

MathSBML.m

The Mathematica SBML Reader

Author: Bruce E. Shapiro

The Systems Biology Workbench Development Group
ERATO Kitano Symbiotic Systems Project
Control and Dynamical Systems, MC 107-81
California Institute of Technology
<http://www.sbml.org>

Introduction

The [MathSBML](#) package provides facilities for reading models expressed in SBML and converting the models to systems of ordinary differential equations for simulation and plotting in *Mathematica*.

The current version of [MathSBML](#) can read SBML Level 1 Version 1 and Version 2.

■ Installation Instructions

[MathSBML](#) requires *Mathematica* 4.2 or above for full XML compatibility. It will not work properly with earlier versions of *Mathematica*.

The only file required is [MathSBML.m](#). Do not edit or modify this file, as it is machine-generated. The file can be copied anywhere on your hard drive. To use the file you need to include the line

```
<< path/MathSBML.m
```

where path is the full path name, relative to your home directory, of the directory in which you have placed [MathSBML](#). An alternative syntax is

```
Get["MathSBML.m", Path → path]
```

Mathematica knows about a set of default directories that it will search when looking for a file. If you put your files in one of these directories, you do not need to specify the path; e.g., you can omit the "path" from the above [Get \(<<\)](#) command:

```
<< MathSBML.m
```

You can determine the current path by entering `$Path` at the *Mathematica* prompt. Additional information on the *Mathematica* path is given in the *Mathematica* help files.

The file `MathSBML.nb` contains the source code for `MathSBML.m`. This file is not required unless you want to change the functionality of `MathSBML`. If you open this file in *Mathematica* you may get a warning message to the effect that the file has been edited outside of *Mathematica*. This is because of the manner in which version information is added to the file by the software repository and does not affect the coding contents of the file. This version information is stored in text boxes within the file.

■ Functions Provided by This Package

`MathSBML` contains three functions that can be invoked directly: `SBMLRead`, `SBMLNDSolve`, and `SBMLPlot`.

`SBMLRead` reads an SBML file and translates it into a *Mathematica* data structure consisting of *Mathematica* differential equations, initial conditions, a list of variables, and replacement rules for constant parameters. `SBMLRead` can also be used to generate an interpretive listing of the SBMLfile. The return value of `SBMLRead` is a *Mathematica* rule list of the form

```
{SBMLODES → list of differential equations ,
  SBMLParameters → list of parameter rules, SBMLIC → list of initial conditions ,
  SBMLSpecies → list of variables, SBMLAlgebraicRules → list of algebraic rules,
  SBMLUnitDefinitions → list of unit definitions ,
  SBMLUnitAssociations → list of unit associations }
```

Here *list of variables* has the form

```
{var1[t], var2[t], ... }
```

where each of `var1`, `var2`, etc., are variables that are governed by rate laws in the SBML models, i.e., any species, parameter, or compartment that is described by a rate law in a rule; and species that are either products or reactants in reactions, and are not boundary conditions. Observe that the variable `t` is reserved for time. Each species in the SBML model is translated into a time-dependent function in the *Mathematica* model.

The *list of differential equations* has the form

```
{var1'[t] == expression1, var2'[t] == expression2, ... }
```

where *expression1*, *expression2*, ... are *Mathematica* expressions formed by applying all of the rules and reactions that affect that corresponding species.

The *list of parameter rules* has the form

```
{par1 → expression1, par2 → expression2, ... }
```

where `par1`, `par2`, etc., are constant parameters or variable parameters described by `scalar` type `ParameterRules`; compartments with volumes that are either fixed or described by `scalar` type `CompartmentVolumeRules`; species that are described by `scalar` type `SpeciesConcentrationRules`; or species that are boundary conditions. The expressions are either constants (for fixed values) or algebraic expressions that give the value of the parameter. It is possible for the same parameter to be listed more than once in this list if the same local parameter name is used in multiple reactions. The parameters are listed in the same order in which they are defined in the SBML model.

The *list of initial conditions* has the form

$$\{\text{var1}[0] == \text{value1}, \text{var2}[0] == \text{value2}, \dots\}$$

where each of `var1`, `var2`, etc., are the same as defined in *list of variables* above. Their lengths of the lists of initial conditions, variables, and differential equations are all the same.

The *list of algebraic rules* has the form

$$\{\text{expression1}==0, \text{expression2}==0, \dots\}$$

where `expression1`, `expression2`, ... are *Mathematica* expressions formed from the corresponding `algebraicRule` in the SBML file.

The *list of unit definitions* has the form

$$\{\text{name1} \rightarrow \text{def1}, \text{name2} \rightarrow \text{def2}, \dots\}$$

where `name1`, `name2`, ... are *Mathematica* variables corresponding to the `name` field of the corresponding `unitDefinition`; and `def1`, `def2`, ... are *Mathematica* expressions defining the units in terms of the basic pre-defined units.

The *list of unit associations* has the form

$$\{\text{name1} \rightarrow \text{units1}, \text{name2} \rightarrow \text{units2}, \dots\}$$

where `name1`, `name2`, ... are *Mathematica* variables (species, parameters, or compartments) and `units1`, `units2`, ... are the units, either predefined or user-defined, corresponding to those variables.

`SBMLNDSolve` solves the system of differential equations produced by `SBMLRead` with `NDSolve`. The user can optionally pass the output of `SBMLRead` directly to `NDSolve` without using `SBMLNDSolve`. Examples of how to perform both of these operations are given below.

`SBMLPlot` can be used to generate plots of the resulting solutions. Plots can also be generated directly with `Plot`.

These functions are illustrated below in detail.

■ Known Limitations of `SBMLRead`

- (1) `SBMLRead` does not perform XML or SBML validation. If invalid SBML or XML is supplied, unexpected results can occur. In general, incorrectly formatted XML will cause *Mathematica's* `Import[...]` function to print an error message indicating the line number for the first error and then *Mathematica* will terminate.
- (2) `SBMLRead` is currently only compatible with SBML Level 1. Subsequent releases will support higher levels. The SBML level and version are detected from the `level` and `version` parameters of the `<sbml ...>` tag. The value of the namespace parameter (`xmlns`) is checked for consistency with these keywords as defined by the appropriate SBML spec and an error message is printed if there is a conflict.
- (3) All of the mathematical functions (e.g., `abs`, `acos`, etc.) are fully supported. Thus `cos(x)` becomes `Cos[x]`, etc.

Predefined rate law functions are recognized as functions but are not implemented. Thus, if the function `umr(argument list)` is specified in the SBML, it will be recognized as a predefined function and will be expanded in the *Mathematica* model as `umr[argument list]`. However, unlike the mathematical functions, no implementation is provided. Thus if the model contains these functions, the user must supply a *Mathematica* implementation for `umr`, etc.

(4) The SBML type `SName` allows the underscore character "_" to be included in an identifier. Unfortunately, this symbol is reserved for patterns in *Mathematica*. `MathSBML` replaces underscore characters in its *Mathematica* output with the value of the option `underscore` to `SBMLRead`. The default value is `underscore→"_"`. Hence an identifier `A_B` in the SBML becomes `A_B` in the *Mathematica* model. To force all underscores to be replaced by a happy face character (`ESC : ESC`), for example, use `SBMLRead[... , underscore→"☺"]`

The under-bracket character can be entered at the keyboard with either of the following keystroke sequences: the backslash character ("\") immediately followed by the string "[UnderBracket]", or the key sequence `ESC u ESC` (with no spaces between the characters). Note that when you hit the escape key, `ESC`, a vertical array of three short lines (`ESC`) will be displayed by *Mathematica*.

With this exception, the *Mathematica* identifiers are identical to the SBML identifiers.

(5) By default, all parameters are replaced with their numerical values as specified in the model. This can be switched off, in which case `SBMLRead` will return a list of *Mathematica* replacement rules of the form `name→value`. However, if the user chooses to switch off parameter evaluation (by setting `evaluateParameters→False`), unexpected results may occur if the same local parameter name is used in more than one reaction or a local parameter has the same name as a global parameter. The reason for this is that each instance of these parameters will generate a rule `name→value` in the returned parameter list, in the order in which they are defined; however, when a rule list is evaluated, *Mathematica* always gives priority to the first occurrence of the rule in the list. Thus `expression/.{a→5, a→10}` will replace every occurrence of the parameter `a` with the value 5.

(6) The topological relationship specified by the `outside` attribute in a compartment definition is ignored, although `SBMLRead` will display the relationship in the verbose listing. If no outside component exists, the *Mathematica* variable `Indeterminate` is displayed.

(7) The `reversible` parameter of the `reaction` type is ignored.

(8) Unspecified initial conditions and parameter values will be labeled as `Indeterminate` if they are not specified, and a warning message will be printed. Models with `Indeterminate` parameters and initial conditions will cause an error in `NDSolve`. To prevent this from happening, the user can optionally specify the options `defaultParameterValue` and `defaultIC` to set all `Indeterminate` parameter values and initial conditions. For example, `defaultParameterValue→1` will set all `Indeterminate` parameters equal to 1. Unspecified units will be labeled as `Indeterminate` in the verbose listing, but no association will be returned for indeterminate units.

(9) Annotations and notes are ignored.

(10) In SBML Level 1 Version 1, at least one compartment must be defined. If no compartments are defined, the program will abort. In the current draft specification of SBML Level 1 Version 2, if no compartments are defined, a fictional compartment "`Indeterminate`" is created.

(11) If a compartment name is omitted from a species reference the species is assigned to compartment "`Indeterminate`". Note that in the current draft specification of SBML Level 1 Version 2, this compartment will have been previously defined by the program if no compartments are defined in the model. In SBML Level 1 Version 1, this will cause a warning message to be printed because the compartment has not been previously defined.

(12) In either SBML Level 1 Version 1 or Version 2, if a species is assigned to an undefined compartment, a warning will be printed but the remainder of the program should complete normally.

(13) algebraicRules (equations of the form $0 == expression$) are not processed by [SBMLNDSolve](#).

(14) units are ignored by [SBMLNDSolve](#).

(15) all model variables -- species, parameters, compartment -- are treated in *Mathematica* as context `Global`` variables. Thus if a variable in the `Global`` context has the same name as a variable in the xml file, a conflict may occur.

(15) if any model variable (parameter, species, compartment) is the same as a predefined *Mathematica* variable (such as D, E, I, Infinity, EulerGamma, Pi, GoldenRatio, etc.) as well as any mathematica function name, a conflict can occur and unexpected results can occur as a result of the collision. Avoiding variables that begin with upper case letters will prevent this from happening. A fix is planned in a later version.

Examples

```
<< MathSBML.m
$MathSBML$Version=1.0.9 (3 May 2003)

Needs["Graphics`"]
```

Import and translate a file

Import the file [smallestHopf.xml](#) from the subdirectory [mathSBMLv1](#) of your home directory

```
r = SBMLRead["sbml-files/smallest.Hopf.xml"];
```

Display the returned Value

```
r
{SBMLODES → {X'[t] == 2.2 X[t] - X[t] Y[t], Y'[t] == -Y[t] + Z[t], Z'[t] == X[t] - Z[t]},
 SBMLParameters → {compartment → 1, A → 1, _void_ → 0.},
 SBMLIC → {X[0] == 2.5, Y[0] == 2.5, Z[0] == 2.5},
 SBMLSpecies → {X[t], Y[t], Z[t]}, SBMLAlgebraicRules → {},
 SBMLUnitDefinitions → {substance → mole, volume → liter, time → second},
 SBMLUnitAssociations → {compartment → volume, X → substance,
  A → substance, Y → substance, Z → substance, _void_ → substance}}
```

Run a simulation on the imported file

Any valid option for [NDSolve](#) such as [MaxSteps](#), can be passed to [SBMLNDSolve](#)

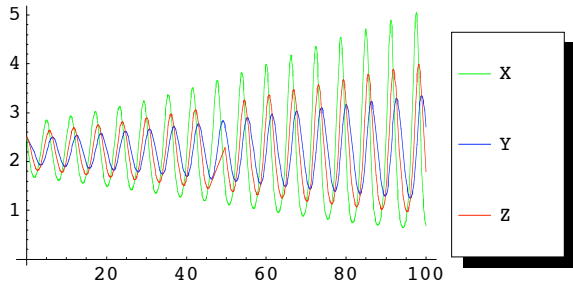
```
n = SBMLNDSolve[r, 300, MaxSteps → 5000]
{{X[t] → InterpolatingFunction[{{0., 300.}}, <>][t],
 Y[t] → InterpolatingFunction[{{0., 300.}}, <>][t],
 Z[t] → InterpolatingFunction[{{0., 300.}}, <>][t]}}
```

Using NDSolve directly instead of SBMLNDSolve

```
NDSolve[Join[SBMLODES /. r, SBMLIC /. r] /. (SBMLParameters /. r),
 SBMLSpecies /. r, {t, 0, 50}]
{{X[t] → InterpolatingFunction[{{0., 50.}}, <>][t],
 Y[t] → InterpolatingFunction[{{0., 50.}}, <>][t],
 Z[t] → InterpolatingFunction[{{0., 50.}}, <>][t]}}
```

Plot the results of the simulation

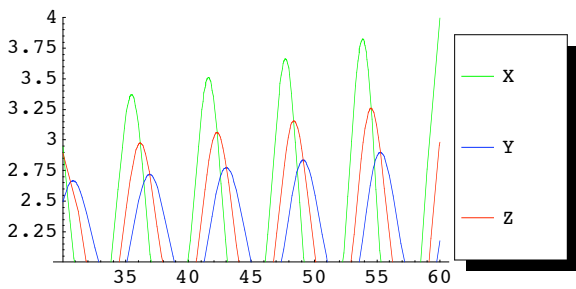
```
SBMLPlot[n, {X, Y, Z}, {0, 100}]
```



- Graphics -

Any valid option for `Plot` can be passed to `SBMLPlot`

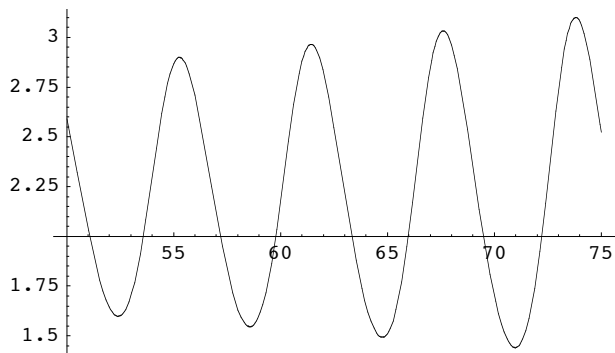
```
SBMLPlot[n, {X, Y, Z}, {30, 60}, PlotRange -> {2, 4}]
```



- Graphics -

The output of `SBMLNDSolve` is directly compatible with `Plot`

```
Plot[Evaluate[Y[t] /. n], {t, 50, 75}]
```



- Graphics -

Using SBMLRead as an SBML interpreter

```
SBMLRead["sbml-files/smallest.Hopf.xml", verbose → True];
```

File Name: sbml-files/smallest.Hopf.xml
SBML Level 1 Version 1

Model: Smallest_chemical_reaction_system_with_Hopf_bifurcation

Unit Definitions

----- None -----

Compartments

| <u>Name</u> | <u>Volume</u> | <u>Units</u> | <u>Derived Units</u> | <u>Outside</u> |
|-------------|---------------|--------------|----------------------|----------------|
| compartment | 1 | volume | liter | Indeterminate |

Species

| <u>Name</u> | <u>I.C.</u> | <u>Units</u> | <u>Derived Units</u> | <u>Charge</u> | <u>Compartment</u> |
|-------------|-------------|--------------|----------------------|---------------|--------------------|
| X | 2.5 | substance | mole | 0 | compartment |
| A | 1 | substance | mole | 0 | compartment |
| Y | 2.5 | substance | mole | 0 | compartment |
| Z | 2.5 | substance | mole | 0 | compartment |
| _void_ | 0. | substance | mole | 0 | compartment |

Global Parameters

----- None -----

Rules

----- None -----

Reactions

| <u>Name</u> | <u>Reaction</u> |
|-------------|---------------------------|
| R1 | $X \rightarrow 2X$ |
| R2 | $X + Y \rightarrow Y$ |
| R3 | $X \rightarrow Z$ |
| R4 | $Z \rightarrow Y$ |
| R5 | $Y \rightarrow \emptyset$ |

Differential Equations from Reactions

Species

X
Y
Z

Differential Equations

$X'[t] == 2.2 X[t] - X[t] Y[t]$
 $Y'[t] == -Y[t] + Z[t]$
 $Z'[t] == X[t] - Z[t]$

Function Reference

■ MathSBML`SBMLRead

SBMLRead[filename, options] returns an option list of the form:

```
{
SBMLODES→{v1'[t]==expression, v2'[t]==expression,...},
SBMLParameters→{k1→value,k2→value,...},
SBMLIC→{v1[0]==value,v2[0]==value,...},
SBMLSpecies→{v1[t],v2[2],...},
SBMLAlgebraicRules→{expression1==0, expression2==0, ...},
SBMLUnitDefinitions→{unitName1→unitDefinition1, unitName2→unitDefinition2,...},
SBMLUnitAssociations→{var1→units1,var2→units2,...}
}
```

where v1,v2,... give all of the species in the SBML file; the expression gives the derived differential equation for that species; value is the SBML value of the associated parameter or initial condition; unitName1,... are user-defined units; unitDefinition1,... are expressions that give the unit definitions in terms of pre-defined units; units1,... are the units that correspond to variable var1,..., which can be either species, parameter, or compartment. Note that SBMLParameters will contain the same parameter more than once if the same parameter name is used in multiple places in the SBML file.

The intention is that this format contains all information necessary to pass the model to NDSolve in the following manner:

```
r=SBMLRead[filename,options]
Apply[NDSolve,{Join[SBMLODES/.r, SBMLIC/.r]/.
(SBMLParameters/.r), SBMLSpecies/.r,{t,0,tmax},NDSolveOptions]}
```

where NDSolveOptions are any valid options for NDSolve and tmax is the duration of the NDSolve Rule

Options for SBMLRead are:

```
concise→False. When True, overrides whatever values are set to verbose, warnings,
and stats to set all of them to False. This option minimizes the written output.
defaultIC→Indeterminate, if reassigned, then all unspecified initial conditions
will be set to the value of defaultIC. Otherwise they will remain Indeterminate.
defaultParameterValue→Indeterminate, if reassigned, then all unspecified parameter values
will be set to the value of defaultIC. Otherwise they will remain Indeterminate.
evaluateParameters→True, immediately evaluate parameters in reactions, otherwise return reactions
with symbolic parameters. If evaluateParameters→False then if the same parameter is defined
in multiple contexts in the SBML then Mathematica will be unable to distinguish between them.
shortenODES→True, ignored unless verbose is True. If shortenODES→False, then
the entire differential equation will be displayed in the verbose listing; if
shortenODES→True (default) then the Mathematica Short[...] version will be used.
showKineticLaw→False, ignored unless verbose→True; if showKineticLaw→True, the
SBML kinetic law is shown in the reaction-listing of the verbose display;
otherwise (default) the kinetic laws are not displayed in the reactions table
stats→False, print a statistical summary of the file
underscore→"_", character (or string) that is
used to replace the underscore ("_") character in SBML identifiers.
verbose→False, if True, print an interpretive table of the SBML
warnings→ True, if False, warning messages will be suppressed.
```

■ Examples

■ Most Concise Output

```
r = SBMLRead["sbml-files/smallest.Hopf.xml"];
```

■ return Value

```
r
{SBMLODES → {X'[t] == 2.2 X[t] - X[t] Y[t], Y'[t] == -Y[t] + Z[t], Z'[t] == X[t] - Z[t]},
 SBMLParameters → {compartment → 1, A → 1, _void_ → 0.},
 SBMLIC → {X[0] == 2.5, Y[0] == 2.5, Z[0] == 2.5},
 SBMLSpecies → {X[t], Y[t], Z[t]}, SBMLAlgebraicRules → {},
 SBMLUnitDefinitions → {substance → mole, volume → liter, time → second},
 SBMLUnitAssociations → {compartment → volume, X → substance,
  A → substance, Y → substance, Z → substance, _void_ → substance}}
```

■ returning file statistics

```
r = SBMLRead["sbml-files/smallest.Hopf.xml", stats → True];
```

File Statistics

| <u>Description</u> | <u>Value</u> |
|--------------------|--------------|
| Compartments | 1 |
| Species | 3 |
| ODES | 3 |
| Global Parameters | 0 |
| Local Parameters | 5 |
| Rules | 0 |
| Reactions | 5 |
| CPU Used | 0.07 |

■ Using SBMLRead to display SBML

```
r = SBMLRead["sbml-files/smallest.Hopf.xml", verbose → True];
```

File Name: sbml-files/smallest.Hopf.xml
SBML Level 1 Version 1

Model: Smallest_chemical_reaction_system_with_Hopf_bifurcation

Unit Definitions

----- None -----

Compartments

| <u>Name</u> | <u>Volume</u> | <u>Units</u> | <u>Derived Units</u> | <u>Outside</u> |
|-------------|---------------|--------------|----------------------|----------------|
| compartment | 1 | volume | liter | Indeterminate |

Species

| <u>Name</u> | <u>I.C.</u> | <u>Units</u> | <u>Derived Units</u> | <u>Charge</u> | <u>Compartment</u> |
|-------------|-------------|--------------|----------------------|---------------|--------------------|
| X | 2.5 | substance | mole | 0 | compartment |
| A | 1 | substance | mole | 0 | compartment |
| Y | 2.5 | substance | mole | 0 | compartment |
| Z | 2.5 | substance | mole | 0 | compartment |
| _void_ | 0. | substance | mole | 0 | compartment |

Global Parameters

----- None -----

Rules

----- None -----

Reactions

| <u>Name</u> | <u>Reaction</u> |
|-------------|---------------------------|
| R1 | $X \rightarrow 2X$ |
| R2 | $X + Y \rightarrow Y$ |
| R3 | $X \rightarrow Z$ |
| R4 | $Z \rightarrow Y$ |
| R5 | $Y \rightarrow \emptyset$ |

Differential Equations from Reactions

| <u>Species</u> | <u>Differential Equations</u> |
|----------------|----------------------------------|
| X | $X' [t] == 2.2 X[t] - X[t] Y[t]$ |
| Y | $Y' [t] == -Y[t] + Z[t]$ |
| Z | $Z' [t] == X[t] - Z[t]$ |

■ MathSBML`SBMLNDSolve

`SBMLNDSolve[model, tmax, options]` evaluates `NDSolve` on an SBML model, where `model` is the output of `SBMLRead`, `tmax` is the duration of the `NDSolve` run, and `options` are any valid options for `NDSolve`.

■ Example

```
r = SBMLRead["smallestHopf.xml", concise → True];
n = SBMLNDSolve[r, 300, MaxSteps → 5000]

{{X[t] → InterpolatingFunction[{{0., 300.}}, <>][t],
  Y[t] → InterpolatingFunction[{{0., 300.}}, <>][t],
  Z[t] → InterpolatingFunction[{{0., 300.}}, <>][t]}}
```

■ MathSBML`SBMLPlot

`SBMLPlot[solution, {var1, var2,...}, {tbegin, tend}, options]` plots the results of a simulation. `solution` is the output of `SBMLNDSolve`.

The variables named `var1, var2,..` are plotted from `t=tbegin` to `t=tend` on a single plot.

`SBMLPlot[solutions,{tbegin,tend}, options]` plots all variables in the solution set.

`SBMLPlot[solution,options]` plots all variables for the entire duration of the run.

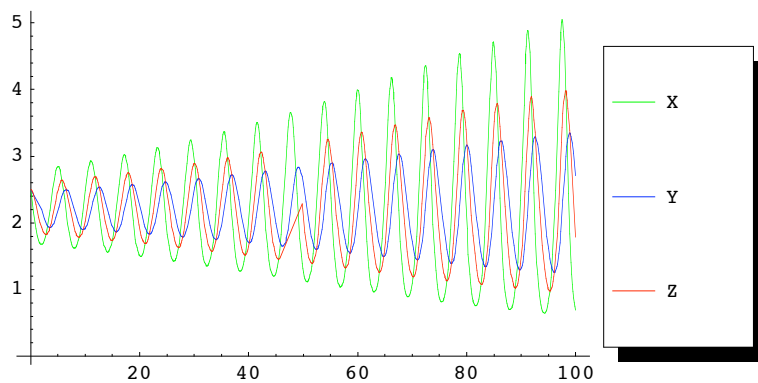
■ Example

```
r = SBMLRead["smallestHopf.xml", concise → True];
```

```
n = SBMLNDSolve[r, 300, MaxSteps → 5000]
```

```
{{X[t] → InterpolatingFunction[{{0., 300.}}, <>][t],  
  Y[t] → InterpolatingFunction[{{0., 300.}}, <>][t],  
  Z[t] → InterpolatingFunction[{{0., 300.}}, <>][t]}}
```

```
SBMLPlot[n, {X, Y, Z}, {0, 100}]
```



- Graphics -