18.03–ESG Notes 2

Pramod N. Achar

Fall 1999

These notes are a brief introduction to the use of computational tools on Athena for solving differential equations. The brevity is due in large part to the author's ignorance of the more sophisticated capabilities of the software in question. Nevertheless, all three have extensive online help available, so those so inclined should be able to exploit the full power of these software packages.

Craig Watkins has prepared some sample files for MATLAB and Maple in conjunction with his 18.03–Independent Study notes. These can be found as follows:

```
athena% add 18.03-esg
athena% cd /mit/18.03-esg/watkins/matlab
```

— or —

```
athena% cd /mit/18.03-esg/watkins/maple
```

MATLAB

MATLAB is a purely numerical tool (as opposed to one capable of symbolic manipulation as well). Full online documentation is available in HTML format; the command

```
helpdesk
```

starts up a web browser pointed to the table of contents of this online documentation.

The main command for solving first-order systems is of the form

`[t,y] = ode23('F', [`t_0`; `t_{final}`], `$\mathbf{y}_0$`);`

(In place of `ode23`, one could also use `ode45`, `ode113`, `ode15s`, and several others. These differ in the algorithm used, the time it takes to compute the solution, and the accuracy of the answer, but the syntax for using any of them is the same.) This solves the system

$$\mathbf{y}' = F(t, \mathbf{y})$$

numerically for t in the range $t_0, \ldots, t_{\mathit{final}}$, subject to the initial condition $\mathbf{y}(t_0) = \mathbf{y}_0$. To enter a vector for $\mathbf{y}_0$, enclose it in square brackets and separate the entries with semicolons:

`[`$y_{0,1}$`; `$y_{0,2}$`; `$\ldots$` ; `$y_{0,n}$`]`

Then, when you run the above `ode23` command, it assigns a vector value to the variable `t`, and a matrix value to `y`. (Of course, you could have used any other letters instead of `t` and `z`.) The size of `t` depends on the algorithm used; its entries will be the values of t from t_0 to t_{final} at which the solution was evaluated.

The variable `y` is a matrix with as many columns as the order of the system, and as many rows as the vector `t`. If $\mathbf{y}(t)$ consists of the component scalar functions $y_1(t), \ldots, y_n(t)$, then the entry at row i, column j of `y` is equal to $y_j(\mathtt{t}_i)$, where $\mathtt{t}_i$ is the ith entry of the vector `t`.

The hardest part is specifying the function $F(t, \mathbf{y})$. For this, you have to create a file, called an odefile, which contains the function definition. For example, to solve the system

$$\mathbf{y}' = \begin{bmatrix} 4 & 5 \\ 6 & 2 \end{bmatrix} \mathbf{y} + \begin{bmatrix} t \\ t^2 \end{bmatrix},$$

one would create a file called `mysystem.m` containing the following lines:

```
function yp = mysystem(t,y)
yp = [ 4*y(1) + 5*y(2) + t; 6*y(1) + 2*y(2) + t^2];
```

Here, yp is a dummy function name that stands for "y-prime"—you could use any name you wanted. (A name like yp becomes important in more complicated odefiles, but that is beyond the scope of this document.) The t and y are also dummy variables; they just stand for the inputs to the function F. One could easily have written

```
function qp = mysystem(s,z)
qp = [ 4*z(1) + 5*z(2) + s; 6*z(1) + 2*z(2) + s^2];
```

and the result would be exactly the same. In particular, the dummy variables in the odefile do not have to be the same as the variables which receive the results of the computation when you enter

```
[t, y] = ode23('mysystem', [0; 6], [1; 0]);
```

You can examine the results of the computation just by typing t or y at the MATLAB prompt; it will then print out the contents of these matrices for you. But of course, you would rather have a graphical plot. The plot command takes two arguments, which should be vectors of the same length, and draws a graph of the entries of the two vectors plotted against one another. Now, y is a matrix of size *something* $\times$ 2; the syntax for extracting just one column of it as a vector is y(:,1) or y(:,2). Thus, to plot $y_1(t)$ against t, the command is

```
plot(t, y(:,1));
```

To plot $y_1(t)$ on the horizontal axis and $y_2(t)$ on the vertical, enter

```
plot(y(:,1), y(:,2));
```

You can also plot more than one pair of variables at a time. To plot $y_1(t)$ and $y_2(t)$ simultaneously on the veritcal axis with t on the horizontal axis, enter

```
plot(t, y(:,1), t, y(:,2));
```

Maple

To get started with Maple, instruct to load the differential equations library with the commmand

```
with(DEtools);
```

Maple contains a number of functions with names like bernoullisol, constantcoeffsol, eulersol, linearsol, *etc.* for analytically solving differential equations that fit a particular form. One function of interest is *matrixDE*, which solves a matrix differential equation. To solve the first-order nonhomogeneous linear matrix equation

$$\mathbf{y}' = \mathbf{A}(t)(y) + \mathbf{f}(t),$$

one uses the syntax

matrixDE($\mathbf{A}(t)$, $\mathbf{f}(t)$, t);

For example, the system discussed in the MATLAB section could be solved explicitly with the following series of commands:

```
A := matrix(2, 2, [4, 5, 6, 2]);
matrixDE(A, [t, t^2], t);
```

(The first command defines the variable A to be the desired matrix. The first two arguments to the matrix command give the number of rows and columns of the matrix.)

Maple also has a general-purpose dsolve command that tries to determine whether the given equation fits one of the types it knows how to solve explicitly, and then tries to solve it according to that.

What about systems that cannot be solved analytically? The following command draws a phase portrait (*i.e.* slope field) and particular solution curves for given initial conditions:

phaseportrait([*eqns*], [*dep. vars*], *indep. var*=t_0..t_{final}, [[*init. conds*]], *options*);

For example, a phase portrait for the system

$$\mathbf{y}' = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \mathbf{y}$$

can be plotted with the command

```
phaseportrait([ diff(x(t),t) = y, diff(y(t),t) = -x ], [x, y], t=0..2*Pi,
[[x(0) = 0, y(0) = 1]], stepsize=0.1);
```

The following example shows how to plot more than solution curve on the phase plane by giving more than one set of initial conditions:

```
f := diff(x(t),t) = y;
g := diff(y(t),t) = -x;
phaseportrait([f, g], [x, y], t=0..2*Pi, [[x(0)=0, y(0)=1],
[x(0)=2, y(0)=0]], stepsize=0.1);
```

Mathematica

Mathematica is arguably the most powerful of the three software packages considered here. It has a very easy-to-use help browser with full information on everything you might ever want to know about Mathematica. One of the most important things to realize about Mathematica is that when you are finished entering a line, press Shift-Return, not just Return, to get Mathematica to evaluate it.

The two main Mathematica functions for solving differential equations are `DSolve` (for solving analytically) and `NDSolve` (for solving numerically). These functions have similar syntax:

DSolve[{*eqns*}, {*dep. vars*}, *indep. var*]
NDSolve[{*eqns*}, {*dep. vars*}, {*indep. var*, t_0, t_{final}}]

For `NDSolve`, the *eqns* must include an appropriate number of initial conditions, whereas `DSolve` can produce general solutions containing arbitrary constants. For example,

```
DSolve[{x'[t] == y, y'[t] == -x}, {x[t], y[t]}, t]
NDSolve[{x'[t] == y, y'[t] == -x, x[0] == 0, y[0] == 1}, {x[t], y[t]},
{t, 0, 2*Pi}]
```

Plotting with Mathematica is slightly tricky, because the output of `DSolve` and `NDSolve` are not functions but "rules." Suppose we have entered the command

```
solution = NDSolve[{x'[t] == y, y'[t] == -x, x[0] == 0, y[0] == 1},
{x[t], y[t]}, {t, 0, 2*Pi}]
```

Then, the following commands would plot $x(t)$ *vs.* t, both $x(t)$ and $y(t)$ *vs.* t, and $y(t)$ *vs.* $x(t)$, respectively:

```
Plot[Evaluate[x[t]/.solution], {t, 0, 2*Pi}]
Plot[{Evaluate[x[t]/.solution], Evaluate[y[t]/.solution]}, {t, 0, 2*Pi}]
ParametricPlot[Evaluate[{x[t], y[t]}/.solution], {t, 0, 2*Pi}]
```

Exercises

1. The motion of a vertically launched projectile, in the absence of air resistance, is determined by the equation $y'' = -9.8$. Suppose instead that we live on a planet with an atmosphere, and the projectile's motion is actually governed by

$$y'' = -9.8 - 0.0006\,(y')^2.$$

Plot some solution curves to this equation using MATLAB. Use $y(0) = 0$, and try a few different values for $y'(0)$. (Of course, the first thing you need to do is convert it from a second-order equation to a first-order system.)

2. Kepler's laws of planetary motion can be used to derive the following equations of motion describing the trajectory of a satellite about a mass located at the origin:

$$\frac{d^2x}{dt^2} = -\frac{x}{(x^2+y^2)^{3/2}} \qquad \frac{d^2y}{dt^2} = -\frac{y}{(x^2+y^2)^{3/2}}$$

Using Maple, plot a phase portrait for this system, and several different trajectories.

3. Let $x(t)$ and $y(t)$ denote the populations of two competing species of animals; suppose that they are governed by the equations

$$\begin{aligned} x' &= 60x - 3x^2 - 4xy \\ y' &= 42y - 3y^2 - 2xy \end{aligned}$$

Investigate what happens in this system under various initial conditions.

4. Recall our discussion of mass-and-spring systems governed by an equation of the form $F(x) = -kx + \beta x^3$, rather than by Hooke's Law. The following equations describe various mechanical systems which deviate in other ways from Hooke's Law. Investigate the properties of each.

$$\begin{aligned} x'' &= -2x' - 20x + 5x^3 \\ x'' &= -4x + x^2 \\ x'' &= -4x + 5x^3 - x^5 \end{aligned}$$