

SBMLToolbox

4 MATLAB

User's manual

Sarah 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.....	3
2. Installation	3
2.1 Windows	3
2.2 Linux	4
3. Contents of SBMLToolbox download	5
4. Importing and exporting SBML	6
5. Validation tests.....	7
5.1 isSBML_Model	7
5.2 isSBML_Compartment	8
5.3 isSBML_XXX	10
6. Storing models in MATLAB	11
6.1 Saving and loading functions	11
6.2 SBMLModels data file functions	12
6.3 Graphical user functions.....	13
7. Viewing models in MATLAB.....	15
7.1 Viewing individual components	16
8. Access model	17
8.1 Getting information functions	17
8.2 Deriving information functions	19
8.3 Overview of model functions.....	20
9. Access to symbols	21
9.1 Getting symbols functions	21
9.2 Getting formula functions	24
9.3 Deriving information functions	25
9.4 Overview of model functions.....	26
9.5 General functions	29
10. Simulation.....	32

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 under development, development that is primarily driven by the goal of providing a user with a biological model access to MATLAB functionality and other tools that have been developed to function within MATLAB.

2. Installation

2.1 Windows

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

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

- 1) Adds this folder (SBMLToolbox/src) 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/src and typing 'install'.

Note: The executable TranslateSBML.dll is included with the download and therefore it is not necessary to build it prior to use. However TranslateSBML.dll can be built within MATLAB by typing 'BuildTranslate_Win32'. Unfortunately some C compilers running through MATLAB fail to link to libsbml and the build will fail. You can change the default C compiler used by MATLAB to any C compiler installed on your system by typing 'mex -setup' at the MATLAB command prompt and following the instructions.

2.2 Linux

To build:

- 1) Change to the directory 'SBMLToolbox/src'.
- 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 LDFLAGS=-L/usr/local/lib
```

In csh or tcsh:

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

- 4) Type 'make'

This should build TranslateSBML.mexglx.

To run:

Ensure the directory containing TranslateSBML.mexglx and the StoreModels, ValidationTexts subdirectories etc... are in your Matlab path. For example, at the Matlab prompt:

```
>> addpath('SBMLToolbox/src');
>> addpath('SBMLToolbox/src/StoreModels');
>> addpath('SBMLToolbox/src/ValidationTests');
>> addpath('SBMLToolbox/src/ViewModelComponents');
>> addpath('SBMLToolbox/src/AccessToSymbols');
```

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

3. Contents of SBMLToolbox download

The SBMLToolbox download includes the following files:

SBMLToolbox

/docs

MATLAB_SBML_Structure.pdf
Manual_SBMLToolbox.pdf

/src

Makefile	// Linux installation
make.bat	// Windows installation
install.m	// Windows installation
BuildTranslate_Win32	// Windows installation
BuildOutput_Win32	// Windows installation
Contents.m	// for MATLAB help
TranslateSBML.m	// for MATLAB help
OutputSBML.m	// for MATLAB help
TranslateSBML.c	// source code
OutputSBML.c	// source code
/AccessModel	// functions that derive information from model
/AccessToSymbols	// functions for conversion to symbols
/Simulation	// functions for producing files for simulation
/StoreModels	// functions for saving/loading models
/ValidationTests	// functions for validating structure
/ViewModelComponents	// functions for GUIs to view model

4. Importing and exporting SBML

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.

This returns a MATLAB_SBML structure named Model within the MATLAB environment. This command and the expected MATLAB response are illustrated in 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.

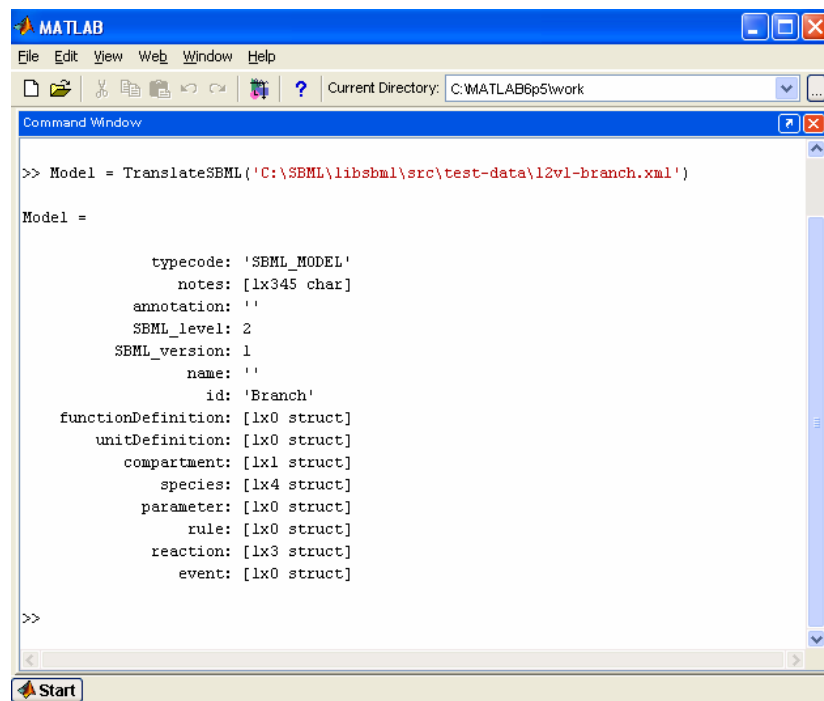


Figure 1: Screenshot of the command `Model = TranslateSBML('../branch.xml')` and the resulting MATLAB_SBML structure returned

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.

5. Validation tests

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.

5.1 isSBML_Model

y = isSBML_Model(MATLAB_SBMLStructure)

returns y = 1 if the input argument MATLAB_SBMLStructure

- is a MATLAB structure type
- has each of the fields listed in Table 1 (appropriate to the level of SBML)
- any fields that are arrays of structures are of appropriate structure
- does not contain any additional fields and
- has the value 'SBML_MODEL' in the **typecode** field.

returns y = 0 otherwise.

Table 1: MATLAB_SBML Structure for a sbml model

Fieldname	Type	
	C	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

5.2 isSBML_Compartment

```
y = isSBML_Compartment(MATLAB_SBMLStructure, SBML_Level)
```

returns $y = 1$ if the input argument `MATLAB_SBMLStructure`

- is a MATLAB structure type
- has each of the fields listed in Table 2 for a Compartment (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 value ‘`SBML_COMPARTMENT`’ in the **typecode** field.

returns $y = 0$ otherwise.

Table 2: Fields contained in the MATLAB_SBML structure defining each sbml component

[illegible]

Table 2: 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
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					

5.3 isSBML_XXX

Validation tests exist for each component of a sbml model.

y = isSBML_XXX(MATLAB_SBMLStructure, SBML_Level)

returns y = 1 if the input argument MATLAB_SBMLStructure

- is a MATLAB structure type
- has each of the fields listed in Table 2 for the component 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 (Table 3).

returns y = 0 otherwise.

Table 3: 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
Species	SBML_RATE_RULE
	SBML_SPECIES
	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.

6. 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 4.

Table 4: 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

Contents.m is a standard MATLAB help file which is typed to the command window when the command help is typed followed by a folder name. Thus 'help StoreModels' will result in the file Contents.m being displayed

6.1 Saving and loading functions

SaveSBMLModel(SBMLModel)

Saves the **SBMLModel** to the data file SBMLModels.mat

performing the following:

- validates the input argument 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

SBMLModel = LoadSBMLModel(inputArgument, SBMLlevel)

returns SBMLModel = MATLAB_SBMLModel structure with sbml level **SBMLlevel**
 from the data file SBMLModels.mat
 where **inputArgument** is a number representing the index of the model in
 the data file
 or is a string representing the name/id of the model

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

6.2 SBMLModels data file functions

ListSBMLModel

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).

DeleteSBMLModel(inputArgument, SBMLlevel)

deletes a MATLAB_SBMLModel of **SBMLlevel** from the data file SBMLModels.mat
 where **inputArgument** is a number representing the index of the model in
 the data file
 or is a string representing the name/id of the model

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

6.3 Graphical user functions

OptionalOutput = BrowseSBML_Models

displays a window that details the contents of the SBMLModels data file (see Figure 2).

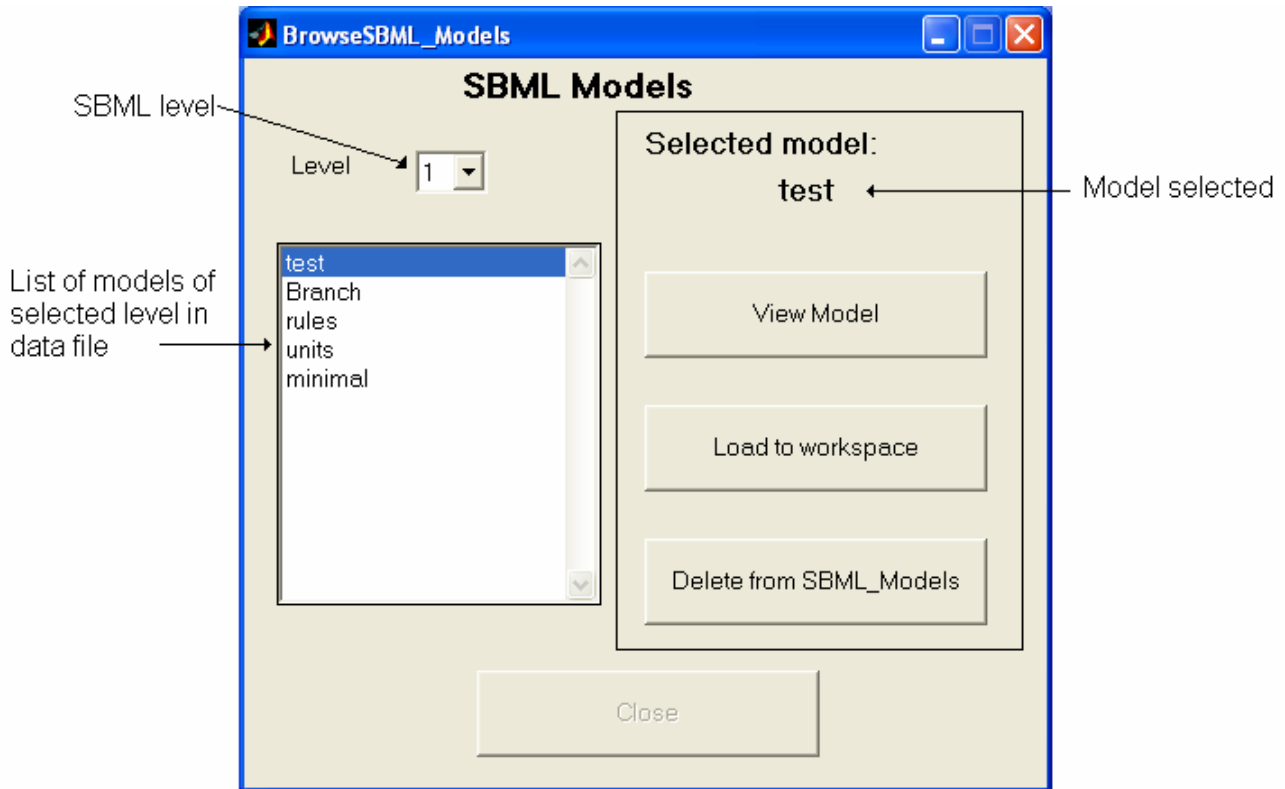


Figure 2: Screenshot of the *BrowseSBML_Models* GUI.

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.

ViewModel(MATLAB_SBMLModel)

displays a window that gives details of the MATLAB_SBMLModel (see Figure 3). This function is located in this directory as it also provides an alternative means of saving the displayed model to the SBMLModels data file.

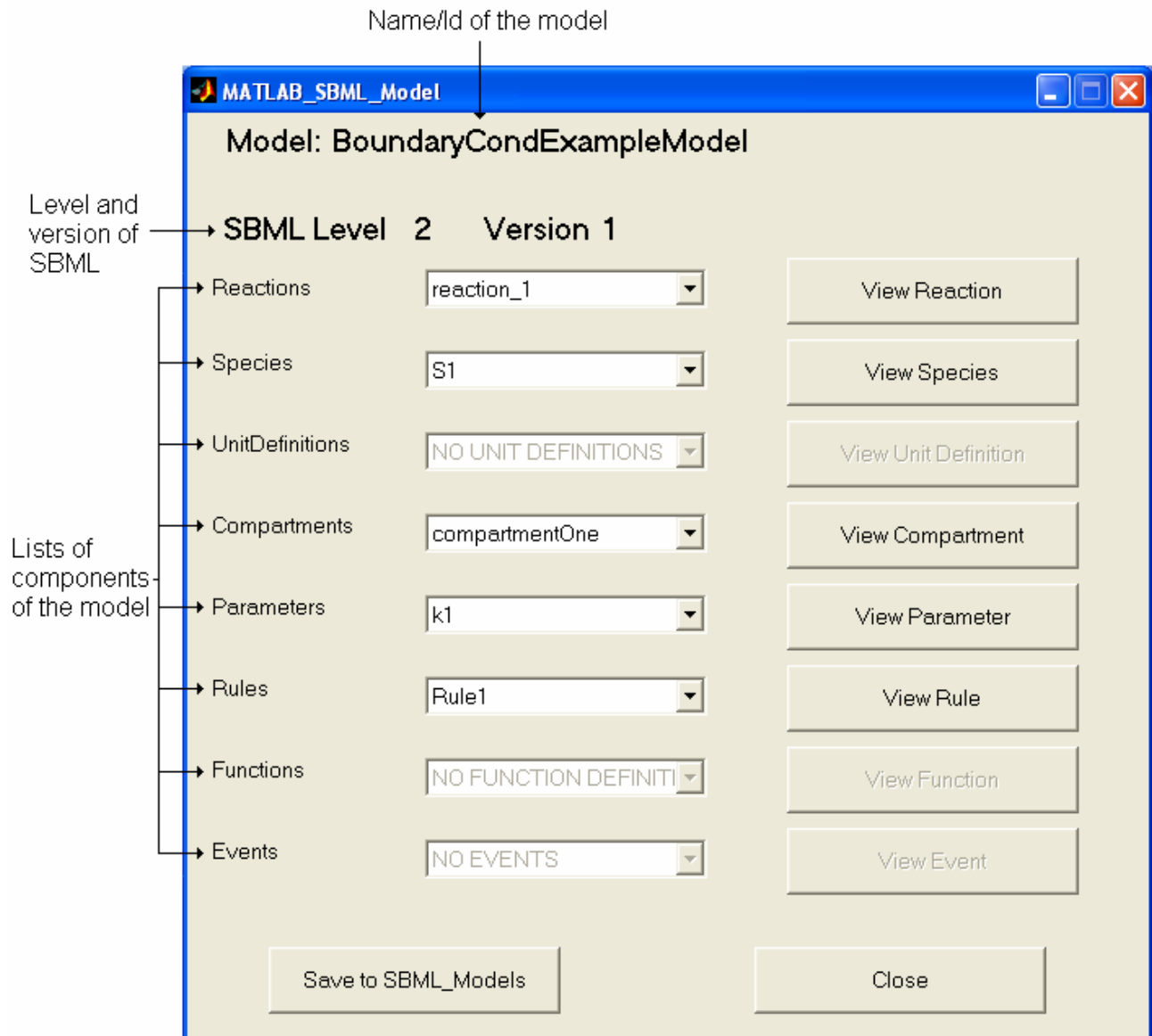


Figure 3: Screenshot of the ViewModel GUI

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.

7. 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 6.3. This GUI (Figure 3) 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 4).

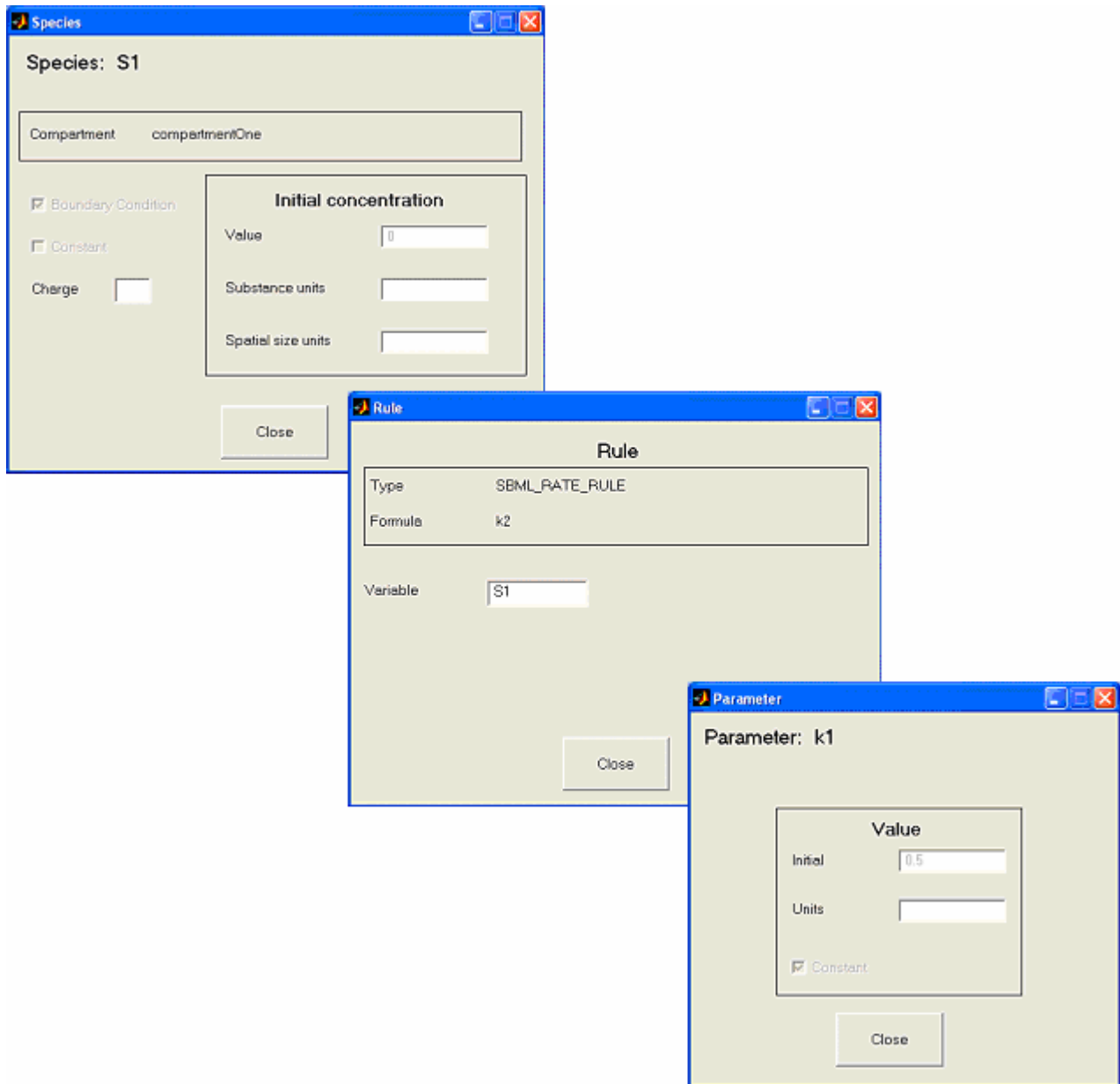


Figure 4: Screenshot of the ViewSpecies, ViewRule & ViewParameter GUIs

7.1 Viewing individual components

The MATLAB_SBML structure is a standard MATLAB structure and thus can be accessed and manipulated in the same way as any structure. Figure 5 shows examples of accessing fields within a MATLAB_SBML Model structure.

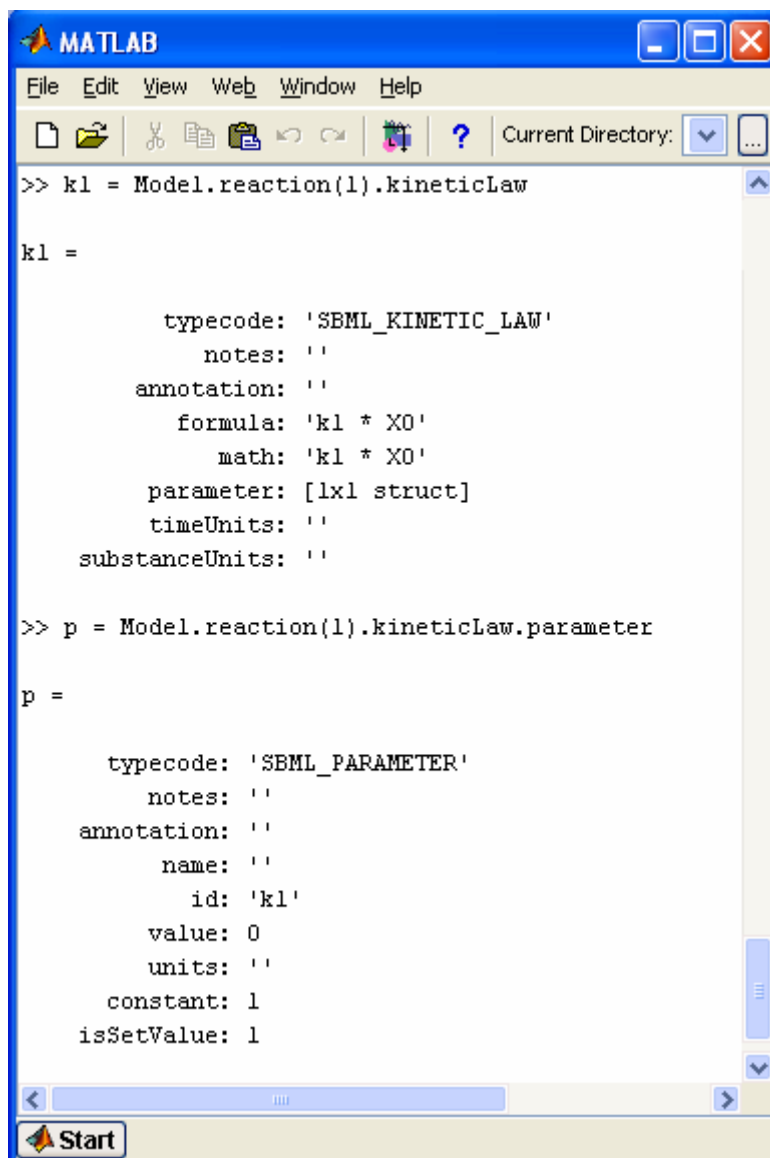


Figure 5: Accessing fields within the MATLAB_SBML Model structure.

Each component of the sbml model can be viewed independently from the ViewModel GUI using the functions in the ViewModelComponents folder. For example ViewKineticLaw(kl) would produce a GUI detailing the kineticLaw structure kl shown in Figure 5.

The variables describing the model cannot be edited using the GUIs as these have initially been developed to facilitate the visualisation of the model. However it would be a short step for a developer to enable the edit boxes should this be a requirement.

8. 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 5.

Table 5: Functions and their type in folder AccessModel

Type of function	Function name
MATLAB help	Contents.m
Getting information	GetAllParameters.m
	GetAllParametersUnique.m
	GetGlobalParameter.m
	GetParameterFromReaction.m
	GetParameterFromReactionUnique.m
	GetSpecies.m
Deriving information	GetStoichiometryMatrix.m
Overview of model	CheckValues.fig
	CheckValues.m

8.1 Getting information functions

[names, values] = GetGlobalParameters(MATLAB_SBMLModel)

returns

names = array of the character string representation of the global parameters
found in MATLAB_SBMLModel i.e. parameters in ListofParameters
(this will be the name field in level 1 SBML models
and the id field in level 2 Models)

values = array of the values of each parameter (if value is not set then -1 is entered in the array)

[names, values] = GetParameterFromReaction(MATLAB_SBMLReaction)

returns

names = array of the character string representation of the parameters
found in MATLAB_SBMLReaction
i.e. parameters in ListofParameters within the kineticLaw of a reaction

values = array of the values of each parameter (if value is not set then -1 is entered in the array)

```
[names, values] = GetParameterFromReactionUnique(MATLAB_SBMLReaction)
```

returns

names = array of the character string representation of the parameters
found in MATLAB_SBMLReaction with the name/id of the reaction appended
values = array of the values of each parameter (if value is not set then -1 is entered in the array)

This function allows a user to generate a list of unique identifiers for the parameters within a model.

```
[names, values] = GetAllParameters(MATLAB_SBMLModel)
```

returns

names = array of the character string representation of each parameter
values = array of the values of each parameter (if value is not set then -1 is entered in the array)

```
[names, values] = GetAllParametersUnique(MATLAB_SBMLModel)
```

returns

names = array of the character string representation of each parameter; with the unique names for
those parameters declared within a reaction
values = array of the values of each parameter (if value is not set then -1 is entered in the array)

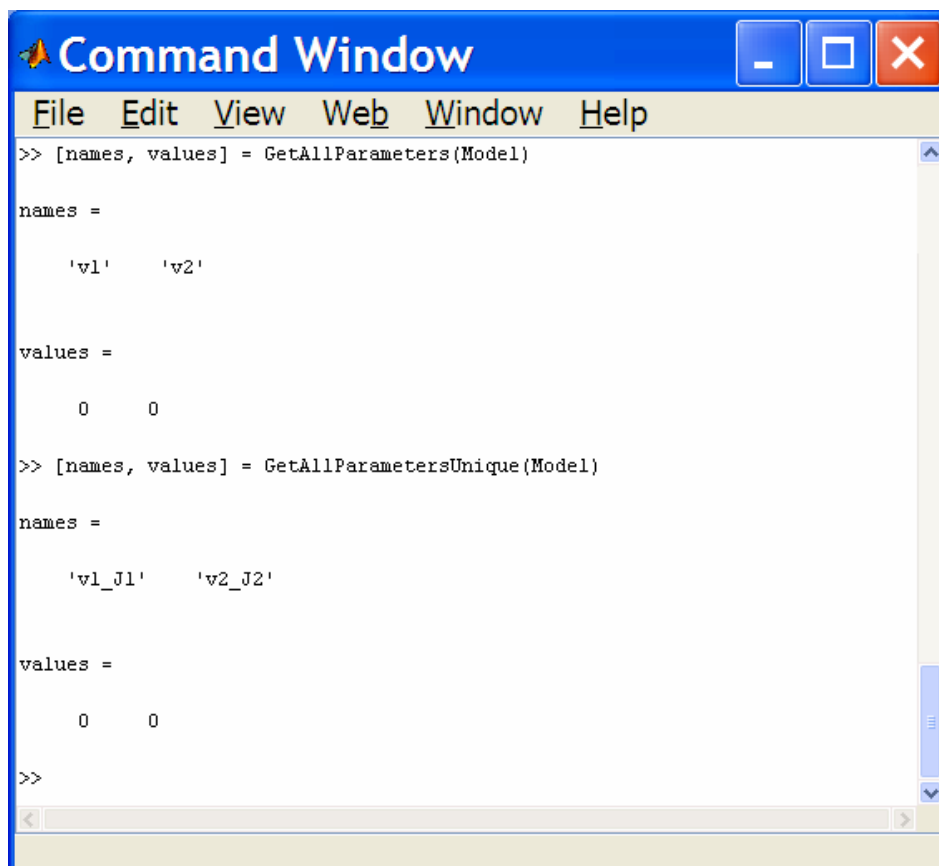


Figure 6: Using the *GetAllParameters* and the *GetAllParametersUnique* functions

```
[names, values] = GetSpecies(MATLAB_SBMLModel)
```

returns

names = array of the character string representation of all species
found in MATLAB_SBMLModel

values = array of the initial concentration/amount of each species
(if value is not set then -1 is entered in the array)

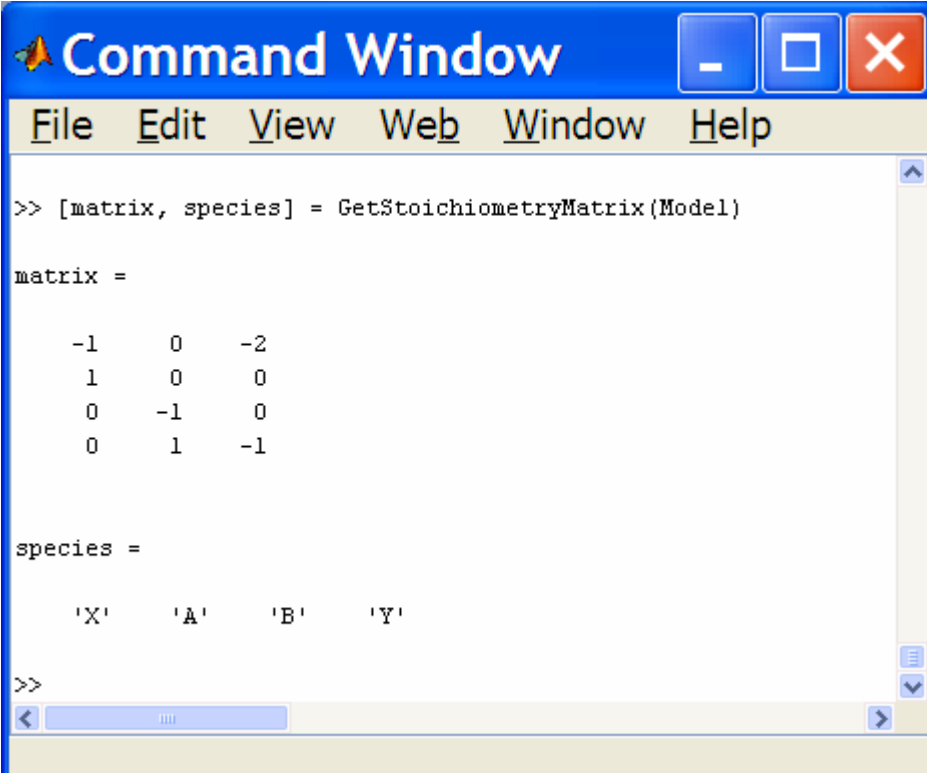
8.2 Deriving information functions

```
[matrix, species] = GetStoichiometryMatrix(MATLAB_SBMLModel)
```

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 matrix deals with them

A screenshot of the MATLAB Command Window. The title bar is blue with the MATLAB logo and the text 'Command Window'. Below the title bar is a menu bar with 'File', 'Edit', 'View', 'Web', 'Window', and 'Help'. The main area is white and contains the following text:

```
>> [matrix, species] = GetStoichiometryMatrix(Model)

matrix =

    -1     0    -2
     1     0     0
     0    -1     0
     0     1    -1

species =

    'X'    'A'    'B'    'Y'
```

The prompt '>>' is at the bottom left of the window.

Figure 7: Typical output from *GetStoichiometryMatrix*

8.3 Overview of model functions

[speciesValues, parameterValues] = CheckValues(MATLAB_SBMLModel)

displays a GUI that allows the user to check that the values for the parameters and the initial amounts of the species are as expected.

returns

speciesValues = array of values for the initial amount of each species

parameterValues = array of values for the parameters

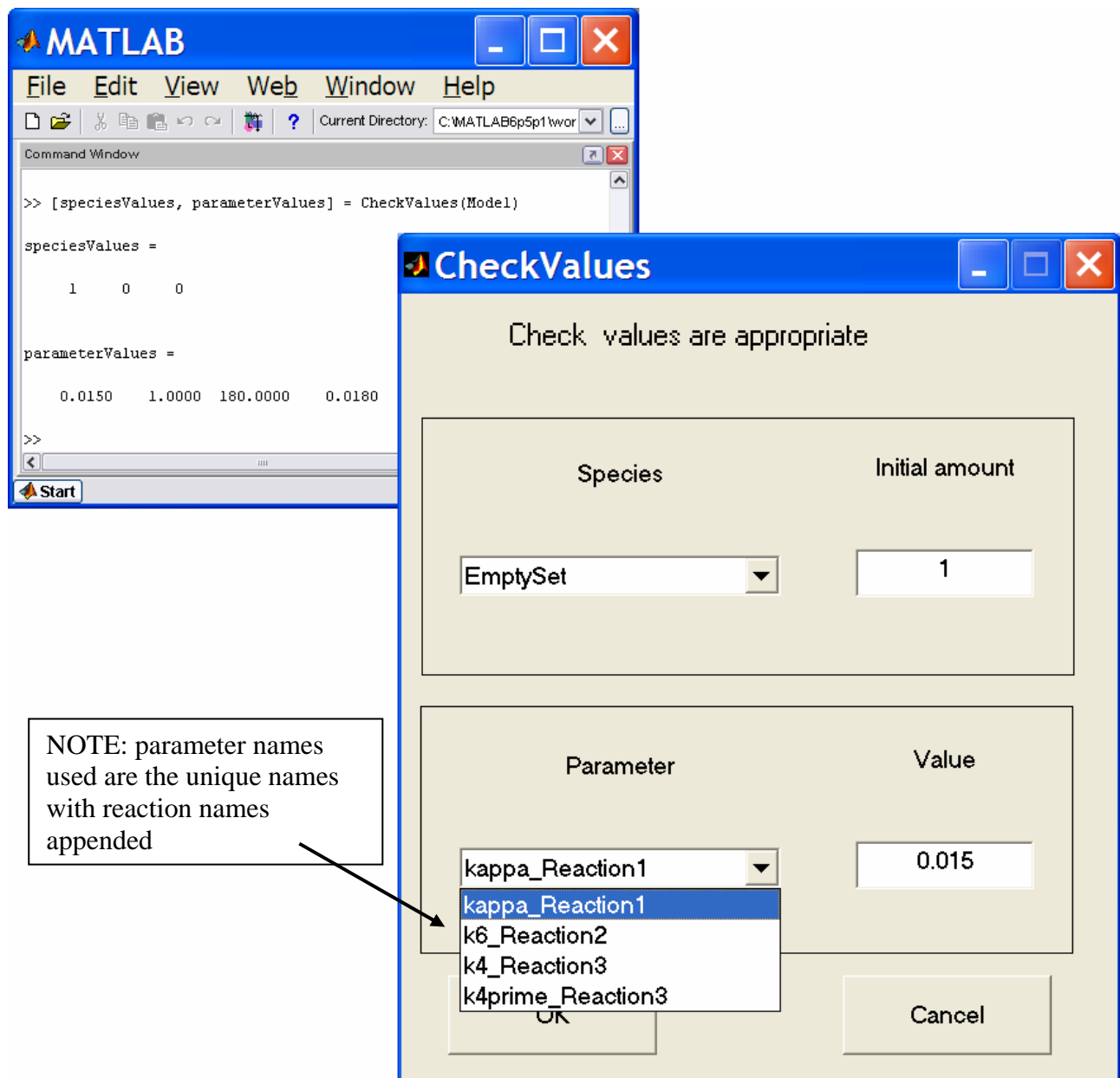


Figure 8: Typical output from CheckValues function

9. 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 6.

Table 6: Functions and their type in folder `AccessToSymbols`

Type of function	Function name
MATLAB help	<code>Contents.m</code>
Getting symbols	<code>GetAllParameterSymbols.m</code>
	<code>GetAllParameterSymbolsUnique.m</code>
	<code>GetGlobalParameterSymbols.m</code>
	<code>GetParameterSymbolsFromReaction.m</code>
	<code>GetParameterSymbolsFromReactionUnique.m</code>
	<code>GetSpeciesSymbols.m</code>
Getting formulas	<code>GetSpeciesRateLaws.m</code>
	<code>GetStoichiometryMatrixSyms.m</code>
Deriving information	<code>GetEquilibrium.m</code>
Overview of model	<code>PlotTimeCourse.m</code>
	<code>PlotSelectedTimeCourse.m</code>
General	<code>charFormula2sym.m</code>
	<code>CreateSymArray.m</code>
	<code>GetDegree.m</code>
	<code>IsSpeciesInReaction.m</code>
	<code>RemoveDuplicates.m</code>

9.1 Getting symbols functions

[symbols, values, names] = GetAllParameterSymbols(MATLAB_SBMLModel)

returns

`symbols` = array of the symbolic representation of all parameters found in `MATLAB_SBMLModel`

`values` = array of the values of each parameter (if value is not set then -1 is entered in the array)

`names` = array of the character string representation of each parameter

[symbols, values, names] = GetAllParameterSymbolsUnique(MATLAB_SBMLModel)

returns

`symbols` = array of the symbolic representation of the unique names of all parameters found in `MATLAB_SBMLModel`

`values` = array of the values of each parameter (if value is not set then -1 is entered in the array)

`names` = array of the character string representation of each parameter; with the unique names for those parameters declared within a reaction

[symbols, values, names] = GetGlobalParameterSymbols(MATLAB_SBMLModel)

returns

symbols = array of the symbolic representation of global parameters

found in MATLAB_SBMLModel i.e. parameters in ListofParameters

values = array of the values of each parameter (if value is not set then -1 is entered in the array)

names = array of the character string representation of each parameter

**[symbols, values, names] =
GetParameterSymbolsFromReaction(MATLAB_SBMLReaction)**

returns

symbols = array of the symbolic representation of parameters found in MATLAB_SBMLReaction
i.e. parameters in ListofParameters within the kineticLaw of a reaction

values = array of the values of each parameter (if value is not set then -1 is entered in the array)

names = array of the character string representation of each parameter

**[symbols, values, names] =
GetParameterSymbolsFromReactionUnique(MATLAB_SBMLReaction)**

returns

symbols = array of the symbolic representation of parameters found in MATLAB_SBMLReaction
with the name/id of the reaction appended

values = array of the values of each parameter (if value is not set then -1 is entered in the array)

names = array of the character string representation of each parameter; with the name/id of the
reaction appended

[symbols, values, names] = GetSpeciesSymbols(MATLAB_SBMLModel)

returns

symbols = array of the symbolic representation of all species found in MATLAB_SBMLModel

values = array of the initial concentration/amount of each species
(if value is not set then -1 is entered in the array)

names = array of the character string representation of each species

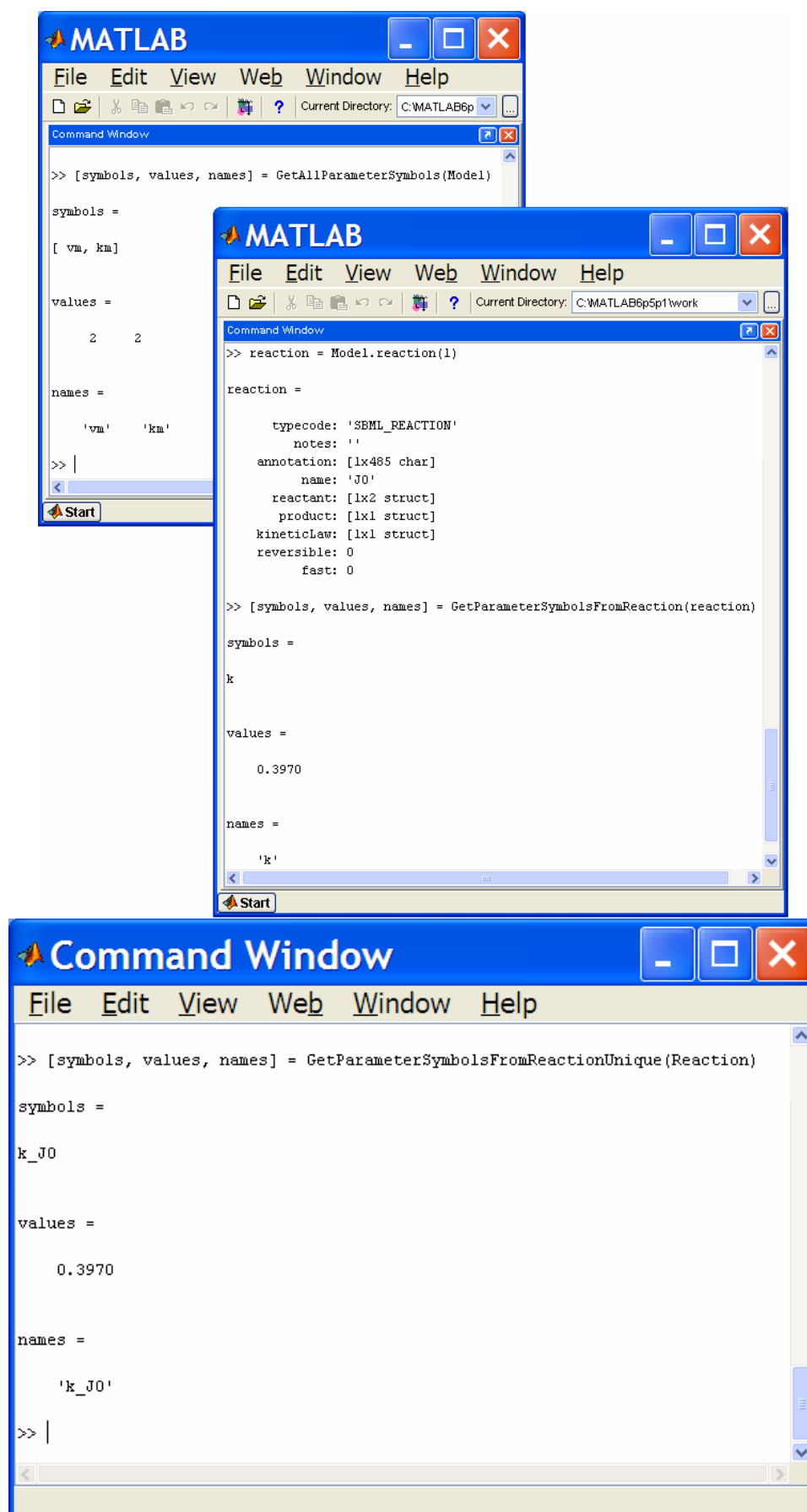


Figure 9: Examples of output from Getting Symbols functions

9.2 Getting formula functions

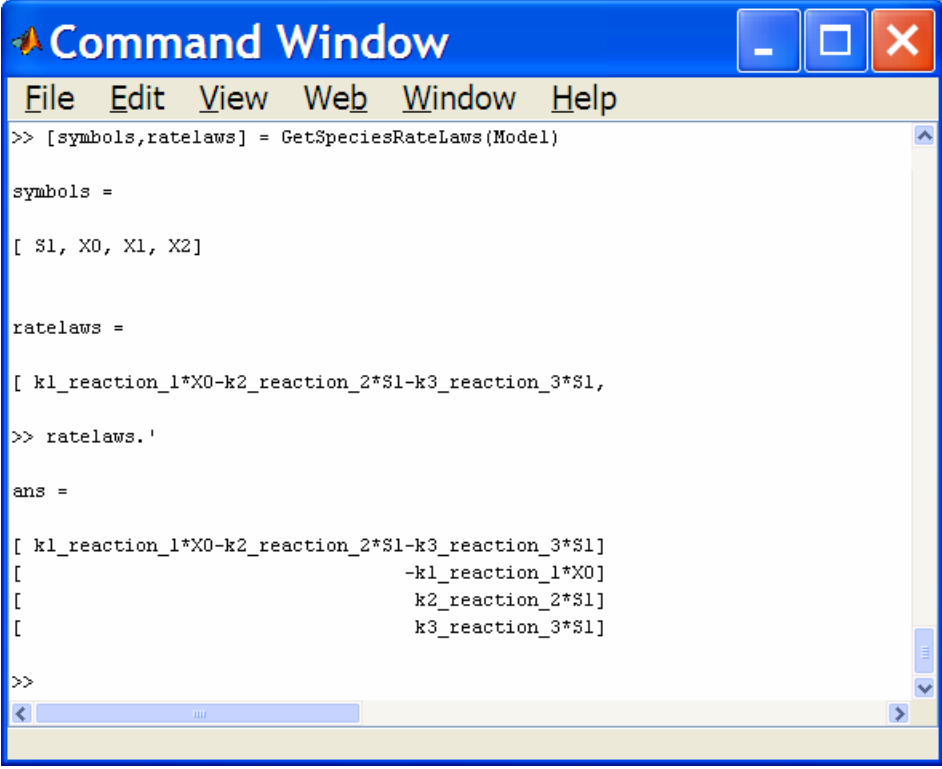
[symbols, ratelaws] = GetSpeciesRateLaws(MATLAB_SBMLModel)

returns

symbols = array of the symbolic representation of all species found in MATLAB_SBMLModel

ratelaws = array of the symbolic representation for the rate law for each species

NOTE: GetSpeciesRateLaws uses the unique parameter names



```

Command Window
File Edit View Web Window Help
>> [symbols, ratelaws] = GetSpeciesRateLaws(Model)

symbols =

[ S1, X0, X1, X2]

ratelaws =

[ k1_reaction_1*X0-k2_reaction_2*S1-k3_reaction_3*S1,

>> ratelaws.'

ans =

[ k1_reaction_1*X0-k2_reaction_2*S1-k3_reaction_3*S1
  -k1_reaction_1*X0
  k2_reaction_2*S1
  k3_reaction_3*S1]

>>

```

Figure 10: Output from GetSpeciesRateLaws function

[matrix, species] = GetStoichiometryMatrixSyms(MATLAB_SBMLModel)

returns

matrix = stoichiometry matrix for the species and reactions in the model

species = array of the symbolic representation of all species
in the order in which the matrix deals with them

9.3 Deriving information functions

[symbols, values, names] = GetAllParameterSymbols(MATLAB_SBMLModel)

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
(equal to 0 if equilibrium not reached)

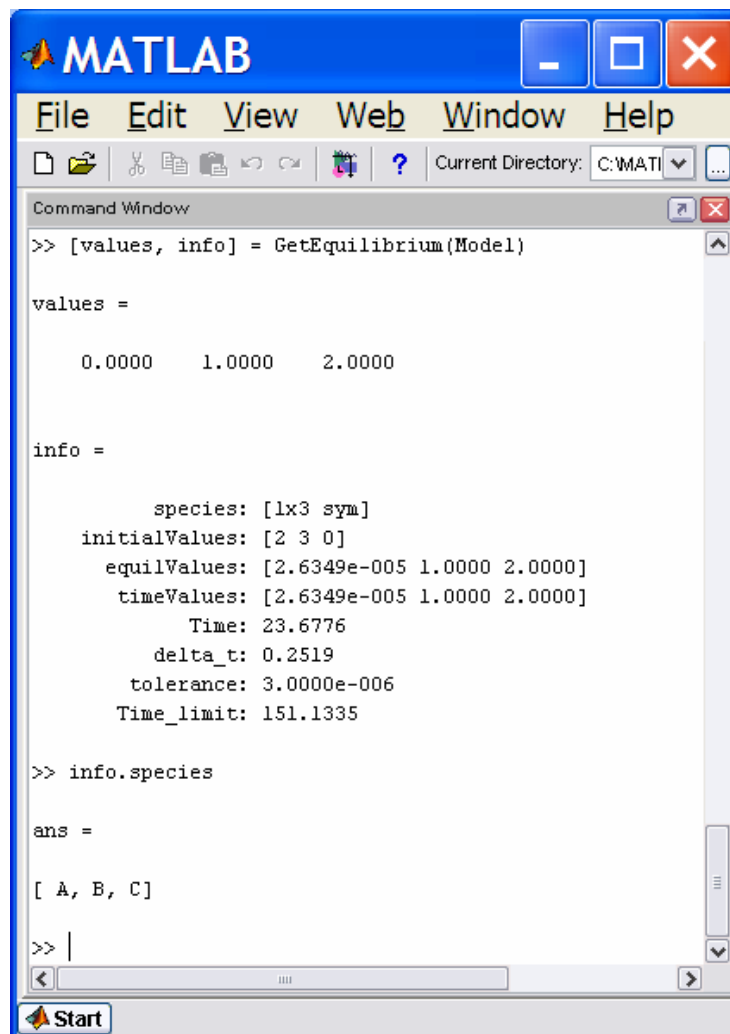
.timeValues – array of the amount of each species at the time shown
(equal to equilValues if equilibrium was reached)

.Time – elapsed time for which the reactions were considered

.delta_t – time step used in the calculations

.tolerance – difference value at which equilibrium was considered to be reached

.Time_limit – length of time reactions would be considered if tolerance not reached



```

MATLAB
File Edit View Web Window Help
Current Directory: C:\MATLAB
Command Window
>> [values, info] = GetEquilibrium(Model)

values =

    0.0000    1.0000    2.0000

info =

    species: [1x3 sym]
  initialValues: [2 3 0]
    equilValues: [2.6349e-005 1.0000 2.0000]
    timeValues: [2.6349e-005 1.0000 2.0000]
          Time: 23.6776
    delta_t: 0.2519
    tolerance: 3.0000e-006
    Time_limit: 151.1335

>> info.species

ans =

[ A, B, C]

>>

```

Figure 11: 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 $A \rightarrow B$ with kinetic law formula $k \cdot B$

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.

9.4 Overview of model functions

[symbols, values, names] = GetAllParameterSymbols(MATLAB_SBMLModel)

PlotTimeCourse uses the same algorithm as that used to calculate equilibrium and displays a plot of the time course for each of the species within the model (see Figure 12 overleaf).

Optionally takes a second argument `time_limit` which limits the length of time the reactions are considered

Returns

`values` = array of species amounts at the end of the plot time
(either at equilibrium or `time_limit` if this has been specified)

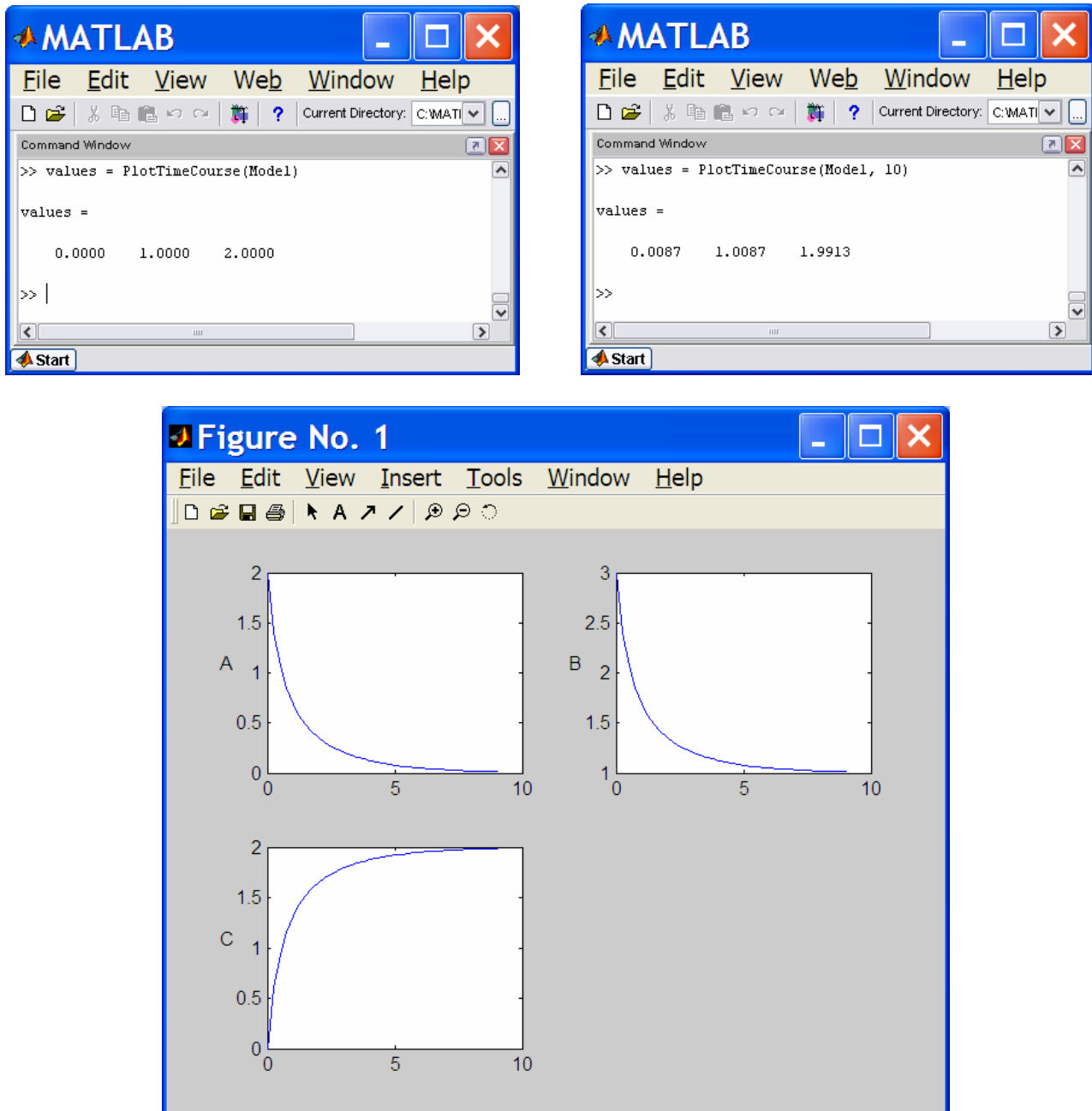


Figure 12: Examples of the output from `PlotTimeCourse` function

Note: The functions `GetEquilibrium`, `PlotTimeCourse` and `PlotSelectedTimeCourse` all call the `CheckValues` function (discussed in section 8.3) with additional error trapping for inappropriate zero or negative values.

```
[values] = PlotSelectedTimeCourse(MATLAB_SBMLModel, (time_limit))
```

PlotSelectedTimeCourse is a variation on PlotTimeCourse where the user is prompted to select the species for which the time course is to be plotted.

All selected species are plotted using the same axes with up to 14 different line types available (Figure 13).

The arguments and returns are identical to those for PlotTimeCourse.

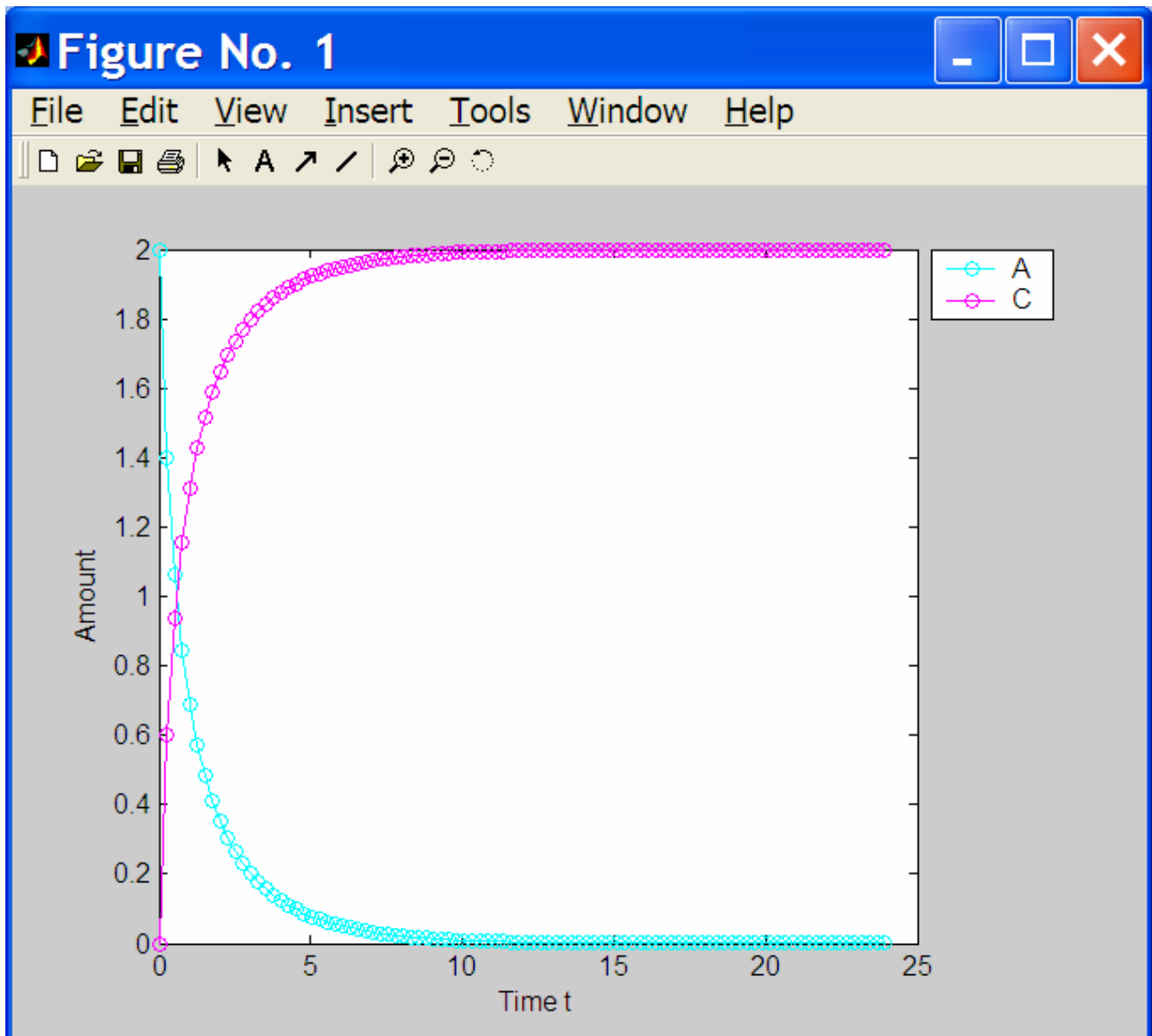


Figure 13: Output from PlotSelectedTimeCourse function

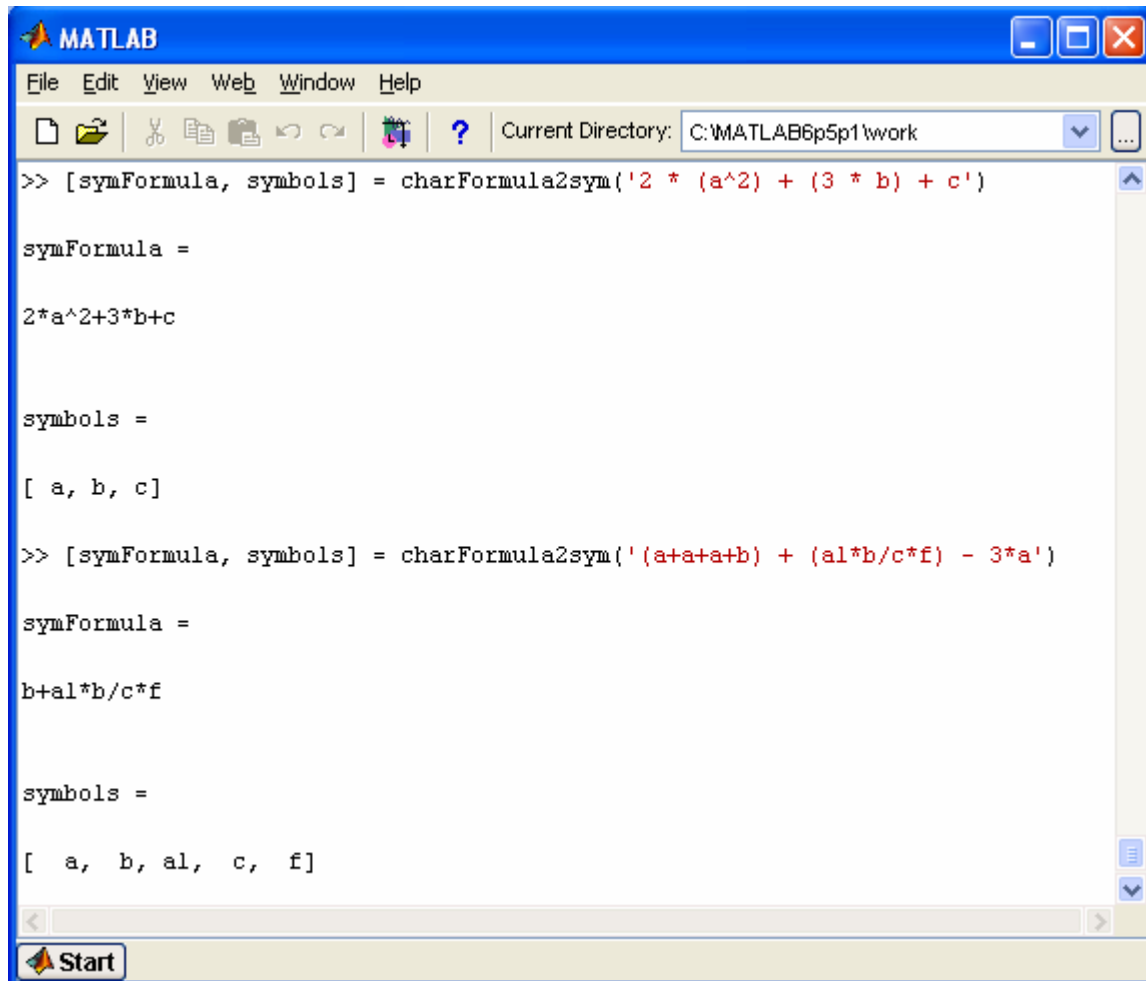
9.5 General functions

`[symFormula, symbols] = charFormula2sym(charFormula)`

returns

`symFormula` = symbolic representation of `charFormula`

`symbols` = array of the symbols used in the formula



The image shows a MATLAB command window with the following content:

```
MATLAB
File Edit View Web Window Help
Current Directory: C:\MATLAB6p5p1\work

>> [symFormula, symbols] = charFormula2sym('2 * (a^2) + (3 * b) + c')

symFormula =

2*a^2+3*b+c

symbols =

[ a, b, c]

>> [symFormula, symbols] = charFormula2sym('(a+a+a+b) + (a1*b/c*f) - 3*a')

symFormula =

b+a1*b/c*f

symbols =

[ a, b, a1, c, f]
```

Figure 14: Sample outputs from `charFormula2sym` function

[symbols] = CreateSymArray(symFormula)

takes a symbolic formula or set of formulas

returns

symbols = array of individual symbols found in symFormula

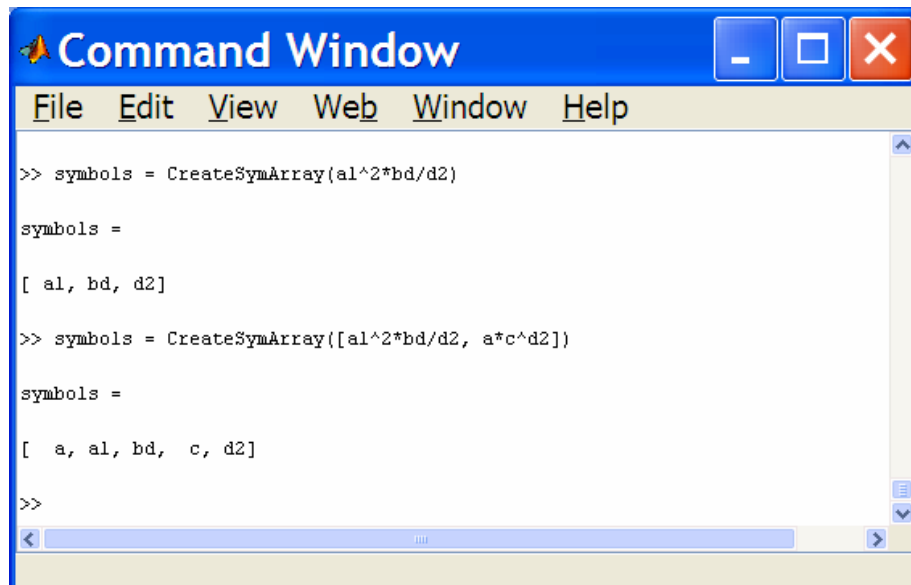


Figure 15: Output from CreateSymArray function

degree = GetDegree(symPolynomial, symVariable)

takes

symPolynomial – a symbolic representation of a polynomial

symVariable – a single symbol

returns

degree = the degree of the single symbol in the polynomial

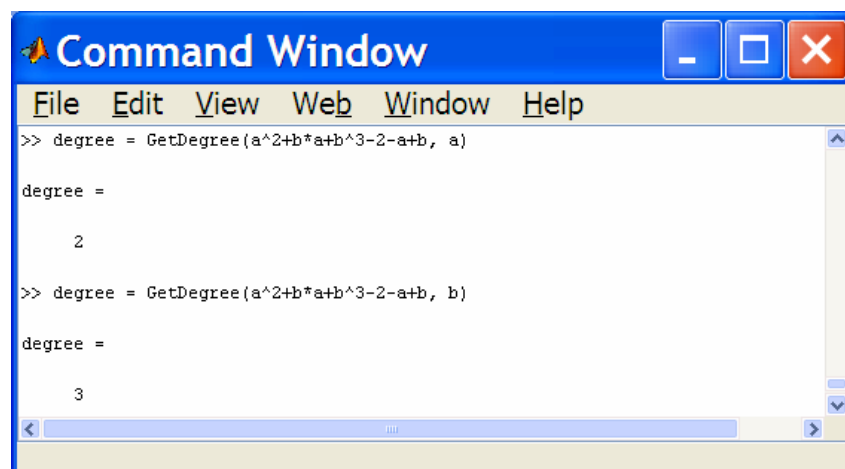


Figure 16: Output from GetDegree function

`y = IsSpeciesInReaction(MATLAB_SBMLSpecies, MATLAB_SBMLReaction)`

returns

`y = 0` if the species given is not in the reaction given

or an array indicating

number of occurrences of species

whether the species is a reactant

stoichiometry if the species is a reactant

whether the species is a product

stoichiometry if the species is a product

whether the species is a modifier

Thus the returned array has the form

[number of occurrence, product_info, reactant_info, modifier]

where product_info or reactant_info may be two numbers if applicable.

Table 7 illustrates some examples.

Table 7: Example outputs from IsSpeciesInReaction function

Output = IsSpeciesInReaction(s, r)	Interpretation
0	species s is not in reaction r
[1,1,2,0,0]	species s is a reactant with stoichiometry 2 $2s + \dots \rightarrow \dots$
[1,0,1,3,0]	species s is a product with stoichiometry 3 $\dots \rightarrow 3s + \dots$
[1,0,0,1]	species s is a modifier in reaction r
[2,1,1,2,0]	species s is a reactant with stoichiometry 1 and a product with stoichiometry 2 $s + \dots \rightarrow 2s + \dots$

`set = RemoveDuplicates(full_set)`

returns

`set` = the array `full_set` with any duplicate members removed

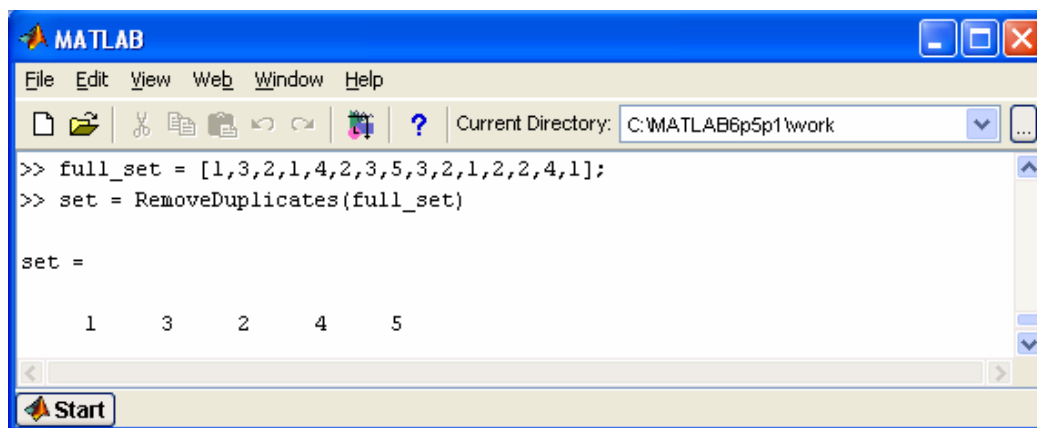


Figure 17: Output from RemoveDuplicates function

10. 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 8: Functions and their type in folder Simulation

Type of function	Function name
MATLAB help	Contents.m
Simulation	WriteODEFunction.m DisplayODEFunction.m
General	SelectSpecies.m SelectSpecies.fig

WriteODEFunction(MATLAB_SBMLModel)

takes a sbml model named 'Test'

and outputs a file named Test.m containing a function Test that can be used with matlabs odeN functions as shown

```
[t,x] = ode23(@Test, [0, t_end], Test)
```

where t_end is the end time of the simulation

DisplayODEFunction(MATLAB_SBMLModel)

graphically outputs the solution derived using the ode23 solver.

NOTE: DisplayODEFunction requires the file output by WriteODEFunction.

The xml model shown in Figure 18 is imported into MATLAB and WriteODEFunction is applied. The resulting output is shown in Figure 19. Results of the DisplayODEFunction are illustrated by Figure 20.


```

<?xml version="1.0" encoding="UTF-8"?>
<sbml level = "1" version = "1" xmlns = "http://www.sbml.org/sbml/level1" xmlns:jd = "http://www.sys-
bio.org/sbml">
  <model name = "AB_C">
    <listOfCompartments>
      <compartment name = "compartment" volume = "1"> </compartment>
    </listOfCompartments>
    <listOfSpecies>
      <specie boundaryCondition = "false" compartment = "compartment" initialAmount = "2" name =
"A">
        </specie>
      <specie boundaryCondition = "false" compartment = "compartment" initialAmount = "3" name =
"B">
        </specie>
      </specie>
    </listOfSpecies>
    <listOfReactions>
      <reaction name = "J0" reversible = "false">
        <listOfReactants>
          <specieReference specie = "A" stoichiometry = "1"/>
          <specieReference specie = "B" stoichiometry = "1"/>
        </listOfReactants>
        <listOfProducts>
          <specieReference specie = "C" stoichiometry = "1"/>
        </listOfProducts>
        <kineticLaw formula = "J0_k*A*B">
          <listOfParameters>
            <parameter name = "J0_k" value = "0.397"/>
          </listOfParameters>
        </kineticLaw>
      </reaction>
    </listOfReactions>
  </model>
</sbml>

```

Figure 18: XML for model of simple reaction $A + B \rightarrow C$

```

function xdot = AB_to_C(time, x)
% function AB_to_C takes
% either      1) no arguments
%              and returns a vector of the initial species concentrations
% or      2) time - the elapsed time since the beginning of the reactions
%           x    - vector of the current concentrations of the species
%              and returns a vector of the rate of change of concentration of each of the species
%
% AB_to_C can be used with matlabs odeN functions as
%   [t,x] = ode23(@AB_to_C, [0, t_end], AB_to_C)
%               where t_end is the end time of the simulation
%
% The species in this model are related to the output vectors with the following indices
%   Index  Species name
%   1      A
%   2      B
%   3      C

% output vector
xdot = zeros(3, 1);

% assign parameter values
J0_k = 3.970000e-001;

% calculate concentration values
if (nargin == 0)
    % initial concentrations
    xdot(1) = 0;
    xdot(2) = 0;
    xdot(3) = 0;
else
    % floating species concentrations
    A = x(1);
    B = x(2);
    C = x(3);
    % species concentration equations
    xdot(1) = -1 * J0_k * A * B;
    xdot(2) = -1 * J0_k * A * B;
    xdot(3) = 1 * J0_k * A * B;
end;

```

Figure 19: *AB_C.m* file output using *WriteODEFunction*

Note: At present this function only deals with the kinetic law specified within a reaction and does not consider rules or SBML level 2 events or functions.

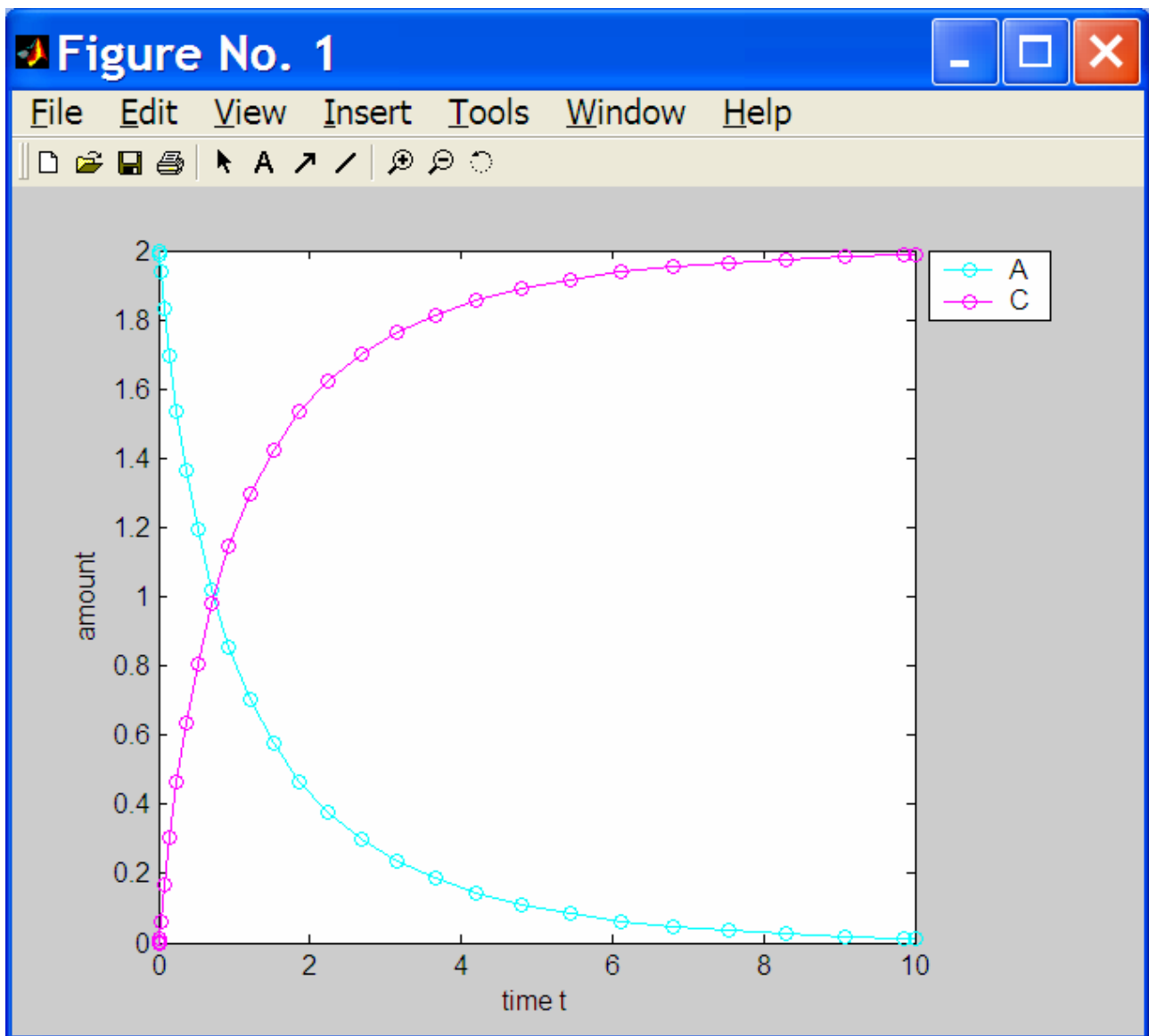


Figure 20: Output from DisplayODEFunction

```
[species] = SelectSpecies(MATLAB_SBMLModel)
```

takes a sbml model

returns

species = array of character names of the species selected

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

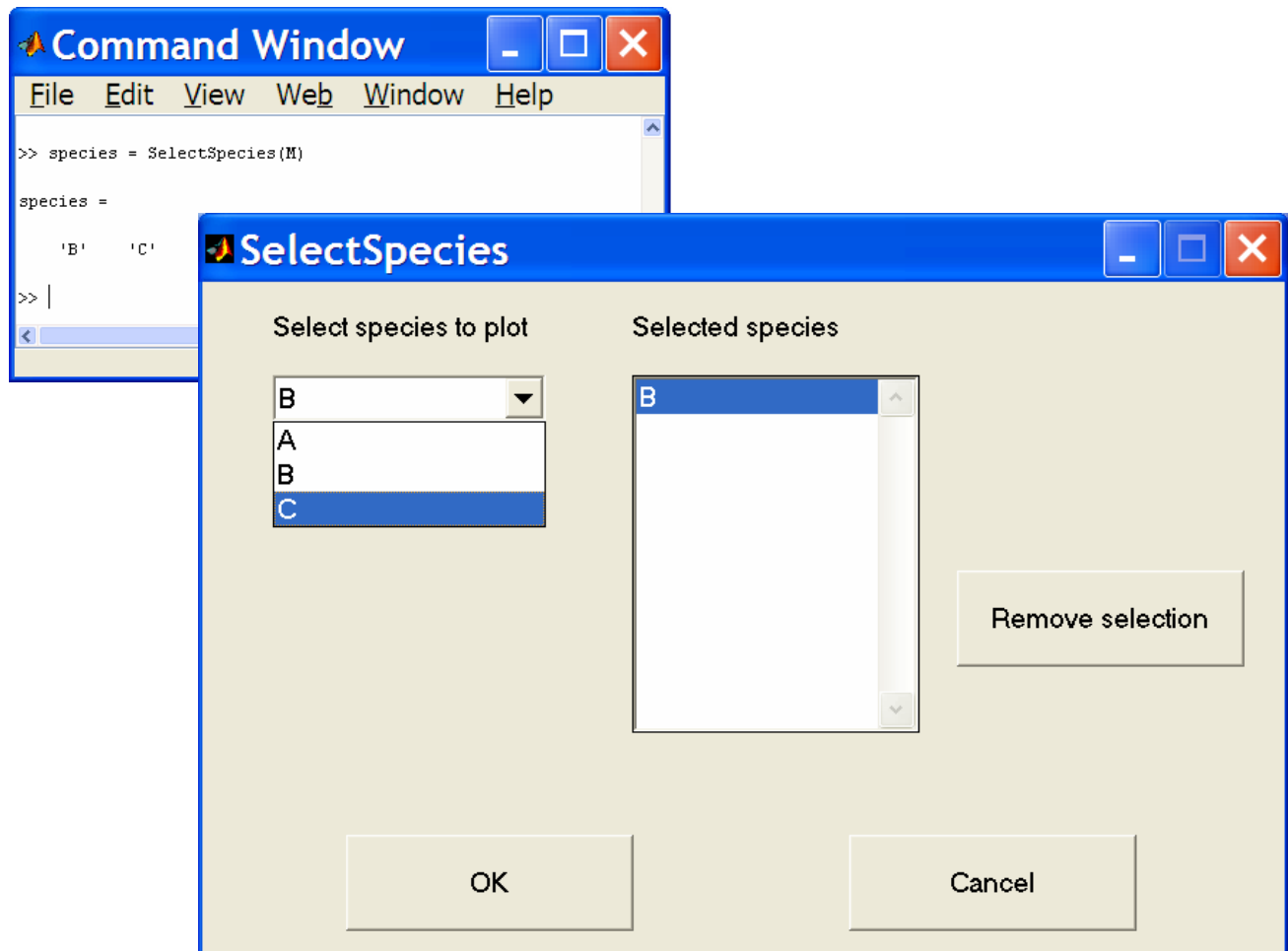


Figure 21: Output from *SelectSpecies* function