EPFL

Numerical Analysis and Computational Mathematics

Fall Semester 2025 – CSE Section

Prof. Laura Grigori

Assistant: Israa Fakih

Session 12 – December 3, 2025

Solutions – Ordinary differential equations

Solution I (MATLAB)

a) The forward Euler method approximates the solution $y(t)$ of equation (1) as:

$$\begin{cases} u_{n+1} = u_n + h\, f(t_n, u_n) & n = 0, \dots, N_h - 1, \\ u_0 = y_0. \end{cases}$$

Since the approximate solution u_{n+1} at the discrete time t_{n+1} appears only on the left-hand side of the first relation, the method is explicit and does not require the solution of any equations. The MATLAB implementation is straightforward and reads:

```
function [ tv, uv ] = forward_euler( fun, y0, t0, tf, Nh )
% FORWARD_EULER Forward Euler method for the scalar ODE in the form
% y'(t) = f(t,y(t)),  t \in (t0,tf)
% y(0) = y_0
%
%  [ tv, uv ] = forward_euler( fun, y0, t0, tf, Nh )
%  Inputs: fun    = function handle for f(t,y), fun = @(t,y) ...
%          y0     = initial value
%          t0     = initial time
%          tf     = final time
%          Nh     = number of time subintervals
%  Output: tv     = vector of time steps (1 x (Nh+1))
%          uv     = vector of approximate solution at times tv
%

tv = linspace( t0, tf, Nh + 1 );
h = ( tf - t0 ) / Nh;

uv = zeros( 1, Nh + 1 );
uv( 1 ) = y0;

for n = 1 : Nh
    uv( n + 1 ) = uv( n ) + h * fun( tv( n ), uv( n ) );
end
```

```
return
```

The Heun method is:

$$\begin{cases} u_{n+1} = u_n + \frac{h}{2}\left[f(t_n, u_n) + f(t_{n+1}, u_n + hf(t_n, u_n))\right] & n = 0, \ldots, N_h - 1, \\ u_0 = y_0, \end{cases}$$

or equivalently:

$$\begin{cases} u^*_{n+1} = u_n + hf(t_n, u_n) & n = 0, \ldots, N_h - 1, \\ u_{n+1} = u_n + \frac{h}{2}\left[f(t_n, u_n) + f(t_{n+1}, u^*_{n+1})\right] & n = 0, \ldots, N_h - 1, \\ u_0 = y_0. \end{cases}$$

The Heun method is explicit and does not require the solution of any equations; the Heun method possesses order of accuracy 2 for solutions $y(t) \in C^3(I)$ (convergence order 2). The MATLAB implementation follows:

```
function [ tv, uv ] = heun( fun, y0, t0, tf, Nh )
% HEUN Heun method for the scalar ODE in the form
% y'(t) = f(t,y(t)),  t \in (t0,tf)
% y(0) = y_0
%
%  [ tv, uv ] = heun( fun, y0, t0, tf, Nh )
%  Inputs: fun    = function handle for f(t,y), fun = @(t,y) ...
%          y0     = initial value
%          t0     = initial time
%          tf     = final time
%          Nh     = number of time subintervals
%  Output: tv     = vector of time steps (1 x (Nh+1))
%          uv     = vector of approximate solution at times tv
%

tv = linspace( t0, tf, Nh + 1 );
h = ( tf - t0 ) / Nh;

uv = zeros( 1, Nh + 1 );
uv( 1 ) = y0;

for n = 1 : Nh
    u_star = uv( n ) + h * fun( tv( n ), uv( n ) );
    uv( n + 1 ) = uv( n ) + h / 2 * ( fun( tv( n ), uv( n ) ) + ...
                                      fun( tv( n + 1 ), u_star ) );
end

return
```

b) The following MATLAB commands allow computing the approximated solutions of $y(t)$ by the forward Euler and the Heun methods implemented at point a) and to plot them in Fig. 1(left).

```
fun = @( t, y ) 1 - y.^2;
t0 = 0;   tf = 5;
y0 = ( exp( 1 ) - 1 ) / ( exp( 1 ) + 1 );
Nh = 10;
```

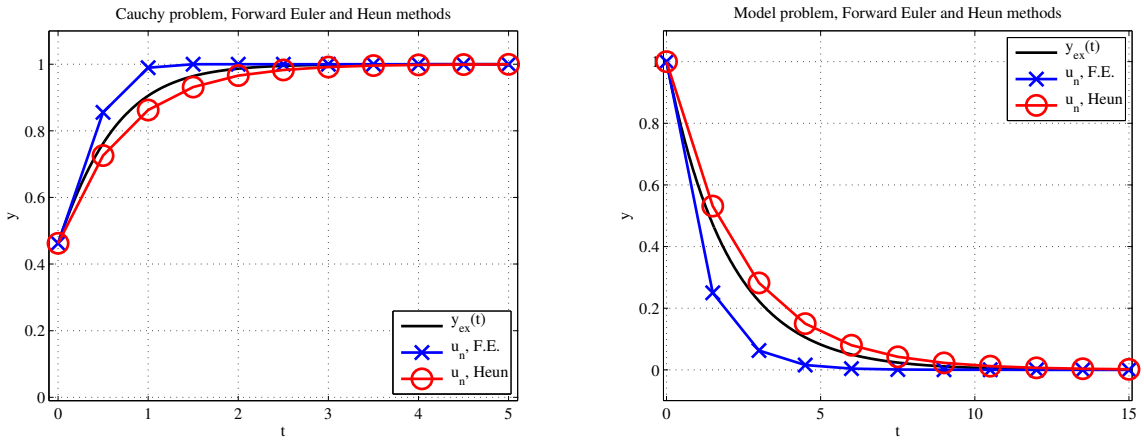


Figure 1: Numerical approximations of $y(t)$ by the forward Euler and Heun methods for $f(t, y) = 1 - y^2$ (left) and for the model problem (2) with $f(t, y) = -y/2$ (right).

```
[ tv, uv_forward_euler ] = forward_euler( fun, y0, t0, tf, Nh );
[ tv, uv_heun ] = heun( fun, y0, t0, tf, Nh );
tv_plot = linspace( t0, tf, 1001 );
y_ex = @( t ) ( exp( 2 * t + 1 ) - 1 ) ./ ( exp( 2 * t + 1 ) + 1 );
figure
plot( tv_plot, y_ex( tv_plot ), '-k', ...
      tv, uv_forward_euler, '-xb', tv, uv_heun, '-or' );
grid on; xlabel('t'); ylabel('y');
legend(' y_{ex}(t)',' u_{n}, F.E.',' u_{n}, Heun' );
```

Qualitatively, the numerical solutions approximate well the exact solution $y(t)$ for $N_h = 10$.

c) The model problem (2) defined on the interval $I = (t_0, t_f)$ has $y(t) = \exp(-t/2)$ as exact solution. The following MATLAB code computes the numerical approximations of $y(t)$ by the forward Euler and the Heun methods. See Fig. 1 (right).

```
lambda = - 0.5;
fun = @( t, y ) lambda * y;
t0 = 0;  tf = 15;
y0 = 1;
Nh = 10;
[ tv, uv_forward_euler ] = forward_euler( fun, y0, t0, tf, Nh );
[ tv, uv_heun ] = heun( fun, y0, t0, tf, Nh );
tv_plot = linspace( t0, tf, 1001 );
y_ex = @( t ) y0 * exp( lambda * t );
figure
plot( tv_plot, y_ex( tv_plot ), '-k', ...
      tv, uv_forward_euler, '-xb', tv, uv_heun, '-or' );
grid on
xlabel('t');
ylabel('y');
legend(' y_{ex}(t)',' u_{n}, F.E.',' u_{n}, Heun' );
```

Again, qualitatively, the numerical solutions approximate well the exact solution $y(t)$ for $N_h = 10$.

d) The backward Euler method approximates the solution y of equation (1) as:

$$\begin{cases} u_{n+1} = u_n + h\, f(t_{n+1}, u_{n+1}) & n = 0, \dots, N_h - 1, \\ u_0 = y_0. \end{cases}$$

Since u_{n+1} appears on both sides of the previous relation, the method is implicit and requires the solution of the nonlinear equation

$$F_n^{\mathrm{BE}}(u_{n+1}) := u_{n+1} - u_n - h\, f(t_{n+1}, u_{n+1}) = 0 \qquad \text{for all } n = 0, \dots, N_h - 1,$$

at each time step t_n with $n = 0, \dots, N_h - 1$. For the model problem (2) this equation is of the form

$$F_n^{\mathrm{BE}}(u_{n+1}) := u_{n+1} - u_n - \lambda h u_{n+1} = 0 \qquad \text{for all } n = 0, \dots, N_h - 1,$$

whose solution is $u_{n+1} = \frac{u_n}{(1-\lambda h)}$ for $n = 0, \dots, N_h - 1$.

Thus, the MATLAB implementation reads:

```
function [ tv, uv ] = backward_euler_modelproblem( lambda, y0, t0, tf, Nh )
% BACKWARD_EULER_MODELPROBLEM Backward Euler method for the model problem
% ODE in the form
% y'(t) = lambda y(t),  t \in (t0,tf)
% y(0) = y_0
%
%  [ tv, uv ] = backward_euler_modelproblem( lambda, y0, t0, tf, Nh )
%  Inputs: lambda = real parameter (negative)
%          y0     = initial value
%          t0     = initial time
%          tf     = final time
%          Nh     = number of time subintervals
%  Output: tv     = vector of time steps (1 x (Nh+1))
%          uv     = vector of approximate solution at times tv
%

tv = linspace( t0, tf, Nh + 1 );
h = ( tf - t0 ) / Nh;

uv = zeros( 1, Nh + 1 );
uv( 1 ) = y0;

for n = 1 : Nh
    uv( n + 1 ) = uv( n ) / ( 1 - h * lambda );
end

return
```

The Crank-Nicolson method reads:

$$\begin{cases} u_{n+1} = u_n + \frac{h}{2}\, [f(t_n, u_n) + f(t_{n+1}, u_{n+1})] & n = 0, \dots, N_h - 1, \\ u_0 = y_0. \end{cases}$$

The nonlinear equation to be solved at each time t_n, with $n = 0, \dots, N_h - 1$, is

$$F_n^{\mathrm{CN}}(u_{n+1}) := u_{n+1} - u_n - \frac{h}{2}\, [f(t_n, u_n) + f(t_{n+1}, u_{n+1})] = 0 \qquad \text{for all } n = 0, \dots, N_h - 1,$$

which, in the case of the model problem (2), simplifies to

$$F_n^{\mathrm{CN}}(u_{n+1}) = u_{n+1} - u_n - \frac{\lambda h}{2}\left[u_n + u_{n+1}\right] = 0 \qquad \text{for all } n = 0, \ldots, N_h - 1.$$

The solution is then $u_{n+1} = \frac{1+\lambda h/2}{1-\lambda h/2}\, u_n$ for all $n = 0, \ldots, N_h - 1$.
We consider the following MATLAB implementation:

```
function [ tv, uv ] = crank_nicolson_modelproblem( lambda, y0, t0, tf, Nh )
% CRANK_NICOLSON_MODELPROBLEM Crank-Nicolson method for the model problem
% ODE in the form
% y'(t) = lambda y(t),  t \in (t0,tf)
% y(0) = y_0
%
%  [ tv, uv ] = crank_nicolson_modelproblem( lambda, y0, t0, tf, Nh )
%  Inputs: lambda = real parameter (negative)
%          y0     = initial value
%          t0     = initial time
%          tf     = final time
%          Nh     = number of time subintervals
%  Output: tv     = vector of time steps (1 x (Nh+1))
%          uv     = vector of approximate solution at times tv
%

tv = linspace( t0, tf, Nh + 1 );
h = ( tf - t0 ) / Nh;

uv = zeros( 1, Nh + 1 );
uv( 1 ) = y0;

alpha = h * lambda / 2;

for n = 1 : Nh
    uv( n + 1 ) = uv( n ) * ( 1 + alpha ) / ( 1 - alpha );
end

return
```

e) We consider the following MATLAB commands:

```
[ tv, uv_backward_euler ] = backward_euler_modelproblem( lambda, ...
                                                          y0, t0, tf, Nh );
[ tv, uv_crank_nicolson ] = crank_nicolson_modelproblem( lambda, ...
                                                          y0, t0, tf, Nh );
figure
plot( tv_plot, y_ex( tv_plot ), '-k', ...
      tv, uv_backward_euler, '-dg', tv, uv_crank_nicolson, '-sm'  );
grid on; xlabel('t'); ylabel('y');
legend(' y_{ex}(t)',' u_{n}, B.E.',' u_{n}, C.-N.' );
```

We observe in Fig. 2 that the numerical approximation given by the Crank-Nicolson method is quite accurate. It is important to understand that even between two methods of the same order of accuracy (convergence order), such as the Heun and the Crank-Nicolson methods, which are both accurate of order 2 for $y(t) \in C^3(I)$, there can be noticeable differences in the quality of the approximate solutions.

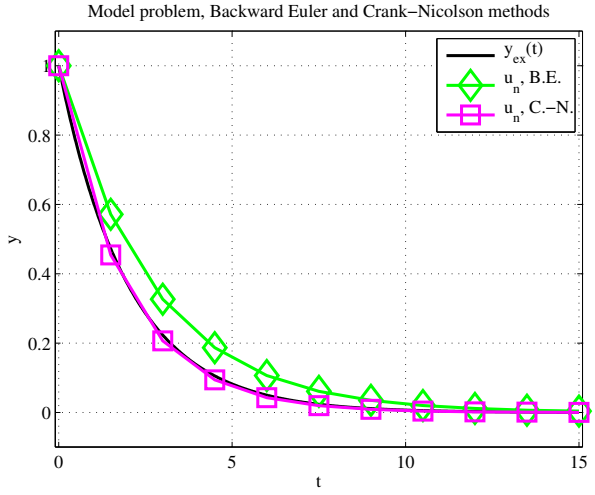


Figure 2: Numerical approximations of the model problem (2) with $f(t, y) = \lambda\, y$ and $\lambda = -0.5$ by the backward Euler and Crank-Nicolson methods.

f) If, for a general method, the error $e_n := |y_n - u_n|$ at the discrete time t_n with $n = 0, \ldots, N_h$ is bounded as:

$$e_n \leq C h^p \qquad \text{for } n = 0, \ldots, N_h,$$

when the time step h is "sufficiently" small ($h \to 0$), we say that the method is *convergent of order* p. Next we shall study the convergence of the four aforementioned methods as $h \to 0$ on the model problem (2) for which $y(t) \in C^\infty(I)$.

Note that, in order to evaluate the errors e_n at time $\bar{t} = 10$, the number of subintervals $N_h = |I|/h$ must be chosen such that $(\bar{t} - t_0)/h \in \mathbb{N}$. For the values used, this implies that we must choose N_h such that $10\, N_h/15 \in \mathbb{N}$, and so we choose $N_h = 15, 30, 45, 60, 120, 240, 480$.

The MATLAB code to compute the four different approximations for different values of N_h and to plot the convergence graph in Fig. 3 is as follows:

```
errv_n_forward_euler = [];
errv_n_backward_euler = [];
errv_n_heun = [];
errv_n_crank_nicolson = [];
Nhv = [ 15 30 60 120 240 480 ];
hv = ( tf - t0 ) ./ Nhv;
t_bar = 10;
for Nh = Nhv
    h = ( tf - t0 ) / Nh;    % h
    n = ( t_bar - t0 ) / h; % step n varies with Nh
   [ tv, uv_forward_euler ] = forward_euler( fun, y0, t0, tf, Nh );
   [ tv, uv_heun ] = heun( fun, y0, t0, tf, Nh );
   [ tv, uv_backward_euler ] = backward_euler_modelproblem( lambda, ...
                                                        y0, t0, tf, Nh );
   [ tv, uv_crank_nicolson ] = crank_nicolson_modelproblem( lambda, ...
                                                        y0, t0, tf, Nh );
   errv_n_forward_euler = [ errv_n_forward_euler, ...
                       abs( uv_forward_euler( n + 1 ) - y_ex( t_bar ) ) ];
   errv_n_heun = [ errv_n_heun, ...
                       abs( uv_heun( n + 1 ) - y_ex( t_bar ) ) ];
```

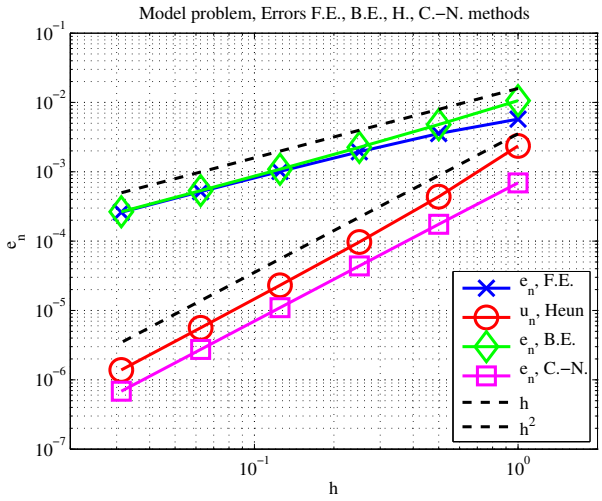


Figure 3: Comparison of the order of convergence of the different methods for the model problem (2).

```
   errv_n_backward_euler = [ errv_n_backward_euler, ...
                        abs( uv_backward_euler( n + 1 ) - y_ex( t_bar ) ) ];
   errv_n_crank_nicolson = [ errv_n_crank_nicolson, ...
                        abs( uv_crank_nicolson( n + 1 ) - y_ex( t_bar ) ) ];
end
figure
loglog( hv, errv_n_forward_euler, '-xb', ...
        hv, errv_n_heun, '-or', ...
        hv, errv_n_backward_euler, '-dg', ...
        hv, errv_n_crank_nicolson, '-sm', ...
        hv, 1.5 * errv_n_backward_euler( 1 ) / hv( 1 ) * hv, '--k',...
        hv, 1.5 * errv_n_heun( 1 ) / hv( 1 )^2 * hv.^2, '--k' );
grid on; xlabel('h'); ylabel('e_n');
legend(' e_{n}, F.E.',' u_{n}, Heun', ' e_{n}, B.E.',' e_{n}, C.-N.', ...
       ' h', ' h^2' );
```

From the convergence plot in Fig. 3, we verify that the forward Euler and backward Euler methods converge with order 1. The Heun and Crank-Nicolson methods converge with order 2.

g) A method for the model problem $y'(t) = \lambda\, y(t)$ with $\lambda < 0$ is called *absolutely stable* (on unbounded intervals) for a given $\lambda\, h$ if the discrete sequence u_n is such that:

$$\lim_{n\to\infty} u_n = 0.$$

In general, a linear one-step method applied to the model problem (2) on the unbounded interval $(t_0, +\infty)$ will produce a sequence in the form $u_{n+1} = R(\lambda\, h)\, u_n$ for $n \geq 0$, where the stability function R is a polynomial if the method is explicit or a rational function if the method is implicit. It follows that $u_n = R(\lambda\, h)^n\, u_0$ for $n \geq 0$. The region of absolute stability is then defined as:

$$\mathcal{A} := \{z = \lambda\, h \in \mathbb{C} \,:\, |R(\lambda\, h)| < 1\}\,.$$

The stability functions $R(\lambda\, h)$ and the conditions for absolute stability of the four methods considered are:

Method	$R(\lambda\, h)$	Region of absolute stability (on $\mathbb{R}^-$)
Forward Euler	$1+\lambda h$	$0 < h < h_{max} = \dfrac{2}{\lvert\lambda\rvert}$
Backward Euler	$\dfrac{1}{1-\lambda h}$	unconditional ($h > 0$)
Heun	$1+\lambda h+\dfrac{(\lambda h)^2}{2}$	$0 < h < h_{max} = \dfrac{2}{\lvert\lambda\rvert}$
Crank-Nicolson	$\dfrac{1+\lambda h/2}{1-\lambda h/2}$	unconditional ($h > 0$)

Table 1: Absolute stability regions of different numerical methods.

We apply the four numerical methods to the model problem (2) with $\lambda = -0.5$, $t_f = 40$, and $t_0 = 0$. We observe that, when $N_h = 9$, we have $|\lambda|\ h \simeq 2.2$, when $N_h = 10$, we have $|\lambda|\ h = 2$, and, when $N_h = 11$, we have $|\lambda|\, h \simeq 1.8$. Therefore, we expect the two explicit methods (forward Euler and Heun methods) to be unstable for $N_h = 9$, *marginally* but not absolutely stable for $N_h = 10$, and absolutely stable for $N_h = 11$.

This prediction is verified by the left column of Fig. 4. If $N_h = 10$ the explicit methods are not absolutely stable, but the approximations remain bounded. For $N_h = 11$ both the forward Euler and Heun methods are absolutely stable, but the forward Euler oscillates greatly and the approximation is very poor.

From the right column of Fig. 4 we can observe that both the implicit methods (backward Euler and Crank-Nicolson) are absolutely stable for all the values of N_h, as expected. Nevertheless, we observe that for h large (N_h small) the approximation given by the Crank-Nicolson method displays some initial oscillation[1].

Exercise II (MATLAB)

a) As mentioned previously (Exercise 1, point d)), the backward Euler method requires the solution of the nonlinear equation:

$$F_n^{\mathrm{BE}}(u_{n+1}) := u_{n+1} - u_n - hf(t_{n+1}, u_{n+1}) = 0 \qquad \text{for all } n = 0, \ldots, N_h - 1.$$

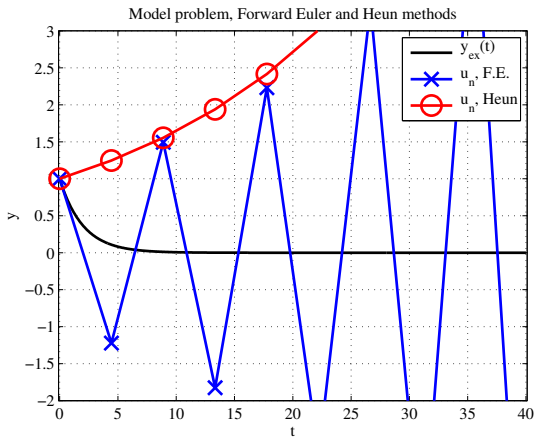
To apply the Newton's method to its solution, we note that:

$$(F_n^{\mathrm{BE}})'(u_{n+1}) := 1 - h\frac{\partial f}{\partial y}(t_{n+1}, u_{n+1}) \qquad \text{for all } n = 0, \ldots, N_h - 1,$$

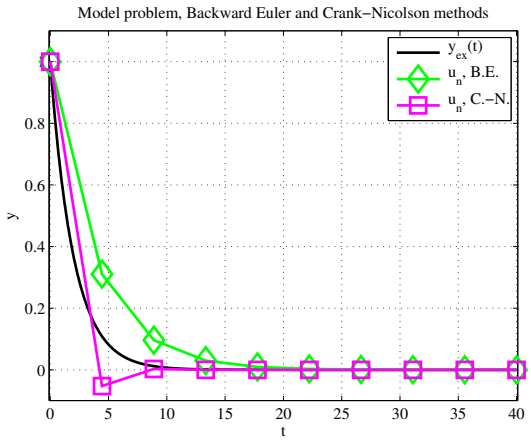
and so we can write the Newton method for all $n = 0, \ldots, N_h - 1$ as:

$$\begin{cases} u_{n+1}^{(k+1)} = u_{n+1}^{(k)} - \dfrac{F_n^{\mathrm{BE}}(u_{n+1}^{(k)})}{(F_n^{\mathrm{BE}})'(u_{n+1}^{(k)})} & k \geq 0, \\ u_{n+1}^{(0)} = u_n, \end{cases}$$

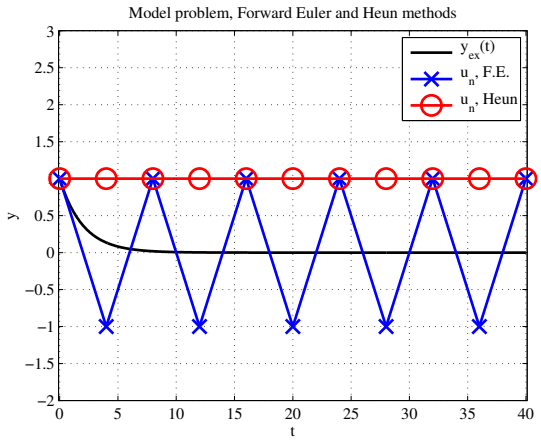
[1]This is due to the fact that for the Crank-Nicolson method the stability function $R(\lambda\, h)$ satisfies $\lim_{|\lambda\, h|\to\infty} |R(\lambda\, h)| = 1$. So, if $\lambda \ll 0$ or the step size h is very large, the Crank-Nicolson method starts to exhibit some oscillations. The drawback of this method is that it contains insufficient damping for oscillations that may arise due to the presence of very fast transients in the solutions.



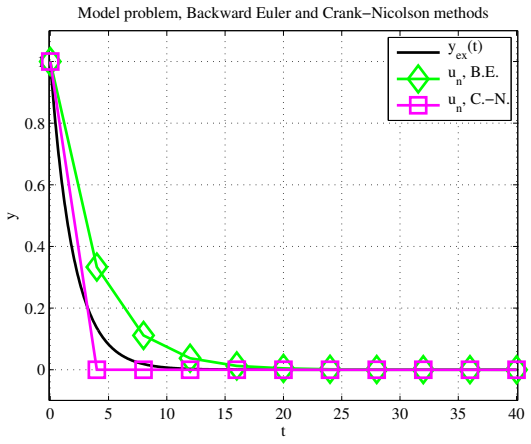
$N_h = 9$



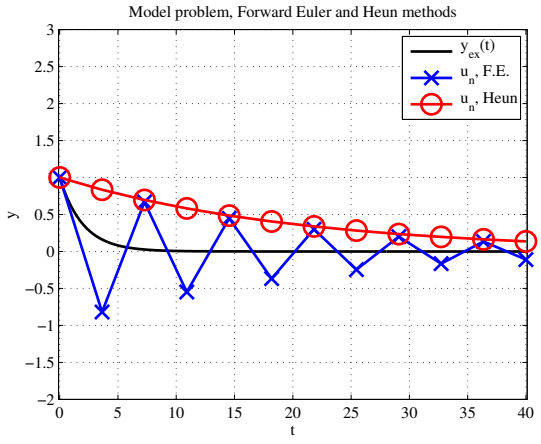
$N_h = 9$



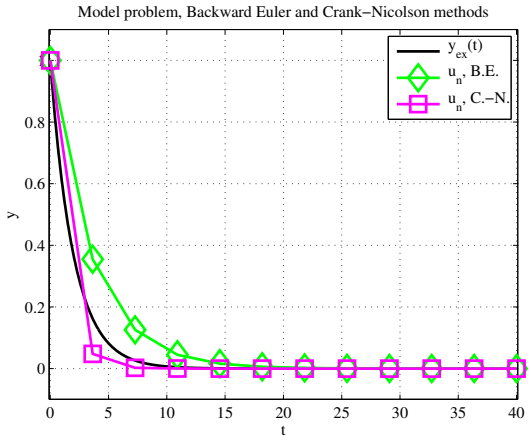
$N_h = 10$



$N_h = 10$



$N_h = 11$



$N_h = 11$

Figure 4: Left column: behavior of the explicit methods (forward Euler and Heun) for $N_h = 9$ (top), $N_h = 10$ (middle), and $N_h = 11$ (bottom). Right column: behavior of the implicit methods (backward Euler and Crank-Nicolson) for $N_h = 9$ (top), $N_h = 10$ (middle), and $N_h = 11$ (bottom).

using as initial guess the solution u_n at the previous time step. The solution at the discrete time t_{n+1} is set to $u_{n+1} = u_{n+1}^{(k+1)}$ for k "sufficiently" large according to some stopping criterion. Then, the MATLAB implementation of the backward Euler method is:

```
function [ tv, uv ] = backward_euler( fun, dfun_y, y0, t0, tf, Nh )
% BACKWARD_EULER Backward Euler method for the scalar ODE in the form
% y'(t) = f(t,y(t)),  t \in (t0,tf)
% y(0) = y_0
%
% The Newton method is used to solve the nonlinear equation at each time
% step. The function newton.m is used.
%
%  [ tv, uv ] = backward_euler( fun, dfun_y, y0, t0, tf, Nh )
%  Inputs: fun    = function handle for f(t,y), fun = @(t,y) ...
%          dfun_y = derivative of f(t,y) w.r.t. y, dfun_y = @(t,y) ...
%          y0     = initial value
%          t0     = initial time
%          tf     = final time
%          Nh     = number of time subintervals
%  Output: tv     = vector of time steps (1 x (Nh+1))
%          uv     = vector of approximate solution at times tv
%

tv = linspace( t0, tf, Nh + 1 );
h = ( tf - t0 ) / Nh;

uv = zeros( 1, Nh + 1 );
uv( 1 ) = y0;

for n = 1 : Nh
    f = @( x ) x - uv( n ) - h * fun( tv( n + 1 ), x );
    df = @( x ) 1 - h * dfun_y( tv( n + 1 ), x );
    [ xv, res, niter ] = newton( f, df, uv( n ), 1e-10, 20 );
    uv( n + 1 ) = xv( end );
end

return
```

b) We obtain the results reported in Fig. 5 by using the following MATLAB commands:

```
alpha = pi/2;  beta = pi/3;
fun = @( t, y ) alpha * y * ( 1 - 1 / beta * y );
dfun_y = @( t, y ) alpha * ( 1 - 2 / beta * y );
t0 = 0;  tf = 20;  y0 = 0.4;
Nh = 20;
[ tv, uv_forward_euler ] = forward_euler( fun, y0, t0, tf, Nh );
[ tv, uv_backward_euler ] = backward_euler( fun, dfun_y, y0, t0, tf, Nh );
tv_plot = linspace( t0, tf, 1001 );
y_ex = @( t ) beta * exp( alpha * ( t - t0 ) + log( y0 / ( beta - y0 ) ) ) ./  ...
           ( 1 + exp( alpha * ( t - t0 ) + log( y0 / ( beta - y0 ) ) ) );
figure
plot( tv_plot, y_ex( tv_plot ), '-k', ...
      tv, uv_forward_euler, '-xb', ...
      tv, uv_backward_euler, '-sr' );
grid on;  xlabel('t');  ylabel('y');
legend(' y_{ex}(t)',' u_{n}, F.E.', ' u_{n}, B.E.' );
```

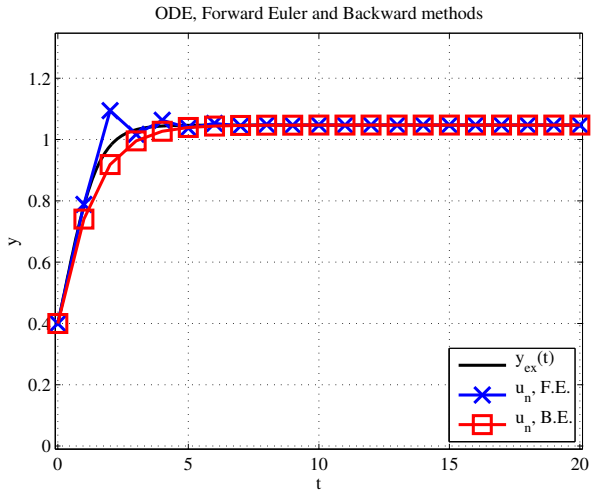


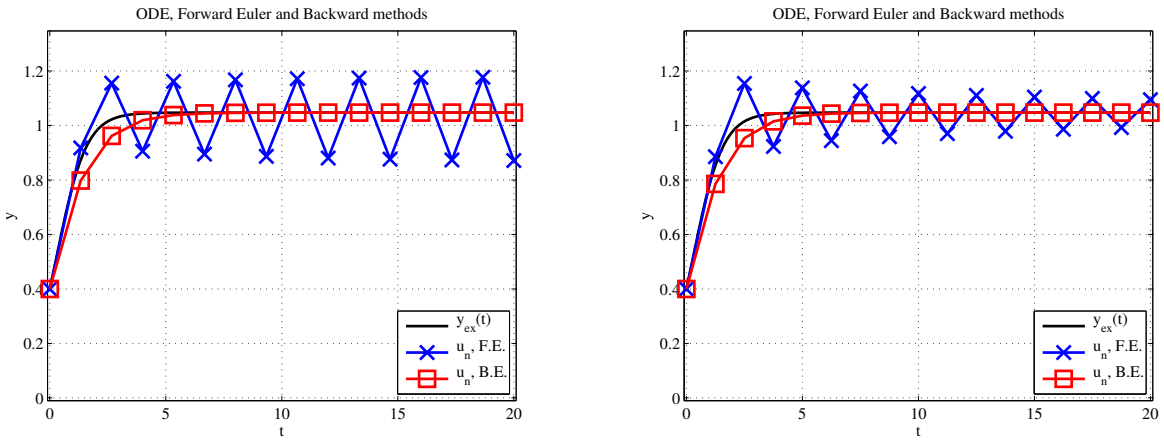
Figure 5: Numerical approximations by the forward and backward Euler methods for $y' = \alpha y \left(1 - \frac{y}{\beta}\right)$.

It can be seen that both the numerical approximations obtained with the forward and backward Euler methods for $N_h = 20$ are (absolutely) stable in the sense that they converge to the asymptotic limit value, $\lim_{t\to\infty} y(t) = \beta = \pi/3$; however, the approximation given by the explicit forward Euler method exhibits some oscillations and is thus a poor approximation of the exact solution.

c) Let $df_{max} = \sup_{t>t_0} \left|\frac{\partial f}{\partial y}(t, y(t))\right|$. For the problem under consideration, we know that $y(t) \in [y_0, \beta]$ for $t \geq 0$ when $\beta > y_0 > 0$. Since $\frac{\partial f}{\partial y}(y) = \alpha \left(1 - \frac{2}{\beta} y\right)$, $y_0 = 0.4$, $\beta = \frac{\pi}{3}$, and $\alpha = \frac{\pi}{2}$, we deduce that $df_{max} = \alpha$. It follows that $h_{max} = \frac{4}{\pi} \simeq 1.2732$. Hence, at least $|I|/h_{\max} = 5\pi \approx 15.7080$ subintervals must be used to guarantee absolute stability. The following MATLAB commands can be considered to obtain the same results:

```
y = linspace( y0, beta, 1001 );
df_val = alpha * ( 1 - 2 / beta * y );
df_max = max( abs( df_val ) )
% df_max =
%     1.5708
h_max = 2 / df_max
% h_max =
%     1.2732
Nh_max = ceil( ( tf - t0 ) / h_max )
% Nh_max =
%     16
```

d) In Fig. 6 we display the numerical solutions obtained with the forward and backward Euler methods for $N_h = 15$ (left) and $N_h = 16$ (right). It can be seen that in the first case the approximation given by the forward Euler method is unstable ($h > h_{max}$). In the second case ($h < h_{max}$), the solution obtained with the forward Euler method is (absolutely) stable. In



$N_h = 15$ $N_h = 16$

Figure 6: Numerical approximations by the forward and backward Euler methods for $y' = \alpha y \left(1 - \frac{y}{\beta}\right)$.

both cases, the approximation given by the explicit forward Euler method oscillates, since, after all, (absolute) stability is not a guarantee of accuracy. On the other hand, we verify that the backward Euler method is, as expected, absolutely stable for all $N_h \neq 1$ ($h > 0$).

The following MATLAB commands can be used to obtain the results in Fig. 6:

```
Nh = 15;  % unstable
[ tv, uv_forward_euler ] = forward_euler( fun, y0, t0, tf, Nh );
[ tv, uv_backward_euler ] = backward_euler( fun, dfun_y, y0, t0, tf, Nh );
tv_plot = linspace( t0, tf, 1001 );
y_ex = @( t ) beta * exp( alpha * ( t - t0 ) + log( y0 / ( beta - y0 ) ) ) ./  ...
              ( 1 + exp( alpha * ( t - t0 ) + log( y0 / ( beta - y0 ) ) ) );
figure
plot( tv_plot, y_ex( tv_plot ), '-k', ...
      tv, uv_forward_euler, '-xb', ...
      tv, uv_backward_euler, '-sr' );
grid on;  xlabel('t');  ylabel('y');
legend(' y_{ex}(t)',' u_{n}, F.E.', ' u_{n}, B.E.' );
```