Chapter 7. Symbolic computation

Symbolic computation allows exact manipulation of mathematical expressions, such as polynomials, equations, derivatives, integrals, and series. Unlike numerical computation, which gives approximate values, symbolic computation preserves the algebraic form of expressions. Scilab can perform symbolic calculations by integrating **Maxima**, a well-established computer algebra system, through the free **Symbolic Toolbox for Scilab**, implemented by **Jean-Francois Magni (2006)** and available at https://github.com/sengupta/scilab-maxima.git. Most functions of this toolbox have a syntax similar to the **Matlab Symbolic Toolbox**, although the functionality of this toolbox is limited to the functionality available in **Maxima**. This chapter is a practical guide to using **Symbolic** computation capabilities in Scilab.

1. Using the SciMax Symbolic Toolbox:

1.1. Symbolic Objects:

The Symbolic Toolbox defines a new Scilab data type called a **symbolic object**. Internally, a symbolic object is a data structure that stores a string representation of a symbol. The toolbox uses symbolic objects to represent symbolic variables, expressions, and matrices. Actual computations involving symbolic objects are primarily performed by **Maxima**, an open-source system for symbolic and numerical mathematics. Maxima was originally developed at the Massachusetts Institute of Technology as part of the **Macsyma** project in the 1960s. It was released under the **GNU** General Public License (**GPL**) in **1998** and continues to be actively maintained by a community of users and developers.

1.2 How to declare symbolic variables and handle symbolic expressions:

To create a symbolic expression that is a constant, you must use the **sym** command. For example, to convert the number **5** into a symbolic constant, enter:

```
--> f = sym('5')
  f =
  5
--> a=sqrt(f)
  a =
  5^(1/2)
```

Use the command **syms** to declare variables as symbolic. For example, the commands:

```
--> syms x y z
```

declare **x**, **y**, and **z** as symbols, not numbers. This allows you to create expressions like:

```
--> expr = x^2 + 3*y - z

 expr  =

z - 3*y + x^2
```

To substitute a symbolic variable with a value, you must use the **subs** command. For example:

```
--> subs(expr, x, 2)

ans=

4 + 3*y - z
```

To simplify an expression, use **simple** `command`. For example:

```
--> syms a b;
--> f = (a + b)^2 - (a - b)^2;
--> simple(f)
ans =
4*a*b
```

2. **Expanding and Functionalizing an Expression:**

The **expand** function develops expressions into their standard form. For example:

```
--> syms x;
--> f = (x + 2)^3;
-->expand(f)
ans=
x^3 + 6*x^2 + 12*x + 8
```

With a trigonometric expansion example:

```
-->syms x y;
-->f = sin(x + y);
--> expand(f)
ans =
 sin(x)*cos(y) + cos(x)*sin(y)
```

3. **Derivatives and Integrals of a Function:**

The **diff** function computes derivatives:

```
// First derivative
--> syms x
--> f = x^3 + 2*x^2 + x;
-->df = diff(f, x)
df =
  3*x^2 + 4*x + 1
```

```
//Higher-order derivatives:
Df2=diff(f, x, 2)  // Second derivative
df2=
     6*x + 4
//Partial Derivatives:
--> syms x y
-->f = x^2*y + sin(y);
-->dx=diff(f, x)//Partial derivative with respect to x:
dx=
   2*x*y
-->dy=diff(f, y)// Partial derivative with respect to y:
dy=
    x^2 + cos(y)
```

The **`integ`** function is used to integrate a symbolic expression:

```
// Indefinite integral:
--> syms x
--> f = x^2;
-->integ(f, x)
ans=
   x^3 / 3
// Definite integral:
--> integ(f, x, 0, 1)
ans=
1/3
```

4. Calculating the Taylor Expansion of a Function:

The **`taylor`** function for Taylor expansion. The syntax of this function is as follow :

```
--> taylor(f, x, x0, n) //  "f": function to expand,
"x": variable,  "x0": point of expansion,  "n": order.
```

For example:

```
// sin(x) around 0 up to order 5

-->syms x

-->taylor(sin(x), x, 0, 5)

ans=

 x - x^3/6 + x^5/120

// log(1 + x) around x = 0 up to order 4

-->syms x

-->taylor(log(1 + x), x, 0, 4)

ans=

    x - x^2/2 + x^3/3 - x^4/4
```