Symbolic Computation

Jiseok Chae

Department of Mathematical Sciences
KAIST

Week 10

Contents

Symbolic computation focuses on manipulating mathematical expressions, or sometimes more abstractly mathematical objects, more than numerical computations.

To perform symbolic computation, one needs a computer algebra system (CAS). Some examples of CAS are:

- Wolfram Mathematica
- Maple
- Sympy package in Python
- ...

MATLAB has an add-on toolbox for symbolic computation, the *Symbolic Math Toolbox*.[1]

There are no additional steps needed to activate the Symbolic Math Toolbox. Just type in the functions/commands provided by the Symbolic Math Toolbox, and MATLAB will load the toolbox automatically.

[1] You should have this installed on your computer if you have followed the installation guide properly.

Contents

The command `syms` is provided by the toolbox, and it generates a *symbolic variable*.

```
>> syms x
% A symbolic variable x is created.

>> syms r theta
% Two symbolic variables r and theta are created.

>> syms A [2 3]
% A symbolic 2x3 matrix A is created.
% The entries A1_1, A1_2, ..., A2_3 are also created.
```

The comments are not printed; they are there to help your understandings.

Let us examine how to manipulate functions.

```
>> y = x^3 - x - 1

y =

x^3 - x - 1

>> z = sin(x)

z =

sin(x)
```

Now the variable y contains the (symbolic) polynomial $x^3 - x - 1$, and the variable z contains the (symbolic) function $\sin(x)$.

The command `subs` performs a substitution.

```
>> subs(y, x, 3)

ans =

23

>> snx = subs(z, x, pi/4)

snx =

2^(1/2)/2
```

Note that the second result returned is the exact value $\sqrt{2}/2$, instead of the (approximated) numerical value 0.7071. This is one of the features of symbolic computation.

To see the result in the usual numerical form, use `eval` or `vpa`.

```
>> eval(snx)

ans =

    0.7071

>> vpa(snx)

ans =

0.70710678118654752440084436210485

>> vpa(snx, 50)

ans =

0.70710678118654752440084436210484903928483593768847
```

The function `sym` can be used to change quantities into symbolic objects.

```
>> x = 123;
>> sx = sym(x)

sx =

123

>> sx^21

ans =

77269364466549865653073473388030061522211723
```

Let us see another example of how symbolic variables can be manipulated.

Have you ever noticed, for any $n \in \mathbb{Z}$, that $\cos(n\theta)$ can be written as a "polynomial" of $\cos(\theta)$?

$$\begin{aligned}
\cos(0) &= 1 \\
\cos(\theta) &= \cos(\theta) \\
\cos(2\theta) &= 2\cos^2(\theta) - 1 \\
\cos(3\theta) &= 4\cos^3(\theta) - 3\cos(\theta) \\
\cos(4\theta) &= 8\cos^4(\theta) - 8\cos^2(\theta) + 1 \\
&\vdots
\end{aligned}
\qquad
\begin{aligned}
T_0(x) &= 1 \\
T_1(x) &= x \\
T_2(x) &= 2x^2 - 1 \\
T_3(x) &= 4x^3 - 3x \\
T_4(x) &= 8x^4 - 8x^2 + 1 \\
&\vdots
\end{aligned}$$

The polynomials $T_n(x)$ are called *Chebyshev polynomials*.

In addition, Chebyshev polynomials satisfy the recurrence

$$\begin{aligned} T_0(x) &= 1, \\ T_1(x) &= x, \\ T_{n+2}(x) &= 2xT_{n+1}(x) - T_n(x). \end{aligned}$$

From the definitions using cosines, one can compute $T_n(x)$ as follows.

cheby_cos.m

```
function T = cheby_cos(n, x)
    T = cos(n * acos(x));
end
```

```
>> syms x
>> cheby_cos(4, x)

ans =

cos(4*acos(x))

>> simplify(cheby_cos(4, x))

ans =

8*x^4 - 8*x^2 + 1
```

From the recurrence relation, one can also compute $T_n(x)$ as follows.

cheby_rec.m

```
function T = cheby_rec(n, x)
    if n == 0
        T = 1;
    elseif n == 1
        T = x;
    else
        T = 2*x*cheby_rec(n-1, x) - cheby_rec(n-2, x);
    end
end
```

```
>> syms x
>> simplify(cheby_rec(4, x))

ans =

8*x^4 - 8*x^2 + 1
```

Contents

We can compute limits of symbolic expressions using `limit`.

```
>> limit(sin(x)/x, x, 0) % lim_{x→0} sin x / x

ans =

1

>> limit((1+1/x)^(x), x, inf) % lim_{x→∞}(1 + x)^x

ans =

exp(1)
```

We can differentiate a variable w.r.t. another variable using `diff`.

```
>> diff(y, x) % computes dy/dx

ans =

3*x^2 - 1

>> diff(z, 2, x) % computes d^2/dx^2 z

ans =

-sin(x)
```

Expressions such as `diff(y)` are also possible, but this can cause confusion when a function depends on multiple variables.

We can integrate a variable w.r.t. another variable using `int`.

```
>> int(y, x) % computes ∫ y dx = ∫(x^3 − x − 1) dx

ans =

-(x*(- x^3 + 2*x + 4))/4

>> int(z, x, 0, pi) % computes ∫_0^π z dx = ∫_0^π sin x dx

ans =

2
```

Of course, not all integrals can be computed. MATLAB only tries its best to compute them.

To solve an equation, you can use the function `solve`, but some minor ad-hoc modifications are often required.

```
>> solve(x^3 - x - 1 == 0, x) % same as solve(y==0, x)

ans =

root(z^3 - z - 1, z, 1)
root(z^3 - z - 1, z, 2)
root(z^3 - z - 1, z, 3)
```

MATLAB does not like solving high degree polynomial equations.
To force MATLAB to compute the solutions in an explicit form, try `solve(x^3 - x - 1 == 0, x, 'MaxDegree', 3)`, then MATLAB will solve up to cubic equations.

To solve an equation, you can use the function `solve`, but some minor ad-hoc modifications are often required.

```
>> solve(sin(x) == x+1, x) % unable to solve explicitly
Warning: Unable to solve symbolically. Returning a numeric
    solution using vpasolve.
> In sym/solve (line 304)

ans =

-1.9345632107520242675632614537689
```

MATLAB falls back to the numerical solver if symbolic computation fails. As in the warning message, `vpasolve` can solve the equation numerically. Compare with the result of `vpasolve(sin(x) == x+1, x)`.

Thank you!