Out: March 9, 2020
Due: April 5, 2020

NTNU – Trondheim
Norwegian University of Science and Technology

TTK4130 Modeling and Simulation
Assignment 9

Introduction

This assignment represents the culmination of the work started in assignment 7, and has as goals:

- To implement and test an arbitrary IRK scheme.
- To study the phenomenon of numerical energy loss.
- To solve numerically implicit DAEs.

Problem 1 (IRK schemes)

(a) Code a Matlab function that implements an arbitrary IRK method based on its Butcher table.
Explain what implicit equations have to be solved at each iteration.
Add the implemented code to your answer.
Hints:

- *A template for such a function can be found on Blackboard.*
- *Use Newton's method to solve the implicit equations. You may want to modify your code from assignment 8 so that it only returns the final root estimate.*
- *It is arguably best to declare the vectors* $\boldsymbol{k}_1, \cdots, \boldsymbol{k}_s$ *as a matrix* $K = [\boldsymbol{k}_1 \cdots \boldsymbol{k}_s]$. *However, when solving the implicit equations, it is probably more advantageous to concatenate this vectors as the column vector* $\boldsymbol{k} = [\boldsymbol{k}_1^\top \cdots \boldsymbol{k}_s^\top]^\top$. *In any case, one can go back and forth these representations using the Matlab command* `reshape`.

Solution: At each iteration k, the vectors $\boldsymbol{k}_1, \cdots, \boldsymbol{k}_s$ have to be found. These vectors are given by the set of equations

$$\boldsymbol{k}_i - \boldsymbol{f}(t_k + \Delta t_k c_i, \boldsymbol{x}_k + \Delta t_k \sum_{j=1}^{s} a_{ij}\boldsymbol{k}_j) = \boldsymbol{0}, \quad i = 1, \cdots, s.$$

Therefore, we define the residual function $\boldsymbol{g}$ as

$$\boldsymbol{g}(\boldsymbol{k}) = \begin{bmatrix} \boldsymbol{k}_1 - \boldsymbol{f}(t_k + \Delta t_k c_1, \boldsymbol{x}_k + \Delta t_k \sum_{j=1}^{s} a_{ij}\boldsymbol{k}_j) \\ \boldsymbol{k}_2 - \boldsymbol{f}(t_k + \Delta t_k c_2, \boldsymbol{x}_k + \Delta t_k \sum_{j=1}^{s} a_{2j}\boldsymbol{k}_j) \\ \vdots \\ \boldsymbol{k}_s - \boldsymbol{f}(t_k + \Delta t_k c_s, \boldsymbol{x}_k + \Delta t_k \sum_{j=1}^{s} a_{sj}\boldsymbol{k}_j) \end{bmatrix},$$

where $\boldsymbol{k} = [\boldsymbol{k}_1^\top \cdots \boldsymbol{k}_s^\top]^\top$.
The Jacobian of $\boldsymbol{g}$ is

$$\begin{aligned}\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{k}}(\boldsymbol{k}) &= \begin{bmatrix} I - \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}(t_k + \Delta t_k c_1, \boldsymbol{x}_k + \Delta t_k \sum_{j=1}^{s} a_{1j}\boldsymbol{k}_j) a_{11}\Delta t_k & -\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}(t_k + \Delta t_k c_1, \boldsymbol{x}_k + \Delta t_k \sum_{j=1}^{s} a_{1j}\boldsymbol{k}_j) a_{12}\Delta t_k & \cdots \\ -\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}(t_k + \Delta t_k c_1, \boldsymbol{x}_k + \Delta t_k \sum_{j=1}^{s} a_{2j}\boldsymbol{k}_j) a_{21}\Delta t_k & I - \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}(t_k + \Delta t_k c_1, \boldsymbol{x}_k + \Delta t_k \sum_{j=1}^{s} a_{2j}\boldsymbol{k}_j) a_{22}\Delta t_k & \cdots \\ \vdots & \ddots & \vdots \\ -\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}(t_k + \Delta t_k c_1, \boldsymbol{x}_k + \Delta t_k \sum_{j=1}^{s} a_{sj}\boldsymbol{k}_j) a_{s1}\Delta t_k & -\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}(t_k + \Delta t_k c_1, \boldsymbol{x}_k + \Delta t_k \sum_{j=1}^{s} a_{sj}\boldsymbol{k}_j) a_{s2}\Delta t_k & \cdots \end{bmatrix} \\ &= \left[I\delta_{ij} - \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}(t_k + \Delta t_k c_i, \boldsymbol{x}_k + \Delta t_k \sum_{j=1}^{s} a_{ij}\boldsymbol{k}_j) a_{ij}\Delta t_k \right],\end{aligned}$$

where I is the $N_{\boldsymbol{x}}$-by-$N_{\boldsymbol{x}}$ identity matrix and δ_{ij} is the Kronecker delta.
Hence, an example implementation is

```matlab
function x = IRK(ButcherArray, f, dfdx, T, x0)
    Nt = length(T);
    Nx = length(x0);
    dT = diff(T);
    x = zeros(Nx,Nt);
    x(:,1) = x0;
    A = ButcherArray.A;
    b = ButcherArray.b(:);
    c = ButcherArray.c(:);
    Nstage = size(A,1);
    xt = x0;
    k = repmat(f(T(1),x0),Nstage,1); % initial guess
    for nt=2:Nt
        t = T(nt-1);
        dt = dT(nt-1);
        g = @(k) IRKODEResidual(k,xt,t,dt,A,c,f);
        G = @(k) IRKODEJacobianResidual(k,xt,t,dt,A,c,dfdx);
        k = NewtonsMethod(g,G,k);
        K = reshape(k,Nx,Nstage);
        xt = xt + dt*(K*b);
        x(:,nt) = xt;
    end
end
function g = IRKODEResidual(k,xt,t,dt,A,c,f)
    Nx = length(xt);
    Nstage = size(A,1);
    K = reshape(k,Nx,Nstage);
    Tg = t+dt*c';
    Xg = xt+dt*K*A';
    g = reshape(K-f(Tg,Xg),[],1);
end
function G = IRKODEJacobianResidual(k,xt,t,dt,A,c,dfdx)
    Nx = length(xt);
    Nstage = size(A,1);
    K = reshape(k,Nx,Nstage);
    TG = t+dt*c';
    XG = xt+dt*K*A';
    dfdxG = cell2mat(arrayfun(@(i) dfdx(TG(:,i),XG(:,i))',1:Nstage,...
        'UniformOutput',false))';
    G = eye(Nx*Nstage)-repmat(dfdxG,1,Nstage).*kron(dt*A,ones(Nx));
end
```

Note that this function generalizes the `ERK.m` function implemented in assignment 7.

(b) The Gauss-Legendre collocation method is an IRK scheme with $s = 2$ stages and of order $2s = 4$, which has the Butcher table :

Table 1: Gauss-Legendre Collocation (IRK4)

$\frac{1}{2} - \frac{\sqrt{3}}{6}$	$\frac{1}{4}$	$\frac{1}{4} - \frac{\sqrt{3}}{6}$
$\frac{1}{2} + \frac{\sqrt{3}}{6}$	$\frac{1}{4} + \frac{\sqrt{3}}{6}$	$\frac{1}{4}$
	$\frac{1}{2}$	$\frac{1}{2}$

Test the code developed in part a. by implementing the IRK4.
Simulate the test system

$$\dot{x} = \lambda x \tag{1}$$

with $\lambda = -2$ and initial condition $x(0) = 1$, using the IRK4 and the RK4 method (assignment 7).
Simulate for a final time $t_f = 2$ and a time-step $\Delta t_k = 0.4$.
Add a plot of the results to your answer, and comment on them.

Solution:
Figure 1 shows the simulation results. The solutions of The RK4 and IRK4 solutions are almost identical. This is as expected since both methods are of order 4.

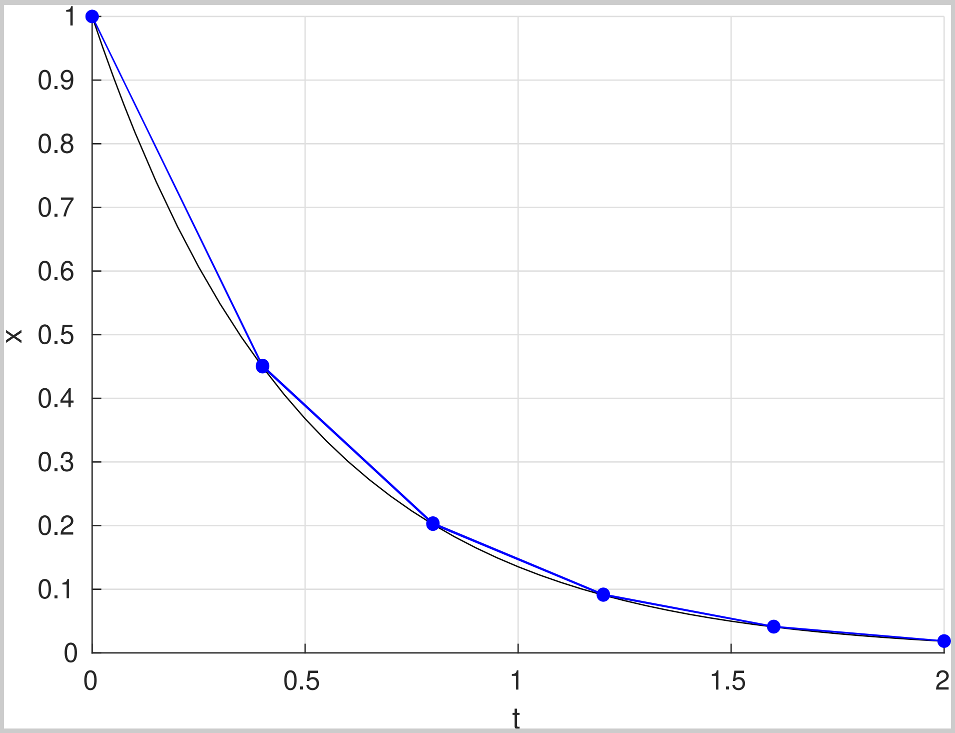

Figure 1: Actual trajectory (black), RK4 solution (red) and IRK4 solution (blue). The RK4 and IRK4 solutions are almost identical.

(c) For what value of $\lambda < 0$ will the IRK4 scheme become unstable?

Solution: Since IRK4 is a Gauss method of order 4, its stability function is $R(s) = P_2^2(s)$. In particular, this method is A-stable, i.e., it is stable for all $\lambda < 0$. Hence, the answer is ”none”.

Problem 2 (Numerical energy loss)

Consider the pneumatic spring without damping:

$$\ddot{x} + g\left(1 - \left(\frac{x_d}{x}\right)^\kappa\right) = 0\,, \tag{2}$$

where $x, x_d, g > 0$ and $\kappa \geq 1$.

Since there is no damping, the physical solution will oscillate around its equilibrium position $x = x_d$.

Moreover, the energy for the system (2) is given by

$$E = \frac{mg}{\kappa - 1}\frac{x_d^\kappa}{x^{\kappa-1}} + mgx + \frac{1}{2}m\dot{x}^2\,, \tag{3}$$

where m is the mass.

(a) Use (2) to show that the energy for the physical solution is constant over time, i.e., $\dot{E}(t) = 0$.

Solution:

$$\dot{E} = -mg\frac{x_d^\kappa}{x^\kappa}\dot{x} + mg\dot{x} + m\dot{x}\ddot{x} = m\dot{x}\left(-g\left(\frac{x_d}{x}\right)^\kappa + g + \ddot{x}\right) = 0.$$

(b) Simulate the pneumatic spring without damping (2) using the explicit Euler method, the implicit Euler method and the implicit midpoint rule (Gauss method of order 2).

Use the initial condition $[x(0),\, \dot{x}(0)] = [2\,\mathrm{m},\, 0\,\mathrm{m\,s^{-1}}]$, a step-size of $\Delta t_k = 0.01\,\mathrm{s}$ and a final time $t_f = 10\,\mathrm{s}$ for each simulation.

Moreover, use the model parameter values $x_d = 1.32\,\mathrm{m}$, $\kappa = 2.40$, $g = 9.81\,\mathrm{m\,s^{-2}}$ and $m = 200\,\mathrm{kg}$.

Add a plot with the simulated positions, x, as well as a plot with the corresponding energies, E, to your answer.

Comment on the results, and give a formal explanation.

Hint: Keywords here are stability region, A- and L-stability.

Solution:

Figure 2 shows the simulated position of the pneumatic spring for the different RK methods, while Figure 3 shows the corresponding energy.

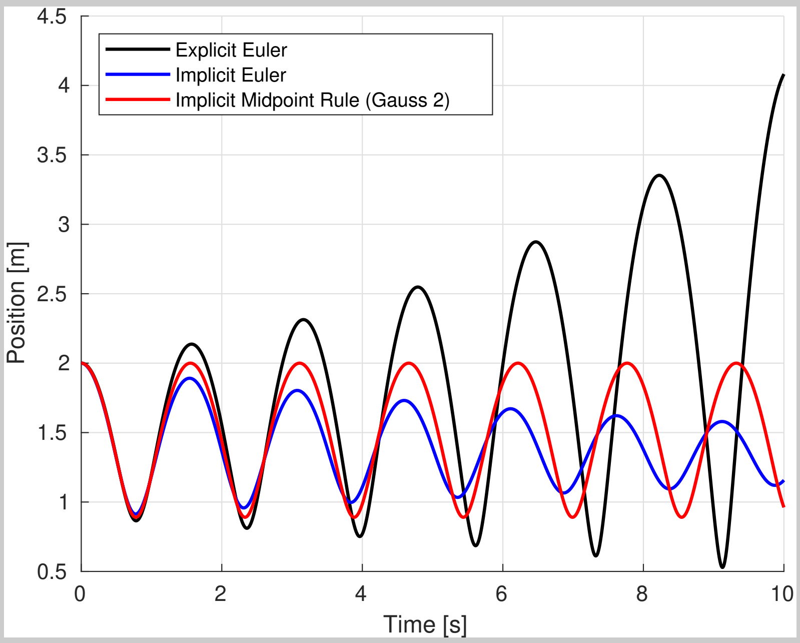

Figure 2: Simulated position.

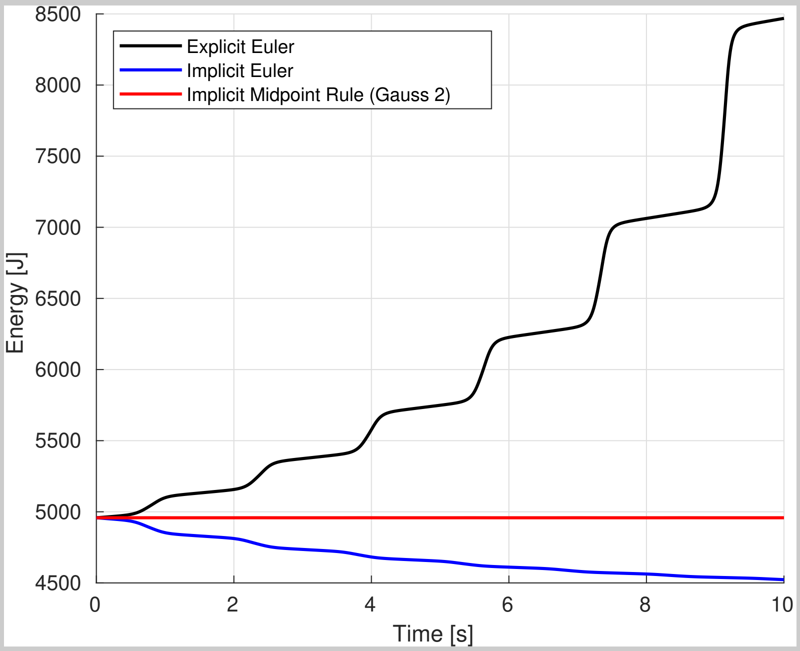

Figure 3: Simulated energy.

Since there is no supply or dissipation of energy in the system, the actual solution should consist of stationary oscillations. In particular, the energy should be constant over time as we have seen in part a.

However, the explicit Euler's solution is unstable, and the energy is increasing. This is as expected since the eigenvalues of the linearization of this model are purely imaginary. Hence, the explicit Euler's method solution is unstable no matter what step size one uses.

On the other side, the solution of the implicit Euler's method is stable. However, the energy is been removed from the simulation. This is in accordance with the integration method being L-stable.

Finally, we have the solution of the implicit midpoint rule, which does not remove nor supply energy from the simulation. This is in accordance with the integration method being A-stable, but not L-stable.

Problem 3 (DAE integration)

In this task, we will use the IRK4 integrator from problem 1 to solve 2 fully-implicit DAEs of the form

$$\boldsymbol{F}\left(\dot{\boldsymbol{x}}, \boldsymbol{x}, \boldsymbol{z}, t\right) = 0 \tag{4}$$

Note that for the sake of simplicity we have assumed that the dynamics have no input.

In order to simulate the DAEs, the function `RKDAE.m` has been delivered. This function integrates a DAE of the form (4) using the selected RK method.

We recommend you to use this solver. However, if you prefer, you are free to implement your own DAE integrator.

(a) Test your code on the 3D pendulum with the constraint

$$C\left(\boldsymbol{q}\right) = \frac{1}{2}\left(\boldsymbol{p}^\top \boldsymbol{p} - L^2\right) = 0\,, \tag{5}$$

where $L = 1$. In this case, the index-reduced model equations read as

$$\dot{\boldsymbol{p}} = \boldsymbol{v} \tag{6a}$$

$$m\dot{\boldsymbol{v}} = -mg\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} - z\boldsymbol{p} \tag{6b}$$

$$0 = \boldsymbol{p}^\top \dot{\boldsymbol{v}} + \boldsymbol{v}^\top \boldsymbol{v} \tag{6c}$$

Put the dynamics in a state-space form first, using e.g.

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{v} \end{bmatrix}, \tag{7}$$

and simulate the DAE using the IRK4. Try e.g. the consistent initial conditions:

$$\boldsymbol{x}(0) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \in \mathbb{R}^6\,, \tag{8}$$

and e.g. the final time $t_\mathrm{f} = 30$. Try different time-steps Δt.

Plot the simulated states and the corresponding contraint values (5). What do you observe? Explain the reason behind this. What would you do to improve the results?

Solution: The state space form will look like:

$$F(\dot{\boldsymbol{x}}, \boldsymbol{x}, z) = \begin{bmatrix} I & 0 & 0 \\ 0 & mI & \boldsymbol{p} \\ 0 & \boldsymbol{p}^\top & 0 \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{x}} \\ z \end{bmatrix} - \begin{bmatrix} \boldsymbol{v} \\ -mg \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{bmatrix} = 0.$$

The simulation present a drift of the constraints, i.e. $C(\boldsymbol{q}(t))$ is less and less satisfied for a "long" simulation time, especially when Δt is not very small. The problem arises from the fact that the index-reduced DAE imposes $\ddot{C} = 0$ instead of $C = 0$. Numerical errors then slowly accumulate such that the constraints become unsatisfied.

The issue can be solved using the so-called Baumgarte stabilization, where one imposes

$$\ddot{C} + \alpha \dot{C} + \beta C = 0$$

for α, $\beta > 0$, instead of $\ddot{C} = 0$.

In this case, the Baumgarte stabilization yields the equation

$$\boldsymbol{p}^\top \dot{\boldsymbol{v}} + \boldsymbol{v}^\top \boldsymbol{v} + \alpha \boldsymbol{p}^\top \boldsymbol{v} + \frac{\beta}{2}\left(\boldsymbol{p}^\top \boldsymbol{p} - L^2\right) = 0.$$

For step size of $\Delta t = 0.01$ and a final time $t_f = 10$, Figure 2 and Figure 5 show the simulated position and velocity, respectively, while Figure 6 shows the corresponding constraint values.

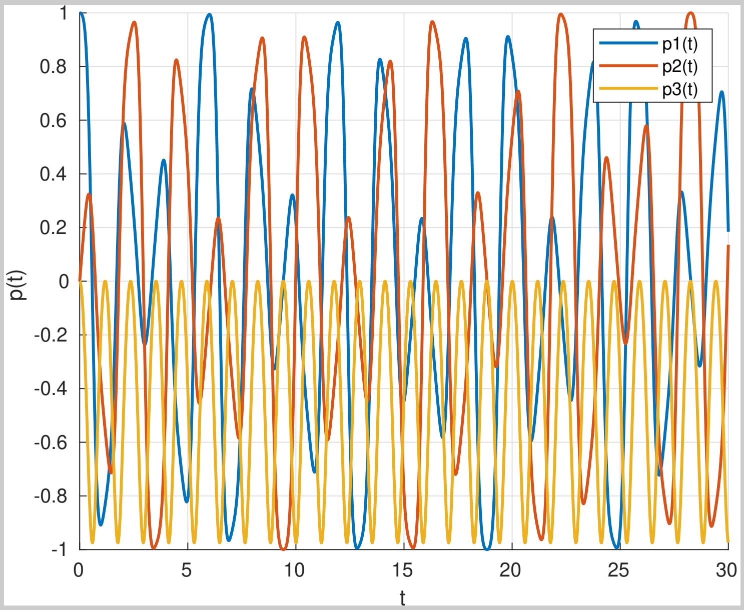

Figure 4: Simulated position.

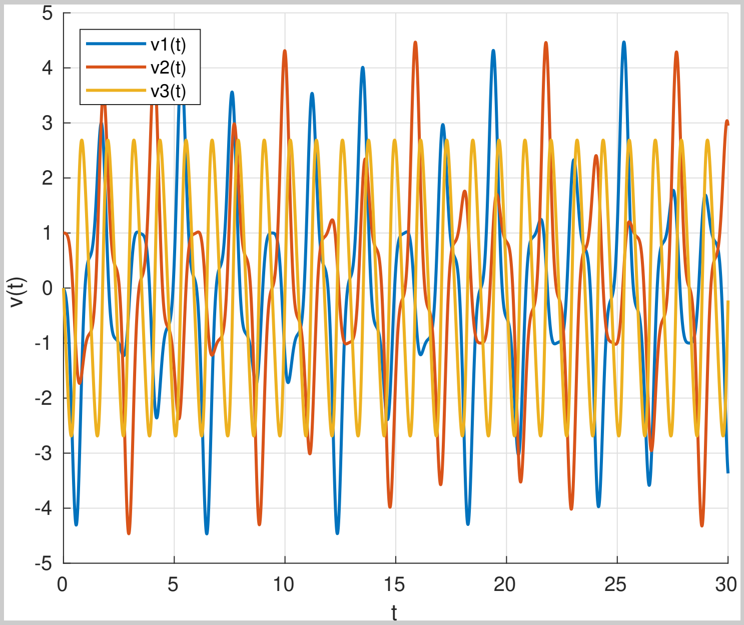

Figure 5: Simulated velocity.

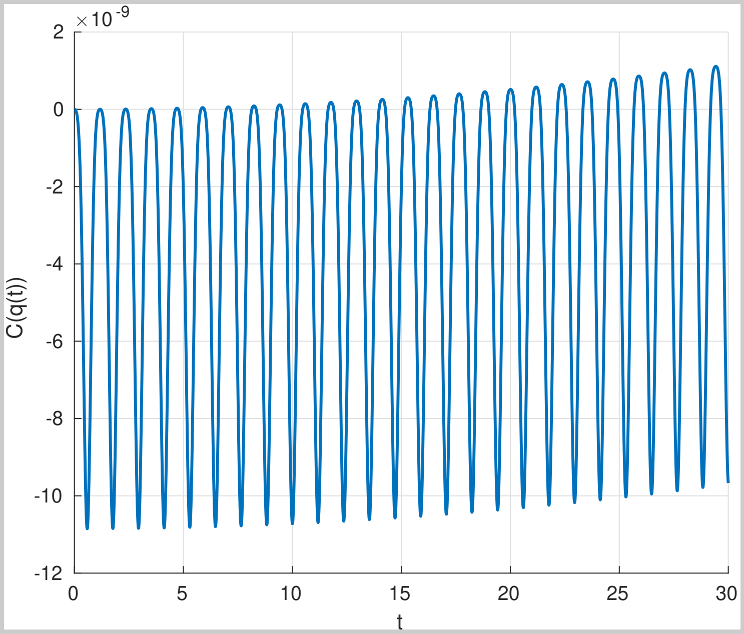

Figure 6: Constraint values.

Figure 7 shows the constraint values using the Baumgarte stabilization, where we have chosen $\alpha = 2\lambda$ and $\beta = \lambda^2$ with $\lambda = 5$. Note that now the constraint values oscillate between $-1 \cdot 10^{-}-8$ and $3 \cdot 10^{-}-8$, and no longer diverge. These results are more than acceptable given that the pendulum length is 1 unit.

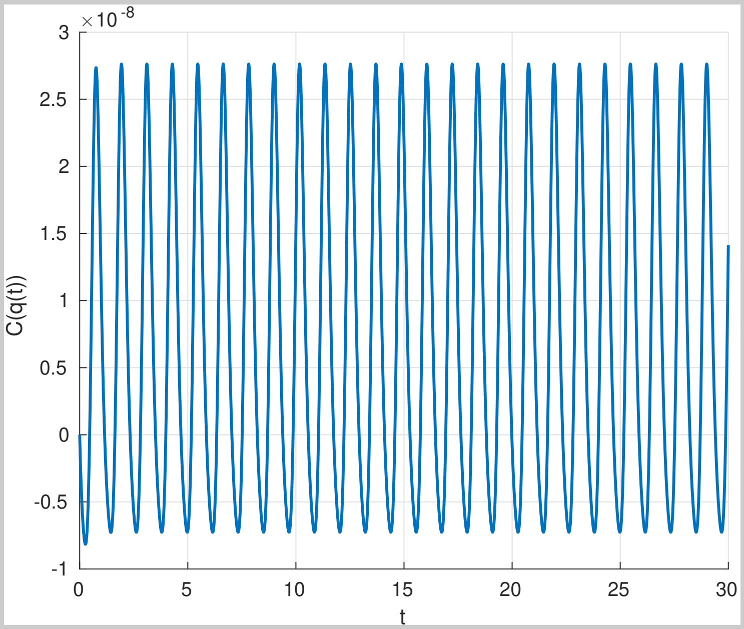

Figure 7: Constraint values using the Baumgarte stabilization.

(b) Test your code on the 3D pendulum model one obtains directly from the Lagrange approach, i.e.

$$\begin{aligned}\dot{\boldsymbol{p}} &= \boldsymbol{v}\\ m\dot{\boldsymbol{v}} &= -mg\begin{bmatrix}0\\0\\1\end{bmatrix} - z\boldsymbol{p}\\ 0 &= \frac{1}{2}\left(\boldsymbol{p}^\top\boldsymbol{p} - L^2\right).\end{aligned}$$

What happens when you try to simulate? Why is that?

Solution: Either the simulations give non-sensical results, or the code stops with an error or throws nasty warnings. The problem here is that we are trying to simulate an index-3 DAE, and our solver does not support that.
In other words, the simulation does not work since the Jacobian

$$\frac{\partial \boldsymbol{F}\left(\dot{\boldsymbol{x}}, \boldsymbol{x}, z, t\right)}{\partial\left(\dot{\boldsymbol{x}}, z\right)} = \begin{bmatrix}\frac{\partial \boldsymbol{F}(\dot{\boldsymbol{x}},\boldsymbol{x},z,t)}{\partial \dot{\boldsymbol{x}}} & \frac{\partial \boldsymbol{F}(\dot{\boldsymbol{x}},\boldsymbol{x},z,t)}{\partial z}\end{bmatrix} = \begin{bmatrix} I_3 & \boldsymbol{0} & \boldsymbol{0}\\ \boldsymbol{0} & mI_3 & \boldsymbol{p}\\ \boldsymbol{0} & \boldsymbol{0} & 0\end{bmatrix}$$

is rank-deficient.