TTK4130 - Modeling and Simulation
Assignement 9
Ingebrigt Stamnes Reinsborg

1a) I have added my code at the end of my submission. I believe the implicit equations to be solved are:

$$k = f(y_n + h(a_i k), t_n + c_i h)$$

$$y_{n+1} = y_n + h(bk)$$

(14.120 - 123 from the book)

which is:

$$x(:, nt) = x(:, nt-1) + delta\_t(k \cdot b');$$

(and the eq's for rf, J_r and k) in my code.

b) I simulated for both IRK4 and ERK4 in figure (1), but the result was more or less the same for both. Now, I'm inclined to not trust this result as I have a gut feeling that they should be slightly different.

I therefore decided to investigate changing the timeframe and -step.

| tf | ts | figure |
|---|---|---|
| 4s | 1s | (2) |
| 10s | 1,395s | (3) |

Increasing ts makes ERK less reliable, but IRK always hits the right values.

I can therefore deduce that this applies when $t_f = 2$ and $t_s = 0.4$. The IRK has correct values for its iterations.

Very cool.

c) The IRK is A-stable, so it won't matter what $\lambda$ is. stable either way you tweak it.

2a)

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

$$x, x_d, g > 1 \qquad k \geq 1$$

$$E = \frac{mg}{k-1}\frac{x_d^k}{x^{k-1}} + mgx + \frac{1}{2}m\dot{x}^2 \quad (3)$$

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

$$+ m\dot{x}\ddot{x}$$

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

$$\cdot\left(\left(\frac{x_d^k}{x^k}\right) - 1\right)$$

$$= 0 \qquad \square$$

b) All code can be found at the end.
see figure(4) and figure(5)
It seems that the only scheme that can show the energy being conserved is Implicit Midpoint or "G2" as I've called it.

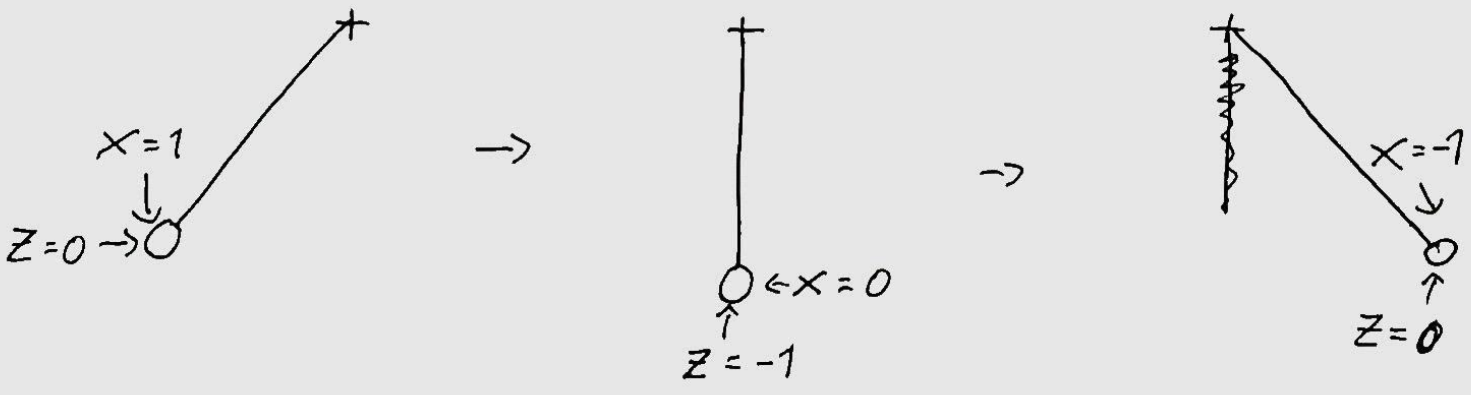
To show the conservation of energy, the scheme has to be stiffly accurate and L-stable.

The scheme has to be able to dampen $Re(j\omega h) \to \infty$

$\Rightarrow b = A^T e_0 \wedge A$ non-singular

Implicit midpoint (G2) is the only scheme that satisfies both of these requirements.

3a) See code for task3 in bottom of "main".
I plotted the simulated states in figure (6) and figure (7)
I couldn't quite understand how the constraint should have been implemented, however, the model/simulation made sense to me anyway.
It behaves like a normal 3D-pendulum, it swings back and forth along the "Y-axis" as I've called it



By decreasing the timestep, the model becomes marginally more accurate, but the computational time increases drastically. Increasing the timestep gives less computational time, but beyond $ts > 0.49$ (or something like that) the model sort of "collapses". I think using a higher order RK-method and a timestep in the goldilock-zone would probably give better results.

b) MatLab gave me a warning about a singular matrix. You can't have singular matrices in Newtons method as you'd end up "dividing by zero"

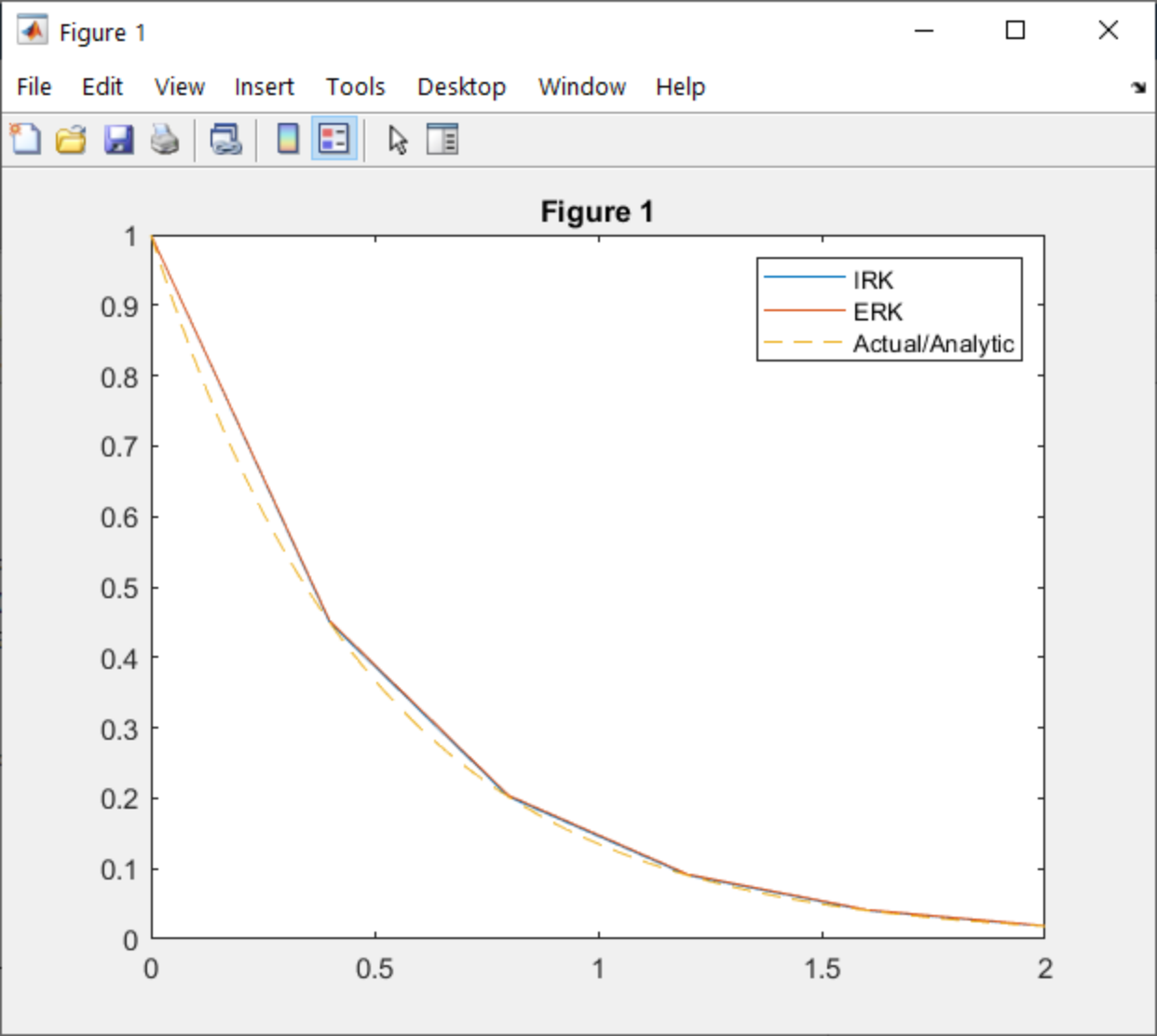


Figure 1
File Edit View Insert Tools Desktop Window Help
Figure 1
IRK
ERK
Actual/Analytic
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2

Figure 1

File Edit View Insert Tools Desktop Window Help

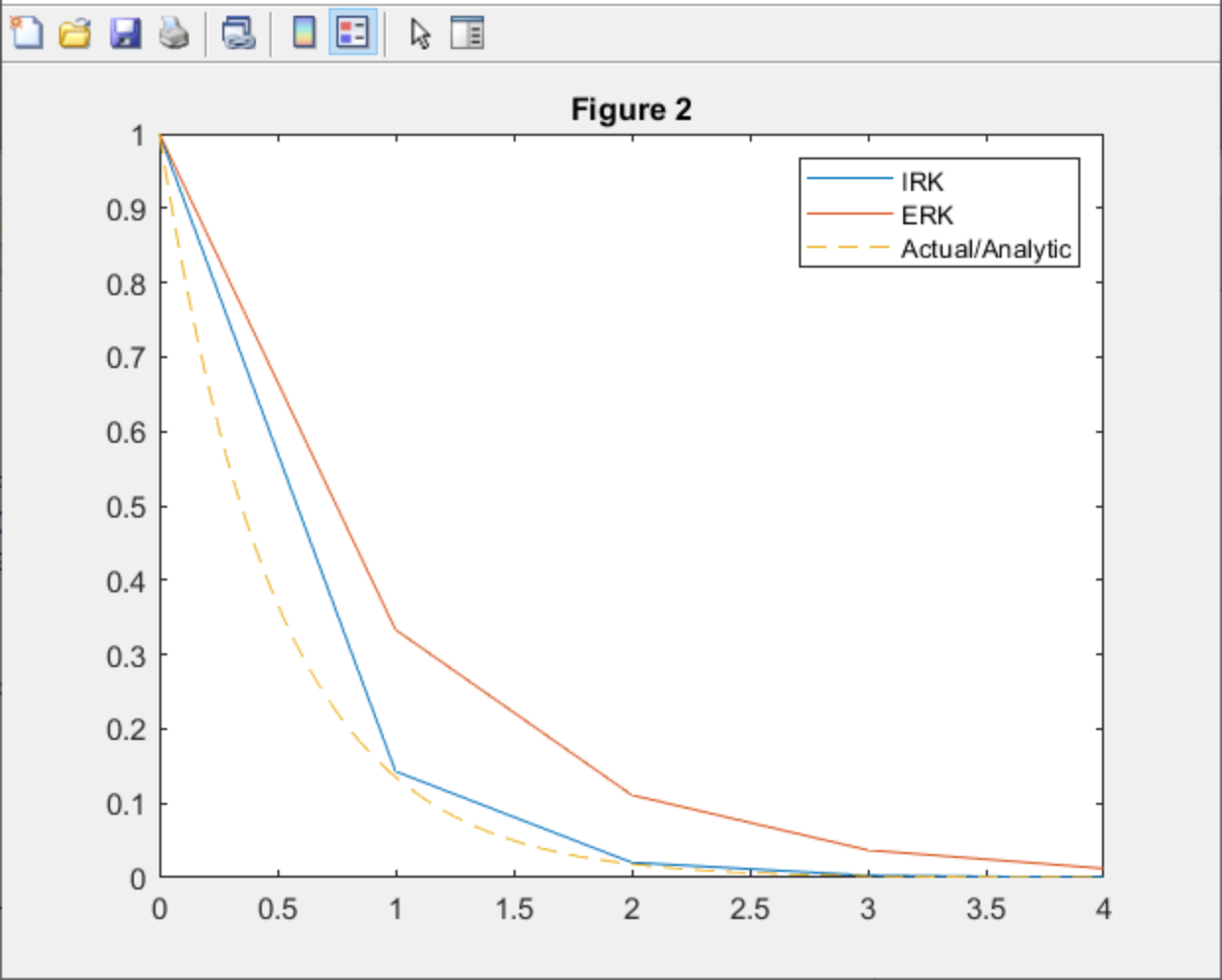


Figure 1

File Edit View Insert Tools Desktop Window Help

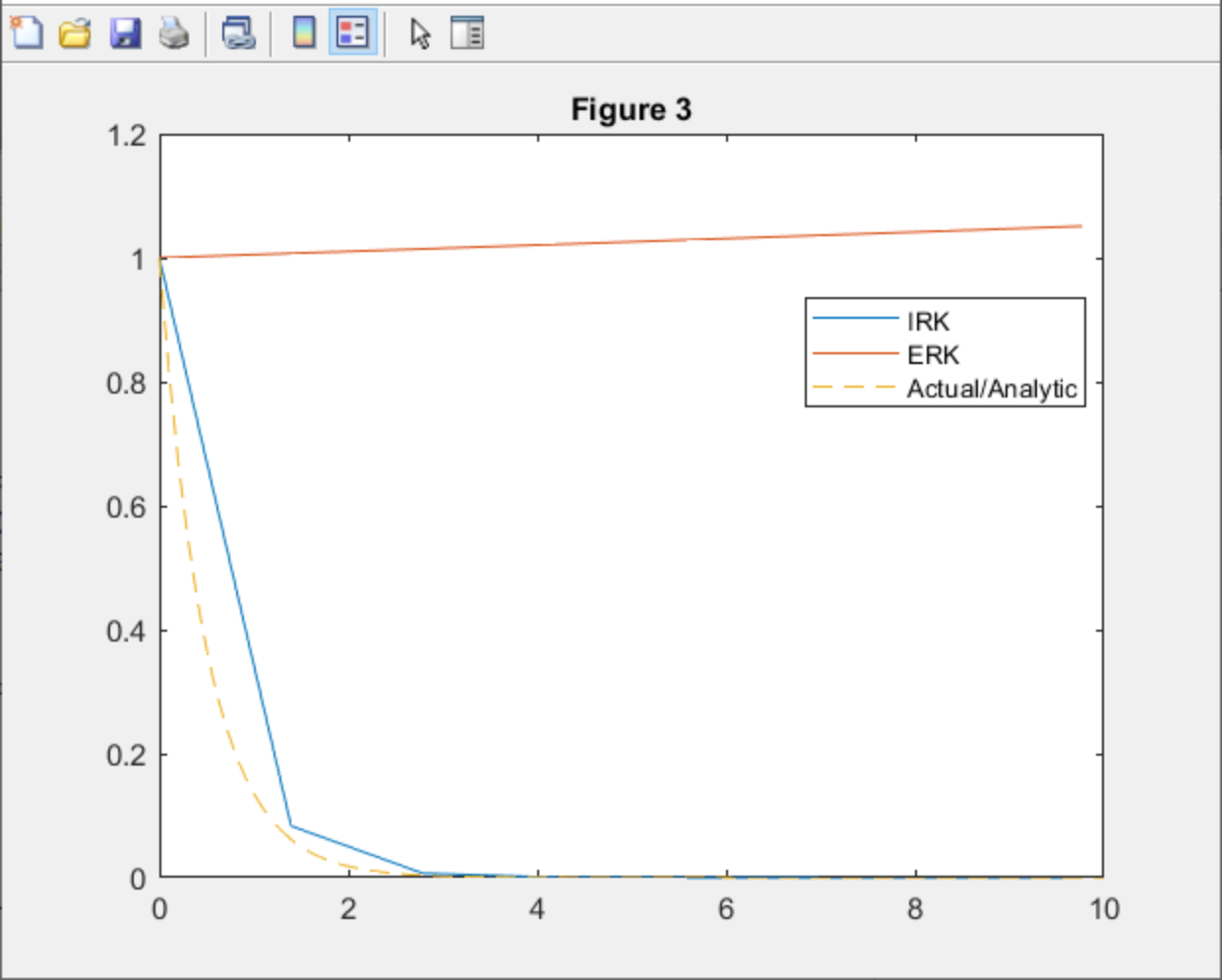


Figure 1

File Edit View Insert Tools Desktop Window Help

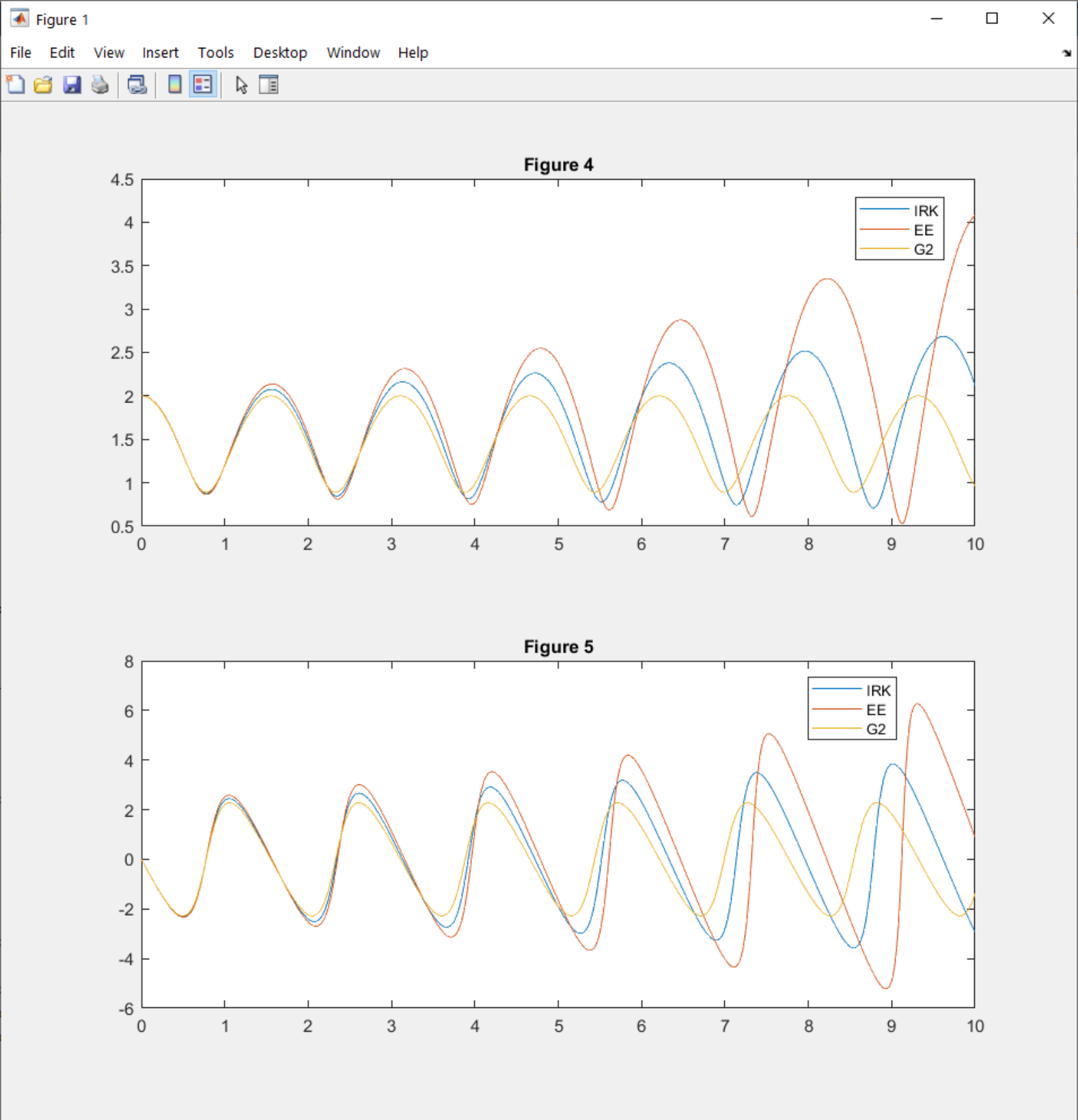
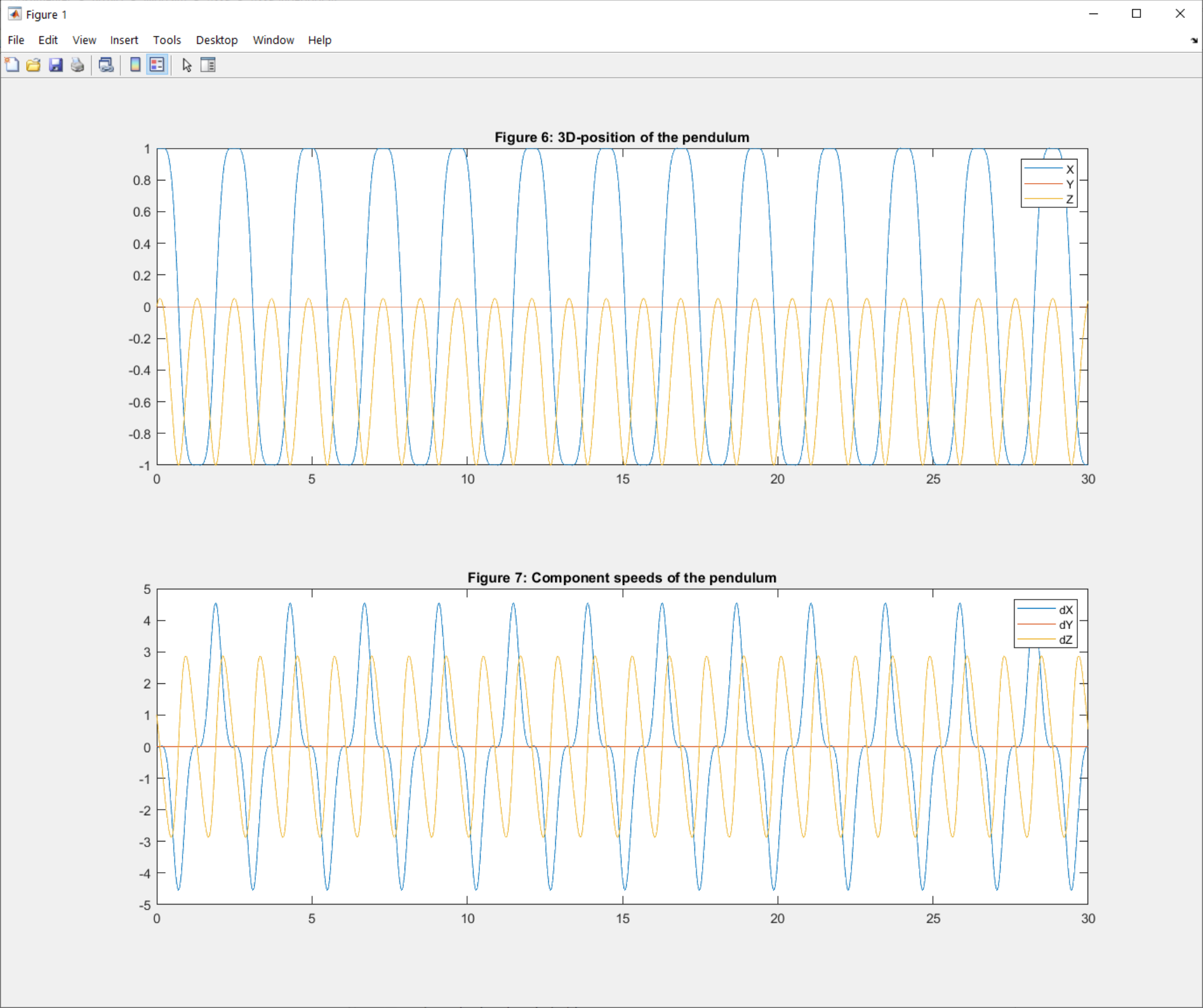


Figure 1

File Edit View Insert Tools Desktop Window Help



```
%IRKTemplate(ButcherArray, f, dfdx, T, x0)
clc;
close all;
clear all;
A_IRK = [(1/4)  (1/4-(sqrt(3)/6));
     (1/4 + (sqrt(3)/6)) (1/4)];
c_IRK = [(1/2 - (sqrt(3)/6));
     (1/2 + (sqrt(3)/6))];
b_IRK = [(1/2) (1/2)];
ButcherExample_IRK = struct('A',A_IRK,'b',b_IRK,'c',c_IRK);

lambda = -2;
t_final = 2;
t_step = 0.4;
T_linspace = linspace(0,t_final,100);
x0 = 1;

f = @(t,x) lambda*x;
J = @(t,x) lambda;
S_anal = exp(lambda*T_linspace);

T = 0:t_step:t_final;

S_IRK = IRKTemplate(ButcherExample_IRK, f, J, T, x0);

% plot(T, S_IRK)
% hold on

A_ERK = [0 0 0 0;
        1/2 0 0 0;
        0 1/2 0 0;
        0 0 1 0];
c_ERK = [0;
         1/2;
         1/2;
         1];
b_ERK = [1/6;
         1/3;
         1/3;
         1/6];

ButcherExample_ERK = struct('A',A_ERK,'b',b_ERK,'c',c_ERK);

S_ERK = ERKTemplate(ButcherExample_ERK, f, T, x0);

% plot(T, S_ERK, T_linspace, S_anal, '--');
% legend('IRK', 'ERK', 'Actual/Analytic');
% title('Figure 3')

%%Task 2)

x_0 = 2;
```

```
xdot_0 = 0;
x_init = [x_0;
          xdot_0];

dt_k = 0.01;
tf = 10;
T2 = 0:dt_k:tf;

x_d = 1.32;
kappa = 2.4;
g = 9.81;
m = 200;

f2 = @(t,x) [x(2);
             (-g*(1-(x_d/x(1))^kappa))];

J2 = @(t,x) [0 1;
             (-(g*kappa*x_d*(x_d/x(1))^(kappa-1))/x(1)^2) 0];

A_EE = 0;
c_EE = 0;
b_EE = 1;
ButcherExample_EE = struct('A',A_EE,'b',b_EE,'c',c_EE);

A_G2 = 1/2;
c_G2 = 1/2;
b_G2 = 1;

ButcherExample_G2 = struct('A',A_G2,'b',b_G2,'c',c_G2);

S2_IRK = IRKTemplate(ButcherExample_IRK, f2, J2, T2, x_init)';
S2_EE = ERKTemplate(ButcherExample_EE, f2, T2, x_init)';
S2_G2 = IRKTemplate(ButcherExample_G2, f2, J2, T2, x_init)';

% subplot(211)
% plot(T2, S2_IRK(:,1))
% hold on
% plot(T2, S2_EE(:,1));
% hold on
% plot(T2, S2_G2(:,1));
% hold on
% legend('IRK', 'EE', 'G2');
% title('Figure 4');
%
% subplot(212)
% plot(T2, S2_IRK(:,2))
% hold on
% plot(T2, S2_EE(:,2));
% hold on
% plot(T2, S2_G2(:,2));
% hold on
% legend('IRK', 'EE', 'G2');
% title('Figure 5');
```

```
%%Task 3)
p  = sym('p', [3,1]);
v  = sym('v', [3,1]);

t = sym('t');
z = sym('z');

dp = sym('dp', [3,1]);
dv = sym('dv', [3,1]);

X  = [p;
      v];
dX = [dp;
      dv];

X_0 = [1 0 0 0 0 1]'; %#2 should be 0
Z_0 = 1;
L = 1;
M = 10;

%Cq = 1/2 * (X(1:3)' * X(1:3) - L^2); for 3b)

f = [X(4:6) - dX(1:3);
    -M*g*[0 0 1]' - z*X(1:3) - M*dX(4:6);
    X(1:3)' * dX(4:6) + X(4:6)'*X(4:6)];

    %X(1:3)' * dX(4:6) + X(4:6)'*X(4:6)];



j_dX = jacobian(f, dX);
j_X = jacobian(f, X);
j_Z = jacobian(f, z);


J_dX = matlabFunction(j_dX, 'Vars', {[dX],[X],z,t});
J_X = matlabFunction(j_X, 'Vars', {[dX],[X],z,t});
J_Z = matlabFunction(j_Z, 'Vars', {[dX],[X],z,t});
f = matlabFunction(f, 'Vars', {[dX],[X],z,t});


dT3 = 0.01;
T3 = [0:dT3:30];



[X, dX, z] = RKDAE(ButcherExample_IRK, f, J_dX, J_X, J_Z, T3, X_0,
 Z_0);


% j_Cq = jacobian(Cq,X(1:3,:)');
% Cq = matlabFunction(Cq, 'Vars', {[dX],[X],z,t});
% J_Cq = matlabFunction(j_Cq, 'Vars', {[dX],[X],z,t});
```

```
% Sol_Cq = IRKTemplate(ButcherExample_IRK, Cq, j_Cq, T3, X_0);

% plot(T3,z);

% subplot(211)
% plot(T3, X(1:3,:)');
% legend('X','Y','Z');
% title('Figure 6: 3D-position of the pendulum')
%
% subplot(212)
% plot(T3, X(4:6,:)');
% legend('dX','dY','dZ');
% title('Figure 7: Component speeds of the pendulum')
```

*Published with MATLAB® R2019a*

```
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;
    c = ButcherArray.c;
    b = ButcherArray.b;

    Nt = length(T);
    Nx = length(x0);
    Nk = size(A,1);

    dt = diff(T);

    k = zeros(Nx*Nk,1);
    x = zeros(Nx, Nt);
    %
    %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    x(:,1) = x0; % initial iteration
    % Loop over time points
    for nt=2:Nt
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % 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
        delta_t = dt(nt-1);
        k = reshape(k, [Nx*Nk,1]);
        rf = @(k)IRKODEResidual(k, x(:,nt-1), nt, delta_t, A, c, f);
        J_r =
 @(k)IRKODEJacobianResidual(k,x(:,nt-1),nt,delta_t,A,c,dfdx);
        k = reshape(NewtonsMethod(rf, J_r, k),[Nx,Nk]);
        x(:,nt) = x(:,nt-1) + delta_t*(k*b');
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    end

end
function g = IRKODEResidual(k_g,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_g,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
```

```
Not enough input arguments.

Error in IRKTemplate (line 15)
    A = ButcherArray.A;
```

*Published with MATLAB® R2019a*

```
function x = ERKTemplate(ButcherArray, f, T, 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
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % 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]

    A = ButcherArray.A;
    c = ButcherArray.c(:);
    b = ButcherArray.b(:);

    Nstage = size(A,1);
    Nt = length(T);
    Nx = length(x0);

    K = zeros(Nx, Nstage);
    x = zeros(Nx, Nt);
    xt = x0;

    dT = diff(T);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    x(:,1) = x0;
    % Loop over time points
    for nt=2:Nt

        t = T(nt-1);
        dt = dT(nt-1);
        K(:,1) = f(t,xt);
        for nstage=2:Nstage
            a = A(nstage, 1:nstage-1)';
            K(:,nstage) = f(t + dt*c(nstage), xt +
 dt*(K(:,1:nstage-1)*a));
        end
        xt = xt + dt*(K*b);
        x(:,nt) = xt;
    end

        %Mine (the following code) was weird, so used the one from the
        %solution for Ass7 instead
%         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         % Update variables
%         x_k = x(:,nt-1);
%         K(:,1) = f(T(nt), x_k);
%         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%         % Loop that calculates k1,k2,...,kNstage
%         for nstage=2:Nstage
%             ksum = 0;
%             for i=1:nstage-1
%                 ksum = ksum + A(nstage,i)*K(:,i);
%             end
%             K(:,nstage) = f(T(nt), x_k+dT*ksum);
%         end
%         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         % Calculate and save next iteration value x_t
%         xsum = 0;
%         for m=1:Nstage
%            xsum = xsum + b(m)*K(:,m);
%         end
%         x(:,nt) = x_k + dT*xsum;
%         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end
```

```
Not enough input arguments.

Error in ERKTemplate (line 15)
    A = ButcherArray.A;
```

*Published with MATLAB® R2019a*

```
function [x,xdot,z] = RKDAE(ButcherArray, F, dFdxdot, dFdx, dFdz, T,
 x0, z0_est)
    % Returns the iterations of a RK method using Newton's method
    % ButcherArray: Struct with the RK's Butcher array
    % F: Function handle.
    %    Implicit function for DAE: F(xdot,x,z,t)=0
    % dFdxdot, dFdx, dFdz: Function handles
    %                      Jacobians of f w.r.t. x_dot,x,z,
 respectively
    % T: Vector of time points
    % x0: Initial state
    % z0_est: Estimate of algebraic variables at initial time
    % x: RK iterations for x, Nx x Nt
    % xdot, z: xdot and z values for x and T, Nx x Nt

    % Definitions and allocations
    Nt = length(T);
    Nx = length(x0);
    Nz = length(z0_est);
    dT = diff(T);
    x = zeros(Nx,Nt);
    xdot = zeros(Nx,Nt);
    z = zeros(Nz,Nt);
    A = ButcherArray.A;
    b = ButcherArray.b(:);
    c = ButcherArray.c(:);
    Nstage = size(A,1);
    % Make sure that the vector function handles can act on
    % concatenations of vectors
    FVec = @(xdot,x,z,t) ExpandFunction2Concatenations(F,xdot,x,z,t);
    dFdxdotVec = @(xdot,x,z,t)
 ExpandFunction2Concatenations(dFdxdot,xdot,x,z,t);
    dFdxVec = @(xdot,x,z,t)
 ExpandFunction2Concatenations(dFdx,xdot,x,z,t);
    dFdzVec = @(xdot,x,z,t)
 ExpandFunction2Concatenations(dFdz,xdot,x,z,t);
    % Start integration
    [xdot0,z0] =
 SolveForXdotAndZGivenX(x0,T(1),FVec,dFdxdotVec,dFdzVec,zeros(Nx,1),z0_est);
    x(:,1) = x0;
    xdot(:,1) = xdot0;
    z(:,1) = z0;
    xt = x0;
    xdott = xdot0;
    zt = z0;
    w = [repmat(xdott,Nstage,1); repmat(zt,Nstage,1)]; % initial guess
    % Integrate
    for nt=2:Nt
        t = T(nt-1);
        dt = dT(nt-1);
        G = @(w) RKDAEResidual(w,xt,t,dt,A,c,FVec);
```

```
        JG = @(w)
 RKDAEJacobianResidual(w,xt,t,dt,A,c,dFdxdotVec,dFdxVec,dFdzVec);
        w = NewtonsMethod(G,JG,w);
        K = reshape(w(1:Nx*Nstage),Nx,Nstage);
        xt = xt + dt*(K*b);
        x(:,nt) = xt;
        [xdott,zt] = SolveForXdotAndZGivenX(xt,t
+dt,FVec,dFdxdotVec,dFdzVec,xdott,zt);
        xdot(:,nt) = xdott;
        z(:,nt) = zt;
    end
end
function [xdot,z] =
 SolveForXdotAndZGivenX(x,t,F,dFdxdot,dFdz,xdotest,zest)
    % Given x and t, returns xdot and z value such that
 F(xdot,x,z,t)=0
    % y = [xdot;z]
    Nx = length(x);
    G = @(y) F(y(1:Nx),x,y(Nx+1:end),t);
    JG = @(y) [dFdxdot(y(1:Nx),x,y(Nx+1:end),t) dFdz(y(1:Nx),x,y(Nx
+1:end),t)];
    y = NewtonsMethod(G,JG,[xdotest;zest]);
    xdot = y(1:Nx);
    z = y(Nx+1:end);
end
function g = RKDAEResidual(w,xt,t,dt,A,c,F)
    % Returns the residual function for the RK scheme iteration
    % w = [K1;K2;...;Knstages;z1;z2;...;znstages];
    Nx = length(xt);
    Nstage = size(A,1);
    K = reshape(w(1:Nx*Nstage),Nx,Nstage);
    Z = reshape(w(Nx*Nstage+1:end),[],Nstage);
    Tg = t+dt*c';
    Xg = xt+dt*K*A';
    g = reshape(F(K,Xg,Z,Tg),[],1);
end
function G = RKDAEJacobianResidual(w,xt,t,dt,A,c,dFdxdot,dFdx,dFdz)
    % Returns the Jacobian of the residual function
    % for the RK scheme iteration
    % w = [K1;K2;...;Knstages;z1;z2;...;znstages];
    Nx = length(xt);
    Nstage = size(A,1);
    K = reshape(w(1:Nx*Nstage),Nx,Nstage);
    Z = reshape(w(Nx*Nstage+1:end),[],Nstage);
    Nz = size(Z,1);
    TG = t+dt*c';
    XG = xt+dt*K*A';
    dFdxdotG = cell2mat(arrayfun(@(i)
 dFdxdot(K(:,i),XG(:,i),Z(:,i),TG(:,i))',...
        1:Nstage,'UniformOutput',false))';
    dFdxG = cell2mat(arrayfun(@(i)
 dFdx(K(:,i),XG(:,i),Z(:,i),TG(:,i))',...
        1:Nstage,'UniformOutput',false))';
```

```
    dFdzG = cell2mat(arrayfun(@(i)
 dFdz(K(:,i),XG(:,i),Z(:,i),TG(:,i))',...
        1:Nstage,'UniformOutput',false))';
    G = [repmat(dFdxdotG,1,Nstage).*kron(eye(Nstage),ones(Nz
+Nx,Nx)) ...
         + repmat(dFdxG,1,Nstage).*kron(dt*A,ones(Nz+Nx,Nx)) ...
         repmat(dFdzG,1,Nstage).*kron(eye(Nstage),ones(Nx+Nz,Nz))];
end
function fVec = ExpandFunction2Concatenations(f,xdot,x,z,t)
    % Returns the concatenation [f(xdot(:,i),x(:,i),z(:,i),t(i)):
 i=1...N]
    % f, function handle that returns column vector
    % xdot, matrix [] x N
    % x, matrix [] x N
    % z, matrix [] x N
    % t, matrix 1 x N
    N = size(t,2);
    fVec = cell2mat(arrayfun(@(i) f(xdot(:,i),x(:,i),z(:,i),t(i))',...
        1:N,'UniformOutput',false))';
end
```

```
Not enough input arguments.

Error in RKDAE (line 15)
    Nt = length(T);
```

*Published with MATLAB® R2019a*

```
function x = NewtonsMethod(f,J,x0,tol,N)
    if nargin < 5
        N = 100;
    end
    if nargin < 4
        tol = 1e-6; %Original 1e-6
    end
    x = x0;
    n = 1;
    fx = f(x);

    iterate = norm(fx,Inf) > tol;
    while iterate
        x = x - J(x)\fx;
        n = n+1;
        fx = f(x);
        iterate = norm(fx,Inf) > tol && n <= N;
    end
end
```

```
Not enough input arguments.

Error in NewtonsMethod (line 8)
    x = x0;
```

*Published with MATLAB® R2019a*