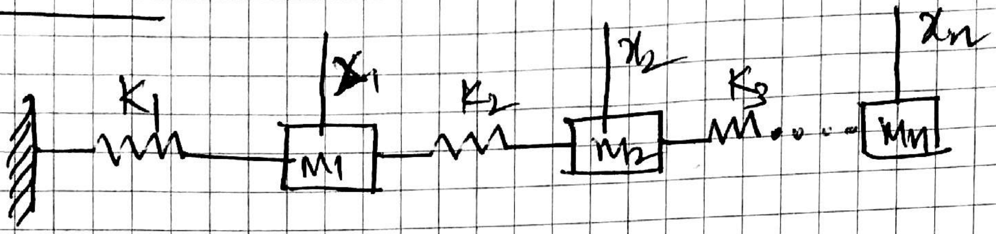
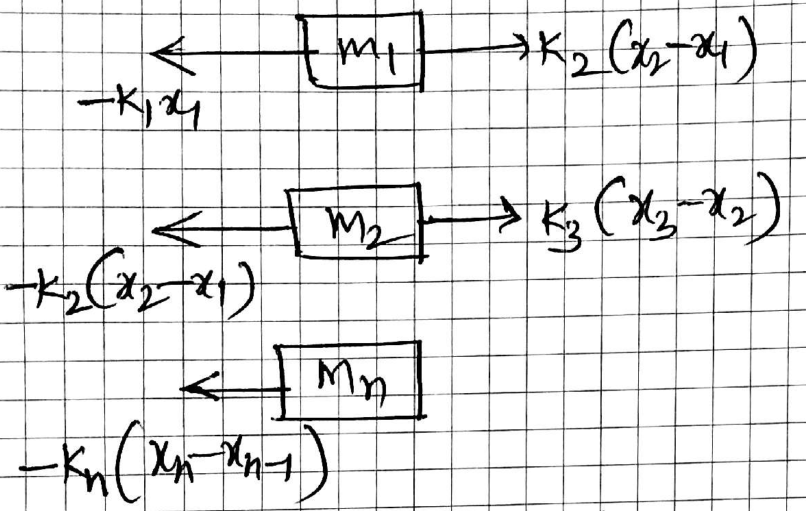
ASSIGNMENT - 2

Exercise - 1





Governing equations:

$$\ddot{x}_1 + \left(\frac{k_1 + k_2}{m_1}\right) x_1 - \frac{k_2}{m_1} x_2 = 0 \longrightarrow (1)$$

$$\ddot{x}_2 - \left(\frac{k_2}{m_2}\right) x_1 + \left(\frac{k_2 + k_3}{m_2}\right) x_2 - \left(\frac{k_3}{m_2}\right) x_3 = 0 \longrightarrow (2)$$

$$\vdots$$

$$\ddot{x}_n - \left(\frac{k_n}{m_n}\right) x_{n-1} + \left(\frac{k_n}{m_n}\right) x_n = 0 \longrightarrow (n)$$

This further can be represented as,

$$
\begin{bmatrix} \ddot{x}_1 \\ \ddot{x}_2 \\ \ddot{x}_3 \\ \vdots \\ \ddot{x}_n \end{bmatrix}_{(n\times 1)} = \begin{bmatrix} -\left(\frac{k_1+k_2}{m}\right) & \left(\frac{k_2}{m_1}\right) & 0 & 0 & 0 & \cdots \\ \left(\frac{k_2}{m_2}\right) & -\frac{(k_2+k_3)}{m_2} & \left(\frac{k_3}{m_2}\right) & 0 & 0 & \cdots \\ \vdots & & \ddots & & & \\ & & & & \frac{k_n}{m_n} & -\frac{k_n}{m_n} \end{bmatrix}_{(n\times n)} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}_{(n\times 1)} \rightarrow (I)
$$

The above equation can be represented as,

$$
\boxed{\ddot{X} = A\,X} \longrightarrow (II)
$$

Eqn (II), has to be transformed to first order.

Considering, $Y_1 = X$, $Y_2 = \dot{X}$

$$
\Rightarrow \begin{bmatrix} \dot{Y}_1 \\ \dot{Y}_2 \end{bmatrix} = \begin{bmatrix} 0 & \underline{\underline{I}} \\ \underline{\underline{A}} & 0 \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix}
$$

, which is in the form

$$
\boxed{\dot{Y} = Q\,Y}
$$

Stability in the sense of Lyapunov

The considered spring mass system has energy.

$$E = E_{kin} + E_{pot}$$

$$= \sum_i \frac{1}{2} m_i v_i^2 + \sum_i \sum_{j>i} \frac{1}{2} k_{ij} \left(\|x_i - x_j\| - l_{0,ij} \right)^2$$

Which is constant.

Here,

k_{ij} = spring constant

l_0 = initial displacement of spring (conditional)

Assuming that the equilibrium point is not stable, then in all $B_\varepsilon(x_{eq})$

$\exists\, x_0 : \|x(t)\| \longrightarrow \infty$ for growing 't'

So, $x = \begin{bmatrix} x \\ v \end{bmatrix}$, $\|x\| \longrightarrow \infty$ or $\|v\| \longrightarrow \infty$

But then, $E \longrightarrow \infty$, which contradicts that E is constant.

So there is no such x_0, $\exists\, B_\varepsilon(x_{eq}) : x_0 \in B_\varepsilon(x_{eq})$

$\|x(t)\| \leq \delta$.

Assignment 2

Exercise 2(a)

```
function u=generalImplicitMethod(my_fun,my_der,t,u,h,A,b,c,maxiter,tol)

% FUNCTION
%
%     general_implicit_method(my_fun,my_der,t,u,h,c,A,b,maxiter,tol)
%
%  solves the ODE equation given by my_fun (the function for ODE)
%  and my_der (the function derivative over u)
%
%  Input:
%
%       my_fun - function (function handle)
%       my_der - derivative of a function (function handle)
%       t - current time step
%       u - the solution from previous step m (vector of dimension n)
%       h - time step
%       A,b,c - Bucher table of the method
%       maxiter - maximal number of iterations
%       tol - the tolerance for Newton method
%
%  Output:
%
%       u - the solution in the next step m+1
%
% IMPORTANT:
%      input functions have two arguments: time, parameter.
%
%
%  B. Rosic
%  wire@tu-bs.de
%  2011

    if nargin<1
        disp('No arguments specified')
    elseif nargin<8
        disp('Not enough input arguments')
    elseif nargin<9
        maxiter=100;
        tol=1e-10;
    elseif nargin<10
        tol=1e-10;
    end

    % check dimensions

    if size(u,2)>size(u,1)
        u=u';
    end

    if size(b,2)<size(b,1)
        b=b';
```

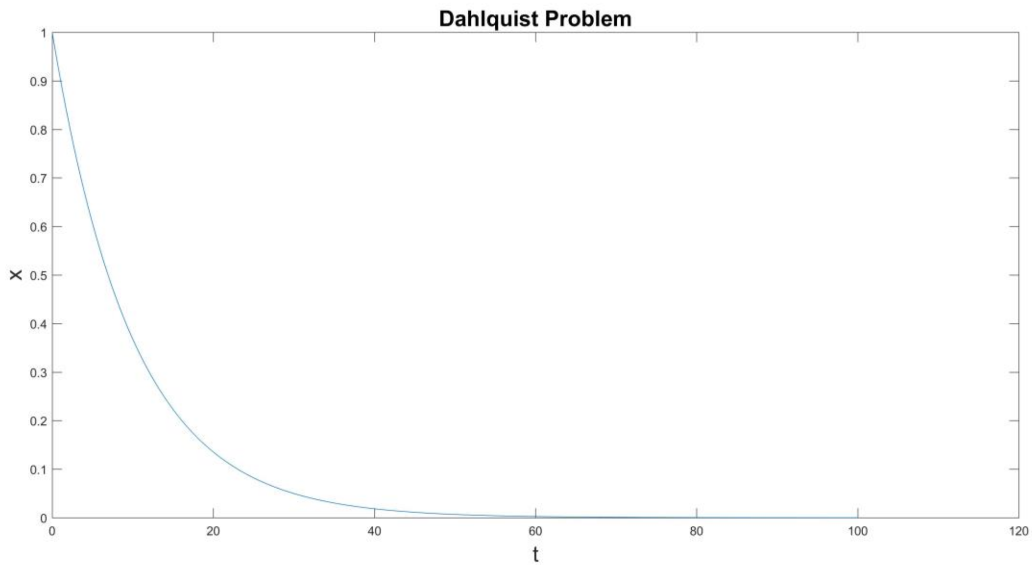
```
    end

     if size(c,2)<size(c,1)
        c=c';
    end

    % Find stage

    s = length(b);

    % Size of the system of equations
    n=length(u)

    % final size of parameters
    w=n*s;

    % Term h(A\kron Id) in Eq. (39)

    Id=eye(n);
    AId=h*kron(A,Id);

    % Term e\kron u_m in Eq. (39)

    e=ones(s,1);
    eu_m=kron(e,u);

    u0=reshape(eu_m,[],1);

    % initial value for v

    v=zeros(n*s,1);
    v=repmat(u,s,1);
    u_old=u;
    % vector t+c*h

    ch=ones(size(c))*t+c*h;

    convg=0;

%     Newton iteration

    niter=0;
    %for ass.4:
    simplifiedFlag =true;
    if simplifiedFlag ==true
        for i=1:s
         % Jacobian
          dGdV(((i-1)*n+1):i*n,((i-1)*n+1):i*n)=feval(my_der,ch(i),v(((i-
1)*n+1):i*n));%%Verify: ch or ch(i)?
        end
    end
    while (niter<maxiter & ~convg)
```

```
         v_old=v;
         niter=niter+1;

        % Term G(t_m,v) in Eq. (39)
        for i=1:s
          G(((i-1)*n+1):i*n,1)=feval(my_fun,ch(i),v(((i-1)*n+1):i*n));

         % Jacobian
            if simplifiedFlag ==false
            dGdV(((i-1)*n+1):i*n,((i-1)*n+1):i*n)=feval(my_der,ch(i),v(((i-
1)*n+1):i*n));%%Verify: ch or ch(i)?
            end
        end

        J=eye(w)-AId*dGdV;

        % compute residual
        R=v-eu_m-AId*G;

        % solve system of equations
        %ok:
        v=v-J\R;
%demanded in assignment 3:
        %v=v-pcg(J,R,tol/10);


     %   v= gmres(J,-R,10,tol);

        if norm(v-v_old)<tol%Criteria checks past progress, not state

            fprintf('Newton method converged in iteration %d with the norm
%1.5e \n',niter,norm(v-v_old));
           convg=1;

            for i = 1:s
            % iteration converged: compute k and return
            %idx = (l-1)*n+1:l*n;
%ok 1d, not my style:                    k(((i-
1)*n+1):i*n,1)=feval(my_fun,ch(i),v(((i-1)*n+1):i*n));   %u_old+h*k_old(((i-
1)*n+1):i*n));
            k(1:n,i)=feval(my_fun,ch(i),v(((i-1)*n+1):i*n));  %each col 1
stage
            %??k(:,l) = feval(my_fun, t + c(l)*h, u0(idx));
           end
        else if (niter==maxiter-1)
            fprintf('Newton method did not converge in iteration %d, the norm
%1.5e \n',niter,norm(v-v_old));
             end
         k=zeros(n,s);  %to enable continuing

        end

    end
% compute u_m+1
```

```
    u = u + h*k*b';
```

Exercise 2(b)

```
clear all
clc

%% Method: Gauss-Legendre 2-stages (Order 4)

A_GaussLegendre=[1/4      1/4-sqrt(3)/6
        1/4+sqrt(3)/6  1/4];
b_GaussLegendre=[1/2  1/2];
c_GaussLegendre=sum(A_GaussLegendre,2);

%% Dahlquist
% funcPtr = @Dahlquist;
% jacFuncPtr = @JacDahlquist;
l = -1e-1;
funcPtr = @(t,u) l.*u;
jacFuncPtr = @(t,u) l;

%Initial condition as demanded:
u0=1;
t(1)=0;
solGaussLegendre(1,:)=u0;
t_end = 100;
h=0.01;
maxiter =6;
tol= 1e-8;

%%

for step = 1:1:t_end/h
        solGaussLegendre(step+1,:)=generalImplicitMethod(funcPtr,jacFuncPtr,
t(step),solGaussLegendre(step,:),h,A_GaussLegendre,b_GaussLegendre,c_GaussLeg
endre,maxiter,tol);
    t(step+1)=t(step)+h;
end
figure
plot(t,solGaussLegendre)
xlabel('t')
ylabel('x')
title('Dahlquist Problem')
```



Exercise 2(c)

```
clear all
clc

%% Method: Gauss-Legendre 2-stages (Order 4)

A_GaussLegendre=[1/4        1/4-sqrt(3)/6
        1/4+sqrt(3)/6   1/4];
b_GaussLegendre=[1/2   1/2];
c_GaussLegendre=sum(A_GaussLegendre,2);

%% Logistic

r = 0.3;
K = 2000;
funcPtr = @(t,u) r.*u.*(1-(u/K));
jacFuncPtr = @(t,u) r-(2.*r.*u/K);

%Initial condition as demanded:
u0=50;
t(1)=0;
solGaussLegendre(1,:)=u0;
t_end = 100;
h=0.01;
maxiter =6;
```

```
tol= 1e-8;
%%

for step = 1:1:t_end/h
        solGaussLegendre(step+1,:)=generalImplicitMethod(funcPtr,jacFuncPtr,
t(step),solGaussLegendre(step,:),h,A_GaussLegendre,b_GaussLegendre,c_GaussLeg
endre,maxiter,tol);
    t(step+1)=t(step)+h;
end
figure
plot(t,solGaussLegendre)
xlabel('t')
ylabel('x')
title('Logistic Problem')
```