**João Vitor Sanches 9833704**

Em conjunto com Victor Chacon Codesseira 9833711

**Aula 5 – Análise Modal**

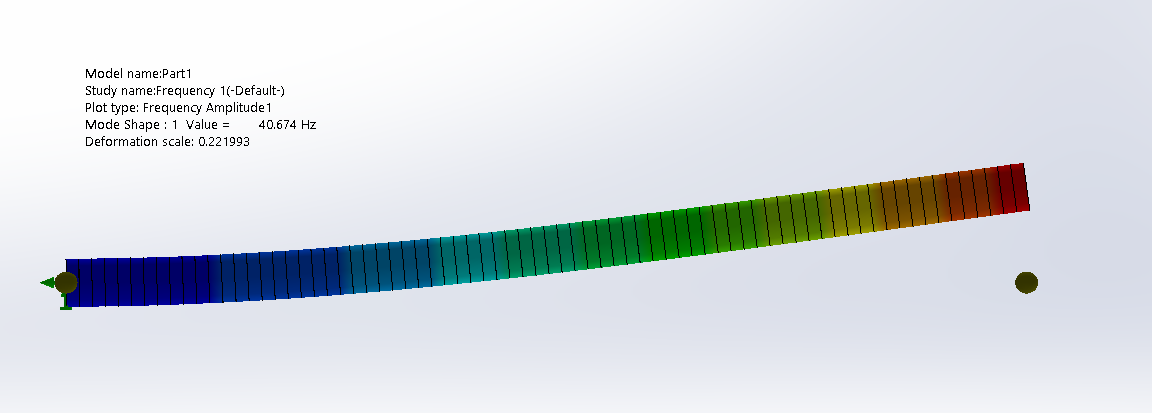
Com a estrutura já modelada e as matrizes calculadas, az sentido completar o programa existente para o cálculo das frequências naturais. Para fins de comparação, analisou-se uma viga de seção quadrada com 1m x 0.05m x 0.05m, engastada em sua extremidade. O estudo foi feito com o programa desenvolvido (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 |

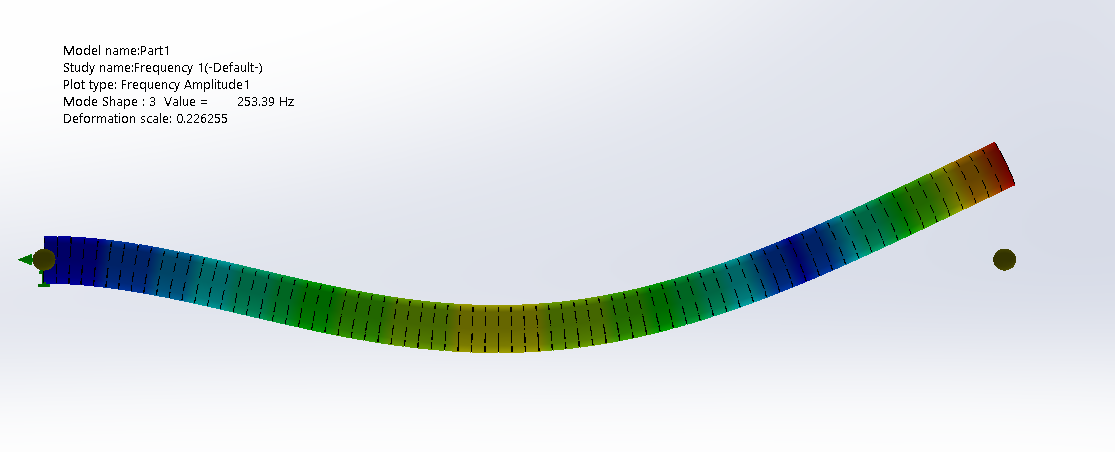
Analiticamente, as frequências (em rad/s) foram obtidas com:

Com

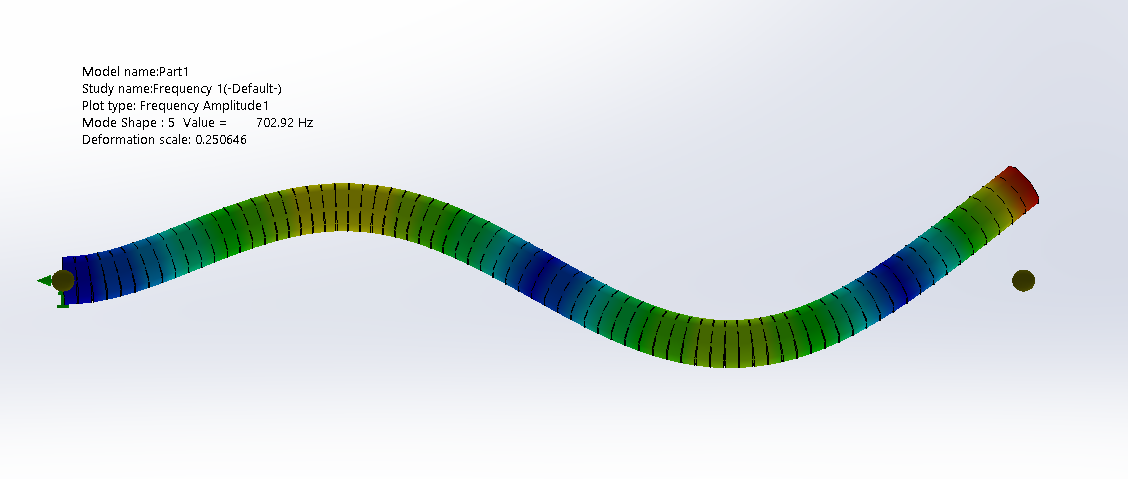
Com o Solidworks, a estrutura em questão foi modelada e simulada, tanto com elementos de viga como com elementos tridimensionais. Algumas imagens do resultado são apresentadas abaixo:



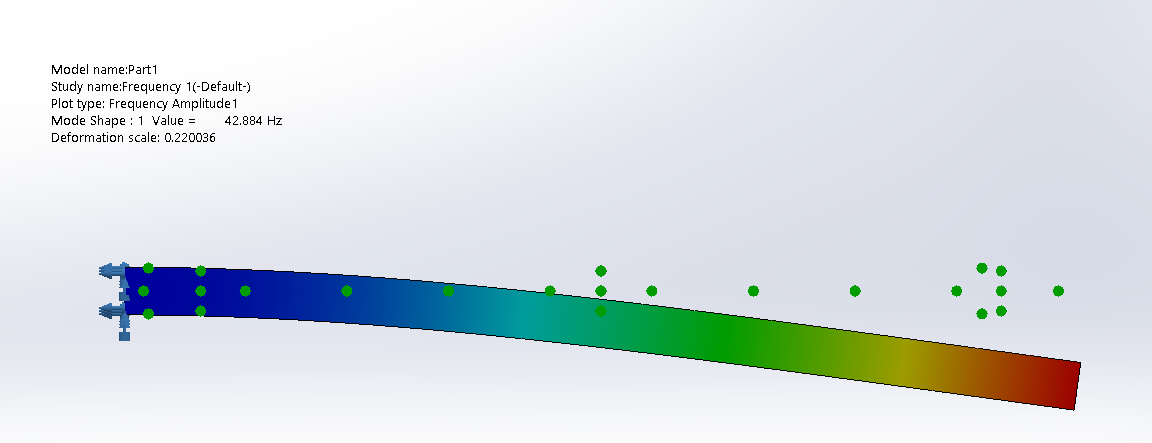
(Simulação com elementos de viga. Primeiro modo de vibrar)



(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

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")

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**

%% 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),part\_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\_data(:,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),part\_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\_data(:,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