João Vitor Sanches 9833704

Em conjunto com Victor Chacon Codesseira 9833711

Aula 5 – Análise Modal

Com a estrutura já modelada e as matrizes calculadas, az sentido completar o programa existente para o cálculo das frequências naturais. Para fins de comparação, analisou-se uma viga de seção quadrada com 1m x 0.05m x 0.05m, engastada em sua extremidade. O estudo foi feito com o programa desenvolvido (variado a quantidade de elementos), analiticamente e com um software comercial de elementos finitos. Os resultados são apresentados na tabela a seguir:

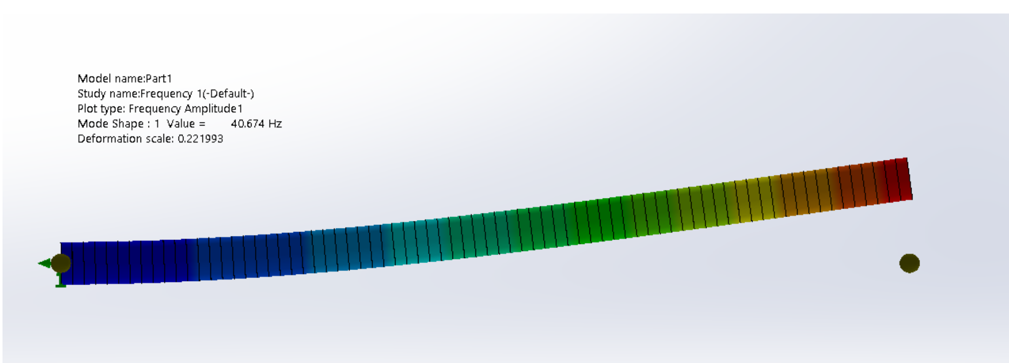
Método		**Modo de vibrar** (freqs. em Hz)		
		1o	2o	3o
	Analítico	40.90	256.34	722.42
Solidworks	Sólido 3D	42.88	256.47	729.56
	Elemento de Viga	40.67	253.39	702.92
Programa	1 elemento	41.1	405.05	1396
	2 elementos	40.93	258.53	874.39
	3 elementos	40.91	257.19	726.74
	1000 elementos	40.91	256.34	717.81

Analiticamente, as frequências (em rad/s) foram obtidas com:

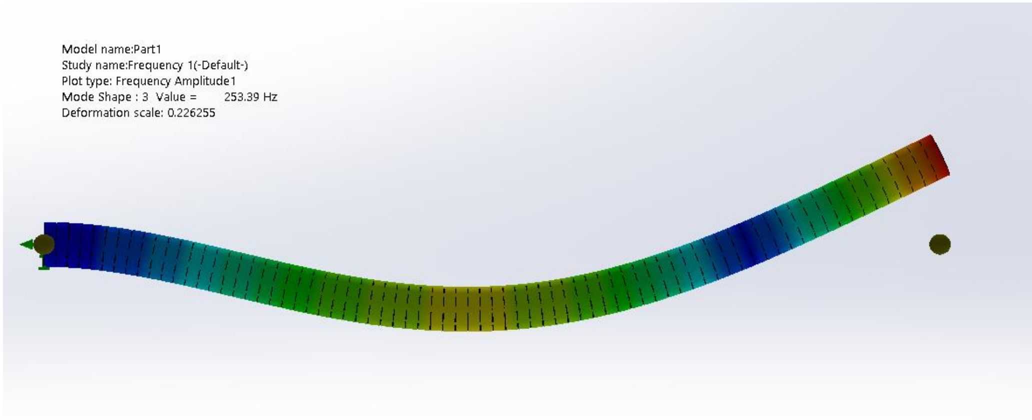
$$\omega_{n1} = \alpha^2 \sqrt{\frac{E\ I}{\rho\ A\ L^4}}$$

Com $\alpha = 1.875$; $\alpha = 4.694$; $\alpha = 7.88$

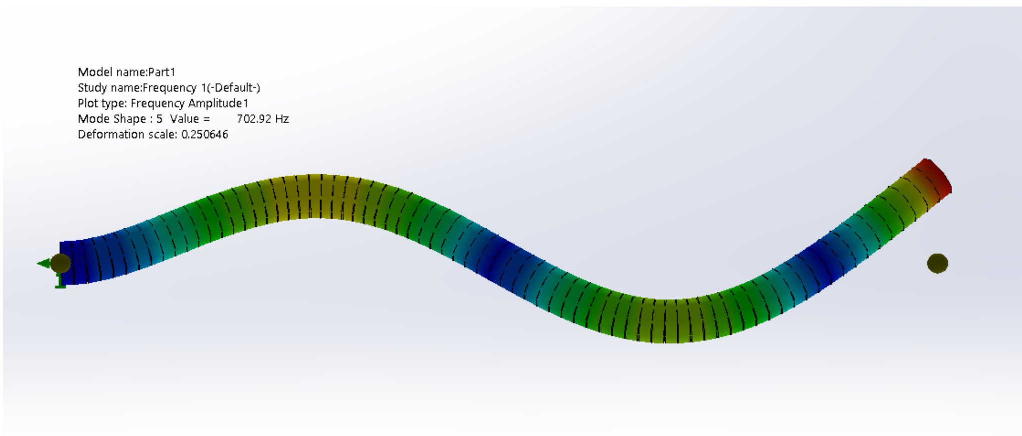
Com o Solidworks, a estrutura em questão foi modelada e simulada, tanto com elementos de viga como com elementos tridimensionais. Algumas imagens do resultado são apresentadas abaixo:



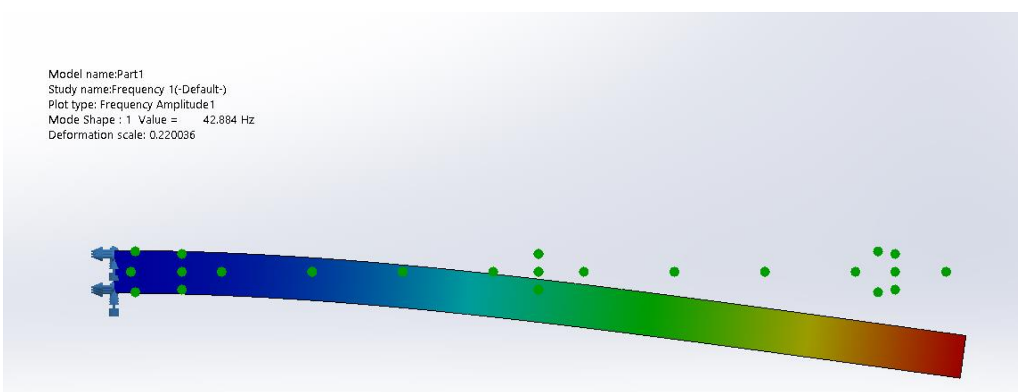
(Simulação com elementos de viga. Primeiro modo de vibrar)



(Simulação com elementos de viga. Segundo modo de vibrar)



(Simulação com elementos de viga. Terceiro modo de vibrar)



(Simulação com elementos tridimensionais. Primeiro modo de vibrar)

Para o uso do programa desenvolvido, algumas alterações foram feitas de modo que a execução de simulações com mais elementos fosse mais rápida. O programa completo está disponível em https://github.com/jvSanches/PMR5026. Algumas seções estão listadas abaixo:

Cantiliever_beam_modal.py

Esse programa foi utilizado para criar arquivos de entrada com vários elementos para a análise modal

```python
import time
import subprocess
import os

# Propriedades da viga
E = 200e9
A = 0.0025
I = 5.21e-7
p = 7800

# Número total de elementos
n_el = int(input("Digite a quantidade de elementos: "))

if n_el < 1:
    raise AttributeError

print("\nStarting file assembly")
start_time = time.time()

# Criando arquvio de entrada
filename = f"balanco_modal_{n_el}.txt"
with open(filename, 'w') as f:
    f.write("#HEADER\n")
    f.write(f"Analise modal de viga em balanco com {n_el} elemento(s)\n")

    f.write("\n#MODAL\n")
    f.write("1\n")

    f.write("\n#NODES\n")
    for x in range(0, n_el+1):
        f.write(f"{x/n_el} 0\n")

    f.write("\n#ELEMENTS\n")
    for x in range(1, n_el+1):
        f.write(f"b {x} {x+1} {A} {E} {I} {p}\n")

    f.write("\n#CONSTRAINTS\n")
    f.write("@1\n")
    f.write("0 0 0\n\n")
```

```python
middle_time = time.time()
print(f"Took {middle_time - start_time} seconds\n")
print("Starting matlab job")

# Rodando matlab
subprocess.run(["matlab", "-batch", f"mefController('{filename}')"])
print(f"\nTook {time.time() - middle_time} seconds")

# Deletando arquivo gerado
os.remove(filename)

input()
```

loader.m

```matlab
%% Open and read the file
disp("Reading file")

fid = fopen(filename);
Text = textscan(fid,'%s','delimiter','\n');
Text = Text{1};
fclose(fid);

disp("Interpreting file")

header_lines =[];
dynamic_mode = 0;
modal_analysis = 0;
nodes = [];
elements = [];

reading_load_on_node = 0;
reading_load_on_element = 0;
reading_constrain_on_node = 0;
reading_disp_on_node = 0;
reading_vel_on_node = 0;
reading_acc_on_node = 0;
part_load_data = [];

initial_disp = [];
initial_vel = [];
initial_acc = [];

state = "";
for i = 1:length(Text)
    line = Text{i};

    if isempty(line)
        if reading_load_on_node

nodes(reading_load_on_node).setLoad(part_load_data(:,1),part_load_data(:,2),p
art_load_data(:,3));
```

```
            reading_load_on_node = 0;
        end
        if reading_load_on_element

elements(reading_load_on_element).setPressure(part_load_data(:,1),part_load_d
ata(:,2));
            reading_load_on_element = 0;
        end
        if reading_constrain_on_node

nodes(reading_constrain_on_node).constrain(constrain_data(1),constrain_data(2
),constrain_data(3));
            reading_constrain_on_node = 0;
        end
        if reading_disp_on_node
            initial_disp(:,reading_disp_on_node) = transpose(disp_data);
            reading_disp_on_node = 0;
        end
        if reading_vel_on_node
            initial_vel(:,reading_vel_on_node) = transpose(disp_data);
            reading_vel_on_node = 0;
        end
        if reading_acc_on_node
            initial_acc(:,reading_acc_on_node) = transpose(disp_data);
            reading_acc_on_node = 0;
        end
        state = "";
        continue;
    elseif line(1) == "#"
        state = line;
        continue;
    end


    switch state
        case "#HEADER"
            header_lines = [header_lines line newline];
        case "#DYNAMIC"
            dynamic_mode = sscanf(line, '%i') == 1;
            timestep = 0;
            simtime = 0;
        case "#MODAL"
            modal_analysis = sscanf(line, '%i') == 1;
        case '#TIMESTEP'
            timestep = sscanf(line, '%f');
        case '#SIMTIME'
            simtime = sscanf(line, '%f');
        case '#NODES'
            a = sscanf(line, '%f %f', [1 2]);
            new_node = node(a(1), a(2));
            nodes = [nodes new_node];
            new_node.setIndex(length(nodes));
        case '#ELEMENTS'
            a = sscanf(line, '%s %f %f %f %f %f %f', [1 7]);
            if a(1) == 't'
                new_element = truss(nodes(a(2)), nodes(a(3)), a(4), a(5),
a(6));
```

```
            elseif a(1) == 'b'
                new_element = beam(nodes(a(2)), nodes(a(3)), a(4), a(5),
a(6), a(7));
            end
            elements = [elements new_element];
        case '#LOADS'
            if a(1)=='@'
                if reading_load_on_node

nodes(reading_load_on_node).setLoad(part_load_data(:,1),part_load_data(:,2),p
art_load_data(:,3));
                end
                reading_load_on_node = str2num(a(2:end));
            else
                a = sscanf(line, '%f %f %f', [1 3]);
                part_load_data = [part_load_data ; a];
            end
        case '#PRESSURES'
            if line(1)=='@'
                if reading_load_on_element

elements(reading_load_on_element).setPressure(part_load_data(:,1),part_load_d
ata(:,2));
                end
                reading_load_on_element = str2num(line(2:end));
            else
                a = sscanf(line, '%f %f %f', [1 3]);
                part_load_data = [part_load_data ; a];
            end
        case '#CONSTRAINTS'
            if line(1)=='@'
                if reading_constrain_on_node

nodes(reading_constrain_on_node).constrain(constrain_data(1),constrain_data(2
),constrain_data(3));
                end
                reading_constrain_on_node = str2num(line(2:end));
            else
                constrain_data = sscanf(line, '%s %s %s', [1 3]);
            end
        case '#INITIALDISP'
            if initial_disp.isempty
                initial_disp = zeros(2,length(nodes));
            end
            if line(1)=='@'
                if reading_disp_on_node
                   initial_disp(:,reading_disp_on_node) =
transpose(disp_data);
                end
                reading_disp_on_node = str2num(line(2:end));
            else
                disp_data = sscanf(line, '%s %s %s', [1 3]);
            end
        case '#INITIALVEL'
            if initial_vel.isempty
                initial_vel = zeros(2,length(nodes));
            end
```

```
            if line(1)=='@'
                if reading_vel_on_node
                   initial_vel(:,reading_vel_on_node) = transpose(vel_data);
                end
                reading_vel_on_node = str2num(line(2:end));
            else
                vel_data = sscanf(line, '%s %s %s', [1 3]);
            end
        case '#INITIALACCEL'
            if initial_acc.isempty
                initial_acc = zeros(2,length(nodes));
            end
            if line(1)=='@'
                if reading_acc_on_node
                   initial_acc(:,reading_acc_on_node) = transpose(acc_data);
                end
                reading_acc_on_node = str2num(line(2:end));
            else
                disp_acc = sscanf(line, '%s %s %s', [1 3]);
            end
    end
end

%% Display loaded info
%clc
disp([filename ' loaded'])
disp('----------------------------------------------------')
disp(header_lines)
disp('----------------------------------------------------')
disp([num2str(length(nodes)) ' nodes ' ])
disp([num2str(length(elements)) ' elements ' ])
disp('Loading Done');

%% Clear workspace
clear ans a fid filename header_lines i j new_element new_node part_load_data
reading_load_on_element reading_load_on_node reading_on_node Text
```

preProcessor.m

```
disp('Building global stiffness matrix...');
Kglobal = sparse(3*length(nodes), 3*length(nodes));
for i=1:length(elements)
    [k11, k12, k22, index1, index2] = elements(i).decomposeStiffnes();
    k21 = transpose(k12);
    index1 = 3 * (index1 - 1);
    index2 = 3 * (index2 - 1);

    for j = 1:3
        for k = 1:3
            Kglobal(index1 + j, index1 + k) = Kglobal(index1 + j, index1 + k)
+ k11(j, k);
            Kglobal(index1 + j, index2 + k) = Kglobal(index1 + j, index2 + k)
+ k12(j, k);
```

```
            Kglobal(index2 + j, index1 + k) = Kglobal(index2 + j, index1 + k)
+ k21(j, k);
            Kglobal(index2 + j, index2 + k) = Kglobal(index2 + j, index2 + k)
+ k22(j, k);
        end
    end

end
disp('Done')

if modal_analysis
    disp('Building global mass matrix...');
    Mglobal = sparse(3*length(nodes), 3*length(nodes));
    for i=1:length(elements)
        [m11, m12, m22, index1, index2] = elements(i).decomposeMass();
        m21 = transpose(m12);

        index1 = 3 * (index1 - 1);
        index2 = 3 * (index2 - 1);

        for j = 1:3
            for k = 1:3
                Mglobal(index1 + j, index1 + k) = Mglobal(index1 + j, index1
+ k) + m11(j, k);
                Mglobal(index1 + j, index2 + k) = Mglobal(index1 + j, index2
+ k) + m12(j, k);
                Mglobal(index2 + j, index1 + k) = Mglobal(index2 + j, index1
+ k) + m21(j,k);
                Mglobal(index2 + j, index2 + k) = Mglobal(index2 + j, index2
+ k) + m22(j,k);
            end
        end
    end
    disp('Done')
end

F = sparse(length(nodes),1);
for i=1:length(nodes)
    F(3*i - 2) = nodes(i).fx;
    F(3*i-1) = nodes(i).fy;
    F(3*i) = nodes(i).mo;
end

clear i index1 index2 k11 k12 k21 k22 m11 m12 m21 m22 Kdist Mdist
```

solver.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,3*i-2) * nodes(i).dx;
```

```
                Kglobal(3*i-2,j) = 0;
                Kglobal(j,3*i-2) = 0;
            end
            F(3*i-2) = nodes(i).dx;
            Kglobal(3*i-2, 3*i-2) = 1;
        end
        if nodes(i).yconstrained
            for j = 1:length(Kglobal)
                F(j) = F(j) - Kglobal(j,3*i-1) * nodes(i).dy;
                Kglobal(3*i-1,j) = 0;
                Kglobal(j,3*i-1) = 0;
            end
            F(3*i-1) = nodes(i).dy;
            Kglobal(3*i-1, 3*i-1) = 1;

        end
        if nodes(i).thetaconstrained
            for j = 1:length(Kglobal)
                F(j) = F(j) - Kglobal(j,3*i) * nodes(i).dtheta;
                Kglobal(3*i,j) = 0;
                Kglobal(j,3*i) = 0;
            end
            F(3*i) = nodes(i).dtheta;
            Kglobal(3*i, 3*i) = 1;

        end
    end

    if modal_analysis
        for i=1:length(nodes)
            if nodes(i).xconstrained
                for j = 1:length(Mglobal)
                    Mglobal(3*i-2,j) = 0;
                    Mglobal(j,3*i-2) = 0;
                end
                Mglobal(3*i-2, 3*i-2) = 1;
            end
            if nodes(i).yconstrained
                for j = 1:length(Mglobal)
                    Mglobal(3*i-1,j) = 0;
                    Mglobal(j,3*i-1) = 0;
                end
                Mglobal(3*i-1, 3*i-1) = 1;

            end
            if nodes(i).thetaconstrained
                for j = 1:length(Mglobal)
                    Mglobal(3*i,j) = 0;
                    Mglobal(j,3*i) = 0;
                end
                Mglobal(3*i, 3*i) = 1;

            end
        end
        return
    end
```

```
D = linsolve(full(Kglobal),full(F));

for i=1:length(nodes)
    nodes(i).dx = D(3*i-2);
    nodes(i).dy = D(3*i-1);
    nodes(i).dtheta = D(3*i);
end

clear i j
```

postProcessor.m

```
    scale_deform = 100;
scale_normal = 5e-6;
scale_shear = 4e-6;
scale_moment = 5e-6;

disp("Calculating results...");

if modal_analysis
    v = eigs(Mglobal\Kglobal, 9, 0, 'maxit', 1e12);
    v = sqrt(v)/(2*pi);
    v = sort(v);
    v = v(4:9);
    disp("Vibrating modes(Hz)")

   disp(num2str(v,8));
   return
end

scatterNodes(nodes, elements, false, true, 0);
title("Não deformado")

fig_u = scatterNodes(nodes, elements, true, false, scale_deform);
title("Deslocamentos")
for i = 1:length(nodes)
    txt = sprintf("dx: %f\ndy: %f\ndr: %f", [nodes(i).dx, nodes(i).dy,
nodes(i).dtheta]);
    text(nodes(i).x + nodes(i).dx*scale_deform - 0.4, nodes(i).y +
nodes(i).dy*scale_deform + 0.3, txt);
end

fig_N = scatterNodes(nodes, elements, false, true, 0);
title("Normal")
fig_V = scatterNodes(nodes, elements, false, true, 0);
title("Cortante")
fig_M = scatterNodes(nodes, elements, false, true, 0);
title("Momento fletor")

for i=1:length(elements)
```

```
    elements(i) = elements(i).calculateStress();
    el = elements(i);
    x = linspace(0, el.L, 101);
    T = [el.l, el.m; -el.m, el.l]^-1;

    %% Display displacement
    figure(fig_u)

    u = x + scale_deform*(eval(subs(el.elasticLineX))-x);
    v = scale_deform*(eval(subs(el.elasticLineY)));
    points = T*[u;v] + [el.n1.x; el.n1.y];

    plot(points(1,:), points(2,:), 'Color', [0 0.447 0.741]);


    %% Display normal
    plotDiagram(fig_N, el, el.Normal, x, T, scale_normal);

    %% Display shear
    plotDiagram(fig_V, el, el.Shear, x, T, scale_shear);

    %% Display moment
    plotDiagram(fig_M, el, el.Moment, x, T, scale_moment);
end

%% Function for scatter of nodes
function fig = scatterNodes(nodes, elements, displaced, lines, scale)
    fig = figure();
    hold on;
    grid on;

    max_x = -inf;
    min_x = inf;
    max_y = -inf;
    min_y = inf;
    for i=1:length(nodes)
        nx = nodes(i).x;
        ny = nodes(i).y;
        if displaced
            nx = nx + scale*nodes(i).dx;
            ny = ny + scale*nodes(i).dy;
        end

        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
    end

    offset = 1;
    axis equal
    axis([min_x-offset max_x+offset min_y-offset max_y+offset])


    if lines
        for i=1:length(elements)
            if displaced
                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];
            else
                line_x = [elements(i).n1.x, elements(i).n2.x];
                line_y = [elements(i).n1.y, elements(i).n2.y];
            end

            line(line_x,line_y)
        end
    end
end

function plotDiagram(fig, element, equation, divisions, rot, scale)
    offset_x = -0.15;
    offset_y = 0.05;
    figure(fig);
    syms x
    if (isnumeric(equation) && abs(equation) < 1e-6) || (~isnumeric(equation) 
&& isnumeric(eval(equation)) && abs(eval(equation)) < 1e-6)
        return
    end
    data = eval(subs(equation, x, divisions));
    if abs(data) < 1e-6
        return
    end

    points = rot*[divisions;scale*data] + [element.n1.x; element.n1.y];

    plot(points(1,:), points(2,:), 'r');
    line([points(1,1), element.n1.x], [points(2,1), element.n1.y], 'Color', 
'r');
    line([points(1,end), element.n2.x], [points(2,end), element.n2.y], 
'Color', 'r');

    text(points(1,1) + offset_x, points(2,1) + offset_y, string(data(1)));
    text(points(1,end) + offset_x, points(2,end) + offset_y, 
string(data(end)));
end
```

mefController.m

```
function mefController(filename)
    close all

    if nargin < 1
        filename = "balanco_modal_1.txt";
    end

    run loader.m
    %run plotter.m
    if dynamic_mode
        run dynamicPreProcessor.m
        run dynamicSolver.m
        %run dynamicPlotter.m
    else
        run preProcessor.m
        run solver.m
        %run plotter.m
        run postProcessor.m
    end
end
```