P. Grohs
W. Wu

Spring Term 2016

# Numerical Analysis II

ETH Zürich
D-MATH

## Problem Sheet 8

### Problem 8.1 Autonomisation and Strang-Splitting

The differential equation

$$\dot{\mathbf{y}} = \mathbf{A}(t)\mathbf{y}, \qquad \mathbf{y}(t_0) = \mathbf{y}_0, \quad \mathbf{A}(t) = t\mathbf{A}_0, \quad \mathbf{A}_0 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{8.1.1}$$

can be autonomised by introducing a variable $\mathbf{z} := \begin{bmatrix} \mathbf{y} \\ t \end{bmatrix}$, which yields the ODE

$$\dot{\mathbf{z}} = F(\mathbf{z}) = \begin{bmatrix} \mathbf{A}(t)\mathbf{y} \\ 1 \end{bmatrix}. \tag{8.1.2}$$

**(8.1a)** Formulate the discrete evolution of a Strang splitting method for (8.1.2) corresponding to the decomposition

$$F(\mathbf{z}) = \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{=:\mathbf{f}(\mathbf{z})} + \underbrace{\begin{bmatrix} \mathbf{A}(t)\mathbf{y} \\ 0 \end{bmatrix}}_{=:\mathbf{g}(\mathbf{z})}$$

based on the exact evolutions for $\mathbf{f}$ and $\mathbf{g}$.

HINT: Use the representation of the exact solution of an IVP for a homogeneous linear differential equation .

**Solution:** The exact evolutions to the linear homogeneous differential equations $\dot{\mathbf{z}} = \mathbf{f}(\mathbf{z})$ and $\dot{\mathbf{z}} = \mathbf{g}(\mathbf{z})$ are given by

$$\boldsymbol{\Phi}^h_{\mathbf{f}} \begin{bmatrix} \mathbf{y} \\ t \end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ t+h \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{\Phi}^h_{\mathbf{g}} \begin{bmatrix} \mathbf{y} \\ t \end{bmatrix} = \begin{bmatrix} \mathrm{e}^{h\mathbf{A}(t)}\mathbf{y} \\ t \end{bmatrix}.$$

The discrete evolutions of the corresponding Strang splitting method is now given by $\boldsymbol{\Psi}^h_{\mathbf{F}}\mathbf{z} = \boldsymbol{\Phi}^{h/2}_{\mathbf{f}} \circ \boldsymbol{\Phi}^h_{\mathbf{g}} \circ \boldsymbol{\Phi}^{h/2}_{\mathbf{f}}\mathbf{z}$. We thus obtain

$$\begin{aligned} \boldsymbol{\Psi}^h_{\mathbf{F}} \begin{bmatrix} \mathbf{y} \\ t \end{bmatrix} &= \boldsymbol{\Phi}^{h/2}_{\mathbf{f}} \circ \boldsymbol{\Phi}^h_{\mathbf{g}} \circ \boldsymbol{\Phi}^{h/2}_{\mathbf{f}} \begin{bmatrix} \mathbf{y} \\ t \end{bmatrix} = \boldsymbol{\Phi}^{h/2}_{\mathbf{f}} \circ \boldsymbol{\Phi}^h_{\mathbf{g}} \begin{bmatrix} \mathbf{y} \\ t+\frac{1}{2}h \end{bmatrix} \\ &= \boldsymbol{\Phi}^{h/2}_{\mathbf{f}} \begin{bmatrix} \mathrm{e}^{h\mathbf{A}(t+\frac{1}{2}h)}\mathbf{y} \\ t+\frac{1}{2}h \end{bmatrix} = \begin{bmatrix} \mathrm{e}^{h\mathbf{A}(t+\frac{1}{2}h)}\mathbf{y} \\ t+h \end{bmatrix}. \end{aligned}$$

**(8.1b)** Implement the resulting method by completing the MATLAB template

```
[t, y] = strangaut(y0, t0, T, h)
```

HINT: The matrix exponential function can be computed using `expm`.

**Solution:** See `strangaut.m`, Listing 8.1.

Listing 8.1: Implementation of `strangaut.m`.

```
function [t, y] = strangaut(y0, t0, T, h)
% Strang splitting method for the solution of an autonomized
   IVP
%
% Input:
% y0: initial value
% t0: initial time
% T: end time
% h: step size
%
% output:
% t: time grid
% y: solution at the points of the grid

% initialization
A0 = [0, 1; -1, 0];

% adapt step size, so that end time is attained precisely
N = ceil((T-t0) / h);
h = (T-t0) / N;

% allocate memory for results
t = zeros(N+1, 1);
y = zeros(2, N+1);

% set starting values
t(1) = t0;
y(:, 1) = y0;

% Timestepping
for k = 1:N

     % apply Strang splitting method

     y0 = expm(h * (t0 + 0.5*h) * A0) * y0;
     t0 = t0 + h;
     % save results
     t(k+1) = t0;
     y(:, k+1) = y0;
```

```
end
```

**(8.1c)** Plot the numerical solution obtained by applying the splitting method to the IVP (8.1.1) where $\mathbf{y}_0 = [1, 0]^\top$, $t_0 = 0$, $T = 10$, $h = 0.01$. To do so, complete the template `strangautplot.m`. Use time $t$ as your $z$-coordinate.

HINT: You might find the function `plot3` to be of use.

**Solution:** See `strangautplot.m`, Listing 8.2. The resulting plot is shown in Figure 8.1.

Listing 8.2: Implementation of `strangautplot.m`.

```
function strangautplot
% plot results from strangaut

% initialization
t0 = 0;
T = 10;
h = 0.01;
y0 = [1; 0];

% compute numerical solution
[t, y] = strangaut(y0, t0, T, h);

% plot numerical solution
figure
plot3(y(1, :), y(2, :), t, 'b-')
xlabel('y_1')
ylabel('y_2')
zlabel('t', 'Rotation', 0)
title('numerical solution of the autonomized IVP via the
   strang splitting method')
```

## Problem 8.2 Lie-Trotter Splitting Method

**(8.2a)** Write a MATLAB function

```
function [t,y] = lietrotter(psi1, psi2, y0, tspan, N)
```

which implements the Lie-Trotter splitting method on the time interval `tspan`.

**Solution:** See Listing 8.3.

Listing 8.3: Implementation of the Lie-Trotter splitting method.

```
function [t,y] = lietrotter(psi1,psi2,y0,tspan,N)
% Single step method based on Lie-Trotter splitting
% psi1 and psi2 must be function of type @(h,y)
% y0: initial value
% tspan: [0,T]
```

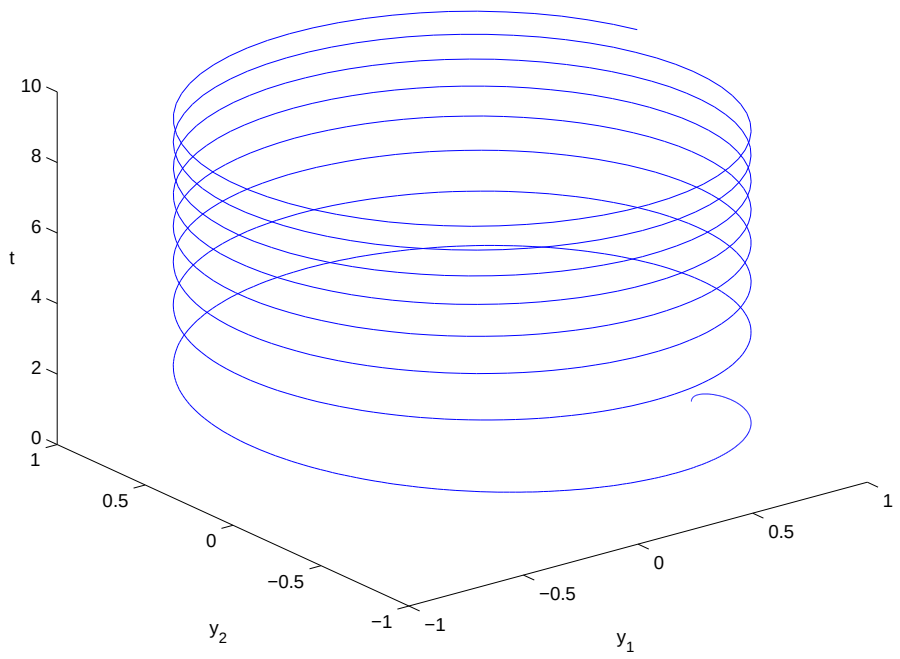


Figure 8.1: Numerical solution of the autonomized linear IVP using the Strang splitting method.

```
% N: number of timesteps
% t: meshpoints of timegrid
% y: solution at meshpoints of timegrid

t = [tspan(1)];
y = [y0];
h = (tspan(2)-tspan(1))/N;

for k=1:N
  y0 = psi1(h,y0);
  y0 = psi2(h,y0);
  y = [y;y0];
  t = [t;k*h];
end
```

**(8.2b)** Write a MATLAB function

```
function [t,y] = strang(psi1,psi2,psi3,y0,tspan,N)
```

which implements the Strang splitting method on the time interval `tspan`.

**Solution:** See Listing 8.4

Listing 8.4: Implementation of the Strang splitting method.

```
function [t,y] = strang(psi1,psi2,psi3,y0,tspan,N)
% Single step method based on Strang splitting
% psi1, psi2 and psi3 must be function of type @(h,y)
% y0: initial value
% tspan: [0,T]
% N: number of timesteps
% t: meshpoints of timegrid
% y: solution at meshpoints of timegrid

t = [tspan(1)];
y = [y0];
h = (tspan(2)-tspan(1))/N;

for k=1:N
  y0 = psi1(h/2,y0);
  y0 = psi2(h/2,y0);
  y0 = psi3(h,y0);
  y0 = psi2(h/2,y0);
  y0 = psi1(h/2,y0);
  y = [y;y0];
  t = [t;k*h];
end
```

## Problem 8.3 Extrapolation of Reversible One-step Methods

We will limit ourselves to the autonomous initial value problem

$$\dot{\mathbf{y}} = -\mathbf{y} \quad \mathbf{y}(0) = 1.$$

**(8.3a)** Implement a MATLAB function

```
y = ImplMpr(y0,h,n),
```

that performs `n` steps of the implicit midpoint rule with step size `h` from the initial value y0.

**Solution:** See Listing 8.5.

Listing 8.5: Implementation of subproblem (8.3a)

```
function y = ImplMpr(y0,h,n)

y=y0;

% n steps of the implicit mid-point rule
for i=1:n
    y = (1 + 0.5*h) \ (y - 0.5*h*y);
end
```

**(8.3b)** In a MATLAB function

```
y = extraImplMpr(y0,T,N,n),
```

implement the extrapolated implicit mid point rule. The input should be given as:`y0`, the initial value, `T`, the end point, `N`, the number of macro steps and the vector `n`, which represents the sequence of the number of micro steps.

HINT: Use the function `extrapolate(Y,n,p)` from `extrapolate.m`.

**Solution:** See Listing 8.6.

Listing 8.6: Implementation of subproblem (8.3b)

```
function y=extraImplMpr(y0,T,N,n)

% macro step
H = T/N;

% length of extrapolation sequence
k = length(n);

% sequence of micro steps
h=H./n;

% consistency order of implicit mid-point rule
p=2;

% initial values
y=y0;

Y=zeros(k,1);

% for each macro step
for ii=1:N

    %find an approximation for each microstep size
    for jj=1:k

        Y(jj)=ImplMpr( y, h(jj) , n(jj));
    end

    %extrapolate a better approximation
    y=extrapolate(Y,n,p);

end
```

**(8.3c)** Complete the template `extraImplMprKonv.m`, which performs a convergence study of the extrapolated implicit mid point rule. Let the method run for different sequences of numbers

of micro steps (for example:$n = 1, 2$; $n = 1, 2, 3$; $n = 2, 4, 8$). What do you observe? Read [NUMODE, Thm. 2.4.22] and comment on your results.

HINT: Recall that the round-off error can be relevant for methods of high order (f.e. for $n = 1, 2, 3, 4, 5, 6, 7$).

**Solution:** With [NUMODE, Thm. 2.4.22] we understand, that the asymptotic evolution of reversible one step methods only preserves even powers. As the implicit mid point rule has order 2, we get

$$y_N = y(T) + \alpha_2 h^2 + \alpha_4 h^4 + \cdots .$$

One extrapolation step with the polynomial

$$p(t) = a_0 + a_2 t^2$$

raises the consistency order by 1. But $\alpha_3 = 0$ in the asymptotic development of reversible methods. The consistency order is thus raised by 2. Naturally, we get the same result for extrapolation with the polynomial

$$p(t) = a_0 + a_2 t^2 + a_3 t^3.$$

This explains why the consistency order is only raised on every second extrapolation steps. See Listing 8.7.

Listing 8.7: Implementation of subproblem (8.3c)

```
%Convergence study of extrapolated mid-point rule
function extraImplMprKonv

% AWP: y' = y, y(0) = 1;
% convergence study for local extrapolated rule
%f=@(y) -y;
y0=1;

% end point
T = 1;

%number of macro steps
N=2.^(1:4);

%sequence of number of micro-steps
n=1:3;

%compute the absolute error for each number of macro steps
err=zeros(size(N));
for i=1:length(N)
    err(i)=abs(exp(-T)-extraImplMpr(y0,T,N(i),n));
end

%loglog-plot
figure
loglog(N,err);
```

```
27
28 %find convergence rate
29 p=polyfit(log(N),log(err),1);
30 fprintf('convergence rate = %3.4f\n',p(1));
```

Published on 18 April 2016.

To be submitted by 26 April 2016.

# References


[NODE] Lecture Notes for the course “Numerical Methods for Ordinary Differential Equations”.

[NUMODE] Lecture Slides for the course “Numerical Methods for Ordinary Differential Equations”, SVN revision # 52913.


Last modified on April 27, 2016