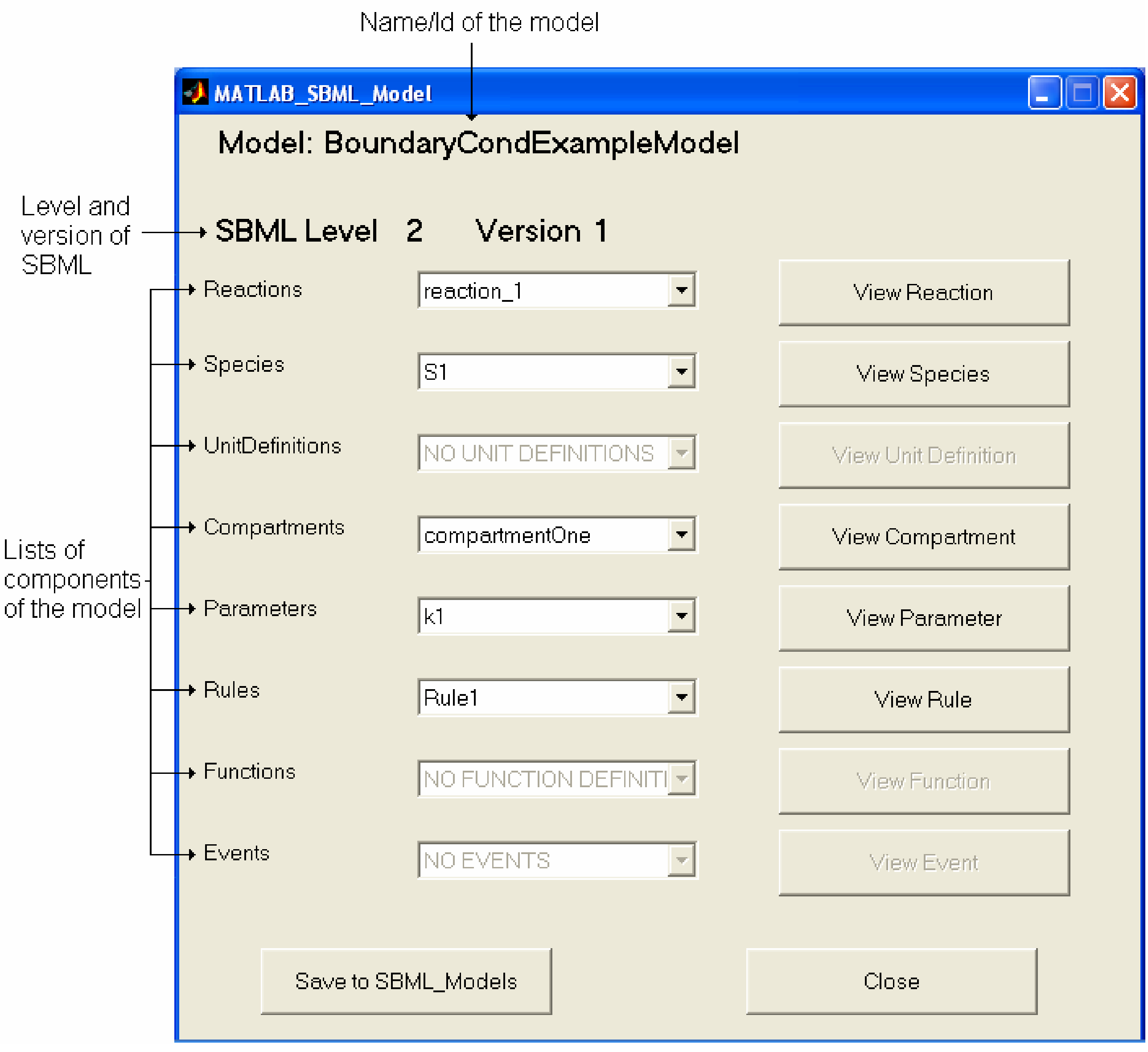


***Figure 19**: Screenshot of the ViewModel GUI*

ViewComponent buttons display further GUIs that detail the component selected. These buttons are greyed if the model does not contain any of the relevant component.

Save to SBML_Models button saves the model to the SBMLModels data file.

Close button closes the window

10. Validate_MATLAB_SBML_Structures

Each of the validation tests checks that the structure supplied as argument is of the appropriate form to represent the intended element of a sbml model.

10.1 isSBML_Model

Format >> y = isSBML_Model(model)
Argument(s) model MATLAB_SBML_Model structure

returns y = 1 if model

- is a MATLAB structure type
- has each of the fields listed in Table 7 (appropriate to the level of SBML)
- any fields that are arrays of structures are of appropriate structure
- has the value 'SBML_MODEL' in the **typecode** field.

returns y = 0 otherwise.

Table 7: MATLAB_SBML Structure for a sbml model

Fieldname	Type C	Type MATLAB
typecode	char *	mxArray of char
notes	char *	mxArray of char
annotation	char *	mxArray of char
name	char *	mxArray of char
level	unsigned int	mxArray of int32
version	unsigned int	mxArray of int32
unitDefinition	List of structures	array of structures of type UnitDefinition
compartment	List of structures	array of structures of type Compartment
species	List of structures	array of structures of type Species
parameter	List of structures	array of structures of type Parameter
rule	List of structures	array of structures of type Rule
reaction	List of structures	array of structures of type Reaction
Additional for Level 2		
id	char *	mxArray of char
functionDefinitions	List of structures	array of structures of type FunctionDefinition
event	List of structures	array of structures of type Event

10.2 isSBML_XXX

Format		>> y = isSBML_XXX(structure, SBML_Level)
Argument(s)	structure	MATLAB_SBML_XXX structure
	SBML_Level	the SBML level of the structure

returns y = 1 if structure

- is a MATLAB structure type
- has each of the fields listed in Table 8 for a elemnt XXX (appropriate to the level supplied as input argument SBML_Level)
- any fields that are arrays of structures are of appropriate structure
- does not contain any additional fields and
- has the appropriate value in the **typecode** field (see Table 9)

returns y = 0 otherwise.

Table 8: Fields contained in the MATLAB_SBML structure defining each sbml component

	Compartment	Event	Event Assignment	Function Definition	Kinetic Law	Modifier Species Reference	Parameter	Reaction	Rule	Species	Species Reference	Unit	Unit Definition
annotation	X	L2	L2	L2	X	L2	X	X	X	X	X	X	X
boundaryCondition										X			
Charge										X			
compartment									X	X			
constant	L2						L2			L2			
delay		L2											
denominator											X		
eventAssignment		L2											
exponent												X	
fast								X					
formula					X				X				
hasOnlySubstanceUnits										L2			
id	L2	L2		L2			L2	L2		L2			L2
initialAmount										X			
initialConcentration										L2			
isSetCharge										X			
isSetFast								L2					
isSetInitialAmount										X			
isSetInitialConcentration										L2			
isSetSize	L2												

Table 8: Fields contained in the MATLAB_SBML structure defining each sbml component

	Compartment	Event	Event Assignment	Function Definition	Kinetic Law	Modifier Species Reference	Parameter	Reaction	Rule	Species	Species Reference	Unit	Unit Definition
L1 – level 1 ONLY		L2 – level 2 ONLY								X – both level 1 & 2			
isSetValue							X						
isSetVolume	X												
kind												X	
kineticLaw								X					
math			L2	L2	L2								
modifier								L2					
multiplier												L2	
name	X	L2		L2			X	X	X	X			X
notes	X	L2	L2	L2	X	L2	X	X	X	X	X	X	X
offset												L2	
outside	X												
parameter					X								
product								X					
reactant								X					
reversible								X					
scale												X	
size	L2												
spatialdimensions	L2												
spatialSizeUnits										L2			
species						L2			X		X		
stoichiometry											X		
stoichiometryMath											L2		
substanceUnits					X					L2			
timeUnits		L2			X								
trigger		L2											
typecode	X	L2	L2	L2	X	L2	X	X	X	X	X	X	X
units	X						X		X	L1			X
value							X						
variable			L2						X				
volume	L1												
L1 – level 1 ONLY		L2 – level 2 ONLY								X – both level 1 & 2			

Table 9: Components in sbml model and appropriate typecode value

Component XXX	typecode
Compartment	SBML_COMPARTMENT
Event	SBML_EVENT
EventAssignment	SBML_EVENT_ASSIGNMENT
FunctionDefinition	SBML_FUNCTION_DEFINITION
KineticLaw	SBML_KINETIC_LAW
ModifierSpeciesReference	SBML_MODIFIER_SPECIES_REFERENCE
Parameter	SBML_PARAMETER
Reaction	SBML_REACTION
Rule	SBML_ALGEBRAIC_RULE
	SBML_SPECIES_CONCENTRATION_RULE
	SBML_COMPARTMENT_VOLUME_RULE
	SBML_PARAMETER_RULE
	SBML_ASSIGNMENT_RULE
	SBML_RATE_RULE
Species	SBML_SPECIES
SpeciesReference	SBML_SPECIES_REFERENCE
Unit	SBML_UNIT
UnitDefinition	SBML_UNIT_DEFINITION

Note: A rule defined by a sbml model may have a number of different types. In order to facilitate the inclusion of rules within the MATLAB_SBML structure any rule structure has the same fields, some of which will be empty depending on the rule type.

11. Viewing models in MATLAB

The SBMLToolbox provides a set of graphics that allow the full definition of the model to be displayed. The ViewModel function was discussed in Section 9.3.2. This GUI (Figure 19) has a range of buttons that allow the sub-structures of the model to be viewed as further GUIs, e.g. the ViewSpecies button brings up a GUI that details the species selected or the ViewRule button brings up a GUI that details the rule selected etc…(Figure 20).

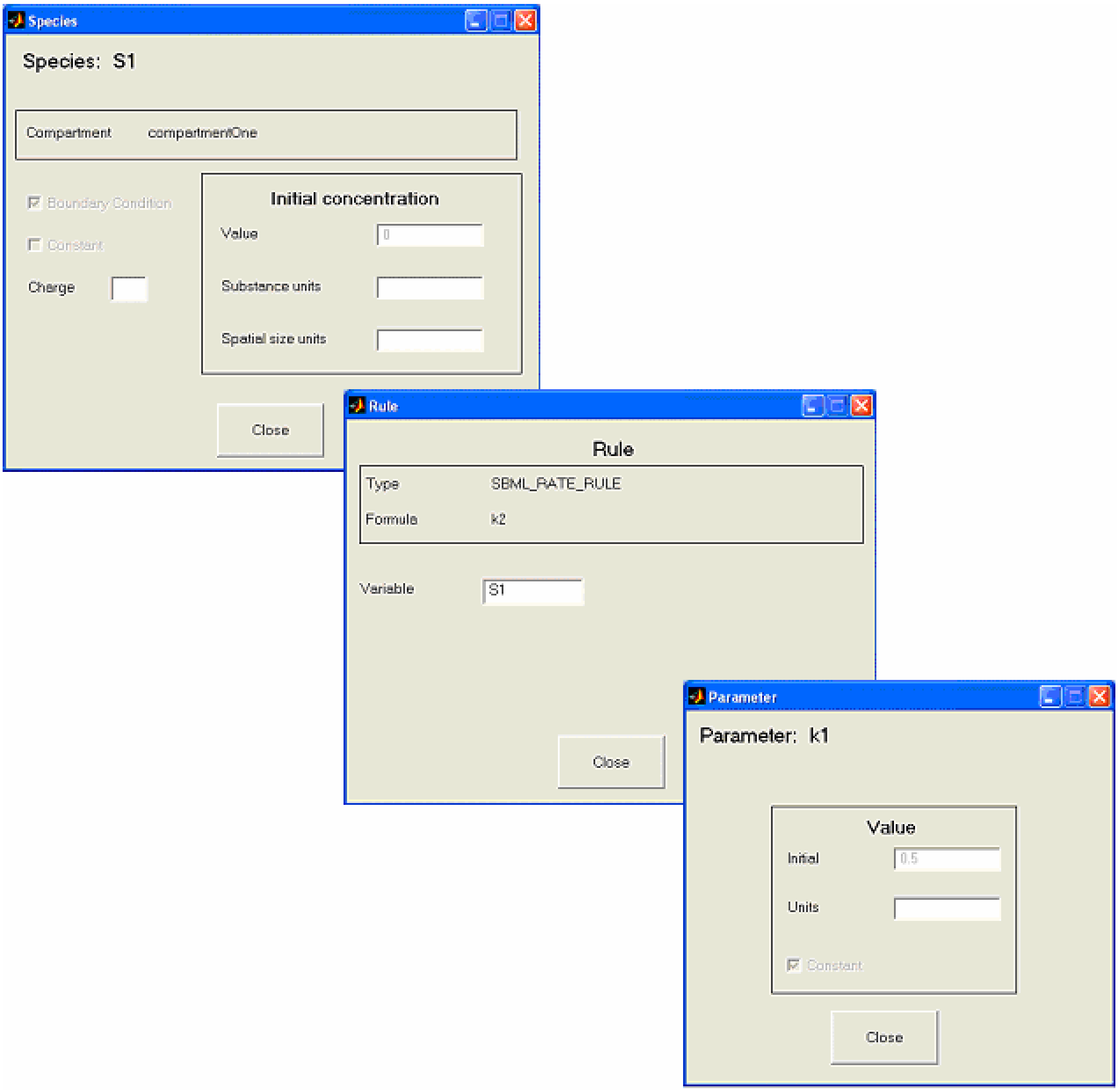


***Figure 20**: Screenshot of the ViewSpecies, ViewRule & ViewParameter GUIs*

Known issues

1. C compilers in windows
The default MATLAB C compiler is lcc. Unfortunately this fails to link to libSBML. You can change the default C compiler used by MATLAB to another C compiler installed on your system by tying ‘mex –setup’ at the MATLAB command prompt and following the instructions.

Using Microsoft VC compilers have proved most reliable.

2. C compilers in linux
There are similar problems with some configurations of linux.