# Numerical Solution of Differential Equations: MATLAB implementation of Euler's Method

The files below can form the basis for the implementation of Euler's method using Matlab. They include `EULER.m`, which runs Euler's method; `f.m`, which defines the function $f(t, y)$; `yE.m`, which contains the exact analytical solution (computed independently), and `ErrorPlot.m`, which plots the errors as a function of $t$ (for fixed $h$).

The structure of these programs is not optimal; for instance, you may wish to modify the calling routine for the Euler method to pass the initial values, step size and number of steps as parameters, instead of modifying them in the `EULER.m` file. I have not attempted to optimize the code (which is taken from the Math 216 labs); my recommended software for Math 256 is Maple, and a Maple implementation for Euler's method is provided in a separate file.

In order to solve a particular differential equation, you will need to define the function $f(t, y)$ in the file `f.m`, and also the exact solution in `yE.m`, if needed. The program computes the step size $h$ from the initial and final $t$ values $a$ and $b$, and the number of steps $N$.

You will need to modify the algorithm in `EULER.m` (inside the `for` loop) to implement the Backward Euler, Improved Euler and Runge-Kutta methods.

### The file `EULER.m`

This program will implement Euler's method to solve the differential equation

$$\frac{dy}{dt} = f(t, y) \qquad y(a) = y_0 \tag{1}$$

The solution is returned in an array `y`. You may wish to compute the exact solution using `yE.m` and plot this solution on the same graph as `y`, for instance by modifying the second-to-last line to read

```
plot(t,y,'-',t,yE(t),'-.')
```

In this program, everything following a `%` is a **comment**. Comments give you information about the program, but are not evaluated by the computer. You may choose whether or not to type these comments into your program, but if you include the comments in your file you must include the `%`, or the computer will try to read them.

```
clear t                          % Clears old time steps and
clear y                          % y values from previous runs
a=0;                             % Initial time
b=1;                             % Final time
N=10;                            % Number of time steps
y0=0;                            % Initial value y(a)
h=(b-a)/N;                       % Time step
t(1)=a;
y(1)=y0;
for n=1:N                        % For loop, sets next t,y values
  t(n+1)=t(n)+h;
  y(n+1)=y(n)+h*f(t(n),y(n));  % Calls the function f(t,y)=dy/dt
end
plot(t,y)
title(['Euler Method using N=',num2str(N),' steps, by MYNAME'])
                                 % Include your own name
```

**The files `f.m` and `yE.m`**

The file `f.m` contains the function $f(t, y)$ for the general differential equation (1) above; the particular form of $f(t, y)$ corresponds to the equation

$$y' = 3 + t - y. \tag{2}$$

To solve a different differential equation with `EULER.m` or another solver, you need only change this file.

```
function f=f(t,y)
f=3+t-y;                          % Defines the function f
```

The file `yE.m` contains the exact solution $y(t) = 2 + t - e^{-t}$ of equation (2), corresponding to the above function $f(t, y)$ defined in the file `f.m`. If you solve a different differential equation with `EULER.m` or one of the other numerical methods described below, and you wish to compare with an analytical expression for the exact solution, you should modify the file `yE.m` as well as `f.m`.

```
function yE=yE(t)
yE=2*ones(size(t))+t-exp(-t);     % Exact solution yE
 % Note the ones() command, creating a vector of ones.
```

**The file** `ErrorPlot.m`

Finally, we plot the error of the Euler method, or compare the error with that of other numerical methods such as Improved Euler or Runge-Kutta. The file ErrorPlot.m helps us do this (note: if the Improved Euler and Runge-Kutta methods have not been implemented in files `impEULER.m` and `RK.m`, giving their solutions in arrays `yI` and `yRK` respectively, then some of the lines below need to be modified or removed; this should just be taken as a sample program). The errors will be plotted on a logarithmic scale, where the logarithm of the error is plotted versus time. A "semilog" plot such as this is particularly helpful when you wish to investigate a function, or several functions, that take on values spanning many orders of magnitude. In order to plot the error (for a fixed time $t$) as a function of $h$, we can produce a similar graph; the scaling of the error with $h$ is best seen in a "log-log" plot.

```
EULER                          % Runs Euler
impEULER                       % Runs ImpEuler
RK                             % Runs RK
yexact=yE(t);                  % Computes the exact solution
errE=abs(yexact-y);            % Euler method error
errI=abs(yexact-yI);           % Improved Euler error
errRK=abs(yexact-yRK);         % Runge Kutta error
semilogy(t,errE,':',t,errI,'-.',t,errRK,'-')
legend('Euler','ImpEuler','RK')
xlabel('t')                    % Labels ``x'' axis
ylabel('Error')                % Labels ``y'' axis
title(['Errors using N=',num2str(N),' steps, by MYNAME'])
```