Advanced Programming in Engineering
Brief Manual to Matlab's `ode45` Routine

Question: numerically solve $y(x)$ from

$$\frac{dy}{dx} = f(x) = \cos x,$$

over the interval $0 \leq x \leq 10$, given that $y(0) = 0$.

Answer: create an m-file function called[1] `DiffEq` that, when called with given values of x and y, returns the value of the derivative:

```
function dydx = DiffEq(x,y)  % invocation must accept two arguments:
                             % first the independent variable,
                             % second the dependent variable
                             %   (even when these are not used).
dydx = cos(x);               % calculate and return the derivative
```

In the command window, initialize auxiliary variables and call `ode45`:

```
range = [0 10];                     % range of independent coordinate
y0 = 0;                             % start value
[T,Yt] = ode45(@DiffEq,range,y0);   % obligatory order of arguments
plot(T,Yt)                          % plot the result
```

You should now see a plot of $\sin x$. The returned data are stored in the column vectors `T` and `Yt`, with `Yt(5)` holding the approximate value of y at time `T(5)`.

Note that the function `DiffEq` is passed to `ode45` as an argument.

Question: numerically solve the coupled differential equations

$$\begin{cases} dr(t)/dt = -r(t) - 0.1s(t) \\ ds(t)/dt = -0.2r(t) - s(t) \end{cases}$$

with the starting point

$$\begin{cases} r(0) = 1 \\ s(0) = 2 \end{cases}$$

over the interval $0 \leq t \leq 10$.

Answer: since ode45 requires a derivate-evaluating function with two arguments, the two coupled equations must be combined into one vector equation:

$$\frac{d\mathbf{z}}{dt} = \mathbf{A}\mathbf{z} \text{ with } \mathbf{z} = \begin{pmatrix} r \\ s \end{pmatrix}, \quad \mathbf{A} = \begin{pmatrix} -1 & -0.1 \\ -0.2 & -1 \end{pmatrix} \text{ and } \mathbf{z}(0) = \begin{pmatrix} 1 \\ 2 \end{pmatrix}.$$

[1] Matlab recognizes external functions by the name of the file; the name in the function declaration is irrelevant (but it is good practice to use the filename there too).

Write an m-file that, for given **z** and t, returns the derivative:

```
function dzdt = DiffEq(t,z)  % invocation must accept two arguments;
                             % ode45 works with column vectors only
global A                     % share matrix A with main program
dzdt = A * z;                % calculate and return the derivative
```

Initialize variables and call `ode45`:

```
global A                            % share matrix A with DiffEq
A = [ -1 -0.1 ; -0.2 -1]            % fill matrix A
range = [0 10];                     % range of independent coordinate
z0 = [1 ; 2];                       % starting point
[T,Zt] = ode45(@DiffEq,range,z0);   % obligatory order of arguments
plot(T,Zt)                          % plot the result
```

You should now see a plot of two curves decaying to zero. The returned data are stored in the column vectors `T` and `Zt`, with `Zt(5,1)` and `Zt(5,2)` holding the approximate values of r and s, respectively, at time `T(5)`.

The use of `global` to pass data to the routine `DiffEq` is potentially dangerous, as this routine can potentially change the value of your data. It is therefore recommended to use the function

```
function dzdt = DiffEqA(t,z,A)
dzdt = A * z;                      % calculate and return the derivative
```

in combination with

```
A = [ -1 -0.1 ; -0.2 -1]            % fill matrix A
DiffEq = @(tt,zz) DiffEqA(tt,zz,A) % create a function handle
                                    %   accepting two arguments
                                    %   and forwarding three arguments
[T,Zt] = ode45(DiffEq,range,z0);    % pass the handle to ode45 (no @)
plot(T,Zt)                          % plot the result
```

Every time matlab calls a function, it reads this function anew from your hard disk. Hence, your code can become very slow when calling a function often. You will then find that your code runs faster if the function is included as a *subfunction* at the end of the m-file or function that invokes ode45.

In stead of a subfunction and a handle, it is also possible to use a *nested* function:

```
function MyCode
A = [ -1 -0.1 ; -0.2 -1]            % fill matrix A
[T,Zt] = ode45(@DiffEq,range,z0);   % pass a function to ode45
plot(T,Zt)                          % plot the result
  function dzdt = DiffEq(t,z)
    dzdt = A * z;                   % calculate and return the derivative
  end                               % end of dzdt
end                                 % end of MyCode
```

Note that the nested function can alter the data in your main code, and is therefore to be used with care.

Question: numerically solve the above differential equation till $s(t) = \frac{1}{2}$.

Answer: create a function that reaches the value *zero* when the integration is to be halted.

```
function [value,isterminal,direction] = criterium(p,q)
                     % again, to be invoked with two arguments
value = q(2) - 0.5;  % crossing zero is the "event"
direction = 0;       %  0 : every crossing is an event
                     % +1 : only increasing through zero is an event
                     % -1 : only decreasing through zero is an event
isterminal = 1;      % stop (1) or don't stop (0) ode at event
```

Initialize variables in the 'old' way, and instruct `ode45` to detect events

```
options = odeset('Events',@criterium);
[T,Zt] = ode45(@diffvgl,interval,z0,options);
plot(T,Zt)
```

The $r(t)$ and $s(t)$ lines in your graph will run till the first t with $s(t) = \frac{1}{2}$.