ECEN203 Circuit Analysis
Symbolic Computation with Matlab

May 2, 2017

Matlab has a powerful toolbox for symbolic computation. Symbolic computation enables describes the capacity to deal directly with mathematical expressions, and lets us do things like differentiate or laplace transform general expressions.

There are times when this is a useful alternative to "normal" numerical computation of Matlab (or other languages), so it is useful to have at least a basic understanding of the language's capabilities. In particular, the symbolic toolbox provides a great tool for checking your algebra when working through problems by hand. We encourage you to use it freely for that purpose, but please don't use it as a crutch to avoid working through things yourself, as that will only lead to trouble later.

This document merely serves as a quick introduction to the most pertinent capabilities for us. It is by no means intended as an exhaustive exploration of the symbolic toolbox. Work through the examples and feel free to explore further options and commands by consulting the matlab help. There is no need to submit anything.

1 Expression manipulation

Before symbolic computation can be used, we must first tell Matlab which symbols it should treat as normal variables, and which it should treat as abstract symbols. In matlab everything is assumed to be a variable unless we say otherwise using the `syms` or `sym` command.

```
>> syms w R C
```

At any time you can use the `whos` command, which will report all of the defined variables and tell which of them are symbols.

Once you have defined some symbols, you can use them to create more complex expressions. For example, let's find the impedance of a parallel resistor and capacitor. We can then use the toolbox to rearrange the expression into a nicer form.

```
>> Z = 1/((1/R)+w*C)
>> Z2 = simplify(Z)
```

The `simplifyFraction` command is generally quicker than `simplify` when dealing with fractions like this, though you won't notice the difference with this simple a fraction. When dealing with more complex expressions you might like to try both, as they sometimes return answers that have slightly different forms.

In any case, we can use the new expression to perform other calculations. For example, let's find the current that would flow through our parallel RC combination if we were to apply a 3 V ac signal.

```
>> i = 3/Z
```

Notice that if you `whos` to examine the new variable `i`, that it has automatically been created as a symbolic expression.

The expressions produced by the symbolic toolbox are sometimes hard to understand, due to the deep nesting of parentheses or divisions. Try using the `pretty` command can help understand such expressions. This command attempts to use the matlab command line to present an expression in form similar to a hand written representation.

```
>> pretty(i)
>> pretty(Z)
>> pretty(Z2)
```

1.1 Substituting for variables

If you know particular numerical values for some symbol in an expression, then the `sub` value let's you do the appropriate substitution. For example, the following examples find the current if $R = 10$ kΩ, or $f = 1$ kHz respectively.

```
>> subs(i, R, 10e3)
>> subs(i, w, 2*pi*1000)
```

Exercise Use the `sub` command to find the current flowing through the RC combination discussed above if $R = 10$ kOhm, $C = 100$ nF and $f = 1000$ Hz.
Hint: You can use `subs(f, [var1 var2 ...], [val1 val2 ...])` to substitute values for multiple variables simulataneously.

1.2 Partial fractions expansion and related matters

A particularly useful command for us is `partfrac`, which performs partial fractions expansion.

```
>> clear
>> syms s
>> Y = (3 * s + 13) / (s^2 + 4 * s + 3)
>> Yp = partfrac(Y)
```

One of the quirks of the toolbox is apparent if we compare the two forms for Y in the code above. One would certainly hope that `Y-Yp` would be equal to zero. Try it! The use of `simplify` can help with this situation. Try `simplify(Y-Yp)`

You can use the `collect` to move in the other direction. In this case the command has the effect of gathering the expression over a common denominator.

```
>> Y2 = collect(Yp)
```

In general `collect` collects the coefficients of various powers of a variable. If necessary you can specify which variable should be used as the variable of interest. Contrast the outputs of the following two uses of `collect`.

```
>> syms s a b
>> Y= 2*a*s^2 + a^2*s^2 + 3
>> collect(Y)
>> collect(Y, a)
```

1.3 Plotting symbolic expressions

The `ezplot` command allows you to quickly plot algebraic expressions.

```
>> y = x^2 + 9*x + 7
>> ezplot(y)
```

You can then modify the resulting figure as you wish, using commands like `xlabel`, `title`, `xlim` and so forth. When you want to constrain the x-axis limits there is a shortcut that let's you include the desired axis limits within the `ezplot` call.

```
>> y = x^2 + 9*x + 7
>> ezplot(y, [0, 10])
```

Exercise Recreate the various mode shapes included in the notes using the symbolic toolbox. That is, plot the modes that correspond to signals $s = \mathrm{e}^{\sigma t}$, $s = \mathrm{e}^{\mathrm{j}\omega t}$ and $s = \mathrm{e}^{(\sigma+\mathrm{j}\omega)t}$. You will need to substitute appropriate values into the general expression to produce the plots. PLot the responses only for $t > 0$.

1.4 Solving equations

Matlab can solve a symbolic equation for you. Notice that you need to use `==` to express the "equals" sign in such expressions. If you omit the equals part, then Matlab will assume that you wish to solve for the specified expression being zero. Sadly it is not always able to solve complicated problems (like cubic polynomial problems) as shown by this example.

```
>> solve(x^2 + 4*x + 20)
>> solve(x^3 + 4*x + 20)
>> solve(x^3 + 6*x^2 +11*x + 6 ==  0)
```

Both the `solve` command and the related `factor` command are useful for when dealing with identifying poles of a transfer function of higher order polyomials in preparation for partial fractions expansion.

```
>> factor(x^3 + 2*x + 20)
```

The symbolic toolbox can also can deal with the soution of systems of equations:

```
>> [x,y]=solve (x + y == 8, x - y == 2)
```

2 Symbolic Laplace and Inverse Laplace transforms

You can quickly find the Laplace and inverse Laplace transforms using Matlab's `laplace` and `ilaplace` commands. Use `heaviside(t)` to define a unit step and `dirac` to specify a dirac delta function.

For example, let's apply a signal $y(t) = 3\mathrm{u}(t-2) + \sin(\omega t)$ to a system having transfer function $G(s) = \frac{3}{s+3}$ and find the output signal.

```
>> clear
>> syms w t s
>> X = laplace(3 * heaviside(t-2) + sin(w * t))
>> G = 3/(s+3)
>> x= ilaplace(X*G)
```

3 Symbolic calculus

If you have a suitably defined symbols, then you can perform symbolic differentiation and integration. Try the following to see how this works:

```
>> clear
>> syms a x
>> diff(a*x^2)
>> int(a*x^2)
```