
SYMDIFF Manual

Release 1.0.0

DEVSIM LLC

Contents

Contents	iv
List of Tables	v
1 Front Matter	1
1.1 Contact	1
1.2 Copyright	1
1.3 Documentation License	1
1.4 Disclaimer	1
1.5 Trademark	1
2 Release Notes	3
2.1 Introduction	3
2.2 Release 1.0.0 (May 7, 2019)	3
3 Introduction	5
3.1 Getting Started	5
3.2 Using the Tool	5
3.3 Tcl version	6
4 Syntax	7
4.1 Variables and Numbers	7
4.2 Basic Expressions	8
4.3 Commands	9
4.4 User functions	10
4.5 Models	10
4.6 Macro Assignment	12
4.7 Handling Exceptions	12
4.8 Table Output	12
5 Generating Source Code	15
5.1 model_list	15
5.2 ordered_list	15
5.3 remove_zeros	16
5.4 subexpression	16
6 Additional Information	17
6.1 Other Examples	17

6.2	Licenses	17
7	Installation	19
7.1	Download	19
7.2	Supported Platforms	19
7.3	Directory Structure	19
	Bibliography	21

List of Tables

4.1	Basic expressions involving unary, binary, and logical operators.	8
4.2	Predefined Functions.	9
4.3	Commands.	9
4.4	Commands for user functions.	10
4.5	Commands for models.	11

Chapter 1

Front Matter

1.1 Contact

Web:	https://devsim.com
Email:	info@devsim.com
Open Source Project:	https://syndiff.org

1.2 Copyright

Copyright © 2009–2019 DEVSIM LLC

1.3 Documentation License

This work is licensed under the Creative Commons Attribution-NoDerivatives 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nd/4.0/>.

1.4 Disclaimer

DEVSIM LLC MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

1.5 Trademark

DEVSIM is a registered trademark and SYMDIFF is a trademark of DEVSIM LLC. All other product or company names are trademarks of their respective owners.

Chapter 2

Release Notes

2.1 Introduction

2.2 Release 1.0.0 (May 7, 2019)

The version number has been updated to having a major revision of 1. We adopt the semantic version numbering presented at <https://semver.org>. The version number can be accessed through the `Python` interface using the `syndiff.__version__` variable.

This is the first versioned release of SYMDIFF. Using the stable ABI, the software is able to run newer `Python 3` releases, without rebuilding the software.

Unlike previous revisions of software, `Python 2.7`, is removed from the build.

Chapter 3

Introduction

3.1 Getting Started

SYMDIFF is a computer algebra tool capable of taking symbolic derivatives. Using a natural syntax, it is possible to manipulate symbolic equations to aid derivation of equations for a variety of applications. Additional commands provide the means to simplify results, create common subexpressions, and order expressions for use as source code in a computer program. With its `Python` and `Tcl` interpreters, you have the ability to create algorithms to generate equations programmatically.

SYMDIFF is available from <https://syndiff.org>. The source code is available under the terms of the Apache License Version 2.0 [ApacheSoftwareFoundation]. Examples are released under the Apache License Version 2.0 [ApacheSoftwareFoundation]. Contributions to this project are welcome in the form of bug reporting, documentation, modeling, and feature implementation.

3.2 Using the Tool

3.2.1 Interactive Mode

The tool is invoked by typing `syndiff` on the command line. On some platforms, the application may also be started by clicking on the application name in a file manager. A `Python` shell is started. In this introduction, we start by importing the module into the global namespace:

```
>>> from syndiff import *
```

We can then start executing SYMDIFF commands.

```
>>> syndiff ('x^y')  
'pow(x, y) '
```

In this expression, both `x` and `y` are independent variables. To differentiate the expression with respect to `x`, we type the following:

```
>>> symdiff('diff(x^y,x)')
'(y * pow(x,(y - 1)))'
```

By default, any non-numeric string which is not already a function name is treated as an independent variable.

If we wish to simplify the expression, we do the following

```
>>> symdiff('simplify(x*x+ 2*x^2)')
'(3 * pow(x,2))'
```

and to expand out an expression

```
>>> symdiff('expand((x+ y)*x)')
'((x * y) + pow(x,2))'
```

A semicolon ; is used to send multiple commands to the interpreter, but it is not recommended, as it makes debugging more difficult.

3.2.2 Script Mode

With its built-in Python interpreter, SYMDIFF will execute a script and can be invoked on the command line of your terminal program as:

```
symdiff myfile.py
```

where `myfile.py` is the name of your input file.

3.2.3 Unicode

The Python interpreter, by default, only allows ASCII characters. In order to enable unicode in your Python scripts, the following line on the first or second line of the script should contain:

```
# -*- coding: utf-8 -*-
```

This assumes that the source file is written using utf8. In interactive mode, using unicode is not recommended, based on issues in setting the environment properly for the Python interpreter.

3.3 Tcl version

A Tcl interface to SYMDIFF is also available by starting `symdiff_tcl`. In order to use SYMDIFF in the Tcl interpreter, the following line is required:

```
% package require symdiff_tcl
```

Chapter 4

Syntax

4.1 Variables and Numbers

Variables and numbers are the basic building blocks for expressions. A variable is defined as any sequence of characters not beginning with a number or underscore, (`_`), followed by any number of characters. Note that the letters are case sensitive so that `a` and `A` are not the same variable. Any other characters are considered to be either mathematical operators or invalid, even if there is no space between the character and the rest of the variable name.

`SYMDIFF` supports `unicode` character sets, so that special characters such as ψ can be used. Note however this feature should be used carefully when generating source code since these characters are not valid input for many computer languages. Please see [Unicode](#) (page 6) for more information.

Examples of valid variable names are:

- `a`, `dog`, `var1`, `var_2`

Numbers can be integer or floating point. Scientific notation is accepted as a valid syntax. For example:

- `1.0`, `1.0e-2`, `3.4E-4`

4.2 Basic Expressions

Table 4.1: Basic expressions involving unary, binary, and logical operators.

Expression	Description
(exp1)	Parenthesis for changing precedence
+exp1	Unary Plus
-exp1	Unary Minus
!exp1	Logical Not
exp1 ^ exp2	Exponentiation
exp1 * exp2	Multiplication
exp1 / exp2	Division
exp1 + exp2	Addition
exp1 - exp2	Subtraction
exp1 < exp2	Test Less
exp1 <= exp2	Test Less Equal
exp1 > exp2	Test Greater
exp1 >= exp2	Test Greater Equal
exp1 == exp2	Test Equality
exp1 != exp2	Test Inequality
exp1 && exp2	Logical And
exp1 exp2	Logical Or
variable	Independent Variable
number	Integer or decimal number

In [Table 4.1](#), the basic syntax for the language is presented. An expression may be composed of variables and numbers joined with mathematical operators. Order of operations is from bottom to top in order of increasing precedence. Operators with the same level of precedence are contained within horizontal lines.

In the expression $a + b * c$, the multiplication will be performed before the addition. In order to override this precedence, parenthesis are used. For example, in $(a + b) * c$, the addition operation is performed before the multiplication.

4.2.1 Functions

Table 4.2: Predefined Functions.

Function	Description
<code>exp(exp1)</code>	exponent
<code>log(exp1)</code>	natural log
<code>pow(exp1, exp2)</code>	take exp1 to the power of exp2
<code>ifelse(test, exp1, exp2)</code>	if test is true, then evaluate exp1, otherwise exp2
<code>if(test, exp)</code>	if test is true, then evaluate exp, otherwise 0

In [Table 4.2](#) are the built in functions of SYMDIFF. Note that the `pow` function uses the `,` operator to separate arguments. In addition an expression like `pow(a, b+y)` is equivalent to an expression like `a^(b+y)`. Both `exp` and `log` are provided since many derivative expressions can be expressed in terms of these two functions. It is possible to nest expressions within functions and vice-versa.

Special care should be used when using the exponentiation operator, since the unary minus has a higher precedence than the exponentiation operator.

```
>>> symdiff('x^-1 + 3')
'(pow(x, (-1)) + 3) '
>>> symdiff('-x^3')
'pow((-x), 3) '
```

Parenthesis are recommended to avoid ambiguity in this situation.

4.3 Commands

Table 4.3: Commands.

Command	Description
<code>diff(obj1, var)</code>	Take derivative of <code>obj1</code> with respect to variable <code>var</code>
<code>expand(obj)</code>	Expand out all multiplications into a sum of products
<code>scale(obj)</code>	Get constant factor
<code>sign(obj)</code>	Get sign as 1 or -1
<code>simplify(obj)</code>	Simplify as much as possible
<code>subst(obj1, obj2, obj3)</code>	substitute <code>obj3</code> for <code>obj2</code> into <code>obj1</code>
<code>unscaledval(obj)</code>	Get value without constant scaling
<code>unsignedval(obj)</code>	Get unsigned value

Commands are shown in [Table 4.3](#). While they appear to have the same form as functions, they are special in the sense that they manipulate expressions and are never present in the expression which results. For example, note the result of the following command

```
> diff(a*b, b)
a
```

4.4 User functions

Table 4.4: Commands for user functions.

Command	Description
<code>clear(name)</code>	Clears the name of a user function
<code>declare(name(arg1, arg2, ...))</code>	declare function name taking dummy arguments <code>arg1, arg2, ...</code> . Derivatives assumed to be 0
<code>define(name(arg1, arg2, ...), obj1, obj2, ...)</code>	declare function name taking arguments <code>arg1, arg2, ...</code> having corresponding derivatives <code>obj1, obj2, ...</code>

Commands for specifying and manipulating user functions are listed in Table 4.4. They are used in order to define new user function, as well as the derivatives of the functions with respect to the user variables. For example, the following expression defines a function named `f` which takes one argument.

```
> define(f(x), 0.5*x)
```

The list after the function prototype is used to define the derivatives with respect to each of the independent variables. Once defined, the function may be used in any other expression. In addition, any expression can be used as an argument. For example:

```
>>> symdiff('diff(f(x*y), x)')
'(5.000000000000000e-01 * x * y * y) '
>>> symdiff('simplify(5.000000000000000e-01 * x * y * y)')
'(5.000000000000000e-01 * x * pow(y, 2) ')
```

$$\frac{\partial}{\partial x} f(u, v, \dots) = \frac{\partial u}{\partial x} \cdot \frac{\partial}{\partial u} f(u, v, \dots) + \frac{\partial v}{\partial x} \cdot \frac{\partial}{\partial v} f(u, v, \dots) + \dots$$

The `declare` command is required when the derivatives of two user functions are based on one another. For example:

```
>>> symdiff('declare(cos(x))')
'cos(x) '
>>> symdiff('define(sin(x), cos(x))')
'sin(x) '
>>> symdiff('define(cos(x), -sin(x))')
'cos(x) '
```

When declared, a functions derivatives are set to 0, unless specified with a `define` command. It is now possible to use these expressions as desired.

```
>>> symdiff('diff(sin(cos(x)), x)')
'(-cos(cos(x)) * sin(x)) '
```

4.5 Models

Models are a feature unique to SYMDIFF. They are used to specify symbolic names which have a definition in a symbolic expression. The most useful property of a model, is that taking the derivative of the model

with respect to a parameter results in a new model being created.

For example:

```
>>> symdiff('define_model(c, (a^2 + b^2)^0.5)')
'c'
>>> symdiff('diff(c,a)')
'c__a'
>>> symdiff('diff(c,b)')
'c__b'
>>> symdiff('model_value(c__a)')
'(a * pow((pow(a,2) + pow(b,2)), (-5.000000000000000e-01)))'
>>> symdiff('model_value(c__b)')
'(b * pow((pow(a,2) + pow(b,2)), (-5.000000000000000e-01)))'
```

For models which are declared, but not defined, it resolves to a special value of UNDEFINED. This example is called `undefined1.py` in the examples

```
symdiff('declare_model(y)')
symdiff('define_model(x, 3 * y + z)')
symdiff('diff(x, z)')
print('%s' % symdiff('model_value(x)'))
print('%s' % symdiff('model_value(x__z)'))
print('%s' % symdiff('model_value(y__z)'))
```

```
((3 * y) + z)
(1 + (3 * y__z))
UNDEFINED
```

Clearing a model removes its name from the list of models. Subsequent evaluation of new expressions will treat this name as a variable or function name. Care should be taken when other models depending on the cleared model remain.

```
>>> symdiff('declare_model(y)')
'y'
>>> symdiff('diff(y,x)')
'y__x'
>>> symdiff('clear_model(y)')
'0'
>>> symdiff('diff(y,x)')
'0'
```

Table 4.5: Commands for models.

Command	Description
<code>clear_model(name)</code>	clear model name
<code>declare_model(name)</code>	declare model name
<code>define_model(name, exp)</code>	define model having expression
<code>model_value(name)</code>	retrieve expression for model

Commands for specifying and manipulating models are listed in [Table 4.5](#).

4.6 Macro Assignment

The use of macro assignment allows the substitution of expressions into new expressions. Every time a command is successfully used, the resulting expression is assigned to a special macro definition, `$_`.

In this example, the result of the each command is substituted into the next.

```
>>> symdiff('a + b')
'(a + b) '
>>> symdiff('$_ - b')
'(a + b - b) '
>>> symdiff('simplify($_)')
'a '
```

In addition to the default macro definition, it is possible to specify a variable identifier by using the `\$` character followed by an alphanumeric string beginning with a letter. In addition to letters and numbers, a `_` character may be used as well. A macro which has not previously assigned will implicitly use 0 as its value.

This example demonstrates the use of macro assignment.

```
>>> symdiff('$a1 = a + b')
'(a + b) '
>>> symdiff('$a2 = a - b')
'(a - b) '
>>> symdiff('simplify($a1+$a2)')
'(2 * a) '
```

4.7 Handling Exceptions

If a SYMDIFF evaluation results in an error an exception of type `symdiff.SymdiffError` will be thrown. It may be caught and printed as a string:

```
try:
    out = symdiff(arg)
except SymdiffError as x:
    out = x
print out
```

4.8 Table Output

`symdiff_table` command is like the `symdiff` command, except that it creates a table for the expression. The last row is the full expression, and it is made up of entries in the previous rows. Each column of a row is

0. The name of the operator
1. The operator type
2. A list of indexes for the operator arguments into the table

3. A list of indexes of operators in other rows using this row as an argument
4. The full string value of the expression composed of this row and the rows it depends on

Example output for the code

```
symdiff('declare_model(x)')
for i, v in enumerate(symdiff_table('a*x + b*c')):
    print '%s %s' % (i, v)
```

is

```
0 ('a', 'variable', (), (2L,)), 'a')
1 ('x', 'model', (), (2L,)), 'x')
2 ('*', 'product', (0L, 1L), (6L,)), '(a * x)')
3 ('b', 'variable', (), (5L,)), 'b')
4 ('c', 'variable', (), (5L,)), 'c')
5 ('*', 'product', (3L, 4L), (6L,)), '(b * c)')
6 ('+', 'add', (2L, 5L), ()), '((a * x) + (b * c))')
```


Chapter 5

Generating Source Code

5.1 model_list

This command prints a list of all models which have been defined up to this point in the execution of SYMDIFF. This is shown in the `modellist1.py` example:

```
syndiff('declare_model(x)')
syndiff('declare_model(y)')
l = model_list()
for i in l:
    print('%s' % i)
```

The resulting output is then:

```
x
y
```

5.2 ordered_list

This command takes a list of 1 or more model names. The resulting list is in the order necessary to ensure that the models are evaluated in the correct order. In this example, `ordered.py`, we define 2 models, and SYMDIFF prints what order the models would have to be defined.

```
syndiff('define_model(b, a)')
syndiff('define_model(d, b * c)')
mylist = ordered_list('d')
for i in mylist:
    print('%s' % i)
```

The resulting output is then:

```
b
d
```

For ordering multiple model names, pass multiple names, or a list using this syntax.

```
ordered_list('a', 'b')
args = ('a', 'b')
ordered_list(*args)
```

5.3 remove_zeros

This command removes all models whose evaluation results in 0. Any models which rely on the definition of models will substitute a 0 in their expression for this model. This is shown in the `remove1.py` example.

```
syndiff('define_model(x, 0)')
syndiff('define_model(y, x + z)')
remove_zeros()
print('%s' % syndiff('model_value(y)'))
```

The resulting output is then:

```
z
```

5.4 subexpression

This command will evaluate all of the currently defined models and find common sub expressions. If more than one dependent model uses the same sub-expression, SYMDIFF will automatically substitute it with a new model with a generated name, as shown in the `subexpression1.py` example.

```
syndiff('define_model(x, simplify(y * z))')
syndiff('define_model(z, simplify(z * y))')
subexpression()
l = model_list()
for i in l:
    print("%s, %s" % (i, syndiff('model_value(%s)' % i)))
```

The resulting output is then:

```
unique0, (y * z)
x, unique0
z, unique0
```

The use of the `simplify` method is important to ensure that the subexpression elimination algorithm can recognize the common expressions.

Chapter 6

Additional Information

6.1 Other Examples

The `arrhenius.py` example demonstrates the use of all of the model manipulation algorithms. The `utf8.py` is an example using unicode encoding.

6.2 Licenses

6.2.1 SYMDIFF

```
Copyright 2012-2016 Devsim LLC
```

```
Licensed under the Apache License, Version 2.0 (the "License");  
you may not use this file except in compliance with the License.  
You may obtain a copy of the License at
```

```
http://www.apache.org/licenses/LICENSE-2.0
```

```
Unless required by applicable law or agreed to in writing, software  
distributed under the License is distributed on an "AS IS" BASIS,  
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
See the License for the specific language governing permissions and  
limitations under the License.
```

6.2.2 Python

The Python license is available by typing `license()` in an interactive session. More information is available from their website at <http://www.python.org>.

6.2.3 Tcl

The `Tcl` license is may be viewed at <http://www.tcl.tk/software/tcltk/license.html>.

Chapter 7

Installation

7.1 Download

SYMDIFF is currently released as a source code distribution. The software download is available at <http://github.com/devsim/symdiff>. Installation notes are in the `INSTALL` file in the source code distribution as well as on the project website.

7.2 Supported Platforms

Currently supported platforms are Linux, Apple Mac OS X, and Microsoft Windows.

7.3 Directory Structure

A `symdiff` directory is created with the following sub directories.

<code>bin</code>	contains the SYMDIFF invocation scripts
<code>doc</code>	contains SYMDIFF documentation
<code>lib</code>	contains runtime libraries
<code>examples</code>	contains example scripts
<code>testing</code>	contains example scripts using the Tcl interface

Bibliography

[ApacheSoftwareFoundation] Apache Software Foundation. Apache License, Version 2.0. URL: <http://www.apache.org/licenses/LICENSE-2.0.html>.