BACKWARD EULER METHOD. Also called implicit Euler method. Linear and nonlinear ODEs can be solved with this method. If the ODE is linear, the discretized equations can be solved directly (i.e., yields explicit algebraic equations) by the y_{i+1} . If the ODE is nonlinear, a root finding method must be used to find y_{i+1} . This example should give the big picture of what an implicit method is about. Let's work out a nonlinear ode example (The same one selected in class).

Nonlinear ODE. Solve the nonlinear ODE:

$$\frac{dy}{dt} = (1+2t)\sqrt{y};\ \ \mathrm{y}(0) = 1;\ \ \mathrm{t} = [0,1];\ \ h = 0.1 \quad \textit{Eq.1}$$

Assuming you will pay attention to all FDM steps, let's focus on the differences of the current method with the forward Euler. Construct the discretized ODE using the implicit Euler method:

$$y_{j+1} = y_j + hf(t_{j+1}, y_{j+1}) \quad \textit{Eq.2}$$

Realizing that $f(t_{j+1}, y_{j+1}) = (1+2t_{j+1})\sqrt{y_{j+1}}$, then the discretized equation is:

$$y_{j+1} = y_j + h\,(1+2t_{j+1})\sqrt{y_{j+1}} \quad \textit{Eq.3} \quad \text{where} \quad t_{j+1} = t_j + h$$

Eq. 3 is a nonlinear algebraic equation which needs to be solved by y_{i+1}. A root finding method needs to be used: Newton's, secant, fixed-point, etc. Eq.3 was used to develop the matlab code in appendix.

QUICK SOLUTION. For a quick solution by hand, the first-time step $(i = 0)$ in Eq.3:

$$y_1 = y_o + h\,(1+2t_1)\sqrt{y_1}; \quad \textit{Eq.4} \quad \text{where } t_1 = t_0 + h$$

In *Eq.4* y_o, h, t_1 are known values. Insert them in,

$$y_1 = 1 + 0.1\,(1+2(0.1))\sqrt{y_1}; \quad \textit{Eq.5} \text{ and } \quad t_1 = 0 + 0.1$$

Simplify Eq.5:

$$y_1 = 1 + 0.12\sqrt{y_1}; \quad \textit{Eq.6} \text{ and } t_1 = 0.1$$

Eq.6 is an implicit nonlinear equation, where the task at hand is to solve by y_1. Newton, secant, fixed-point or any other root finding methods can be used.

NEWTON'S METHOD (NM). Due to its fans club, let's start with the Newton's method:

$$x_{i+1} = x_i - \frac{f_i}{f_i'} \quad \textit{Eq.7}$$

Where x_{i+1} is the new root approximation of the function f_i. This is an iterative method where the first value of the root x_i is usually extracted by inspection from a graph of $f_i\ vs\ x_i$.

Let's change the variable notation of the *Eq.6*, so it looks more familiar (e.g., call the unknown $x = y_1$):

$$x = 1 + 0.12\sqrt{x} \quad \text{or} \quad 0 = 1 + 0.12\sqrt{x} - x \quad \textit{Eq.8}$$

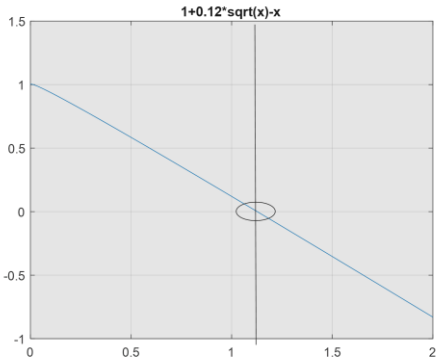
Arrange Eq.8 to get an equation suitable for NM:

$$f(x) = 1 + 0.12\sqrt{x} - x \quad \text{Eq.9}$$

In Eq.9 the goal is to get values of x for which $f(x) = 0$ is satisfied. The first derivative of the function (Eq.9) is required in NM:

$$f'(x) = -1 + \frac{0.06}{\sqrt{x}} \quad \textit{Eq. 10}$$

A plot of the Eq.9 is shown, the oval region is where the root is:



To apply the *Eq. 7* we need:

$$x_{i+1} = x_i - \frac{f_i}{f_i'}$$

where:

$$f_i = 1 + 0.12\sqrt{x_i} - x_i \quad \textit{Eq. 9}$$
$$f_i' = -1 + \frac{0.06}{\sqrt{x_i}} \quad \textit{Eq.10}$$

A sample computation of Eq.7 with $i = 0$;

$$x_{i+1} = x_i - \frac{f_i}{f_i'};\ ==>\quad x_1 = x_o - \frac{f_0}{f_0'} = x_o - \frac{1+0.12\sqrt{x_o}-x_o}{-1+\frac{0.06}{\sqrt{x_o}}} \qquad \textit{Eq.11}$$

NM needs an initial estimate of the root. This can be any point close to the root. The obvious is $x_o = 1$. Please note that this is just the initial condition $y_0 = 1$. The new root shouldn't be too far away this value, then:

$$x_1 = x_o - \frac{1+0.12\sqrt{x_o}-x_o}{-1+\frac{0.06}{\sqrt{x_o}}} = 1 - \frac{1+0.12\sqrt{1}-1}{-1+\frac{0.06}{\sqrt{1}}} = 1.12766 \quad \textit{Eq. 12}$$

The problem doesn't end here, the value of $x_1 = 1.12766$ is not the root of Eq. 6 or Eq 8, it just a closer estimate. Newton's is an iterative method and requires that the estimated error to be:

$$e_i = |x_{i+1} - x_i| < error\ tolerance \quad \text{Eq.13}$$

The error tolerance is assumed by the customer. If we assume an error tolerance of $1x10^{-5}$ (this means 0.00001). For the current iteration:

$$e_0 = |x_1 - x_o| = |1.12766 - 1.0| = 0.12766 \quad \text{Eq.14}$$

Because $e_0 = 0.12766 > 1x10^{-5}$, the process needs to be repeated, assuming that the new root estimated is now $x_1 = 1.12766$, thus *Eq 7* for a new iteration yields $(i = 1)$:

$$x_2 = x_1 - \frac{1+0.12\sqrt{x_1}-x_1}{-1+\frac{0.06}{\sqrt{x_1}}} = 1.12766 - \frac{1+0.12\sqrt{1.12766}-1.12766}{-1+\frac{0.06}{\sqrt{1.12766}}} = 1.127415807 \quad \text{Eq.15}$$

The new error is (Eq.13):

$$e_1 = |x_2 - x_1| = |1.127415807 - 1.12766| = 0.000243768 \quad \text{Eq.16}$$

Still $e_1 = 0.000243768 > 1x10^{-5}$ then, repeat the process until Eq.13 is satisfied:

$$e_i < error\ tolerance\ (1x10^{-5})$$

Better use Excel for the next computations:

i	x(i)	f(i)	f'(i)	error(i)
0	1	0.12	-0.94	
1	1.127659574	-0.000229994	-0.943498205	0.127659574
2	1.127415807	-7.4443E-10	-0.943492097	0.000243768
3	1.127415806	0	-0.943492097	7.89015E-10

Therefore, the last value of $x = 1.127415806$ in the above table is the root and solves *Eq.8*, and *Eq.6*. This is the value of y_1 for the first-time step in the solution of ODE by Backward Euler in *Eq.6*.

NEXT TASKS: This means you are ready to compute the next time step. In our matlab code, all the burden of the root finding step is alleviated by the function *fsolve*. Professor: *didn't you say 'quick solution by hand'? where is the 'quick'*part*, shouldn't be re-labeld as the 'slow solution by hand'?*

Summarizing our results so far:

j	t_j	y_j
0	0	1
1	0.1	1.127415806
…	…	…
11	1.0	

The equation for the next time step

$$y_2 = y_1 + h\,(1 + 2t_2)\sqrt{y_2}$$

Where $y_1 = 1.127415806$, $h = 0.1$, and $t_2 = 0.2$

ASSIGNMENT: Student should:

(i) Work on one more time step (e.g., use Excel for the root finding step)
(ii) Run matlab code to generate the graph (appendix-1) and have a feeling where the root is at.
(iii) Run matlab code in appendix-2 and study the results and graph

APPENDIX-1

```
% Plot

clc, clear, clf

f=@(x) 1+0.12*sqrt(x)-x;
```

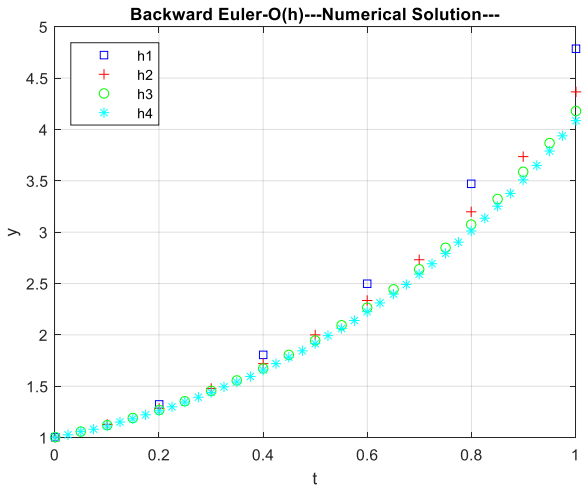
```
fplot(f,[0,2]);
grid on
title('1+0.12*sqrt(x)-x');
```

APPENDIX-2

In the code below four different values of h are used, each new one is half the previous one: $h_{i+1} = \frac{h_i}{2}$. The nonlinear algebraic equation is solved with the matlab function ***fsolve***. The estimated error is computed as $e_i = |y_i^k - y_i^{k+1}|$, where k and $k+1$ are two successive iterations and i is the node label. Also, GDE is computed with the maximum error in each iteration.

```
% Backward Euler Method with MATLAB
% Solves IVP-ODE using implicit Euler's method
% Equation to solve:    y'=(1+2*t)*sqrt(y);   y(0)=1; t=[0,1];
% Author: Marco Arocha; File: EulerBackward.m

clc, clear
close all
clf

% instruction to write results on external file
fid=fopen('EulerOut2.m','w');

% Some parameters and auxiliary variables
h=0.2; a=0; b=1;d=2; dd=1;

% table title
fprintf(fid,'%7s %7s %7s %10s\n','i','t(i)','y(i)','er(i)');
for jj=1:1:4        % Four runs with different h's.

   t = a:h:b;       % time vector
   y = zeros(1,length(t));

   y(1) = 1;        % initial condition

  fprintf(fid,'-------------------------------------------------\n');
  fprintf(fid,' h =%f \n',h);
  for ii=1:1:(length(t)-1)
     % Root finding with matlab function fzero
     yo=y(ii);  tn=t(ii+1);
     fun=@(yn) yo+h*(1+2*tn)*sqrt(yn)-yn;
     y(ii+1)=fzero(fun,yo);
  end

  % Local Estimated Error + Output
  if jj==1
```

```
    Y=y;
    for ii=1:1:numel(t)
      fprintf(fid,'%7d %7.2f %7.3f\n',ii, t(ii), y(ii));
    end
  else
    k=1;
    for ii=1:d:numel(t)
      er(ii)=abs(y(ii)-Y(k));
      fprintf(fid,'%7d %7.2f %7.3f %10.3e\n',ii, t(ii), y(ii),er(ii));
      k=k+dd;
    end
    GDE=max(er);
    fprintf(fid,'GDE= %.3e \n',GDE);
    Y=y;
  end
  h=h/2.0;                          % In each new iteration program halves the step size

  data=['sb';'+r';'og';'*c'];       % 4 different markers
  celldata = cellstr(data);         % cell array of strings
  tt=numel(t), yy=numel(y)
  plot(t,y,celldata{jj});
  hold on                           % allows to draw several curves in single plot

  if jj>=3, d=2*d; dd=dd*2; end

end

% Chulerias adicionales para la grafica
title('Backward Euler-O(h)---Numerical Solution---');
ylabel('y');    xlabel('t');
grid on
leyendas=[' h1 ';' h2 ';' h3 ';' h4 '];
leyendata=cellstr(leyendas);
legend(leyendata,'Location','northwest');

hold off
fclose(fid);
```



OUTPUT ('EulerOut2.m') Selected data points are shown:

```
    i     t(i)      y(i)      er(i)
-------------------------------------------------
 h =0.200000
    1    0.00    1.000
    2    0.20    1.322
    3    0.40    1.806
    4    0.60    2.502
    5    0.80    3.470
    6    1.00    4.782
```

```
    i     t(i)      y(i)      er(i)
-------------------------------------------------
 h =0.100000
    1    0.00    1.000    0.000e+00
    3    0.20    1.286    3.574e-02
    5    0.40    1.717    8.894e-02
    7    0.60    2.336    1.658e-01
    9    0.80    3.197    2.729e-01
   11    1.00    4.366    4.168e-01
GDE= 4.168e-01
```

```
    i     t(i)      y(i)      er(i)
-------------------------------------------------

h =0.050000
    1    0.00    1.000    0.000e+00
    3    0.10    1.120    7.384e-03
    5    0.20    1.270    1.634e-02
    7    0.30    1.454    2.722e-02
    9    0.40    1.676    4.038e-02
   11    0.50    1.943    5.619e-02
   13    0.60    2.261    7.503e-02
   15    0.70    2.635    9.731e-02
   17    0.80    3.074    1.234e-01
   19    0.90    3.585    1.538e-01
   21    1.00    4.177    1.888e-01
GDE= 1.888e-01
```

```
    i     t(i)      y(i)      er(i)
-------------------------------------------------
 h =0.025000
    1    0.00    1.000    0.000e+00
    5    0.10    1.116    3.548e-03
    9    0.20    1.262    7.831e-03
   13    0.30    1.441    1.301e-02
   17    0.40    1.657    1.927e-02
   21    0.50    1.917    2.678e-02
   25    0.60    2.225    3.574e-02
   29    0.70    2.589    4.632e-02
   33    0.80    3.015    5.874e-02
   37    0.90    3.512    7.319e-02
   41    1.00    4.087    8.988e-02
GDE= 1.538e-01
```