Lecture 30

Euler Methods

Numerical Solution of an IVP

Suppose we wish to numerically solve the initial value problem

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}), \qquad \mathbf{y}(a) = \mathbf{y}_0, \tag{30.1}$$

on an interval of time $[a, b]$.

By a numerical solution, we must mean an approximation of the solution at a finite number of points, i.e.

$$(t_0, \mathbf{y}_0), (t_1, \mathbf{y}_1), (t_2, \mathbf{y}_2), \ldots, (t_n, \mathbf{y}_n),$$

where $t_0 = a$ and $t_n = b$. The first of these points is exactly the initial value. If we take n steps as above, and the steps are evenly spaced, then the time change in each step is

$$h = \frac{b-a}{n}, \tag{30.2}$$

and the times t_i are given simply by $t_i = a + ih$. This leaves the most important part of finding a numerical solution: determining $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n$ in a way that is as consistent as possible with (30.1). To do this, first write the differential equation in the indexed notation

$$\dot{\mathbf{y}}_i \approx \mathbf{f}(t_i, \mathbf{y}_i), \tag{30.3}$$

and then replace the derivative $\dot{\mathbf{y}}$ by a difference. There are many ways we might carry this out and in the next section we study the simplest.

The Euler Method

The most straight forward approach is to replace $\dot{\mathbf{y}}_i$ in (30.3) by its forward difference approximation. This gives

$$\frac{\mathbf{y}_{i+1} - \mathbf{y}_i}{h} = \mathbf{f}(t_i, \mathbf{y}_i).$$

Rearranging this gives us a way to obtain $\mathbf{y}_{i+1}$ from $\mathbf{y}_i$ known as Euler's method:

$$\mathbf{y}_{i+1} = \mathbf{y}_i + h\mathbf{f}(t_i, \mathbf{y}_i). \tag{30.4}$$

With this formula, we can start from $(t_0, \mathbf{y}_0)$ and compute all the subsequent approximations $(t_i, \mathbf{y}_i)$. This is very easy to implement, as you can see from the following program (which can be downloaded as `myeuler.m`).

```
function [T , Y] = myeuler(f,tspan,y0,n)
    % Solves dy/dt = f(t,y) with initial condition y(a) = y0
    % on the interval [a,b] using n steps of Euler's method.
    % Inputs: f -- a function f(t,y) that returns a column vector the
    %              same length as y
    %         tspan -- a vector [a,b] with the start and end times
    %         y0 -- a column vector of the initial values, y(a) = y0
    %         n  -- number of steps to use
    % Outputs: T -- a n+1 column vector containing the times
    %          Y -- a (n+1) by d matrix where d is the length of y
    %               Y(j,i) is the ith component of y at time T(j)
    a = tspan(1); b = tspan(2); % parse starting and ending points
    h = (b-a)/n;    % step size
    t = a; T = a;  % t is the current time and T will record all times
    y = y0;     % y is the current variable values, as a column vector
    Y = y0';    % Y will record the values at all steps, each in a row
    for i = 1:n
        y = y + h*f(t,y); % Euler update of y.
        Y = [Y; y'];      % y' becomes the next row in Y.
        t = t + h;        % The next time.
        T = [T; t];       % Record t into T.
    end
end
```

To use this program we need a function, such as the vector function for the pendulum:

```
≫ dy = @(t,y)[y(2);-.1*y(2)-sin(y(1))+sin(t)]
```

Save this and then type

```
≫ [T Y] = myeuler(dy,[0 20],[1;-1.5],5);
```

Here `[0 20]` is the time span you want to consider, `[1;-1.5]` is the initial value of the vector `y` and `5` is the number of steps. The output `T` contains times and `Y` contains values of the vector at the times. Try

```
≫ size(T)
≫ size(Y)
```

Since the first coordinate of the vector is the angle, we only plot its values:

```
≫ theta = Y(:,1);
≫ plot(T,theta)
```

In this plot it is clear that $n = 5$ is not adequate to represent the function. Type

```
≫ hold on
```

then redo the above with 5 replaced by 10. Next try 20, 40, 80, and 200. As you can see the graph becomes increasingly better as n increases. We can compare these calculations with MATLAB's built-in function with the commands

```
≫ [T Y]= ode45(dy,[0 20],[1;-1.5]);
≫ theta = Y(:,1);
≫ plot(T,theta,'r')
```

The problem with the Euler method

You can think of the Euler method as finding a linear approximate solution to the initial value problem on each time interval. An obvious shortcoming of the method is that it makes the approximation based on information at the beginning of the time interval only. This problem is illustrated well by the following IVP:

$$\ddot{x} + x = 0 \quad \text{with} \quad x(0) = 1 \quad \text{and} \quad \dot{x}(0) = 0\,. \tag{30.5}$$

You can easily check that the exact solution of this IVP is

$$x(t) = \cos(t).$$

If we make the standard change of variables

$$y_1 = x \quad \text{and} \quad y_2 = \dot{x},$$

then we get

$$\dot{y}_1 = y_2 \quad \text{and} \quad \dot{y}_2 = -y_1.$$

Then the solution should be $y_1(t) = \cos(t)$ and $y_2(t) = \sin(t)$. If we then plot the solution in the (y_1, y_2) plane, we should get exactly a unit circle. We can solve this IVP with Euler's method:

```
≫ dy = @(t,y)[y(2);-y(1)]
≫ [T Y] = myeuler(dy,[0 4*pi],[1;0],20)
≫ y1 = Y(:,1);
≫ y2 = Y(:,2);
≫ plot(y1,y2)
```

As you can see the approximate solution goes far from the true solution. Even if you increase the number of steps, the Euler solution will eventually drift outward away from the circle because it does not take into account the curvature of the solution.

The Modified Euler Method

An idea which is similar to the idea behind the trapezoid method would be to consider f at both the beginning and end of the time step and take the average of the two. Doing this produces the Modified (or Improved) Euler method represented by the following equations:

$$\begin{aligned} \mathbf{k}_1 &= h\mathbf{f}(t_i, \mathbf{y}_i) \\ \mathbf{k}_2 &= h\mathbf{f}(t_i + h, \mathbf{y}_i + \mathbf{k}_1) \\ \mathbf{y}_{i+1} &= \mathbf{y}_i + \frac{1}{2}(\mathbf{k}_1 + \mathbf{k}_2)\ . \end{aligned} \tag{30.6}$$

Here $\mathbf{k}_1$ captures the information at the beginning of the time step (same as Euler), while $\mathbf{k}_2$ is the information at the end of the time step.

A program that implements the Modified method can be downloaded as `mymodeuler.m`.

Test this program on the IVP above:

```
≫ [T Ym] = mymodeuler(dy,[0 4*pi],[1;0],20)
≫ ym1 = Ym(:,1);
≫ ym2 = Ym(:,2);
≫ plot(ym1,ym2)
```

You will find that the results are much better than for the plain Euler method.

Exercises

30.1 Download `myeuler.m` and `mymodeuler.m`.

(a) Type the following commands:

```
≫ dy = @(t,y) sin(t)*cos(y);
≫ hold on
≫ [T Y] = myeuler(dy,[0,12],.1,20);
≫ plot(T,Y)
```

Position the plot window so that it can always be seen and type

```
≫ [T Y] = myeuler(dy,[0,12],.1,30);
≫ plot(T,Y)
```

(You can use the up button to reduce typing.) Continue to increase the last number in the above until the graph stops changing (as far as you can see). Record this number and print the final graph. Type `hold off` and kill the plot window.

(b) Follow the same procedure using `mymodeuler.m` .

(c) Describe what you observed. In particular compare how fast the two methods converge as n is increased (h is decreased).

30.2 The equation of motion of a damped, unforced pendulum is

$$\ddot{\theta} + \frac{\gamma}{m}\dot{\theta} + \frac{g}{\ell}\sin\theta = 0.$$

The total energy of the pendulum is $E = m\ell^2\dot{\theta}^2/2 + mg\ell(1 - \cos(\theta))$. Suppose a pendulum has $m = 2$, $g = 9.81$, $l = 3$, and coefficient of friction $\gamma = 0.5$, all in SI units. Write a well-commented **script** program that uses the Modified Euler's method with $h = 0.01$ to simulate the movement of the pendulum from a starting position of $(\theta(0), \dot{\theta}(0)) = (.9\pi, 0)$ until the energy falls below 0.01 Joules (use a `while` loop). Record the steps in matrices Y and T and plot the motion. Turn in your program and plot.