SYMDIFF

Version: 1.00

User Guide

Contact

Company Website:
http://www.devsim.com

Email:
info@devsim.com

Project Website:
http://github.com/devsim/symdiff

Copyright

Disclaimer

License

Please see *SYMDIFF* on page 13 for the license terms of this software.

Trademark

Contents

List of Tables

Chapter 1

Introduction

1.1 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 programatically.

Where to get help   If you need assistance, please contact us using the information in the front cover of this manual.

1.2 Using the Tool

1.2.1 Interactive Mode

The tool is invoked by typing `symdiff` 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 symdiff import *
```

We can then start executing **SYMDIFF** commands.

```
>>> symdiff ("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.

1.2.2 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.

1.2.3 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 UTF-8. In interactive mode, using Unicode is not recommended, based on issues in setting the environment properly for the Python interpreter.

1.3 Tcl version

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

```
% package require symdifftcl
```

Chapter 2

Syntax

2.1 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* φ *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* Section *Unicode* on page 2 *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
```

2.2 Basic Expressions

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

Table 2.1: Basic expressions involving unary and binary operators.

In Table 2.1, the basic syntax for the language is presented. An expression is 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.

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

Table 2.2: Predefined Functions.

2.3 Functions

In Table 2.2 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.

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

Table 2.3: Commands.

2.4 Commands

Commands are shown in Table 2.3. While they appear to have the same form as functions, they are special in the sense that they manipulate expressions immediated and are never present in the resulting expression. For example, note the result of the following command

```
>>> symdiff('diff(a*b, b);')
'a'
```

6

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

Table 2.4: Commands for user functions.

2.5 User Function Commands

Commands for specifying and manipulating user functions are listed in Table 2.4. 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.

```
>>> symdiff('define(f(x), 0.5*x);')
'f(x)'
```

The list after the function protoype 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)'
```

The chain rule is applied to ensure that the derivative is correct. This can be expressed as

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

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))'
```

Command	Description
`declare_model(name)`	declare model name
`define_model(name, exp)`	define model having expression
`model_value(name)`	retrieve expression for model

Table 2.5: Commands for Models.

2.6 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
```

Commands for specifying and manipulating models are listed in Table 2.5.

2.7 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)'
```

Chapter 3

Generating Source Code

3.1 `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:

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

The resulting output is then:

```
x
y
```

3.2 `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.

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

The resulting output is then:

```
b
d
```

3.3 `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.

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

The resulting output is then:

```
z
```

3.4 `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.

```
symdiff('define_model(x, simplify(y * z));')
symdiff('define_model(z, simplify(z * y));')
subexpression()
l = model_list()
for i in l:
  print("%s, %s" % (i, symdiff('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 4

Additional Information

4.1   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.

4.2   Licenses

4.2.1   SYMDIFF

```
Copyright 2012 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.
```

4.2.2   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.

4.2.3   Tcl

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

Chapter 5

Installation

5.1 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.

5.2 Supported Platforms

Currently supported platforms are Linux and Mac OS X. A Microsoft Windows version will be available in an upcoming release.

5.3 Directory Structure

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

`bin` contains the **SYMDIFF** invocation scripts

`doc` contains **SYMDIFF** documentation

`lib` contains runtime libraries

`examples` contains example scripts

`testing` contains example scripts using the Tcl interface.