Out: January 30, 2018
Recommended completion: February 06, 2018

NTNU – Trondheim
Norwegian University of
Science and Technology

Exercise 4
TTK4130 Modeling and Simulation

Problem 1 (Taylor expansions and order conditions)

The Butcher array for an explicit Runge-Kutta method with two stages is

$$\begin{array}{c|cc} 0 & & \\ c_2 & a_{21} & \\ \hline & b_1 & b_2 \end{array}$$

Assume (for simplicity) that the problem is scalar, that is, the dimension of $\mathbf{y}$ is 1. The stage computations are

$$\begin{aligned} k_1 &= f(y_n, t_n), \\ k_2 &= f(y_n + ha_{21}k_1, t_n + hc_2). \end{aligned}$$

Recall that the Taylor expansion of a function of two variables can be written

$$f(y+\Delta, t+\delta) = f(y,t) + \Delta\frac{\partial f(y,t)}{\partial y} + \delta\frac{\partial f(y,t)}{\partial t} + O(\Delta^2) + O(\delta\Delta) + O(\delta^2).$$

(a) Derive a first order taylor expansion of k_2, assuming that $a_{21} = c_2$, and using that

$$\frac{\mathrm{d}f(y_n,t_n)}{\mathrm{d}t} = \frac{\partial f(y_n,t_n)}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t} + \frac{\partial f(y_n,t_n)}{\partial t} = \frac{\partial f(y_n,t_n)}{\partial y}f(y_n,t_n) + \frac{\partial f(y_n,t_n)}{\partial t}.$$

Solution: By letting

$$\begin{aligned} \Delta &= ha_{21}k_1 = ha_{21}f(y_n,t_n), \\ \delta &= hc_2, \end{aligned}$$

we can write

$$\begin{aligned} k_2 &= f(y_n + ha_{21}k_1, t_n + hc_2) \\ &= f(y_n,t_n) + ha_{21}f(y_n,t_n)\frac{\partial f(y_n,t_n)}{\partial y} + hc_2\frac{\partial f(y_n,t_n)}{\partial t} + O(h^2) \\ &= f(y_n,t_n) + ha_{21}\left(\frac{\partial f(y_n,t_n)}{\partial y}f(y_n,t_n) + \frac{\partial f(y_n,t_n)}{\partial t}\right) + O(h^2) \\ &= f(y_n,t_n) + ha_{21}\frac{\mathrm{d}f(y_n,t_n)}{\mathrm{d}t} + O(h^2). \end{aligned}$$

(b) Derive the conditions on b_1, b_2 and $c_2 = a_{21}$ for the method to be of order 2.

Solution: Inserting the above result into $y_{n+1} = y_n + h(b_1k_1 + b_2k_2)$, we get

$$\begin{aligned} y_{n+1} &= y_n + hb_1k_1 + hb_2k_2 \\ &= y_n + hb_1f(y_n,t_n) + hb_2\left(f(y_n,t_n) + ha_{21}\frac{\mathrm{d}f(y_n,t_n)}{\mathrm{d}t} + O(h^2)\right) \\ &= y_n + h(b_1+b_2)f(y_n,t_n) + h^2b_2a_{21}\frac{\mathrm{d}f(y_n,t_n)}{\mathrm{d}t} + O(h^3). \end{aligned}$$

Comparing this to the Taylor expansion of the exact solution starting at y_n (the local solution),

$$\begin{aligned} y(t_n + h) &= y(t_n) + h\frac{\mathrm{d}y}{\mathrm{d}t} + \frac{h^2}{2!}\frac{\mathrm{d}^2 y}{\mathrm{d}t^2} + O(h^3) \\ &= y(t_n) + hf(y_n, t_n) + \frac{h^2}{2}\frac{\mathrm{d}f(y_n, t_n)}{\mathrm{d}t} + O(h^3), \end{aligned}$$

we see that the local error is $O(h^3)$ (the order of the method is 2) if

$$\begin{aligned} b_1 + b_2 &= 1, \\ b_2 a_{21} &= \frac{1}{2}. \end{aligned}$$

Remark 1: From the above, we can conclude that if the order conditions are satisfied, the order of the method is at least two. To see that it cannot be higher, one must calculate the second order Taylor expansion of k_2, and see it is impossible to match the third-order terms. This is a nice exercise for those who like to differentiate!

Remark 2: This procedure (matching the expansions of the stage computations with the expansion of the exact solution) is (as we have seen) fairly easy using first order expansions, doable (but involved) for second order expansions but practically impossible (by hand) for higher order expansions. Special computer programs for this have been developed to find order conditions for higher order Runge-Kutta methods.

Problem 2 (Implementing solvers for the pneumatic spring)

This problem continues with the pneumatic spring, described by

$$\ddot{x} + g\left[1 - \left(\frac{1}{x}\right)^{\kappa}\right] = 0,$$

where $\kappa = 1.40$ and $g = 9.81$. Defining $y_1 = x$ and $y_2 = \dot{x}$, this is on state-space form

$$\begin{aligned} \dot{y}_1 &= y_2, \\ \dot{y}_2 &= -g\left[1 - \left(\frac{1}{y_1}\right)^{\kappa}\right]. \end{aligned}$$

Recall (from the book or previous exercises) that since there is no damping, the physical solution is that the position oscillates around the equilibrium position $y_1 = 1$.

(a) Implement the explicit Euler method (or an explicit two-stage Runge-Kutta method) in Matlab and simulate from $t = t_0 = 0$ to $t = 10$ s, with step length $h = 0.01$ s. Use initial condition $y_0 = (2, 0)^\top$. Plot the position, and comment.

Solution: The position is shown in Figure **??**. Since there is no supply or dissipation of energy in the system, we would expect standing oscillations. However, this explicit Euler solution is unstable (the energy is, unphysically, increasing). This is as expected for this method: In a previous problem we saw that the eigenvalues of the linearization of this model is on the imaginary axis, and Euler's method cannot be stable for purely imaginary eigenvalues.

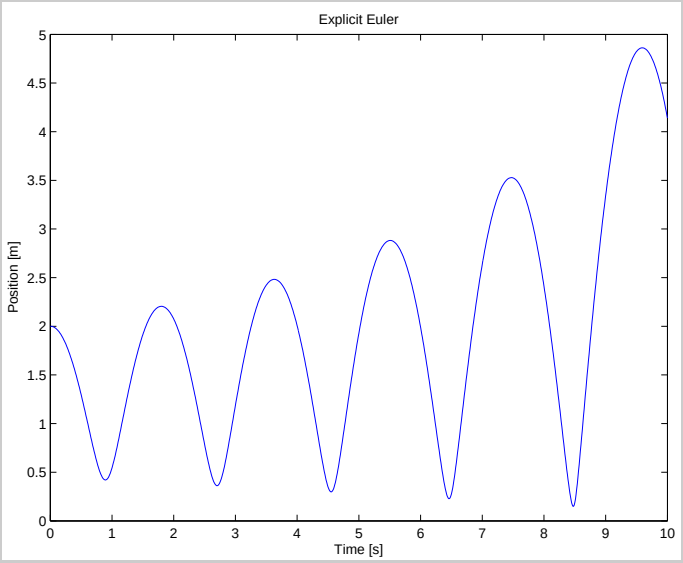

Figure 1: Simulation using explicit Euler

The code in the below listing contains code for solving all part of this problem.

```
% Script to simulate pneumatic spring using explicit Euler, implicit Euler
% and implicit midpoint rule (Gauss order 2).

%% Parameters
g = 9.81; K = 1.4;
y10 = 2; y20 = 0; y0 = [y10;y20]; % Initial values
h = 0.01; % Step size
t0 = 0; tstop = 10; % Time start and stop
time = t0:h:tstop; % Generate time vector
nstep = ((tstop-t0)/h)+1;
opt = optimset('Display','off','TolFun',1e-8); % Options for fsolve

%% Create storage
y_EE = zeros(size(y0,1),size(time,2)); % Explicit Euler
y_IE = zeros(size(y0,1),size(time,2)); % Implicit Euler
y_IM = zeros(size(y0,1),size(time,2)); % Implicit Midpoint rule

%% Function y' = f(y,t)
f = @(y,t) [ y(2); -g*(1-(1/y(1))^K) ];

%% EXPLICIT EULER
y_EE(:,1) = y0; % Initial value
for i = 1:nstep-1,
    y_EE(:,i+1) = y_EE(:,i) + h*feval(f,y_EE(:,i),time(i));
end
% Plots the results for Explicit Euler
figure(1); plot(time,y_EE(1,:));
xlabel('Time [s]'); ylabel('Position [m]'); title('Explicit Euler')

%% IMPLICIT EULER
y_IE(:,1) = y0; % Initial value is set.
```

```
for i = 1:nstep-1,
    r = @(y) (y_IE(:,i) + h*feval(f,y,time(i+1)) - y); % Root function
    [y_IE(:,i+1),fval,exitflag,output] = fsolve(r,y_IE(:,i),opt);
end
% Plots the results for Implicit Euler
figure(2); plot(time,y_IE(1,:));
xlabel('Time [s]'); ylabel('Position [m]'); title('Implicit Euler')

%% IMPLICIT MIDPOINT RULE
y_IM(:,1) = y0; % Initial value is set.
for i = 1:nstep-1,
    r = @(y) (y_IM(:,i) + h*feval(f,(y_IM(:,i) + y)/2,time(i)+h/2) - y);
    [y_IM(:,i+1),fval,exitflag,output] = fsolve(r, y_IM(:,i), opt);
end
% Plots the results for Implicit Midpoint Rule
figure(3); plot(time,y_IM(1,:));
xlabel('Time [s]'); ylabel('Position [m]'); title('Implicit Midpoint Rule')

%% Plot energies
p0 = 200000; A = 0.01; g =10; m = 200;
E_EE = 1/(K-1)*p0*A*y_EE(1,:).^(-(K-1)) + m*g*y_EE(1,:) + 1/2*m*y_EE(2,:).^2;
E_IE = 1/(K-1)*p0*A*y_IE(1,:).^(-(K-1)) + m*g*y_IE(1,:) + 1/2*m*y_IE(2,:).^2;
E_IM = 1/(K-1)*p0*A*y_IM(1,:).^(-(K-1)) + m*g*y_IM(1,:) + 1/2*m*y_IM(2,:).^2;
figure(4); plot(time,E_EE); hold on;
plot(time,E_IE,'m--'); plot(time,E_IM,'c:'); hold off;
legend('Explicit Euler','Implicit Euler',...
    'Implicit Midpoint Rule (Gauss 2)','Location','NorthWest');
xlabel('Time [s]'); ylabel('Energy [J]');
```

(b) Implement the implicit Euler method for the same problem. Use fsolve from the optimization toolbox (or implement a Newton-type algorithm yourself) to solve the nonlinear equation. For example: Define the model as

```
f = @(y,t) [ y(2); -g*(1-(1/y(1))^K) ];
```

Then, in each iteration of the for-loop, define the function to be solved ($r(y_{n+1}) = y_n + hf(y_{n+1}, t_{n+1}) - y_{n+1} = 0$), and call fsolve.

```
r = @(ynext) (y(:,i) + h*feval(f, ynext, time(i+1)) - ynext);
y(:,i+1) = fsolve(r, y(:,i), opt);
```

To get this to work, it is important to set small tolerances for the solution:

```
opt = optimset('Display','off','TolFun',1e-8); % Options for fsolve
```

Remark: Using fsolve for this is not particularly efficient. If you want, you can provide the Jacobian of r to fsolve to speed up the solutions (the Jacobian of f was calculated in an earlier exercise). You may also experiment by using the procedure outlined in Chapter 14.8.1 in the book (not required).

Solution: See listing above. A simulation is shown in Figure **??**. We see that the solution now is stable, but that energy is removed from the simulation. This is in accordance with the method being L-stable.

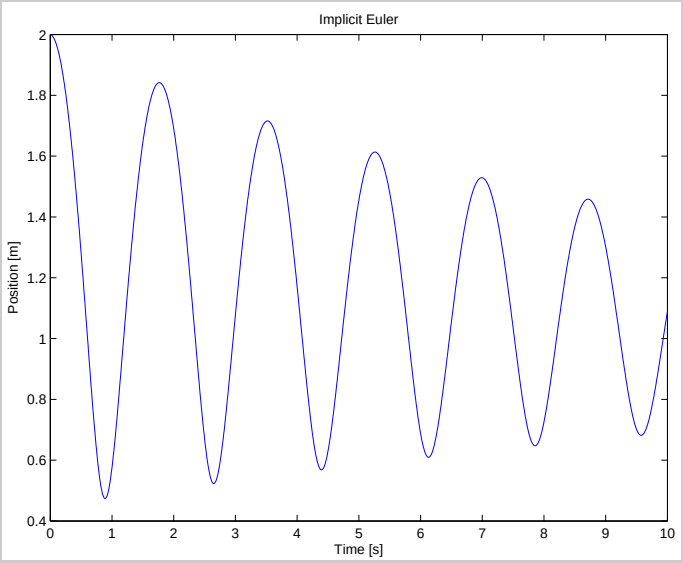

Figure 2: Simulation using implicit Euler

(c) Implement the implicit midpoint rule (Gauss order 2) for this problem, $y_{n+1} = y_n + hf((y_n + y_{n+1})/2, t_n + h/2)$.

Solution: See listing above, and simulation in Figure **??**. We see that this method does not remove energy from the simulation, in accordance with the method being A-stable but not L-stable.

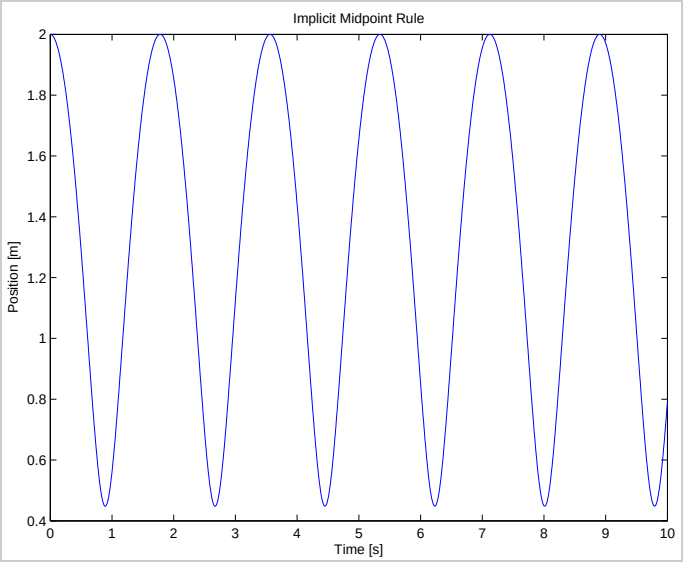

Figure 3: Simulation using implicit midpoint rule (Gauss order 2)

(d) The energy for the system is

$$E = \frac{1}{\kappa - 1} p_0 A x^{-(\kappa-1)} + mgx + \frac{1}{2} m v^2$$

Plot the energy for all three solutions above. Assume $A = 0.01$ m^2, $m = 200$ kg og $p_0 = 2 \cdot 10^5$

$\mathrm{N/m^2}$.

Solution: See Figure **??**. This plot agrees with the insights obtained above.

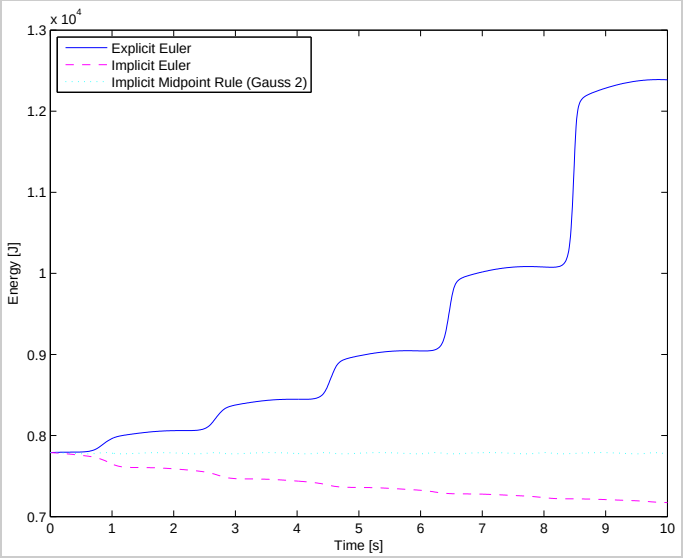

Figure 4: Energies

Problem 3 (Tank and valve-model)

In this problem, we will model a tank that is being emptied through valve. See Figure 5.

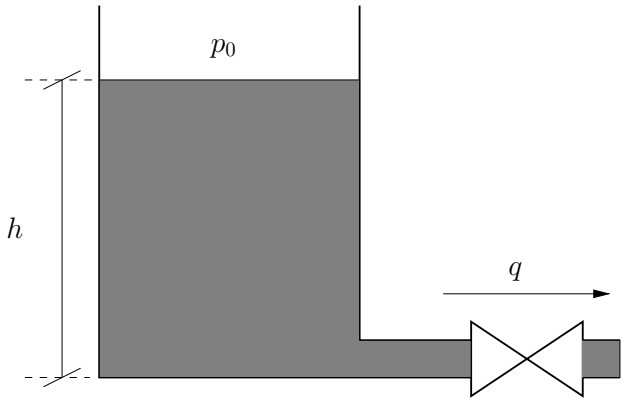

Figure 5: Tank being emptied through a valve

We model the valve with the valve equation

$$q = C_v \sqrt{p - p_0}.$$

We assume constant density $\rho = 1000\mathrm{kg/m^3}$, tank area $A = 4.5\mathrm{m^2}$, gravity of acceleration $g = 10\mathrm{m/s^2}$, valve constant $C_v = 0.15\mathrm{m^3}/(\mathrm{s} \cdot \sqrt{\mathrm{Pa}})$, and outside pressure is athmospheric $p_0 = 10^5\mathrm{Pa}$.

(a) Using a mass balance for the tank, show that the level h is modeled by the differential equation

$$\frac{\mathrm{d}h}{\mathrm{d}t} = -\frac{C_v}{A}\sqrt{\rho g h}.$$

Solution: A mass balance is given by

$$\frac{\mathrm{d}}{\mathrm{d}t}(\rho A h) = -\rho q.$$

Together with the valve equation and $p = p_0 + \rho g h$, this gives the result.

(b) Linearize the system. What happens to the eigenvalue as $h \to 0$?

Solution: The linearized model (around $h = h^*$) is

$$\Delta\dot{h} = -\frac{C_v \rho g}{2A\sqrt{\rho g h^*}}\Delta h.$$

The eigenvalue is

$$\lambda = -\frac{C_v \rho g}{2A\sqrt{\rho g h^*}},$$

which goes to minus infinity as $h^* \to 0$ (from above).

(c) Implement and simulate the model in Matlab, using the ODE solver `ode45`. Use initial condition $h(t = 0) = 2$, and simulate for one second. To show the steplengths, use the plot-command **plot**`(t,h,'o-')`. Comment on the steplengths in view of the answer in (b).

Solution: A Matlab script simulating the model, is:

```
Cv = .15; A = 4.5; rho = 1000;
g = 10; Po = 1;

% ODE
f = @ (t,h) ( -(Cv/A) * sqrt(rho*g*h) );

% Simulate
[t,h] = ode45(f,[0 1],2,options);
```

A plot is shown in Figure **??**.

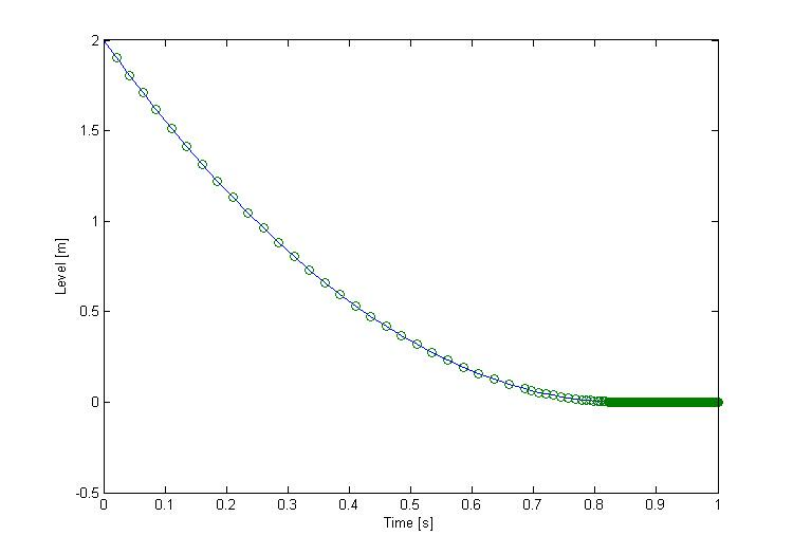

Figure 6: Simulating the level. The 'O' marks each step by the ODE solver.

As we can see, as $h \to 0$, the step sizes becomes smaller. This is in correspondence with the eigenvalues approaching infinity.

Remark: This valve model is not good for small flows, for instance, the solution will typically become complex as the model (square root) is evaluated for negative h. An explanation for this problem is that the valve equation is based on an assumption of turbulent flows, and for smaller flows the flow will at some point become laminar. The solution to this problem is to regularize the valve model, for instance as explained in Section 4.2.3 in the book.