SCHOOL OF PHYSICS, ASTRONOMY & MATHEMATICS

4PAM1008 MATLAB

5 – Symbolic Computing

Dr Richard Greenaway

5 Symbolic Computing (Symbolic Math Toolbox)

5.1 Introduction

Back in workshop 2 we introduced the idea of function *overloading*, where a particular MATLAB function behaves differently depending on the type of input. In the example we used the function `diff` with a string input defining the function x^2 and the output was $2x$, ie the differential.

```
>> diff('x^2')

ans =

2*x
```

This is an example of a symbolic computation using the *symbolic toolbox*.

The Symbolic Math toolbox is a tool to help you do *analytical* maths, simplifying and solving equations, differentiation, integration, limits and so forth. MATLAB contains a very powerful user interface for doing symbolic math called the **MuPAD Notebook** (MuPAD is the symbolic computation 'engine') which generates fully formatted mathematical expressions . While you can use the MATLAB command window with the symbolic toolbox in exactly the same way as you would for doing numerical computation, the MuPAD Notebook provides a much more elegant and useful interface for the user. However, the syntax and 'modus operandi' are somewhat different to MATLAB so, to keep things consistent, we will use the symbolic math toolbox through the, now familiar, MATLAB interface. Once you are familiar with using the toolbox you might want to explore the MuPAD Notebook for yourself.

5.2 Creating Symbolic Expressions

When doing symbolic maths, we are not dealing with numbers but symbolic variables, '*x*', '*y*' etc. Symbolic objects are a special MATLAB data type introduced by the Symbolic Math Toolbox software. They allow you to perform mathematical operations in the MATLAB workspace analytically, without calculating numeric values.

Before you can create and use a symbolic expression you must create the symbolic objects that will be used to make up the expression

There are two ways to declare variables. You can use the `syms` command or the `sym` function. `syms` allows you to declare one or more variables in a single call, eg

```
≫ syms x y
```

If you look in the workspace window you will see that the variables are defined as type *sym* and have a different icon associated with them. You will also notice that they have a dimension of 1-by-1 which is the default for symbolic objects.

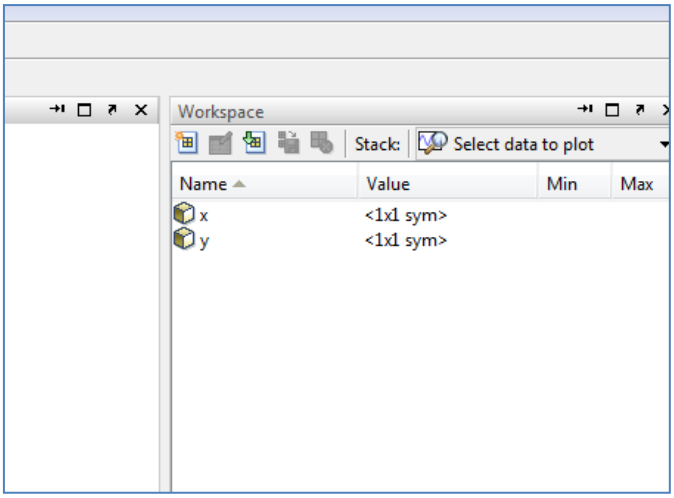


Figure 1 Symbolic objects defined in the MATLAB workspace

The `sym` command can only create one symbolic object at a time and you must use parentheses and place the object in quotes.

```
x = sym('x')
```

For simple variables this doesn't look so useful but it allows you to create entire symbolic expressions and 'symbolic numbers'.

So, for instance.

```
>> a = sym('2');
>> sqrt(a)

ans =

2^(1/2)
```

Here we see another instance of overloading. `sqrt` does not calculate the square root of 2 but returns the symbolic 'value' $2^{½}$ which can be incorporated into a symbolic expression

You can manipulate the symbolic objects according to the usual rules of mathematics. For example:

```
>> syms x y;
>> x + y + x

ans =

2*x + y
```

Having defined symbolic variables you can now create expressions by typing them at the command line, for example.

```
>>
>> syms a b c x
>> f = (x^2 - 3)*(x + 4)

f =

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

>>
>>
>> y = a*sin(b/2) - 3*cos(a)

y =

a*sin(b/2) - 3*cos(a)
```

Example. 5-1

5.3 Simplifying Expressions

The toolbox provides a number of functions which can do various kinds of algebraic manipulation.

`collect`	Collect coefficients
`expand`	Symbolic expansion of polynomials and functions
`horner`	Transforms a polynomial into its nested (Horner) representation
`factor`	Factorisation of polynomials
`simplify`	A general purpose function for simplifying expressions.
`simple`	This is a slightly odd function which attempts to find a representation of an expression in the fewest possible characters.

Consider the following polynomial.

$$x^3 + 4x^2 - 3x - 12$$

We can generate it in MATLAB as follows

```
>> syms x
>> f = x^3 + 4*x^2 - 3*x - 12

f =

x^3 + 4*x^2 - 3*x - 12
```

We can factorise it using `factor`.

```
>> g = factor(f)

g =

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

Yes, it's the same polynomial as we generated in Example. 5-1. We can return to the original representation by expanding the factorised form.

```
>> expand(g)

ans =

x^3 + 4*x^2 - 3*x - 12
```

If we had used `simplify` on our function `f`, it would have given us the same result as `factor`.

```
>> simplify(f)

ans =

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

What would `collect` do to `f`?

```
>> collect(f)

ans =

x^3 + 4*x^2 - 3*x - 12
```

Nothing of course, the terms are already collected. But on `g`?

```
>> collect(g)

ans =

x^3 + 4*x^2 - 3*x - 12
```

In this trivial case, it's the same as `expand`.

It's pretty straight forward really. The following exercise will provide some practice.

Exercise 5-1

1. Simplify, $3\cos^2 4x + 3\sin^2 4x$

2. Simplify, $e^{(-3\ln(x)+1)}$

3. Simplify, $ln\left(\frac{e^{2x}}{e^{3y}e^{x}}\right)$
   This one doesn't quite work. You'll see that the result requires one further trivial step to get it into its simplest form.

4. Expand $(u+v)^5$

5. Expand $e^{(2a-b)}$

6. Factorise $x^2 - 4y^2$

7. Find the roots of the following equation.
   $f(x) = x^3 - 2x^2 - 5x + 6$

5.4 Solving Equations

In question 7 for Exercise 5-1 you were asked to find the roots (that is solve) a cubic equation. In fact there is a dedicated function, `solve`, which attempts to solve a symbolic expression *S* for which *S* = 0.

So let's use solve to try to find the roots of the cubic presented in question 7.

```
>> f = x^3-2*x^2-5*x+6

f =

x^3 - 2*x^2 - 5*x + 6

>> solve(f)

ans =

 -2
  1
  3
```

Let's see what happens when we solve a general quadratic equation.

```
>> syms x a b c
>> f = a*x^2 + b*x + c

f =

a*x^2 + b*x + c

>> solve(f)

ans =

 -(b + (b^2 - 4*a*c)^(1/2))/(2*a)
 -(b - (b^2 - 4*a*c)^(1/2))/(2*a)
```

The answer is the standard formula you would find in your maths handbooks.

The `solve` function assumes that you want to solve for *x*. We could solve the quadratic for *b* as follows.

```
>> b = solve(f,b)

b =

-(a*x^2 + c)/x
```

All the above examples assume that $f(x) = 0$. If you you have a function of the form $f(x) = g(x)$ then you must either re-arrange the function to equal zero if possible or pass the expression to `solve` as a *string*.

Consider now the equation $x^2 + 3x - 1$. The solution to this equation is…

```
>> solve(x^2+3*x-1)

ans =

 - 13^(1/2)/2 - 3/2
   13^(1/2)/2 - 3/2
```

Note that the results are constants, ie values but represented in 'symbolic' form. We can easily convert them to numbers as follows.

```
>> double(ans)

ans =

   -3.3028
    0.3028
```

In workshop 3 you saw how to solve systems of equations using matrix division. You can use the `solve` function to do the same thing. Here is the example given in section 3.4.3 again.

Solve

$$x + 3y = 7$$
$$3x - 2y = -12$$

```
>> syms x y
>> f1 = x +3*y - 7

f1 =

x + 3*y - 7

>> f2 = 3*x - 2*y + 12

f2 =

3*x - 2*y + 12

>> s = solve(f1,f2)

s =

    x: [1x1 sym]
    y: [1x1 sym]
```

```
>> s.x

ans =

-2

>> s.y

ans =

3
```

⚠ **NOTE**

The answer in the above example is returned as a *struct*. This is a special type which can group multiple variables together. So, the *struct* `s` contains two variables, `x` and `y`. They are symbolic variables just like any other but to access them, you must use the syntax shown.

Exercise 5-2

Solve the following equations for *x*.

1. $3x^2 = 4$

2. $y = 2x^3 - 4$
   How many real roots does it have?
   Generate numeric values for this function for $-2 \leq x \leq 2$ in steps of 0.1 and plot the data to verify the result.

3. $x^2 + ax + 3 = 0$

Solve the following simultaneous equations.

4. $x^2 + y^2 = 2$
   $x + y = 1$

5. $3x - 2y = 7$
   $6x - 4y = 4$
   You've done this one before, numerically (Ex. 3.6, Q2).

6. The distance moved by an object is measured at two points. After 3 seconds it has moved 66m and after 5 seconds it has moved 160m

   Using the equation

   $$s = ut + \frac{1}{2}at^2,$$

   determine the initial velocity *u* and the acceleration *a*.

5.5 Differentiation

Differentiate an expression using the function `diff`.

```
>> syms x
>> diff(x^3)

ans =

3*x^2
```

It's as simple as that.

You do not have to use the symbolic variable, `x`, any name can be used.

```
>> syms theta
>>
>> diff(3*cos(2*theta))

ans =

(-6)*sin(2*theta)
```

To compute the second order differential you could simply call `diff` twice.

```
>> diff(diff(3*cos(2*theta)))

ans =

(-12)*cos(2*theta)
```

However, it is easier to calculate the n^{th} order differential as follows

```
diff(S,n)
```

where `S` is the expression to differentiate.

```
>> diff(3*cos(2*theta),2)

ans =

(-12)*cos(2*theta)
```

5.5.1 Differentiating Expressions with Several Variables

When you have an expression of more than one variable you must specify which variable you wish to differentiate with respect to.

You can do this simply as shown in the following example.

```
>> syms x t
>> f = t*cos(t*x)
```

```
f =

t*cos(t*x)

>> diff(f,x)

ans =

-t^2*sin(t*x)
```

To differentiate w.r.t `t`.

```
>> diff(f,t)

ans =

cos(t*x) - t*x*sin(t*x)
```

Exercise 5-3

1. Determine the following and simplify if possible

   a) $\frac{d}{dx}(e^{ax}\sin bx)$

   b) $\frac{\mathrm{d}}{\mathrm{dx}}\left(\mathrm{x}^2-\sqrt{\mathrm{x}}-3\right)$

   c) $\frac{\mathrm{d}}{\mathrm{dx}}((\mathrm{x}^3+4)^2(\mathrm{x}^2+\mathrm{e})^3)$

2. Show that the function $y = Ae^{-2x}\cos(3x + b)$, where *A* and *b* are constants, satisfies the equation

$$\frac{d^2y}{dx^2} + 4\frac{dy}{dx} + 13y = 0$$

5.6 Integration

The function **`int`** can be used to determine the indefinite integral of a symbolic function.

```
>> syms x
>> int(sin(x))

ans =

-cos(x)
```

To determine a *definite* integral between limits *a* and *b*, use the following syntax.

```
int(function, a, b)
```

For example,

$$\int_0^1 t^2 dt$$

```
>> syms t
>> int(t^2,0,1)

ans =

1/3
```

If one or both limits are infinite then use the special value `inf`.

For example.

$$\int_1^\infty \frac{1}{t^2} dt$$

```
>> syms t
>> int(1/t^2,1,inf)

ans =

1
```

Remember that some integrals do not have an analytic solution.

$$\int_1^2 \sin x^2 e^{\sqrt{x}}\, dx$$

```
>> syms x
>> int(sin(x^2)*exp(sqrt(x)),1,2)
Warning: Explicit integral could not be found.

ans =

int(sin(x^2)*exp(x^(1/2)), x = 1..2)
```

Later on you will see how to numerically integrate this function to obtain an approximate answer.

Exercise 5-4

1. Determine the indefinite integral $\int (2+3x)^9\, dx$

2. Evaluate

$$\int \frac{dt}{1+t+t^2}$$

The result is quite difficult to read. Try using the function `pretty` *to make the result more readable.*

3. Show that

$$\int_0^1 \frac{\tan^{-1} x}{\sqrt{x^3}} dx = 1.897$$

4. Evaluate the following definite integrals.

$$\int_2^4 \frac{2x+3}{x(x-1)(x+2)} dx$$

$$\int_4^9 \frac{dx}{(\sqrt{x}-1)\sqrt{x}}$$

5.7 Using Symbolic Results in Numerical Computations

5.7.1 Working with Symbolic Constants

We will now look at how symbolic expressions can be used in numerical computations.

We have seen already in some of the previous examples that even when a result from a symbolic computation is a constant (ie numeric value) the result is represented in symbolic form.

Consider again the previous example where we solved the following equation.

$$x^2 + 3x - 1$$

```
>> syms x
>> r = solve(x^2+3*x-1)

r =

 - 13^(1/2)/2 - 3/2
   13^(1/2)/2 - 3/2
```

The result returned is

$$\pm\frac{\sqrt{13}-3}{2}$$

In fact you can use the result directly in a numerical expression, for instance.

```
>> 4 * r(1)

ans =

- 2*13^(1/2) - 6
```

Notice that the result is still 'symbolic'. We could of course convert the symbolic value to a standard numeric using `double` as was demonstrated earlier.

```
>> val = double(r(1))

val =

   -3.3028

>> 4 * val

ans =

  -13.2111
```

5.7.2 Generating Functions

You can convert a symbolic expression into a standard MATLAB function using `matlabFunction` which can then be used to process data like any other function.

`matlabFunction` can generate two types of output. The first is a standard m-file and the second is known as a *function handle*. The latter is a slightly more advanced concept but it is very easy to use. We can demonstrate with the following simple example.

Let's define a simple symbolic expression.

```
>> syms x
>> y = 2*sin(x)

y =

2*sin(x)
```

If we want to calculate `y` for various values of `x` we must generate the MATLAB function equivalent.

```
>> myFunc = matlabFunction(y)

myFunc =

    @(x)sin(x).*2.0
```

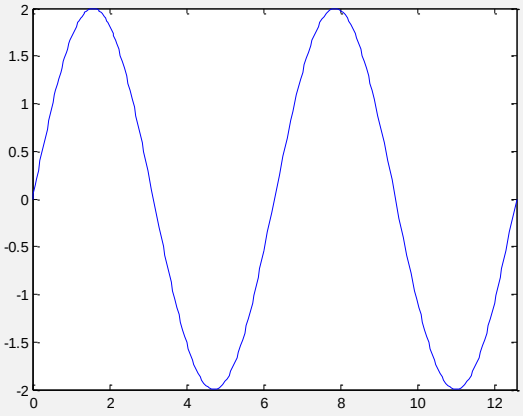
Here we have used `matlabFunction` to create a function handle called `myFunc`. The output might look a little complicated but all you need to do is treat `myFunc` as if it were a standard MATLAB function.

So for instance, you can calculate a value.

```
>> myFunc(pi/2)

ans =

     2
```

Or plot the function.

```
>> figure;
>> fplot(myFunc,[0 4*pi])
```



So you can see that it is very easy to use the results from a symbolic computation with real data.

The following exercise will give you some practice.

Exercise 5-5

Earlier in this workshop you saw an example of an integral for which MATLAB can't find an analytical solution.

$$\int_1^2 \sin x^2 e^{\sqrt{x}}\, dx$$

1. Generate the symbolic expression

$$y = \sin x^2 e^{\sqrt{x}}$$

2. Convert this to a numeric function which can be executed by via a function handle, `yFunc`.

3. Use `fplot` to plot this function over the domain $0 \leq x \leq 6$.

4. Use the `quad` function to numerically evaluate the integral over the *integral* range specified.

   `quad` is one of a suite of MATLAB functions which can be used to numerically evaluate a definite integral. Use the help system to find the correct syntax.

   Answer: 1.4825

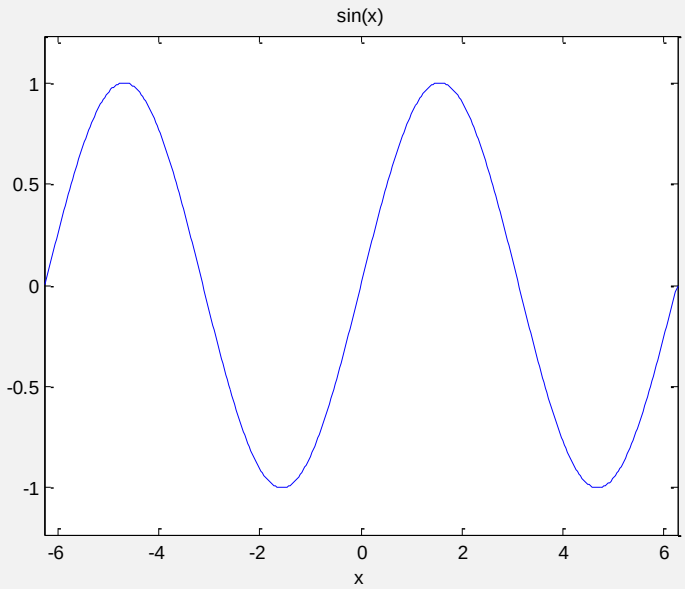
5.8 Plotting Symbolic expressions

In the previous section we could plot the symbolic expression using `fplot` only after converting it with `matlabFunction` . If you tried plotting the symbolic expression directly with `fplot` you would get an error.

However, there are various easy-to-use plotting functions which provide a very quick way to visualise expressions, including symbolic expressions. These functions are named **ez***xxx* (**ezplot**, **ezpolar** …).

For example.

```
>> syms x
>> f = sin(x)

f =

sin(x)

>> ezplot(f)
```

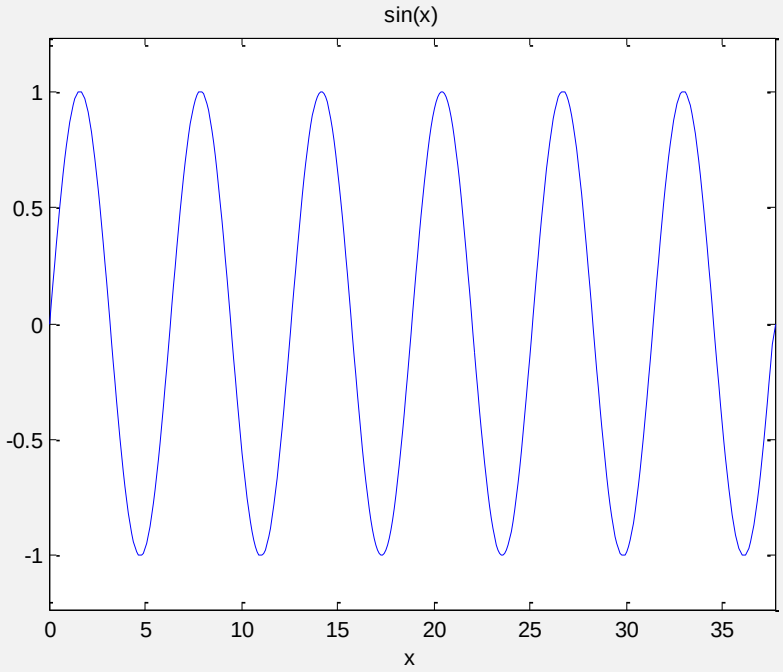


Notice how it very helpfully creates a title from the expression.

By default `ezplot` will plot the function over the domain $-2\pi < x < 2\pi$, however you can always set your own using the following syntax.

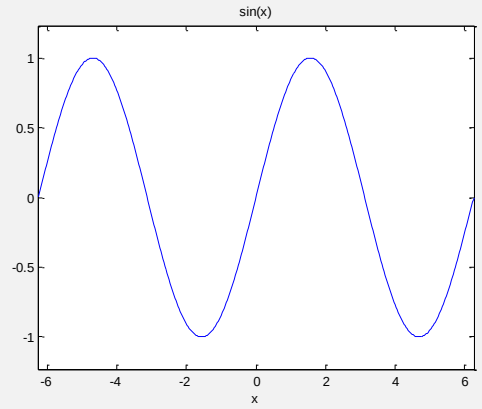
```
ezplot(func, [min max]);
```

```
>> syms x
>> f = sin(x)

f =

sin(x)

>> ezplot(f, [0 12*pi])
```

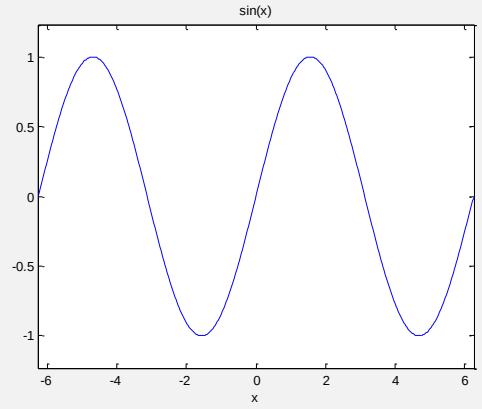


`ezplot` also works with function handles and strings.

```
>> fhandle = matlabFunction(f)

fhandle =

    @(x)sin(x)

>> ezplot(fhandle)
```




```
>> ezplot('sin(x)')
```



You can also specify a y-range using the following syntax

```
ezplot(fun,[xmin,xmax,ymin,ymax])
```

IMPORTANT

There appears to be a bug when attempting to plot a symbolic expression with a custom y-range. It works fine when the function is represented by a numeric function/function handle or a string. So if you wish to set the Y-axis limits you must first convert your function using `matlabFunction` and then pass the function handle to `ezplot` (or any of the numeric plotting functions if you wish).

Alternatively you can manually adjust axis limits on any graph using the zoom options in the figure.

Exercise 5-6

Find the stationary (zero gradient) values for the following function

$$f(x) = 2x^3 - 5x^2 + 4x + 1$$

1. First generate the expression and plot it to see what the function looks like.

2. Determine the differential and solve it to find the solution

3. Plot $\frac{df}{dx}$ to confirm the solution visually
   You will need to zoom in to find the crossing points.