21-124 MODELING WITH DIFFERENTIAL EQUATIONS

LECTURE 3: MATLAB, M-FILES AND NUMERICAL METHODS

1. Working in Matlab

You can use Matlab as a fancy calculator by simply typing in the expression you want to evaluate, using

`+ - * / ^`

as is standard practice. Matlab also has a number of standard commands, like `sqrt()`, `sin()`, `cos()` and `exp()`.

One of the things we will frequently want Matlab to do for us is plot graphs. In order to do so, we will need to use vectors. A short vector can be defined by typing in the entries. Entering `v=[1,2,3]` creates the row vector

$$v = [\ 1 \quad 2 \quad 3\ ].$$

You can also use spaces instead of commas. Entering `v=[1;2;3]` creates the row vector

$$w = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}.$$

You can achieve the same thing by entering `w=v'`.

The command `t=3:.2:7` creates the row vector `t` whose entries are

$$[\ 3 \quad 3.2 \quad 3.4 \quad \dots \quad 6.8 \quad 7\ ].$$

Now, if you type `y=sin(t)` you will get the numerical equivalent of the vector

$$y = [\ \sin(3) \quad \sin(3.2) \quad \sin(3.4) \quad \dots \quad \sin(6.8) \quad \sin(7)\ ].$$

Now you can plot a graph of the sin function by using the `plot` command. Enter `plot(t,y)`, and Matlab will produce a separate window with the graph in it. The command `plot(u,v)` takes the vectors u and v as input. These vectors must be the same length. Matlab plots the points $(u_1, v_1), (u_2, v_2), \dots$ and connects them with straight lines.

If you want to plot $y = t^2$, you need to define y using

`y=t.^2`

The dot before the carrot tells Matlab to square each entry separately, rather than using matrix multiplication.

There is an optional third argument for the `plot` command. If you enter `plot(t,y,'r')` the graph will be plotted in red and `plot(t,y,'m')` produces a magenta graph. On the other hand, `plot(t,y,'--')` produces a graph with a dashed line. Experiment with some of the other options: c, y, k, g, w, o, x, +, *, -. and :. They can be combined as well, like `plot(t,y,'g*:')`.

You can get a complete description of the `plot` command by entering `help plot`. In fact most of Matlab's commands have a help file.

One last thing is MATLAB's `diary` command. When you enter `diary on`, MATLAB begins saving everything that transpires to a text file named "diary". This continues until you enter `diary off`. You can then edit this text file using emacs. If you enter `diary filename` instead of `diary on`, the session will be saved to a text file called "filename".

2. Function M-files

The basic format for a function M-file is:

```
function w=foo(x,y,z)

w= x.*sin(y) + z.^2
```

If this file is saved as `foo.m` in the same directory you use to run MATLAB, then entering `foo(5,3,11)` on the MATLAB command line will cause MATLAB to compute $5\sin(3) + 11^2$.

I should call to your attention one of MATLABs built in functions:

$$\texttt{sign(x)} = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases}$$

This function can be used to define some other useful functions. I introduced two of these in class:

```
function y=heavy(a,x)

y = ( sign(x-a) + 1 )./2;
```

which steps up from 0 to 1 at $x = a$, and

```
function y=lght(b,x)

y = ( sign(b-x) + 1 )./2;
```

which steps down from 1 to 0 at $x = b$. We can also define

```
function y=rect(a,b,x)

y = ( sign(x-a) + sign(b-x) )./2;
```

this steps up from 0 to 1 at $x = a$ and then back down to 0 at $x = b$.

These three can then be used to define piecewise continuous functions, such as

$$f(x) = \begin{cases} 4 & x \le -2 \\ x^2 & -2 < x \le 1 \\ 1 & x > 1 \end{cases}$$

This function can be defined using the function M-file

```
function y=fctn(x)

y=lght(-2,x).*4 + rect(-2,1,x).*x.^2 + heavy(1,x).*1;
```

3. Numerical Methods

Euler's method for computing numerical solutions is usually described as follows: to compute an approximate solution to the differential equation $\frac{dy}{dt} = f(t, y)$ satisfying the initial condition $y(t_0) = y_0$, we choose a timestep Δt and for each $t_k = t_0 + k\Delta t$ the value

$$y_k = y_{k-1} + \Delta t \cdot f(t_{k-1}, y_{k-1})$$

is computed which approximates $y(t_k)$.

If we can compute an analytic solution $y(t)$, then $y'(t) = f(t, y(t))$. Using the Fundamental Theorem of Calculus, we see that

$$y(t_k) - y(t_{k-1}) = \int_{t_{k-1}}^{t_k} y'(t)dt = \int_{t_{k-1}}^{t_k} f(t, y(t))dt.$$

Solving this for $y(t_k)$ we get the exact equation

$$y(t_k) = y(y_{k-1}) + \int_{t_{k-1}}^{t_k} f(t, y(t))dt.$$

Now, $\int_{t_{k-1}}^{t_k} f(t, y(t))dt$ is the area under the curve $y'(t)$ between t_{k-1} and t_k. The quantity $\Delta t \cdot f(t_{k-1}, y_{k-1})$ approximates this area using a rectangle of height $f(t_{k-1}, y_{k-1}) \approx y'(t_{k-1})$ and width $\Delta t = t_k - t_{k-1}$.

When a number of steps are combined, what we are actually doing is approximating a definite integral using a Riemann Sum (with the left endpoint rule).

In 21-117 several methods of numerical integration are introduced: the Left Endpoint Rule, the Midpoint rule, the Trapezoid Rule, and Simpson's Rule. When the Trapezoid rule is adapted for solving differential equations, the result is the second order Runge-Kutta method (or improved Euler's method). An adaptation of Simpson's rule becomes the fourth order Runge-Kutta method. These are both available in dfield. In the Options menu, clicking on "Solver" allows you to choose Euler's method, Runge-Kutta 2, Runge-Kutta 4 or the Dormond-Prince method. The Dormond-Prince solver is a more sophisticated method that uses a variable step size to improve accuracy.

There are M-files available that will compute Euler's method, and Runge-Kutta method approximations to solutions of differential equations. You can download the M-files `eul.m`, `rk2.m` and `rk4.m`, all of which are provided (for academic use) by John Polking.

The use of these files is the same in each case. Suppose you wish to compute a solution to the differential equation

$$\frac{dy}{dt} = \mathtt{funct}(t, y),$$

on the interval $[s, t]$ with initial condition $y(s) = b$, and timestep d. First, you must write a function M-file for the function `funct`, which accepts two input arguments (even if it is an autonomous equation). Then enter at the MATLAB prompt

```
[t,y]=eul('funct',[s,t],b,d)
```

The output will be two vectors, `t` and `y`. The vector `t` contains all the t_i's, i.e. `t`$= [s, s+d, s+2d, \dots]$. The vector `y` contains the approximations $y_i \approx y(t_i)$. To produce a graph of the approximate solution, use the command `plot(t,y)`.

4. Appendix: Built-in Functions

MATLAB makes available many commonly used functions. Below is a partial list. If you wish to learn more about one of them use the `help` command to learn more, e.g. `help log10`.

4.1. **Elementary Functions.** `abs()`, `sqrt()`, `sign()`.

4.2. **Trigonometric Functions.** `sin()`, `cos()`, `tan()`, `cot()`, `sec()`, `csc()`.

4.3. **Inverse Trigonometric Functions.** `asin()`, `acos()`, `atan()`, `acot()`, `asec()`, `acsc()`.

4.4. **Exponential and Logarithm Functions.** `exp()`, `log()`, `log10()`