EPFL

Numerical Analysis and Computational Mathematics

Fall Semester 2025 – CSE Section

Prof. Laura Grigori

Assistant: Israa Fakih

Session 13 – December 10, 2025

Solutions – Ordinary differential equations

Solution I (Theoretical)

Proposition 1 (Cauchy-Lipschitz) *If $f(t, y)$ is continuous w.r.t. both arguments and in addition globally Lipschitz-continuous w.r.t. the second argument y, i.e. there exists $L > 0$ s.t.*

$$|f(t, y_1) - f(t, y_2)| \leq L\,|y_1 - y_2| \qquad \text{for all } t \in I, \text{ for all } y_1, y_2 \in \mathbb{R},$$

for some interval $I = [t_0, t_f]$, then there exists a unique solution $y(t)$ of the Cauchy problem:

$$\begin{cases} y'(t) = f(t, y(t)), & t \in I, \\ y(0) = y_0, \end{cases}$$

for all $y_0 \in \mathbb{R}$.

We verify the conditions of the Cauchy-Lipschitz theorem for the case $f(t, y) = f(y) = -\arctan(y)$, $t_0 = 0$, $t_f = \infty$. First, we note that f is continuous w.r.t. both arguments. We observe that if $f \in C^1(t_0, t_f)$, then:

$$|f(t, y_1) - f(t, y_2)| \leq \sup_{y \in \mathbb{R}, t \in I} \left| \frac{\partial f}{\partial y}(t, y) \right| \cdot |y_1 - y_2|\,,$$

so that, if $\sup_{y \in \mathbb{R}, t \in I} \left| \frac{\partial f}{\partial y}(t, y) \right| < \infty$, we can choose $L := \sup_{y \in \mathbb{R}, t \in I} \left| \frac{\partial f}{\partial y}(t, y) \right|$ as the global Lipschitz constant. Indeed, this is the case, as the partial derivative is:

$$\left| \frac{\partial f}{\partial y}(t, y) \right| = \left| \frac{1}{1 + y^2} \right| \leq 1 \qquad \text{for all } y \in \mathbb{R}, t \in \mathbb{R}.$$

As the ODE $y'(t) = -\arctan(y(t))$ is autonomous ($f(t, y) = f(y)$), the Lipschitz constant L does not depend on the time interval I and we can take $L = 1$.
Since the hypotheses of the Proposition are satisfied, the solution $y(t)$ of the Cauchy problem exists and it is unique for all the times $t \in I$.

Solution II (MATLAB)

a) We consider the following MATLAB implementation:

```
function [ tv, uv ] = runge_kutta_4( fun, y0, t0, tf, Nh )
% RUNGE_KUTTA_4 Runge-Kutta 4, explicit method for the scalar ODE in the
% form:
% y'(t) = f(t,y(t)),  t \in (t0,tf)
% y(0) = y_0
%
%  [ tv, uv ] = runge_kutta_4( 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
    K1 = fun( tv( n ), uv( n ) );
    K2 = fun( tv( n ) + h / 2, uv( n ) + h / 2 * K1 );
    K3 = fun( tv( n ) + h / 2, uv( n ) + h / 2 * K2 );
    K4 = fun( tv( n + 1 ), uv( n ) + h * K3 );
    uv( n + 1 ) = uv( n ) + h / 6 * ( K1 + 2 * K2 + 2 * K3 + K4 );
end

return
```

b) We use the following MATLAB commands to obtain the results in Fig. 1:

```
alpha = 1.5;  beta = - 2e-1;
t0 = 0;    tf = 30;
y0 = alpha + sin( t0 );
fun = @( t, y ) y * ( cos( t ) / ( alpha + sin( t ) ) + beta );
Nh = 30;
%[ tv, uv_forward_euler ] = forward_euler( fun, y0, t0, tf, Nh );
%[ tv, uv_heun ] = heun( fun, y0, t0, tf, Nh );
[ tv, uv_runge_kutta_4 ] = runge_kutta_4( fun, y0, t0, tf, Nh );
tv_plot = linspace( t0, tf, 5001 );
y_ex = @( t ) ( alpha + sin( t ) ) .* exp( beta * ( t - t0 ) );
plot( tv_plot, y_ex( tv_plot ),  '.-r', tv, uv_runge_kutta_4, '.-g' );
grid; axis([-0.1+t0 tf+0.1 -0.25 2.5]);
legend( ' y_{ex}(t)', ' u_{n}, F.E.', ' u_{n}, Heun', ' u_{n}, R.-K. 4' );
```

We observe that the numerical approximation by means of the RK4 method is representing well the exact solution $y(t)$ already for $N_h = 30$, better than the Heun and forward Euler methods do.

c) We recall that the forward Euler method has convergence order 1, while the Heun and RK4

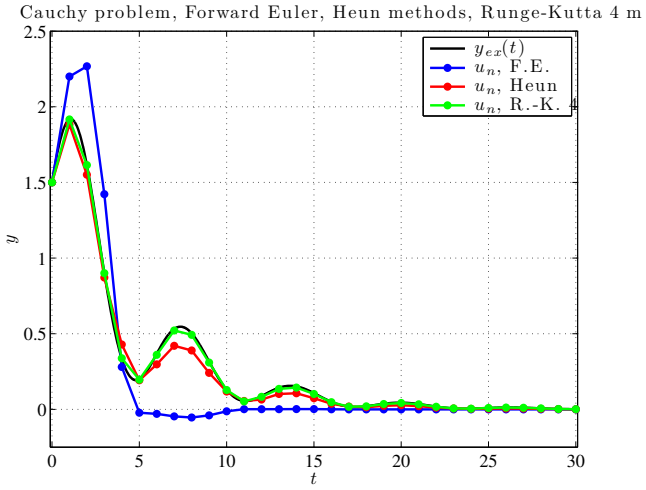


Figure 1: Numerical approximations by the forward Euler, Heun, and RK4 methods for $y' = f(t, y(t))$ and $N_h = 30$.

methods are of orders 2 and 4, respectively, assuming $y(t)$ to be "sufficiently" regular. In this case, for the setup considered at point b), we have $y(t) \in C^\infty(I)$. In order to graphically verify the convergence orders, we compare the errors with the curves (h, h), (h, h^2), and (h, h^4). We consider the following MATLAB commands:

```
errv_n_forward_euler = [];
errv_n_heun = [];
errv_n_runge_kutta_4 = [];
Nhv = [ 30 60 120 240 480 960 ];
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_runge_kutta_4 ] = runge_kutta_4( fun, y0, t0, tf, Nh );
   % notice that vector index starts from 1 in Matlab
   % while n from zero, n = 0,1,...
   errv_n_forward_euler = [ errv_n_forward_euler, ...
                       abs(  y_ex( t_bar ) - uv_forward_euler( n + 1 ) ) ];
   errv_n_heun = [ errv_n_heun, ...
                       abs(  y_ex( t_bar ) - uv_heun( n + 1 ) ) ];
   errv_n_runge_kutta_4 = [ errv_n_runge_kutta_4, ...
                       abs(  y_ex( t_bar ) - uv_runge_kutta_4( n + 1 ) ) ];
end
% scale factors for visualization of the curves  (h,h), (h,h^2), (h,h^4)
scale_factor_1 = 2 * errv_n_forward_euler( end ) / hv( end );
scale_factor_2 = 2 * errv_n_heun( end ) / hv( end )^2;
scale_factor_3 = 2 * errv_n_runge_kutta_4( end ) / hv( end )^4;
figure;
loglog( hv, errv_n_forward_euler, '-xb', hv, errv_n_heun, '-or', ...
        hv, errv_n_runge_kutta_4, '-dg', ...
        hv, scale_factor_1 * hv, '--k', hv, scale_factor_2 * hv.^2, '.-k',...
```

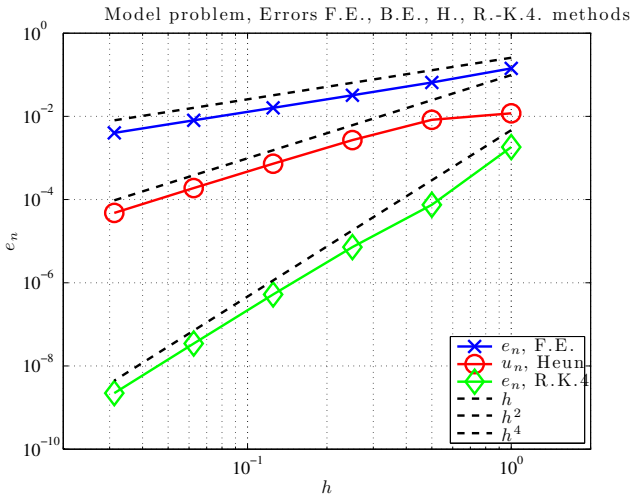


Figure 2: Errors for the forward Euler, Heun, and RK4 methods vs h for $y' = f(t, y(t))$.

```
        hv, scale_factor_3 * hv.^4, ':k' );
grid; axis( [ 2e-2 2 1e-10 1e0 ])
legend(' e_{n}, F.E.',' u_{n}, Heun', ' e_{n}, R.K.4', ...
       ' h', ' h^2', ' h^4' );
```

We obtain the results in Fig. 2. The curves of the errors are parallel (in logarithmic scales) to the curves (h, h), (h, h^2), and (h, h^4) for the forward Euler, Heun, and RK4 methods, respectively. Hence, the results confirm the convergence orders expected from the theory.

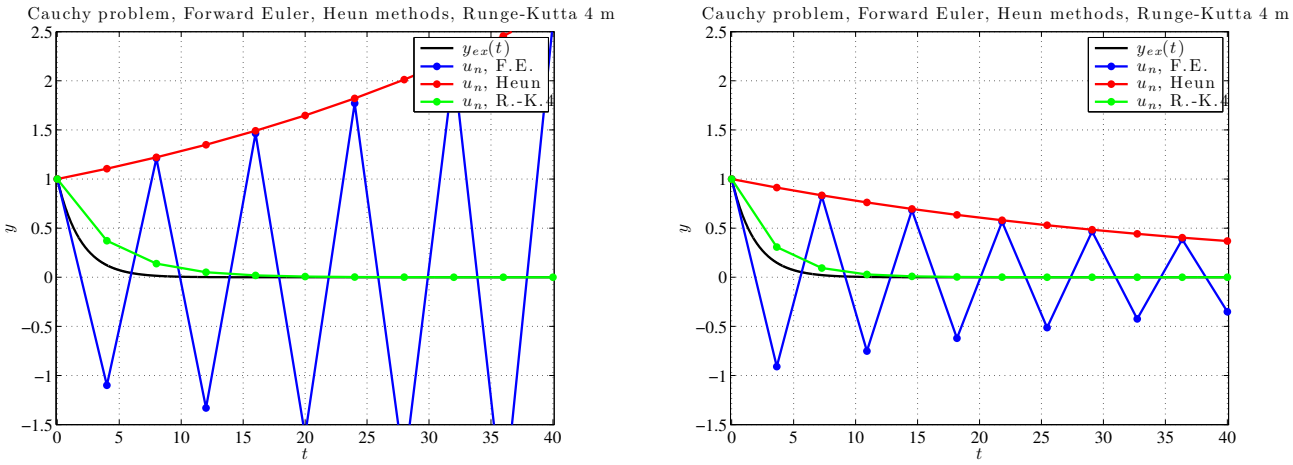
Alternatively, the convergence order of the methods can be estimated e.g. by means of the following MATLAB commands for h "sufficiently" small, since $e_n \leq C\, h^p$ for a general method.

```
conv_ord_Forward_Euler = log( errv_n_forward_euler(end-1) / ...
                              errv_n_forward_euler(end) ) / ...
                                  log( hv(end-1) / hv(end) )
%  conv_ord_Forward_Euler =
%      0.9996
```

We obtain that the estimated convergence order for the forward Euler method is $p_{FE} = 0.9996$. Similarly, for the Heun method we obtain the convergence order $p_H = 1.9971$, while for RK4 we have $p_{RK4} = 3.9618$, as expected.

d) We consider the following MATLAB commands to obtain the results in Fig. 3 (left):

```
t0 = 0;  tf = 40;
lambda = -0.525;  y0 = 1;
fun = @( t, y ) lambda * y;
Nh = 10; % or 11
[ tv, uv_forward_euler ] = forward_euler( fun, y0, t0, tf, Nh );
[ tv, uv_heun ] = heun( fun, y0, t0, tf, Nh );
[ tv, uv_runge_kutta_4 ] = runge_kutta_4( fun, y0, t0, tf, Nh );
tv_plot = linspace( t0, tf, 5001 );
y_ex = @( t ) y0 * exp( lambda * ( t - t0 ) );
figure;
```



$N_h = 10$ $N_h = 11$

Figure 3: Model problem solved with the forward Euler, Heun, and RK4 methods.

```
plot( tv_plot, y_ex( tv_plot ), '-k', tv, uv_forward_euler, '.-b', ...
      tv, uv_heun, '.-r', tv, uv_runge_kutta_4, '.-g' );
grid; axis([-0.1+t0 tf+0.1 -1.5 2.5 ]);
legend( ' y_{ex}(t)', ' u_{n}, F.E.', ' u_{n}, Heun', ...
     ' u_{n}, R.-K.4',  'Location', 'NorthEast' );
```

We deduce from Fig. 3 that for, $N_h = 10$ (i.e. for $h = \frac{t_f - t_0}{N_h} = 4$), the forward Euler and Heun methods are not absolutely stable, while the RK4 method is. In particular, for $\lambda < 0$, we have that the forward Euler and Heun methods are absolutely stable for $0 < h < h_{max}^{FE,H} = \frac{2}{|\lambda|}$, while the RK4 method for $0 < h < h_{max}^{RK4} \simeq \frac{2.7853}{|\lambda|}$. For the model problem under consideration we have $\lambda = -0.525$, so that $h_{max}^{FE,H} = 3.8095$ and $h_{max}^{RK4} = 5.3053$. Since, $h = 4 > h_{max}^{FE,H}$, the forward Euler and Heun methods are not abolutely stable, while the RK4 method is. In order to recover the absolute stability for all the methods, we calculate $N_{h_{max}^{FE,H}}$ as the smallest integer larger than $\frac{t_f - t_0}{h_{max}^{FE,H}}$: $N_{h_{max}^{FE,H}} = 11$. We use the following MATLAB commands:

```
h_max_FE_H = 2 / abs( lambda )
%  h_max_FE_H =
%       3.8095
N_h_max_FE_H = ceil( ( tf - t0 ) / h_max_FE_H )
%  N_h_max_FE_H =
%       11
h_max_RK4 = 2.7853 / abs( lambda )
%  h_max_RK4 =
%       5.3053
N_h_max_RK4 = ceil( ( tf - t0 ) / h_max_RK4 )
%  N_h_max_RK4 =
%        8
```

We report in Fig. 3 (right) the numerical solutions obtained for $N_h = 11$. We verify that all methods are absolutely stable since the condition $h < h_{max}$ is satisfied in all cases. We

remark that, for $N_h < N_{h_{max}^{RK4}} = 8$, all methods would provide (absolutely) unstable numerical solutions.

Exercise III (MATLAB)

a) We consider the following implementation of the MATLAB function `forward_euler_system.m`:

```
function [ tv, uv ] = forward_euler_system( fun, y0, t0, tf, Nh )
% FORWARD_EULER_SYSTEM Forward Euler method for solving a system of ODEs
% in the form:
% y'(t) = F(t,y(t)),  t \in (t0,tf)
% y(0) = y_0
%
% y, y_0 are vectors of size (m x 1)
%
%  [ tv, uv ] = forward_euler_system( fun, y0, t0, tf, Nh )
%  Inputs: fun    = function handle for F(t,y), fun = @(t,y) ...
%                   a vector of size (m x 1) must be returned by fun
%          y0     = initial vector of size (m x 1)
%          t0     = initial time
%          tf     = final time
%          Nh     = number of time subintervals
%  Output: tv     = vector of time steps (1 x (Nh+1))
%          uv     = matrix of the approximate solution at times tv
%                   size (m x (Nh+1))

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

m = length( y0 );
uv = zeros( m, Nh + 1 );
uv( :, 1 ) = y0;

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

return
```

b) We implement the MATLAB function `backward_euler_system_nhcc.m` as:

```
function [ tv, uv ] = backward_euler_system_nhcc( A, g, y0, t0, tf, Nh )
% BACKWARD_EULER_SYSTEM_NHCC Backward Euler method for solving a system of
% ODEs in the nonhomogeneous with constant coefficients form:
% y'(t) = A y(t) + g(t),  t \in (t0,tf)
% y(0) = y_0
%
% y, y_0 are vectors of size (m x 1)
%
%  [ tv, uv ] = backward_euler_system_nhcc( A, g, y0, t0, tf, Nh )
%  Inputs: A      = square matrix of size (m x m)
%          g      = function handle for g(t), g = @(t) ...
%                   a vector of size (m x 1) must be returned by g
```

```
%           y0     = initial vector of size (m x 1)
%           t0     = initial time
%           tf     = final time
%           Nh     = number of time subintervals
%  Output:  tv     = vector of time steps (1 x (Nh+1))
%           uv     = matrix of the approximate solution at times tv
%                    size (m x (Nh+1))

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

m = length( y0 );
uv = zeros( m, Nh + 1 );
uv( :, 1 ) = y0;

I = speye( m, m );
M = ( I - h * A );

for n = 1 : Nh
    uv( :, n + 1 ) = M \ ( uv( :, n ) + h * g( tv( n + 1 ) ) );
end

return
```

c) We solve the problem by using the following MATLAB commands to obtain the results in Fig. 4.

```
t0 = 0;   tf = 5;
y0 = [ 1; 0.6 ];
A = - [ 3 1; 1 1 ];
g = @( t ) [ 0; 0 ] * t;
fun = @( t, y ) A * y;
Nh = 25; % 8 % 9

[ tv, uv_forward_euler ] = forward_euler_system( fun, y0, t0, tf, Nh );
figure;
plot( tv, uv_forward_euler( 1, : ), '.-b', ...
          tv, uv_forward_euler( 2, : ), '.-r' );
grid; axis([-0.1+t0 tf+0.1 -1 1.25]);
title('System of ODEs, Forward Euler');
legend('y(1)','y(2)','Location','NorthEast');

[ tv, uv_backward_euler ] = backward_euler_system_nhcc( A, g, y0, t0, tf, Nh );
figure;
plot( tv, uv_backward_euler( 1, : ), '.-b', ...
          tv, uv_backward_euler( 2, : ), '.-r' );
grid;  axis([-0.1+t0 tf+0.1 -1 1.25]);
title('System of ODEs, Backward Euler');
legend('y(1)','y(2)','Location','NorthEast');
```

The results show that, as expected, both components of the solution tend to zero for t "sufficiently" large for both forward and backward methods.

d) We repeat point c) by considering $N_h = 8$. We obtain the results in Fig. 5. We observe that the numerical solution obtained with the forward Euler method is (absolutely) unstable, while that obtained with the backward Euler method is absolutely stable.

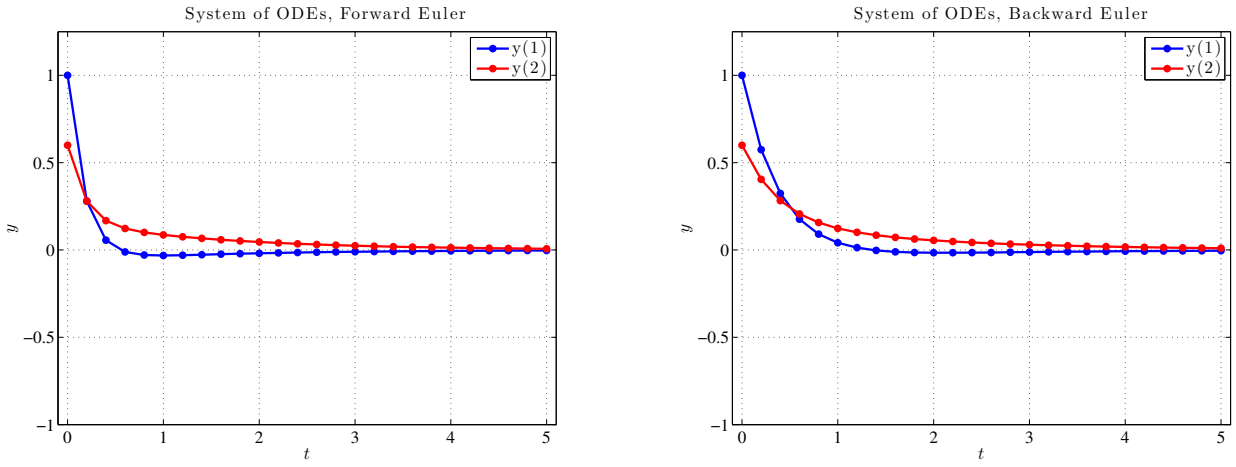
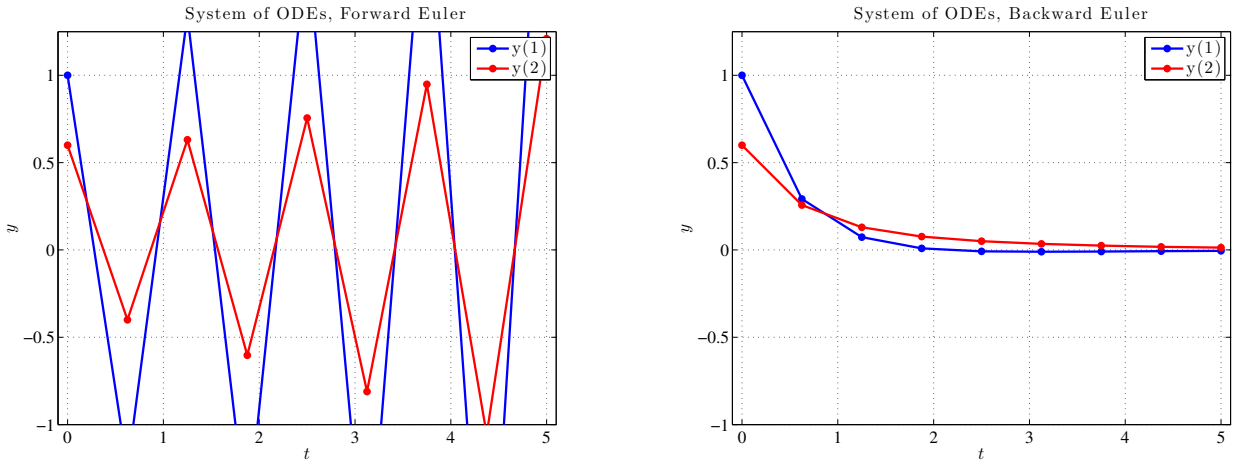


Figure 4: Numerical approximation of the system of ODEs $\mathbf{y}'(t) = \mathbf{F}(t, \mathbf{y}(t)) = A\,\mathbf{y}(t)$ with $\mathbf{y}(t_0) = \mathbf{y}0$; components of the numerical solution $\mathbf{u}_n$ vs. t obtained with the forward Euler method (left) and backward Euler method (right) for $N_h = 25$.

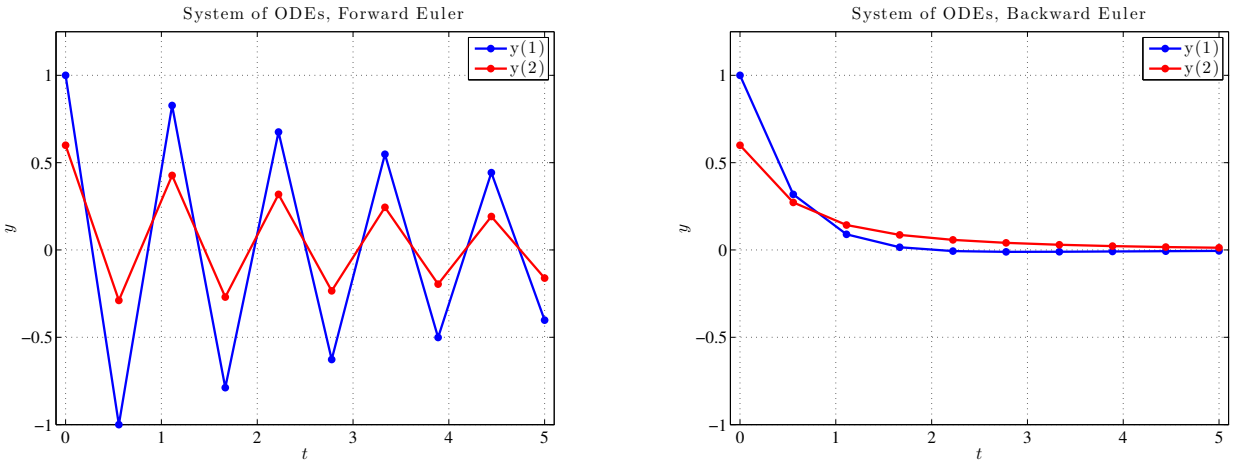


Forward Euler Backward Euler

Figure 5: Numerical approximation of the system of ODEs $\mathbf{y}'(t) = \mathbf{F}(t, \mathbf{y}(t)) = A\,\mathbf{y}(t)$ with $\mathbf{y}(t_0) = \mathbf{y}0$; components of the numerical solution $\mathbf{u}_n$ vs. t obtained with the forward Euler method (left) and backward Euler method (right) for $N_h = 8$.

The result was expected, since the backward Euler method is unconditionally absolutely stable, while the forward Euler method is only conditionally stable. In particular, since the system of ODEs is in the form $\mathbf{y}'(t) = A\,\mathbf{y}(t)$, the stability condition on the step size h for the forward Euler method depends on the eigenvalues of the matrix $A \in \mathbb{C}^{m\times m}$. If all the eigenvalues of A are real and negative, we have the following stability condition for the forward Euler method (and also for the Heun method): $0 < h < h_{max} = \frac{2}{\max_{i,\ldots,m}|\lambda_i(A)|}$.

By using the following MATLAB commands we verify that the eigenvalues of A are real and



Forward Euler Backward Euler

Figure 6: Numerical approximation of the system of ODEs $\mathbf{y}'(t) = \mathbf{F}(t, \mathbf{y}(t)) = A\,\mathbf{y}(t)$ with $\mathbf{y}(t_0) = \mathbf{y}0$; components of the numerical solution $\mathbf{u}_n$ vs. t obtained with the forward Euler method (left) and backward Euler method (right) for $N_h = 9$.

negative, so that the condition above applies: $h < 0.5858$, which corresponds to $N_h \geq N_{h_{max}} = 9$.

```
lambda = ( eig( A ) )'
%  lambda =
%     -3.4142    -0.5858
h_max = 2 / max( abs( lambda ) )
%  h_max =
%     0.5858
N_h_max = ceil( ( tf - t0 ) / h_max )
%  N_h_max =
%     9
```

We report in Fig. 6 the results obtained for $N_h = 9$. We verify that, in this case, the forward Euler method is also absolutely stable, since $0 < h < h_{max}$.