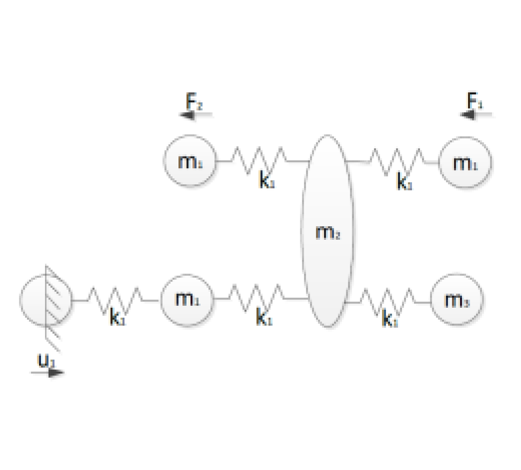
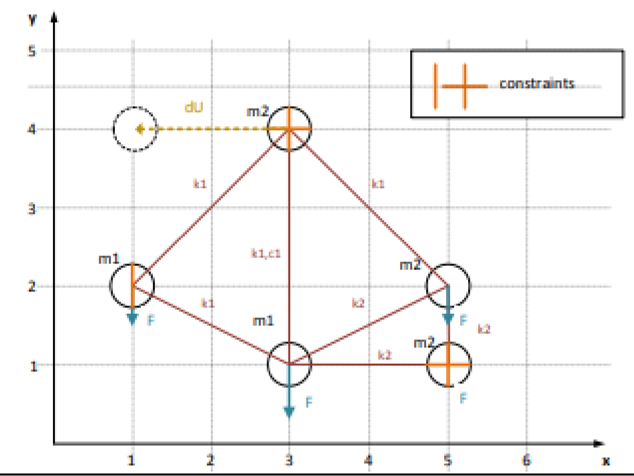
Martynas Kemežys
Physically Based Animation
Individual project 1

1. Problem

2.		

For all tasks set physical parameters as:

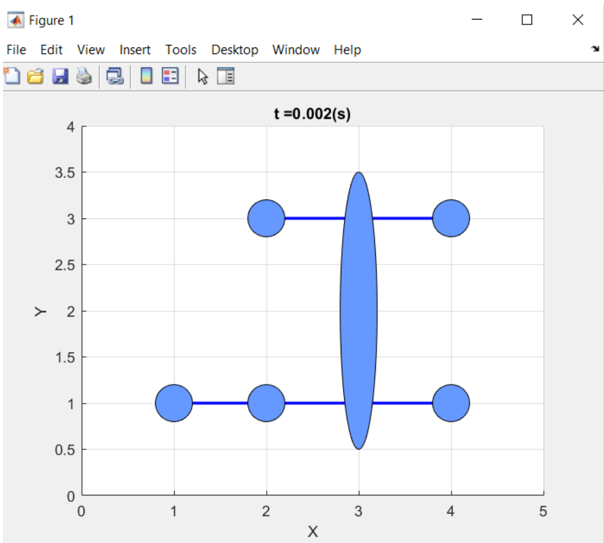
1. Force F starts immediately in time tF. Choose values F and tF freely.
2. Kinematic excitation for node i starts at time $tu1$ and ends at time $tu2$. Choose values $tu1$, $tu2$ and ui freely.
3. Damping is proportional for masses of particles. Choose proportionality coefficient α ($0 < \alpha < 1$) freely.

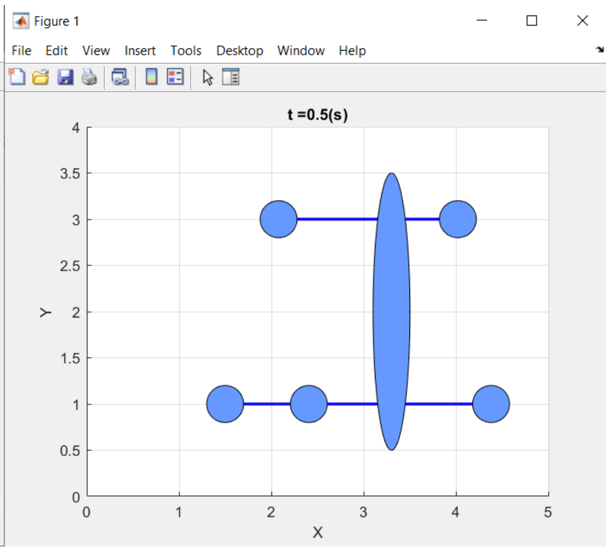
2. Theory background

All pratical material is based on second Newton's law. If the resultant of all forces acting on a particle is non zero, the particle moves with acceleration in the direction of the resultant, the magnitude of the acceleration is directly proportional to the magnitude of the resultant and inversely proportional to the mass of the particle.

3. Visualization of structure before and after animation

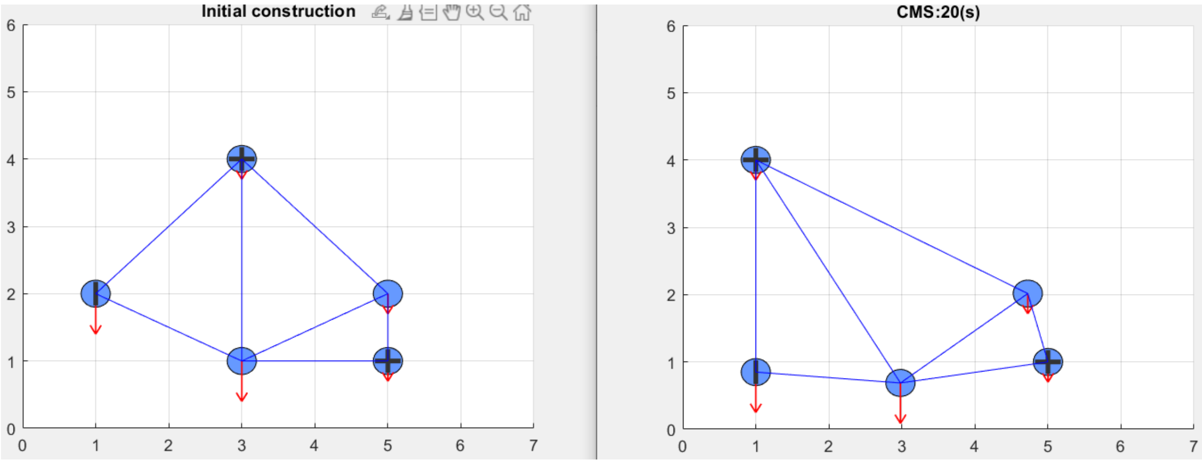
Uni-dimensional elastic structure:





Before After

2D structures of elastically bounded particles:



Before After

4. Displacements of particles in time

Uni-dimensional elastic structure:

```
po =
        0.0215
        0.5000
        0.0792
        0.4066
        0.3024
        0.3812
```

2D structures of elastically bounded particles:

```
po CMS: U =
        y: 0   -1.1510   -0.0195   -0.3118        0
        x: 0   -0.2762    0.0158   -2.0000        0
```

5. Programming code

Uni-dimensional elastic structure:

```
function Dalis1
clc, close all, clear all

% ---- construction data ----
% mass constants
m1 = 1; m2 = 1.5; m3 = 2;
% stiffness of elements
k1 = 4000;
% forces
f1=400; f2=800;
% time moments when forces are added to the construction
tf1 = 0.05; tf2 = 0.1;

%  --- information about nodes ---
% array of node constraints
IS = [  0,    1,    0,   0,   0,    0];
 % displacements of node nr. 2
u2_deltaU  =  0.5;
```

```
 % time moment, when kinematic condition starts
u2_t_start =  0.2;
% how long the motion takes
u2_deltaT  =  0.1;
% masses of the particles
m=   [  m1, 1, m1, m1, m2, m3];
% forces applied to the particles
F =  [  -f1,   0, -f2,  0,  0,  0];
% time moments when the forces start impacting the structure
tf = [  tf1,   0, tf2,  0,  0, 0];

%  --- information about elements ---
% stiffness coefficients
k=[k1, k1, k1, k1, k1];
 % damping coefficients
c=[10, 10, 10, 10, 10];

% information about elements (node1, node2, visualization level in y axis)
ind=[1, 3, 3;
     2, 4, 1;
     3, 5, 3;
     4, 5, 1;
     5, 6, 1];

%  --- visualization data ---

% Coordinates of particles
% x axis 2-4-5-6-3
x=[4, 1, 2, 2, 3, 4];
% y axis
y=[3, 1, 3, 1, 2, 1];
% Diameters of particles
rad1=0.2; rad2=1.5;
rads = [rad1, rad1;
        rad1, rad1;
        rad1, rad1;
        rad1, rad1;
        rad1, rad2;
        rad1, rad1];

nmz=length(m); % total number of nodes (in this example = 6)
nel=length(k); % total number of elements (in this example = 5)
% initial displacements and velocities
U=zeros(nmz,1);  % displacements
DU=zeros(nmz,1); % velocities

% initial data visualization
visualization(x, y, ind, U,  rads, 0);


% numerical integration  - dynamic modelling
Urez=zeros(nmz,1);      % array to save the data of nodes displacements
TT = 0.5;     % numerical integration time (dynamic modelling time)
dt=0.001;     % numerical integration step
for t=0:dt:TT       % loop of numerical integration
    % updating acceleration
    DDU=acceleration(m,k,c,ind,F,tf, U,DU,t,@F_time_function);
    % updating velocities
    DU=DU+dt*DDU;
    % boundary condition of velocities
    DU(find(IS))=0;
    DU(2) = du_time_function(u2_t_start, u2_deltaT, u2_deltaU, t);
    % updating displacements
    U=U+dt*DU;
    % saving intermediate results
    jj=round(t/dt)+1;
    Urez(:,jj)=U;
    % animation condition of the structure
    figure(1);   visualization(x, y, ind, U,  rads,t);
    pause(0.01);
 end


% representing displacements of particles in time
figure(2);hold on; grid on;
color={'b-';'r-';'g-';'m-';'c-';'k-';};
% plotting displacements in time of each node
```

```
for i=1:nmz
    plot([0:dt:TT],Urez(i,:),color{i});
end

plot([tf1, tf1], [-0.2, 0.2], 'g--');
plot([tf1, tf1], [-0.2, 0.2], 'k--');
plot([u2_t_start, u2_t_start], [-0.2, 0.2], 'r--');
legend('1 node','2 node', '3 node', '4 node', '5 node', '6 node');
xlabel('Time (s)');
ylabel('Displacements (m)');
end
% *********************************************
% function that governs application of forces
function Ft=F_time_function(F, tf, t)
    nmz=length(F);
    Ft = zeros(nmz, 1);
    for i=1:nmz,
        if(tf(i) < t)
            Ft(i) = F(i);
        end
    end
end
% *********************************************
% time function of velocities
function du=du_time_function(t_start, deltaT, deltaU, t)
    if t >= t_start  % if addition of velocity started
        if  t <= t_start + deltaT, % if the node is moving
            % velocity is prescribed
            b = deltaT; a = deltaU;


            du = (a*pi*cos((3*pi)/2 + (pi*t)/b))/(2*b);


            if(t == t_start || t == t_start + deltaT),
               du = du/2;
            end
        else,
            du = 0; % displacement has already been applied, velocity of particle = 0
        end
    else
        du = 0; % motion has not started, velocity of particle = 0
    end
    return
end

% *********************************************
function DDU=acceleration(m,k,c,ind,F,tf,U,DU,t,Ft)
    nel=length(k);nmz=length(m);
    DDU=zeros(nmz,1);
    DDU= Ft(F, tf, t); % adding external forces
    for iii=1:nel      % adding forces of elements
        i=ind(iii,1);j=ind(iii,2);
        T=(U(j)-U(i))*k(iii)+(DU(j)-DU(i))*c(iii);
        DDU(i)=DDU(i)+T;
        DDU(j)=DDU(j)-T;
    end
    % now variable DDU represents all forces (F) in the structure
    % solving F = m * a, and getting acceleration
    DDU=DDU./m';
return
end

% *********************************************
function visualization(x,y,ind,U,rads,t)
    % prepare figure for visualization
    clf; hold on; grid on;axis equal;
    axis([min(x)-1 ,max(x)+1,min(y)-1 ,max(y)+1]);
    title(['t =', num2str(t), '(s)']);
    xlabel('X');
    ylabel('Y');
    % calculate number of nodes and elements
    nmz=length(x);nel=size(ind,1);
    % visualizing elements
    for i=1:nel
        line([x(ind(i,1))+U(ind(i,1)), ...
            x(ind(i,2))+U(ind(i,2)) ], ...
```

```
            [ind(i,3), ind(i,3)],...
            'Color','blue','Linewidth',2);
    end
    % visualizing nodes
    for i=1:nmz
        rectangle('Position',[x(i)+U(i)-rads(i,1), ...
            y(i)-rads(i,2),rads(i,1)*2,rads(i,2)*2]....
            ,'Curvature',[1,1],'FaceColor',[0.4 0.6 1]);
    end
    disp('po = ');disp(U);
return
end
```

2D structures of elastically bounded particles:

```
function Dalis2

    close all; clf; clc; clear all;

    % ---- construction data ----
    % mass constants
    m1 = 2; m2 = 1;
    % stiffness constrants
    k1 = 100; k2 = 2500;
    % spring dampers
    c1 =10; c2 = 20;
    % masses of particles
    m = [m1;  m1;  m2; m2; m2];
    % coordinates of particles
    cords = [  1,  2;  3,  1;  5,  1;  5,  2;  3,  4];

    % constraints array
    IS = logical([  1,   0,   0,   0,   1,   1,   0,   0,   1,   1]);
    % particles displacements start times
    U_t_start = [   0,   0,   0,   0,   0,   0,   0,   0,   0.1,   0.1];
    % particle displacements
    deltaU    = [   0,   0,   0,   0,   0,   0,   0,   0,   -2,   0];
    % particles kinematics times
    U_deltaT = [    0,   0,   0,   0,   0,   0,   0,   0,   2,   2];

    % elements (springs) array 10 liko 7
    elm = [1, 2; 2, 3; 2, 4;2 , 5; 3 , 4;  4 , 5; 5,1];
    % elements (springs) stiffness
    k = [k1, k2 ,k2, k1 , k2, k1, k1];
    % elements (springs) dampings
    c = [0, 0 ,0, c1 , 0, 0, 0];

    % ading m*g forces to the particles
    g = -9.8;
    nNodes = length(m);
    F = zeros(nNodes *2,1);
    for(i = 1:nNodes)
        F(i*2) = m(i) *g;
    end

    % total number of particles in construction
    nNodes = length(m);
    % degree of freadom in the particles
    DOF = 2;
    % displacements array
    U  = zeros(nNodes*DOF,1); % displacement vector
    % rendering construction data
    rendering(U,elm,cords,F,IS,1, 'Initial construction');
    pause(0.1);


    % numerical integration  - dynamic modelling
    TT=20; dt=0.01;    % intrgration time and step
    U  = zeros(nNodes*DOF,1); % displacement vector
    DU = zeros(nNodes*DOF,1); % velocities

    for t=0:dt:TT
          % updating acceleration
        DDU=acceleration(U,DU,t,m, F,elm,cords, k, c);
```

```matlab
        % updating velocities
        DU=DU+dt*DDU';
        % boundary condition of velocities
        DU(IS)=0;
        % updating velocities of particles of kinematic motion
        for i=1:nNodes*DOF
           if IS(i) == 1
                [ xxx DU(i) xxx] = ...
                    du_time_function(U_t_start(i), deltaU(i), U_deltaT(i), t);
           end
        end
        % updating displacements
        U=U+dt*DU;
        % rendering construction
        rendering(U,elm,cords,F,IS, 3, strcat('CMS:  ', num2str(t), '(s)'));
        pause(0.01);
    end
    disp('po CMS: U = ');disp(U');
    rendering(U,elm,cords,F,IS, 3, strcat('CMS:  ', num2str(t), '(s)'));
end



% acceleration function
function DDU = acceleration(U,DU,t,mass, F,elm,cords, k, c)
    dof=2;
    nmz=length(mass);NN=nmz*dof;
    siz=size(elm);nel=siz(1);
    T=F;  % adding external forces
    % assembling forces of elements to the global forces vectors
    for i=1:nel   %
        r=elm(i,1);s=elm(i,2);  % nubers of nodes of spring ends
        ur=U((r-1)*dof+1);vr=U(r*dof);     % displacements of nodes of spring ends
        us=U((s-1)*dof+1);vs=U(s*dof);
        xr=cords(r,1)+ur;yr=cords(r,2)+vr;
        xs=cords(s,1)+us;ys=cords(s,2)+vs;
        dur=DU((r-1)*dof+1);dvr=DU(r*dof); % velocities of spring particles
        dus=DU((s-1)*dof+1);dvs=DU(s*dof);

        l0=sqrt((cords(s,1)-cords(r,1))^2+(cords(s,2)-cords(r,2))^2); % initial spring length
        lrs=sqrt((xs-xr)^2+(ys-yr)^2);  % current lenght of spring
        n= [xs-xr , ys-yr]/lrs;         % element forcwe normal vector

        Trs=k(i)*(lrs-l0)+c(i)*dot( n, [dus-dur,dvs-dvr ] ); % force created by element

        % adding spring forces to the global node forces array
        T((r-1)*dof+1)=T((r-1)*dof+1)+Trs*n(1);
        T(r*dof)=T(r*dof)+Trs*n(2);
        T((s-1)*dof+1)=T((s-1)*dof+1)-Trs*n(1);
        T(s*dof)=T(s*dof)-Trs*n(2);

    end
DDU(1:dof:NN)=T(1:dof:end)./mass;   % forces dividing from masses
DDU(2:dof:NN)=T(2:dof:end)./mass;
end

% kinematic function
function [ddu du u]=du_time_function(U_t_start, deltaU,U_deltaT, t)
    % U_t_start(i), deltaU(i), U_deltaT(i)
    ddu =0; du =0;u =0;
    if t >= U_t_start  % if displacement starts
        if  t <= U_t_start + U_deltaT, % if displacement in progress
            if(U_deltaT > 0)
                % updating acceleration, velocities and displacements
                omega=(pi)/U_deltaT;
                ddu = (deltaU /2) * omega^2 * sin (omega *(t - U_t_start)+(3/2)*pi);
                du = (deltaU /2) * omega * cos(omega *(t - U_t_start)+(3/2)*pi);
                u = (deltaU /2) * (sin(omega *(t - U_t_start)+(3/2)*pi)+1);
            else,
               % node displacement = 0 = 0
               ddu =0;du =0;u =0;
            end
        else,
            % displacements already done
            ddu = 0; du = 0; u = deltaU;
        end
    end
```

```
    return
end

% rendering function

function rendering(U,elm,cords,F,IS, fig, strTitle)
    DOF=2;
    xx = size(cords); nNodes = xx(1);;
    xx = size(elm); nElm = xx(1);
    NN = nNodes * DOF;
    ff = figure(fig);
    clf(ff);
    axis([0 7 0 6]);
    hold on; grid on;
    title(strTitle);

    % geting the construction dimention, to visualize the forces as arrows
    % in the construction
    xlim=get(gca,'XLim'); ylim=get(gca,'YLim');
    xn=xlim(2)-xlim(1);yn=ylim(2)-ylim(1); % axis diapazons
    range=min(xn,yn);            % geting minimum of axis
    maxForce=max(abs(F));        % maximum foce
    % scaling coeficients
    mast= range/maxForce*0.1;
    constrLength=range/17;

    for i=1:nNodes
        % rendering particles
        u=U((i-1)*DOF+1);v=U(i*DOF); % displacements of i-th particle
        r=0.2;  % i-os daleles spindulys
        % plotting particle:
        rectangle('Position',[cords(i,1)+u-r,cords(i,2)+v-
r,2*r,2*r],'Curvature',[1,1],'FaceColor',[0.4 0.6 1]);
        % ploting forces of exact particle
        fx=F((i-1)*DOF+1)*mast;fy=F(i*DOF)*mast; % length of ith force particle arrow
        x1=cords(i,1)+u;x2=cords(i,1)+u+fx;y1=cords(i,2)+v;y2=cords(i,2)+v+fy;
        line([x1,x2],[y1,y2],'Color','red','LineWidth',1);
        varr=[x1-x2;y1-y2]; varr =varr/norm(varr)*range/40; % ploting arrow end
        alf=pi/6; transf = [cos(alf) sin(alf);-sin(alf) cos(alf)];
        varr1=transf*varr; line([x2, x2+varr1(1)],[y2,
y2+varr1(2)],'Color','red','LineWidth',1);
        varr1=transf'*varr;line([x2, x2+varr1(1)],[y2,
y2+varr1(2)],'Color','red','LineWidth',1);
        % constraints of velocities :
        ix=IS((i-1)*DOF+1)*mast;iy=IS(i*DOF)*mast;
        if ix ~= 0, line(([cords(i,1)+u, cords(i,1)+u]),([cords(i,2)+v-constrLength/2,
cords(i,2)+v+constrLength/2]),'Color',[ 0.2 0.2 0.2],'LineWidth',3);end
        if iy ~= 0, line(([cords(i,1)+u-constrLength/2,
cords(i,1)+u+constrLength/2]),([cords(i,2)+v, cords(i,2)+v]),'Color',[ 0.2 0.2
0.2],'LineWidth',3);end

    end
    % rendering springs
    for i=1:nElm
        r=elm(i,1);s=elm(i,2); % numbers of spring nodes (particles)
        ur=U((r-1)*DOF+1);vr=U(r*DOF);  % displacements of spring particles
        us=U((s-1)*DOF+1);vs=U(s*DOF);
        xr=cords(r,1)+ur;yr=cords(r,2)+vr;
        xs=cords(s,1)+us;ys=cords(s,2)+vs;
        plot([xr,xs] , [yr,ys],'b-');      % spring rendering as segment
    end
return
end
```