Solving Ordinary Differential Equations with Matlab

CS 310 — Fall 1999

In this handout we'll discuss how to solve differential equations using Matlab. All of the files described in this handout are available on the course Web page under the Matlab link. You may copy them from there if you prefer not to type them in.

1 Euler's Method

To solve the differential equation

$$y'(t) = f(t, y)$$

with initial condition $y(t_0) = y_0$, we can use Euler's formula,

$$y_{k+1} = y_k + hf(t_k, y_k) \tag{1}$$

With this formula, we can use the given values t_0 and y_0 to calculate y_1. To get t_1, we add h to t_0. Now we have two points on the curve $y(t)$ — we were given (t_0, y_0), now we also have (t_1, y_1). Then, we can do the same thing over again to calculate t_2 and y_2, and then t_3 and y_3, and so on. Each time we use Euler's formula, we get another point on the curve $y(t)$.

What we're doing is getting an approximation to the solution $y(t)$, a continuous function, by finding lots of discrete points that are close to the graph of $y(t)$.

In this section, we're going to write a program in Matlab which will do exactly this for the differential equation

$$\frac{dy(t)}{dt} = y(t) + t - 1$$

with initial condition $y(0) = 2$. The goal is to obtain a graph of the function $y(t)$ by generating points (t_k, y_k) with Euler's method.

We will store the values y_k that are calculated using Euler's method in a vector `U`. So, `U(1)` is the first value, $y(0)$ in this example. `U(2)` is the second value, $y(h)$. Remember that `U(k)` in the program below is *not* the value of the function at $t = k$! `U(k)` is the k^{th} element in the MATLAB vector `U`, which represents the approximation to the function y; $y(t)$ is the function y with respect to time. The variable k represents the index here and must always be positive (starting at 1), while t represents time and could be positive or negative (or zero). The following table illustrates the differences:

Step	t	$y(t)$	`U(Step)`	y'
1	0	2	2	1
2	0.1	2.1	2.1	1.2
3	0.2	2.22	2.22	1.42

To start, use your favorite editor to create a file called `euler1.m` with the following lines in it, or copy it from the course Web page Matlab folder.

```
% This program uses Euler's method to solve the D.E.
%    dy/dt = y + t - 1
% where y(0)=2.
clear;

h = 0.1;
U(1) = 2;

k = 1;

for t = 0:h:2-h
   Uprime = U(k) + t - 1;
   U(k+1) = U(k) + h*Uprime;
   k = k+1;
end

t = 0:h:2;
Uexact = 2*exp(t) - t;

plot(t,U,t,Uexact,':');
```

Now start up Matlab, and run the script `euler1.m`. If you have any mistakes in your program, Matlab will tell you about them. You might need to edit the file to remove all of your typos.

A plot should appear with two curves. The dotted one is the exact solution, and the solid one is the numerical approximation generated by Euler's method.

Let's go through the program step by step.

1. The first statement is a `clear`. It is always a good idea to begin your Matlab programs with `clear`. (In this program, if we'd had a previously existing variable `U`, the `plot` statement might not have worked — can you figure out why?)
2. Next we set the value of `h` and set the first element of the `U` vector to be the given initial value. The variable `k` is used to keep track of where we are in `U`. It is the

same `k` that appears in Euler's formula, (1). The initial condition is $y(0) = 2$, but we set `U(1)=2`.

3. Now the main loop begins. Each time through the loop, we calculate one more entry in the vector `U`, based on the current value in the vector. The loop starts at `0` and stops at `2-h`. We want the last value calculated to correspond to $t = 2$, so the loop must stop at `2-h` since this is the next-to-last value.

   Each time through the loop, we calculate the right-hand side of the differential equation (`Uprime`), then use Euler's formula to calculate the new value of `U`, `U(k+1)`. We also need to increment `k`.

4. After the loop is finished, we're essentially done. The rest of the program is just for plotting the answer and the known analytical solution. To do that, we create the vector `t`, so that we can evaluate the function $2e^t - t$, the analytical solution to the problem. We call it `Uexact`.

   In real life, the analytical answer will probably not be known. If it were, we wouldn't need a numerical solution — we could use the analytical one!

5. The last thing we do is make the plot. The extra `':'` as the last argument to `plot` indicates that the second curve should be made with dots.

To test your understanding of the program, see if you can make the following modifications.

☞ Add a title and axis labels to the graph.

☞ Change the program so that instead of stopping at $t = 2$, it stops at $t = 5$.

If you want, create a new variable called `tfinal`, which is assigned a value near the beginning of the program and is used to correctly determine `U` and `Uexact` from $t = 0$ to $t =$`tfinal`.

☞ Instead of the initial condition $y(0) = 2$, solve the same equation using $y(-1) = 3$. In this case, you will need to use

$$u(t) = 2e^{t+1} - t$$

for the exact solution. Do you see why?

Make sure that the exact solution agrees with the solution calculated using Euler's method.

☞ Instead of plotting the exact answer and the approximate answer, let's modify the program to calculate and plot the error between them. To do this, add the following line just before the `plot` statement:

```
err = Uexact - U;
```

Also change the plot statement to

```
plot(t,err);
```

What happens to the error as a function of time? Why? Can you think of a different measure of the error? What happens if you use that instead?

2 Solving Coupled Differential Equations

We will now consider solving coupled differential equations, such as those that arise when modeling predator-prey problems. Consider the populations of rabbits and foxes on a secluded island: how will their populations interact?

The following system models the situation, where $u(t)$ and $v(t)$ are the rabbit and fox populations, respectively, at time t:

$$\frac{du(t)}{dt} = 0.5\ u(t) - 0.2\ u(t)\ v(t)$$
$$\frac{dv(t)}{dt} = -0.7\ v(t) + 0.3\ u(t)\ v(t)$$

with the initial conditions $u(0) = 2$, and $v(0) = 3$. These equations are coupled in that they can't be solved separately, since they depend on each other. We have to solve them both at the same time.

We can do this with Euler's method using the same technique as before. From the values of t_0, u_0 and v_0, we can calculate the values of t_1, u_1 and v_1 with Euler's formula. We must use Euler's formula twice, once for u_{k+1} and again for v_{k+1}. Both of these calculations will involve t_k, u_k and v_k.

Then, after we have gotten t_1, u_1 and v_1, we can repeat the procedure to calculate t_2, u_2 and v_2, and so on. We can keep going as long as we want. Keep in mind, though, that after a while the errors will (probably) have accumulated to a significant degree.

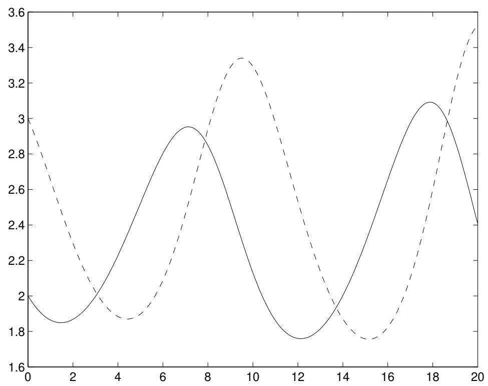
Here is a Matlab program that will do this. Type it in, or copy it from the course Web page, and call it `euler2.m`

```
% This program uses Euler's method to solve the coupled D.E.'s
%    du/dt =  0.5 u - 0.2 uv
%    dv/dt = -0.7 v + 0.3 uv
% where u(0)=2 and v(0)=3
clear;
```

```
h = 0.02;
tstart = 0;
tend = 20;

U(1) = 2;
V(1) = 3;

k = 1;

for t = tstart:h:tend-h
   Uprime = 0.5*U(k) - 0.2*U(k)*V(k);
   Vprime = -0.7*V(k) + 0.3*U(k)*V(k);

   U(k+1) = U(k) + h*Uprime;
   V(k+1) = V(k) + h*Vprime;

   k = k+1;
end

t = tstart:h:tend;
plot(t,U,t,V,'--');
```

You should understand what the program is doing inside the loop. At each step, the goal is to calculate `U(k+1)` and `V(k+1)` based on the previous values, `U(k)` and `V(k)`.

If all goes well, you should see a graph that looks something like this:



Since we replaced `':'` with `'--'`, the second curve (the one for v) is drawn with dashes instead of dots. Also, notice that we added the variables `tstart` and `tend`, so it should be easy to change the program to start and end at different times. As it is, the equation is solved from $t = 0$ to $t = 20$.

Recall that v represents the population of foxes (the dashed curve in the graph), and u represents the population of rabbits (the solid curve). Does the graph make sense with this interpretation?

☞ Try to make the following modifications to `euler2.m`. It may be a good idea to copy `euler2.m` to `euler3.m` before you begin, and make your changes to `euler3.m`

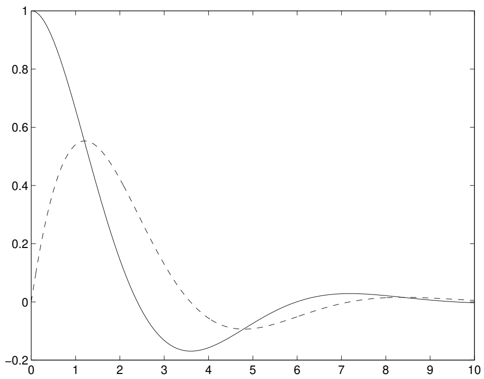
Change the program so that it solves the coupled differential equations

$$\frac{dy_1(t)}{dt} = -y_2(t)$$
$$\frac{dy_2(t)}{dt} = y_1(t) - y_2(t)$$

with initial conditions $y_1(0) = 1$ and $y_2(0) = 0$, from $t = 0$ to $t = 10$.

The plot of the solution should look something like the one shown to the right. The solid curve is the solution for $y_1(t)$, the dashed one is the solution for $y_2(t)$. You can see that $y_1(0) = 1$ and $y_2(0) = 0$.

Put a title and axis labels on your plot. Try different initial conditions. What happens if you choose too large a value for `h`?



A solution to this problem is `euler3.m`, located in the Matlab folder on the course Web page. You should look at the solution even if you are able to get the correct plot on your own.

3 Modified or Improved Euler (optional)

☞ Modify the program `euler1.m` to use the one of the variants on Euler's method: the Modified Euler or the Improved Euler Method.

Solutions are available on the course Web page under the Matlab link, in the files `modified1.m` and `improved1.m`. Each of these solves the same differential equation solved with `euler1.m`, so your graph should look similar. How do you expect your errors to compare with those obtained using the normal Euler's method?

☞ *(optional)* Modify the program to solve the problem with *both* Euler's method and one of the variants (modified or improved), and make a plot comparing the errors for each method.

Matlab has built-in routines that solve differential equations. One is called `ode45`. In the next lab, we'll use this to solve some differential equations that arise from real problems.

`ode45` uses 4^{th} and 5^{th} order Runge-Kutta formulas to solve the differential equations.

If you're interested, read the help information on `ode45` by typing `help ode45`. You can also type `odedemo` to have Matlab show you a short demonstration of solving differential equations using `ode45` (and a similar routine, `ode23`).