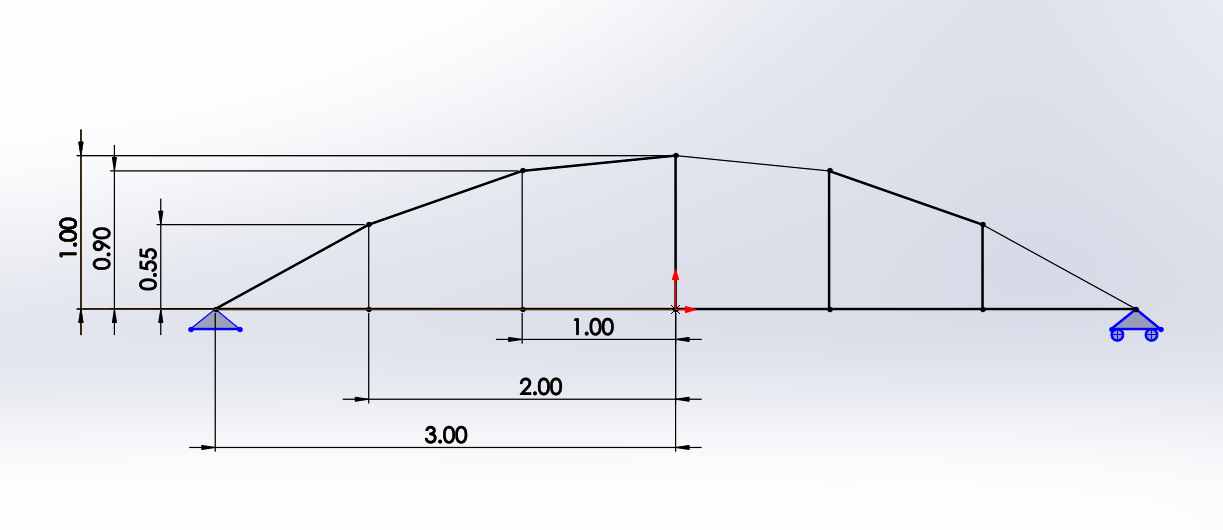
**João Vitor Sanches 9833704**

Em conjunto com Victor Chacon Codesseira 9833711

**Aula 3 – Dinâmica de Treliças**

Com o desenvolvimento anterior do código em matlab, é possível fazer a análise estática da ponte de pedestres, cujas dimensões são:

****

Adotou-se área da seção transversal 0.0015 m2, módulo de elasticidade 206 GPa e massa específica 7800 Kg/m3.

Aplicando 50KN no ponto central superior, obtém-se os resultados apresentados abaixo



Estrutura como modelada

Arquivo de entrada utilizado:

#HEADER

Análise estática - ponte de pedestres

#DYNAMIC

0

#NODES

0 1

0 0

1 0.9

1 0

2 0.55

2 0

3 0

-1 0.9

-1 0

-2 0.55

-2 0

-3 0

#ELEMENTS

1 2 0.0015 206843000000 7800

1 4 0.0015 206843000000 7800

3 4 0.0015 206843000000 7800

3 6 0.0015 206843000000 7800

5 6 0.0015 206843000000 7800

1 9 0.0015 206843000000 7800

8 9 0.0015 206843000000 7800

8 11 0.0015 206843000000 7800

10 11 0.0015 206843000000 7800

1 3 0.0015 206843000000 7800

3 5 0.0015 206843000000 7800

5 7 0.0015 206843000000 7800

7 6 0.0015 206843000000 7800

6 4 0.0015 206843000000 7800

2 4 0.0015 206843000000 7800

9 2 0.0015 206843000000 7800

9 11 0.0015 206843000000 7800

11 12 0.0015 206843000000 7800

12 10 0.0015 206843000000 7800

10 8 0.0015 206843000000 7800

8 1 0.0015 206843000000 7800

#LOADS

@1

0 -50000

#CONSTRAINTS

@12

0 0

@7

u 0

Saída do programa:

Displacement on node 1 is 0.00056729 in x and -0.0027257 in y

Displacement on node 2 is 0.00056729 in x and -0.0027257 in y

Displacement on node 3 is 0.00043308 in x and -0.0022503 in y

Displacement on node 4 is 0.00080902 in x and -0.0023067 in y

Displacement on node 5 is 0.00051115 in x and -0.0015295 in y

Displacement on node 6 is 0.00098808 in x and -0.0015456 in y

Displacement on node 7 is 0.0011346 in x and 0 in y

Displacement on node 8 is 0.0007015 in x and -0.0022503 in y

Displacement on node 9 is 0.00032556 in x and -0.0023067 in y

Displacement on node 10 is 0.00062343 in x and -0.0015295 in y

Displacement on node 11 is 0.0001465 in x and -0.0015456 in y

Displacement on node 12 is 0 in x and 0 in y

Stress on element 1 is 0 MPa

Stress on element 2 is 18.3211 MPa

Stress on element 3 is -12.981 MPa

Stress on element 4 is 9.0139 MPa

Stress on element 5 is -6.1384 MPa

Stress on element 6 is 18.3211 MPa

Stress on element 7 is -12.981 MPa

Stress on element 8 is 9.0139 MPa

Stress on element 9 is -6.1384 MPa

Stress on element 10 is 37.2001 MPa

Stress on element 11 is 32.0595 MPa

Stress on element 12 is 34.3702 MPa

Stress on element 13 is -30.55 MPa

Stress on element 14 is -37.0969 MPa

Stress on element 15 is -50.0182 MPa

Stress on element 16 is -50.0182 MPa

Stress on element 17 is -37.0969 MPa

Stress on element 18 is -30.55 MPa

Stress on element 19 is 34.3702 MPa

Stress on element 20 is 32.0595 MPa

Stress on element 21 is 37.2001 Mpa

Ilustração da estrutura deformada:



(Escala de deformação 100X)

**Análise dinâmica**

O programa foi alterado para lidar com problemas dinâmicos. Para o exercício 1, o arquivo de entrada definido foi:

#HEADER

Exercicio 1 - dinamico

Dynamic values are zero, unless defined

#DYNAMIC

1

#TIMESTEP

0.0001

#SIMTIME

1

#NODES

0 0

-0.508 0

#ELEMENTS

1 2 0.000625 206843000000 7800

#LOADS

@2

0 0

450 0

#CONSTRAINTS

@1

0 0

@2

u 0

#INITIALDISP

#INITIALVEL

#INITIALACCEL

Como as condições iniciais são nulas, as seções foram deixadas vazias. O degrau na força é definido descrevendo a força inicial e a do primeiro passo (que é mantido no decorrer da simulação).

Calcula-se o timestep crítico com

Com , a posição do nó é mal calculada e o gráfico aponta a não convergência da resposta:



Já com , a posição converge exatamente para o valor do equilíbrio estático:



Em <https://github.com/jvSanches/PMR5026> é possível ter acesso ao programa completo...

O código utilizado nos exercícios é listado a seguir:

**DynamicreProcessor.m**

steps = round(simtime/timestep);

dof = 2\*length(nodes);

D = zeros(steps+1, dof);

Ddot = zeros(steps+1, dof);

Dddot = zeros(steps+1, dof);

F = zeros(steps+1, dof);

D(1,1:dof) = initial\_disp(1:dof);

Ddot(1,1:dof) = initial\_vel(1:dof);

Dddot(1,1:dof) = initial\_accel(1:dof);

for i=1:length(nodes)

for j=1:steps+1

if j > length(nodes(i).fx)

F(j,2\*i-1) = F(j-1,2\*i-1);

else

F(j,2\*i-1) = nodes(i).fx(j);

end

if j > length(nodes(i).fy)

F(j,2\*i) = F(j-1,2\*i);

else

F(j,2\*i) = nodes(i).fy(j);

end

end

end

%% Builds system matrices

disp('Building global stiffness matrix...');

Kglobal = zeros(2\*length(nodes));

for i=1:length(elements)

Kdist = zeros(2\*length(nodes));

[k11, k12, k22, index1, index2] = elements(i).decomposeStiffnes();

index1 = 2 \* index1 -1;

index2 = 2 \* index2 -1;

Kdist(index1,index1) = k11(1,1);

Kdist(index1,index1+1) = k11(1,2);

Kdist(index1+1,index1) = k11(2,1);

Kdist(index1+1,index1+1) = k11(2,2);

Kdist(index1,index2) = k12(1,1);

Kdist(index1,index2+1) = k12(1,2);

Kdist(index1+1,index2) = k12(2,1);

Kdist(index1+1,index2+1) = k12(2,2);

Kdist(index2,index1) = k12(1,1);

Kdist(index2,index1+1) = k12(1,2);

Kdist(index2+1,index1) = k12(2,1);

Kdist(index2+1,index1+1) = k12(2,2);

Kdist(index2,index2) = k22(1,1);

Kdist(index2,index2+1) = k22(1,2);

Kdist(index2+1,index2) = k22(2,1);

Kdist(index2+1,index2+1) = k22(2,2);

Kglobal = Kglobal + Kdist;

end

disp('Done')

disp('Building global mass matrix...');

Mglobal = zeros(2\*length(nodes));

for i=1:length(elements)

Mdist = zeros(2\*length(nodes));

[m11, m12, m22, index1, index2] = elements(i).decomposeMass();

index1 = 2 \* index1 -1;

index2 = 2 \* index2 -1;

Mdist(index1,index1) = m11(1,1);

Mdist(index1,index1+1) = m11(1,2);

Mdist(index1+1,index1) = m11(2,1);

Mdist(index1+1,index1+1) = m11(2,2);

Mdist(index1,index2) = m12(1,1);

Mdist(index1,index2+1) = m12(1,2);

Mdist(index1+1,index2) = m12(2,1);

Mdist(index1+1,index2+1) = m12(2,2);

Mdist(index2,index1) = m12(1,1);

Mdist(index2,index1+1) = m12(1,2);

Mdist(index2+1,index1) = m12(2,1);

Mdist(index2+1,index1+1) = m12(2,2);

Mdist(index2,index2) = m22(1,1);

Mdist(index2,index2+1) = m22(1,2);

Mdist(index2+1,index2) = m22(2,1);

Mdist(index2+1,index2+1) = m22(2,2);

Mglobal = Mglobal + Mdist;

end

disp('Done')

Cglobal = 0.0004 \* (0.3\*Mglobal + 0.03\*Kglobal);

**DunamicSolver.m**

disp('Starting Solver')

%% Reduces system with given constraits

for i=1:length(nodes)

if nodes(i).xconstrained

for j = 1:length(Kglobal)

F(:, j) = F(:, j) - Kglobal(j,2\*i-1) \* nodes(i).dx;

Kglobal(2\*i-1,j) = 0;

Kglobal(j,2\*i-1) = 0;

Mglobal(2\*i-1,j) = 0;

Mglobal(j,2\*i-1) = 0;

end

F(:,2\*i-1) = nodes(i).dx;

Kglobal(2\*i-1, 2\*i-1) = 1;

Mglobal(2\*i-1, 2\*i-1) = 1;

end

if nodes(i).yconstrained

for j = 1:length(Kglobal)

F(:, j) = F(:, j) - Kglobal(j,2\*i) \* nodes(i).dy;

Kglobal(2\*i,j) = 0;

Kglobal(j,2\*i) = 0;

Mglobal(2\*i,j) = 0;

Mglobal(j,2\*i) = 0;

end

F(:,2\*i) = nodes(i).dy;

Kglobal(2\*i, 2\*i) = 1;

Mglobal(2\*i, 2\*i) = 1;

end

end

%Cglobal = 1\*Mglobal;

M\_inv = (Mglobal./timestep^2)^-1;

MC1 = (2/timestep^2)\*Mglobal - (1/timestep)\*Cglobal;

MC2 = (1/timestep^2)\*Mglobal - (1/timestep)\*Cglobal;

D0 = D(1,:) - timestep\*Ddot(1,:) + (timestep^2/2)\*Dddot(1,:);

D(2, :) = M\_inv\*(F(1,:)' - Kglobal\*D(1,:)' + MC1\*D(1,:)' - MC2\*D0');

for i=2:steps-1

Dn = D(i,:)';

Dn\_1 = D(i-1,:)';

R\_int = Kglobal\*Dn;

Fn = F(i,:)';

Dn\_new = M\_inv\*(Fn- R\_int + MC1\*Dn - MC2\*Dn\_1);

D(i+1,:) = Dn\_new;

end

disp('Done')

%% Plot dos valores desejados

% limit = 10000; figure; plot(0:timestep:(limit-1)\*timestep,D(1:limit,2))

T2 = 0.0128;

t = 0:timestep:2\*T2;

figure; plot(t,D(1:length(t),4))

**truss.m**

classdef truss

%UNTITLED truss element

% Detailed explanation goes here

properties

n1,

n2,

A,

E,

L,

m,

l,

K,

M,

ro,

end

methods

function obj = truss(node1, node2, area, e\_modulus, density)

%UNTITLED Construct an instance of this class

% Detailed explanation goes here

obj.n1 = node1;

obj.n2 = node2;

obj.A = area;

obj.E = e\_modulus;

obj.ro = density;

obj.L = sqrt((node2.x - node1.x)^2 + (node2.y - node1.y)^2);

l = (node2.x-node1.x) / obj.L;

m = (node2.y-node1.y) / obj.L;

obj.K = (obj.E\*obj.A/obj.L)\*[l\*l l\*m 0 -l\*l -l\*m 0;

l\*m m\*m 0 -l\*m -m\*m 0;

0 0 1 0 0 0;

-l\*l -l\*m 0 l\*l l\*m 0;

-l\*m -m\*m 0 l\*m m\*m 0;

0 0 0 0 0 1];

obj.M = (obj.A \* obj.ro \* obj.L/6)\*[2\*l\*l 2\*l\*m l\*l l\*m;

2\*l\*m 2\*m\*m l\*m m\*m;

l\*l l\*m 2\*l\*l 2\*l\*m;

l\*m m\*m 2\*l\*m 2\*m\*m];

end

function [k11, k12, k22, index1, index2] = decomposeStiffnes(obj)

%METHOD1 Summary of this method goes here

% Detailed explanation goes here

k11 = obj.K(1:3,1:3);

k12 = obj.K(1:3,4:6);

k22 = obj.K(4:6,4:6);

index1 = obj.n1.index;

index2 = obj.n2.index;

end

function [m11, m12, m22, index1, index2] = decomposeMass(obj)

%METHOD1 Summary of this method goes here

% Detailed explanation goes here

m11 = obj.M(1:2,1:2);

m12 = obj.M(1:2,3:4);

m22 = obj.M(3:4,3:4);

index1 = obj.n1.index;

index2 = obj.n2.index;

end

function tension = getTension(obj)

nx1 = obj.n1.x+obj.n1.dx;

ny1 = obj.n1.y+obj.n1.dy;

nx2 = obj.n2.x+obj.n2.dx;

ny2 = obj.n2.y+obj.n2.dy;

L\_strain = sqrt((nx2 - nx1)^2 + (ny2 - ny1)^2);

tension = (obj.L-L\_strain)/obj.L \* obj.E;

end

end

end

**node.m**

classdef node < handle

%NODE Node for mef

% Detailed explanation goes here

properties

index,

x,

y,

theta,

fx,

fy,

mo,

dx,

dy,

dtheta,

xconstrained,

yconstrained,

thetaconstrained,

end

methods

function obj = node(n\_x,n\_y)

%NODE Construct an instance of this class

% Detailed explanation goes here

obj.x = n\_x;

obj.y = n\_y;

obj.theta = 0;

obj.fx = 0;

obj.fy = 0;

obj.mo = 0;

obj.dx = 0;

obj.dy = 0;

obj.xconstrained = 0;

obj.yconstrained = 0;

obj.thetaconstrained = 0;

end

function setLoad(obj, nfx, nfy, nmo)

obj.fx = obj.fx + nfx;

obj.fy = obj.fy + nfy;

obj.mo = obj.mo + nmo;

end

function setIndex(obj, ind)

obj.index = ind;

end

function constrain(obj, c\_x, c\_y, c\_theta)

if c\_x ~= 'u'

c\_x = str2num(c\_x);

obj.xconstrained = 1;

obj.dx = c\_x;

end

if c\_y ~= 'u'

c\_y = str2num(c\_y);

obj.yconstrained = 1;

obj.dy = c\_y;

end

if c\_theta ~= 'u'

c\_theta = str2num(c\_theta);

obj.thetaconstrained = 1;

obj.dtheta = c\_theta;

end

end

function setDeltas(obj, ndx, ndy)

obj.dx = ndx;

obj.dy = ndy;

end

end

end

plotter.m

figure

hold on;

grid on;

scale = 100;

%%Display nodes

max\_x = -inf;

min\_x = inf;

max\_y = -inf;

min\_y = inf;

for i=1:length(nodes)

nx = nodes(i).x+scale\*nodes(i).dx;

ny = nodes(i).y+scale\*nodes(i).dy;

max\_x = max(max\_x, nx);

max\_y = max(max\_y, ny);

min\_x = min(min\_x, nx);

min\_y = min(min\_y, ny);

if nodes(i).xconstrained

if nodes(i).yconstrained

scatter(nx, ny, 's', 'filled', 'red');

else

scatter(nx, ny, '>', 'filled', 'red');

end

else

if nodes(i).yconstrained

scatter(nx, ny,'^' ,'red');

else

scatter(nx, ny, 'red');

end

end

text(nx+0.1, ny+0.1, string(i));

end

offset = 1;

axis([min\_x-offset max\_x+offset min\_y-offset max\_y+offset])

%display trusses

for i=1:length(elements)

line\_x = [elements(i).n1.x+scale\*elements(i).n1.dx, elements(i).n2.x+scale\*elements(i).n2.dx];

line\_y = [elements(i).n1.y+scale\*elements(i).n1.dy, elements(i).n2.y+scale\*elements(i).n2.dy];

line(line\_x,line\_y)

end