Problem 1

Table of Contents

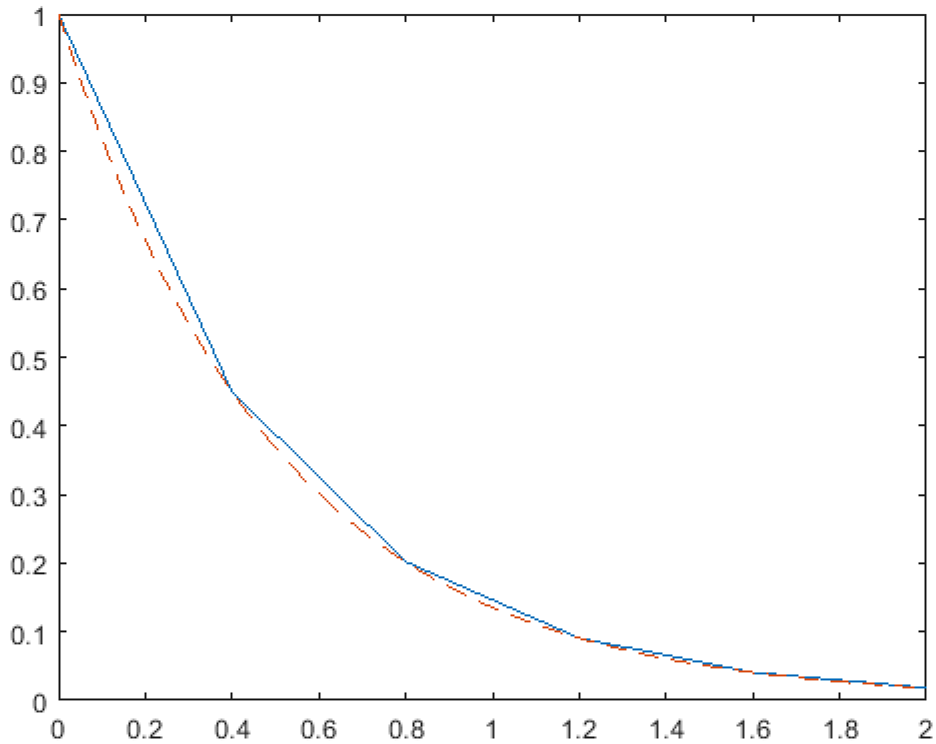
Implicit

```
A_GL = [1/4, 1/4-sqrt(3)/6;
    1/4+sqrt(3)/6, 1/4];
c_GL = [1/2-sqrt(3)/6,1/2+sqrt(3)/6]';
b_GL = [0.5, 0.5];
ButcherArray_GL = struct('A', A_GL, 'b', b_GL, 'c', c_GL);
lambda = -2;
x0 = 1;
f = @(t,x) lambda*x;
J = @(t,x) lambda;
T = [0:0.4:2];

x_iter = IRKTemplate(ButcherArray_GL, f, J, T, x0);
```

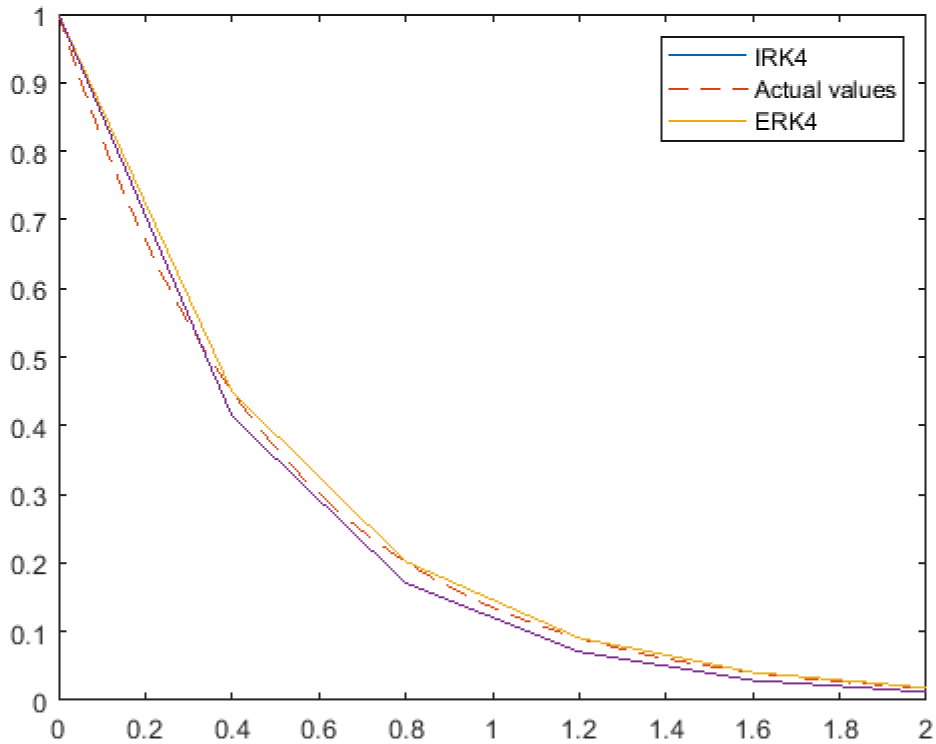
b) The plot shows that the implicit scheme is able to calculate accurate values, while the explicit scheme has some error.

```
plot(T,x_iter)
hold on
T2 = linspace(0,2,100);
x = exp(lambda*T2);
plot(T2,x, '--');
```




```
clear x
% Explicit
A = [0,0,0,0;
    0.5,0,0,0;
    0,0.5,0,0;
    0,0,1,0];
c = [0,0.5,0.5,1]';
b = [1/6,1/3,1/3,1/6]';
ButcherArray_RK4 = struct('A',A,'b',b,'c',c);

x = ERKTemplate(ButcherArray_RK4, f, T, x0);
plot(T,x_iter)
plot(T,x);
legend('IRK4', 'Actual values', 'ERK4');
snapnow
```



c) The IRK-scheme is A-stable, and will regardless of delta_t find accurate values. It will however not be able to track high-frequent transient responses with low delta_t's

Problem 2

a)

```
clf
im = imread('modsim.png');
imshow(im);
snapnow
```

$$\dot{E} = \frac{mg x_d^k}{k-1} \cdot (-(k-1)) \cdot \frac{1}{x^k}\,\dot{x} + mg\dot{x} + m\dot{x}\ddot{x}$$

$$\ddot{x} = g\left(\left(\frac{x_d}{x}\right)^k - 1\right)$$

$$\dot{E} = -\frac{mg x_d^k}{x^k} \cdot \dot{x} + \cancel{mg\dot{x}} + mg\dot{x}\left(\left(\frac{x_d}{x}\right)^k \cancel{-1}\right)$$

$$\Rightarrow \underline{\dot{E}(t) = 0}$$

b) Function:

```
x0 = [2,0]';
g = 9.81;
k = 2.40;
xd = 1.32;
m = 200;
f = @(t,x) [x(2);
        -g*(1-(xd/x(1))^k)];
J = @(t,x) [0,1;
    -(g*k*xd*(xd/x(1))^(k-1))/x(1)^2,0];
```

RK-Schemes:

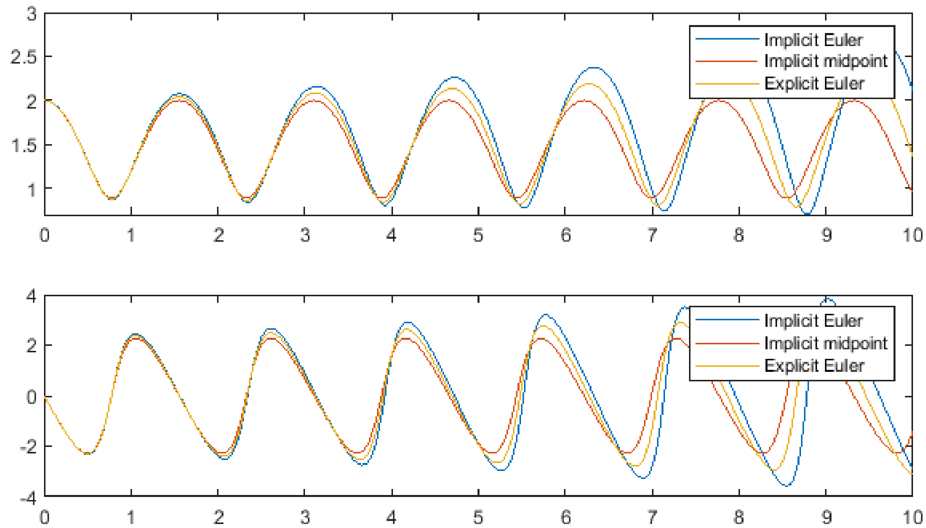
```
A_G2 = 0.5;
c_G2 = 0.5;
b_G2 = 1;
ButcherArray_G2 = struct('A', A_G2, 'b', b_G2, 'c', c_G2);

T = 0:0.01:10;
x_iter_GL = IRKTemplate(ButcherArray_GL, f, J, T, x0)';
x_iter_G2 = IRKTemplate(ButcherArray_G2, f, J, T, x0)';
x_iter_ERK4 = ERKTemplate(ButcherArray_RK4, f, T, x0);
```

Plot

```
subplot(211)
plot(T,x_iter_GL(:,1))
hold on
plot(T, x_iter_G2(:,1));
plot(T, x_iter_ERK4(:,1));
legend('Implicit Euler', 'Implicit midpoint', 'Explicit Euler');
subplot(212)
plot(T, x_iter_GL(:,2))
hold on
```

```
plot(T, x_iter_G2(:,2));
plot(T, x_iter_ERK4(:,2));
legend('Implicit Euler', 'Implicit midpoint', 'Explicit Euler');
snapnow
```

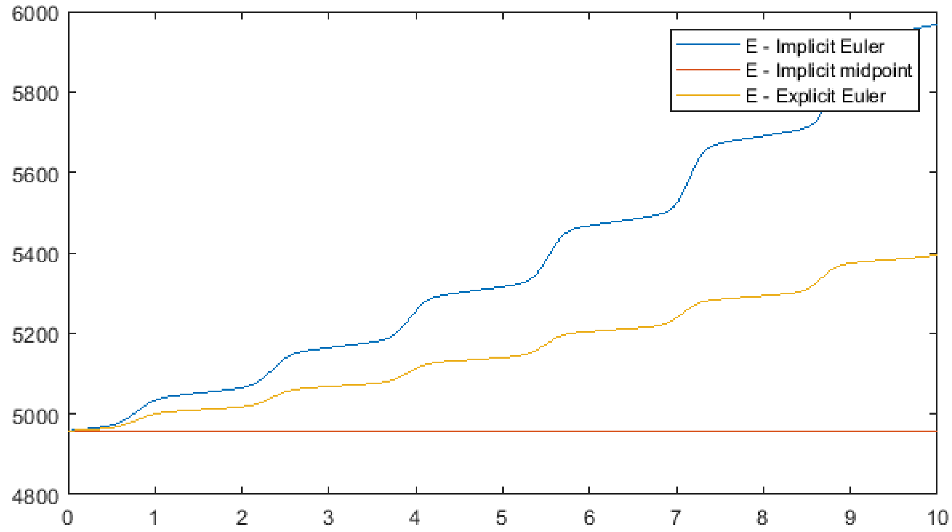



```
clf
% Energy:
Nx = size(x_iter_GL,1);
Eval = NaN(Nx,3);
E = @(x1,x2) m*g*xd^k/((k-1)*x1^(k-1)) + m*g*x1 + 0.5*m*x2^2;
Eval(:,1) = arrayfun(E, x_iter_GL(:,1), x_iter_GL(:,2));
Eval(:,2) = arrayfun(E, x_iter_G2(:,1), x_iter_G2(:,2));

Eval(:,3) = arrayfun(E, x_iter_ERK4(:,1), x_iter_ERK4(:,2));
```

Plotting the Energy-functions shows that two of the schemes are unable to conserve it, probably due to inaccuracy in the calculations.

```
plot(T,Eval)
legend('E - Implicit Euler', 'E - Implicit midpoint', 'E - Explicit
 Euler');
snapnow
```



Conservation of energy requires the RK-scheme to be stiffly accurate in this case. (And L-stable). This means that the RK-scheme has to be able to dampen Re(j*omega*h) as they tend to infinity. Re(j*omega*h) -> inf = 0 requires:

$b = A^T e_\sigma$ And that A is nonsingular. Only Implicit midpoint satisfies both rules

Problem 3

```
A_GL = [1/4, 1/4-sqrt(3)/6;
    1/4+sqrt(3)/6, 1/4];
c_GL = [1/2-sqrt(3)/6,1/2+sqrt(3)/6]';
b_GL = [0.5, 0.5];
ButcherArray_GL = struct('A', A_GL, 'b', b_GL, 'c', c_GL);
x0 = [0,0,0,0,0,1]';
m = 10;
g = 9.81;
L = 1;
x = sym('x', [6,1], 'Real');
xdot = sym('xdot', [6,1], 'Real');
syms t z;

f = [x(4:6)-xdot(1:3);
    -m*g*[0,0,1]'-z*x(1:3)-m*xdot(4:6);
    x(1:3)'*xdot(4:6) + x(4:6)'*x(4:6);
    1/2*(x(1:3)'*x(1:3)-L^2)]
J_xdot = jacobian(f,xdot);
```

```
J_x = jacobian(f,x);
J_z = jacobian(f,z);
    %    Implicit function for DAE: F(xdot,x,zx[,t)=0


Jxdot = matlabFunction(J_xdot, 'Vars', {[xdot],[x],z,t});
Jx = matlabFunction(J_x, 'Vars', {[xdot],[x],z,t});
Jz = matlabFunction(J_z, 'Vars', {[xdot],[x],z,t});
f = matlabFunction(f, 'Vars', {[xdot],[x],z,t});
delta_t = 0.2;
T = [0:delta_t:30];
z0 = 1;
[x, xdot, z] = RKDAE(ButcherArray_GL, f, Jxdot, Jx, Jz, T, x0, z0);
```

```
f =

                                          x4 - xdot1
                                          x5 - xdot2
                                          x6 - xdot3
                                   - 10*xdot4 - x1*z
                                   - 10*xdot5 - x2*z
                          - 10*xdot6 - x3*z - 981/10
 x4^2 + x5^2 + x6^2 + x1*xdot4 + x2*xdot5 + x3*xdot6
                     x1^2/2 + x2^2/2 + x3^2/2 - 1/2


Error: File: \\sambaad.stud.ntnu.no\jonashj\Documents\ModSim
\O9\RKDAE.m Line: 36 Column: 24
Invalid expression. When calling a function or indexing a variable,
 use parentheses. Otherwise, check for mismatched delimiters.

Error in Problem1 (line 162)
[x, xdot, z] = RKDAE(ButcherArray_GL, f, Jxdot, Jx, Jz, T, x0, z0);
```

IRKTemplate.m

```
function x = IRKTemplate(ButcherArray, f, dfdx, T, x0)
    % Returns the iterations of an IRK method using Newton's method
    % ButcherArray: Struct with the IRK's Butcher array
    % f: Function handle
    %    Vector field of ODE, i.e., x_dot = f(t,x)
    % dfdx: Function handle
    %       Jacobian of f w.r.t. x
    % T: Vector of time points, 1 x Nt
    % x0: Initial state, Nx x 1
    % x: IRK iterations, Nx x Nt
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Define variables
    % Allocate space for iterations (x) and k1,k2,...,ks
    %
    %
    A = ButcherArray.A;
```

```
    b = ButcherArray.b;
    c = ButcherArray.c;
    Nk = size(A,1);
    Nx = length(x0);
    Nt = length(T);
    delta_t = diff(T);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    x = zeros(Nx, length(T));
    x(:,1) = x0; % initial iteration
    k = zeros(Nx*Nk,1);
    % Loop over time points
    for nt=2:Nt
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        dt = delta_t(nt-1);
        k = reshape(k, [Nx*Nk,1]);
        g = @(k) IRKODEResidual(k, x(:,nt-1), nt, dt, A, c, f);
        G = @(k) IRKODEJacobianResidual(k,x(:,nt-1), nt, dt, A, c,
 dfdx);

        k = reshape(NewtonsMethod(g,G,k),[Nx,Nk]);
        % Update variables
        % Get the residual function for this time step
        % and its Jacobian by defining adequate functions
        % handles based on the functions below.
        % Solve for k1,k2,...,ks using Newton's method
        % Calculate and save next iteration value x_t
        %

        x(:,nt) = x(:,nt-1) + dt*(k*b');
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    end
end
function g = IRKODEResidual(k,xt,t,dt,A,c,f)
    % Returns the residual function for the IRK scheme iteration
    % k: Column vector with k1,...,ks, Nstage*Nx x 1
    % xt: Current iteration, Nx x 1
    % t: Current time
    % dt: Time step to next iteration
    % A: A matrix of Butcher table, Nstage x Nstage
    % c: c matrix of Butcher table, Nstage x 1
    % f: Function handle for ODE vector field
    Nx = size(xt,1);
    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)
    % Returns the Jacobian of the residual function
    % for the IRK scheme iteration
    % k: Column vector with k1,...,ks, Nstage*Nx x 1
    % xt: Current iteration, Nx x 1
    % t: Current time
```

```
    % dt: Time step to next iteration
    % A: A matrix of Butcher table, Nstage x Nstage
    % c: c matrix of Butcher table, Nstage x 1
    % dfdx: Function handle for Jacobian of ODE vector field
    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
```

ERKTemplate.m

```
function x = ERKTemplate(ButcherArray, f, time_vec, x0)
    % Returns the iterations of an ERK method
    % ButcherArray: Struct with the ERK's Butcher array
    % f: Function handle
    %    Vector field of ODE, i.e., x_dot = f(t,x)
    % T: Vector of time points, 1 x Nt
    % x0: Initial state, Nx x 1
    % x: ERK iterations, Nx x Nt
    a = ButcherArray.A;
    b = ButcherArray.b;
    c = ButcherArray.c;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Nx = length(x0);
    Nt = length(time_vec);
    Nstage = length(ButcherArray.b);
    K = NaN(Nx,Nstage);
    % Define variables
    % Allocate space for iterations (x) and k1,k2,...,kNstage
    % It is recommended to allocate a matrix K for all kj, i.e.
    % K = [k1 k2 ... kNstage]
    %
    %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    xt = x0; % initial iteration
    x(1,:) = x0';
    % Loop over time points
    for nt=2:Nt
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Update variables
        K(:,1) = f(nt, xt);
        T = time_vec(nt)-time_vec(nt-1);
        aK = zeros(Nx, 1);
        bK = zeros(Nx,1);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
        % Loop that calculates k1,k2,...,kNstage
        for nstage=2:Nstage
            for j = 1:(nstage-1)
                aK = aK + a(j)*K(:,j);
            end
            K(:,nstage) = f(nt + c(nstage-1),xt + T*aK);

        end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Calculate and save next iteration value x_t
        for m = 1:Nstage
            bK = bK + b(m)*K(:,m);
        end
        xt_1 = xt + T*bK;
        xt = xt_1;
        x(nt,:) = xt';
    end
end
```

Published with MATLAB® R2019a