Lecture 5 for Math 398 Section 952: Caculus and Differential Equations

Thomas Shores
Department of Math/Stat
University of Nebraska
Fall 2002

Most of the functions that we study in this lesson can be found in one category of the help files. Start by issuing a "helpwin" command and clicking on "matlab/funfun".

Calculus

Matlab provides many numerical tools that are useful in solving problems of calculus. Note the term "numerical." There is an add-on package that we will discuss briefly at the end of this lesson, but most of the applications here provide numerical (and therefore approximate) answers to certain problems. This is good enough for most purposes, including high quality graphics.

Polynomials and polynomial interpolation.

Matlab has some special functions for polynomials. First, there is the matter of storage. Matlab views a polynomial as having the form $c_1x^n + c_2x^{n-1} + \cdots + c_nx + c_{n+1}$ and stores the vector of coefficients $[c_1, c_2, \ldots, c_{n+1}]$ of coefficients of the polynomial. Fire up Matlab and execute the following commands (sans comments)

```
> p = [1 -2 1 3] % polynomial x^3-2*x^2+x+3
> q = [1 0 -1] % polynomial x^2-1
> x = linspace(-1,1,100);
```

Notice polyval evaluates the polynomial at each coordinate of a matrix argument

```
> plot(x, polyval(p,x));
> hold on, grid
> plot(x,polyval(q,x))
> % differentiation is also easy
> polyder(p)
> A = [1 2;0 3]
```

Here is an example of evaluating a polynomial at a matrix argument

```
> polyvalm(p,A)
```

An example of long division and polynomial multiplication :

```
> [quot, r] = deconv(p,q)
> conv(quot, q)
> ans + r
```

Finally, here's an entertaining application of these tools: find a "best fit" cubic to the function $x*\sin(x)$, $0 \leq x \leq 2\pi$ by selecting 4 points to thread your cubic through. Actually, you can thread a cubic through as many data points as you want in Matlab, which does a least squares fit of data to polynomial coefficients.

```
> t = linspace(0,2*pi,100);
> plot(t, t.*sin(t));
> [x,y] = ginput(4) % pick four points
> p = polyfit(x,y,3)
> plot(t, polyval(p,t))
```

If you don't like the fit, select other points and another polynomial. If your figure gets too busy, use Figure Window tools to erase earlier guesses.

Root Finding and optimization.

Matlab has a number of useful commands for these purposes. Do the following:

```
> clear
> fzero('x*sin(x)',1)
> fzero('x*sin(x)',0.5)
> fzero('x*sin(x),0.01) % note:  fzero only looks for sign changes
```

On the other hand, we could do this:

```
> [x, y] = fminbnd('(x*sin(x))^2',1,4)
> [x, y] = fminbnd('(x*sin(x))^2',-1,3)
```

We can also minimize functions of more than one variable. Edit the file fcn so that the return value is x(1)^2 + y(1)^2 - x(1)*y(1). Then enter:

```
> [x, fval] = fminsearch(@fcn, ones(2,1))
```

Quadrature.

Numerical integration is easy in Matlab as well.

```
> clear
> x = 0:.1:5;
> plot(x, humps(x))
> quad(@humps, 0, 5)
```

Now create a function fcn with fcn(x)=x.*log(x).

```
> quad(@fcn, 1,3)
```

Differential Equations

Many tools are available here. We will use just a few. In this lesson we'll focus on some general purpose numerical method for solving odes. In the next lesson, we'll examine some problems that require more specialized methods.

Ordinary differential equations.

Start by creating a function of two variables in the .m file fcn.m, where $fcn(t, x) = -x - 5e^{-t}\sin(5t)$. Now execute the commands

```
> tspan = [0 3]
> yzero = 1
> [t, x] = ode45(@fcn, tspan, yzero);
> plot(t, x);
```

Now close the plot and examine the error function $x - e^{-t}\cos(5t)$ (you can check that the exact solution is $x(t) = e^{-t}\cos(5t)$.)

```
> plot(t, x - exp(-t).*cos(5*t))
> norm(x - exp(-t).*cos(5*t),inf) % gives the largest coordinate in absolute value
```

Next, we examine a second order system. Consider the ode $y'' + y - y^2 = 0$, where $y = y(t)$. First, we convert this system to a first order vector system by making the substitutions $x_1 = y$ and $x_2 = y'$ to obtain

$$\begin{aligned} \frac{dx_1}{dt} &= x_2 \\ \frac{dx_2}{dt} &= -x_1 + x_1^2. \end{aligned}$$

Now modify the function file fcn.m so that the value returned is [x(2),-x(1)+x(1)^2]. Enter the following to get a graph of a few solutions and a direction field.

```
> tspan = [0 10]
> [ta, xa] = ode45(@fcn, tspan, [0.5 0]);
> [tb, xb] = ode45(@fcn, tspan, [0.75 0]);
> plot(ta, xa,)
> plot(tb, xb, 'r:')
```

I have written a simple routine for plotting direction fields. Use it here

```
> dirfield(0, -1:0.2:1, -1: 0.2:1, @fcn)
```

Symbolic Math and Matlab

Matlab add-on packages are called "toolboxes." Some of these are freely available on the net and others are marketed, mostly by MathWorks (the company that produces Matlab.) One particularly useful one is the symbolic toolbox, This toolbox contains the Maple kernel, which performs all the symbolic and variable precision computations of which the toolbox is capable. FYI, the Toolbox is included with the Student Edition of Matlab, but not with commercial versions. The following will look familiar to Maple users:

```
> clear
```

```
> syms a b c x % declare these variables to be symbols
> y = solve(a*x^2+b*x+c,x) % output y is a vector of symbols
> % now let’s check our answer
> a*y.^2 + b*y + c
> simplify ans
> % now for some calculus
>y = int(’sqrt(tan(2*x))’) % try an integral
> prettyprint(ans) % print it out in a more readable fashion
> double(y) % convert it to a float
> diff(y,x) % check our answer
```

End of Lesson