# Breve relatório com os resultados obtidos durante o modulo-4

### Aluno: Ruben Esteche Araújo

### CPF: 109.429.904-98

Dando continuidade ao que foi abordado no módulo passado, os problemas vistos nesse módulo foram os mesmos, todavia, a abordagem foi diferente em alguns aspectos, na intenção de aprendermos uma forma de resolução que pode eventualmente vir a ser mais eficiente quanto a resolução de problemas em que haja necessidade (ou praticidade maior) em poder observar a evolução das interações do programa.

Ainda utilizando um método simplético de segunda ordem (EULER\_CROMER), montamos em cima do programa elaborado na aula passada arrays para salvar nossas variáveis dentro das interações; dessa forma, podemos então desrever a evolução temporal de diversos estados simultâneamente a execução o programa. O programa que desenvolvi pode ser conferido logo a seguir:

**Programa Arrays: C/C++**

#include<stdlib.h>

#include<math.h>

#include<stdio.h>

int main(){

//Definindo variáveis de movimento

int n,i,p, r, g=10,j,q,k=10,c;

double l=10,t, dt=0.001, gama=0.9, w;

//Arrays

double x[10], vx[10], w0[10], teta[10];

//matriz de condições iniciais

r=0;

for(q=0;q<k;q++){

for(c=0;c<k;c++){

vx[r]= 1 + (2\*q);

x[r]=1 + (2\*c);

r= r+1;

}

}

r=0;

for(q=0;q<k;q++){

for(c=0;c<r;c++){

w0[r]= 1 + (2\*q);

teta[r]=1 + (2\*c);

r= r+1;

}

}

//quantidade de interações

n=50000;

//Seleção de qual programa executar

printf("Selecione qual problema voce deseja resolver:\n01.Pendulo simples(1)\n02.Oscilador(2)\n");

scanf("%d",&p);

if(p==1){

//arquivo

FILE \*Rx\_pendulo;

Rx\_pendulo = fopen("posição (pendulo).txt","w+");

//Variáveis do pêndulo

//printf("Digite velocidade inicial:\n");

//scanf("%lf",&w0);

printf("Digite o angulo inicial:\n");

scanf("%lf",&teta);

//interações

for(i=0;i<n;i++){

for(r=0;r<10;r++){

fprintf(Rx\_pendulo,"%lf %lf", w0[r], teta[r]);

w0[r] = w0[r] -((g/l)\*sin(teta[r]))\*dt;

teta[r] = teta[r] + w0[r]\*dt;

t= t+dt;

}

fprintf(Rx\_pendulo,"\n");

}

//fechando o arquivo

fclose(Rx\_pendulo);

}if(p==2){

//arquivo

FILE \*Rx\_oscilador;

Rx\_oscilador = fopen("posição (oscilador).txt","w+");

//Variáveis do oscilador

//printf("Digite velocidade inicial:\n");

//scanf("%lf",&vx);

printf("Digite a frequência angular inicial:\n");

scanf("%lf",&w);

//interações

for(i=0;i<n;i++){

for(r=0;r<10;r++){

fprintf(Rx\_oscilador,"%lf %lf", vx[r], x[r]);

vx[r] = vx[r] - (w\*w\*x[r])\*dt; //-(vx\*gama)\*dt + (cos(1\*t))\*dt; //força resistiva com gama e força motriz externa em cosseno

x[r]= x[r] + vx[r]\*dt;

t= t+dt;

}

fprintf(Rx\_oscilador,"\n");

}

//fechando o arquivo

fclose(Rx\_oscilador);

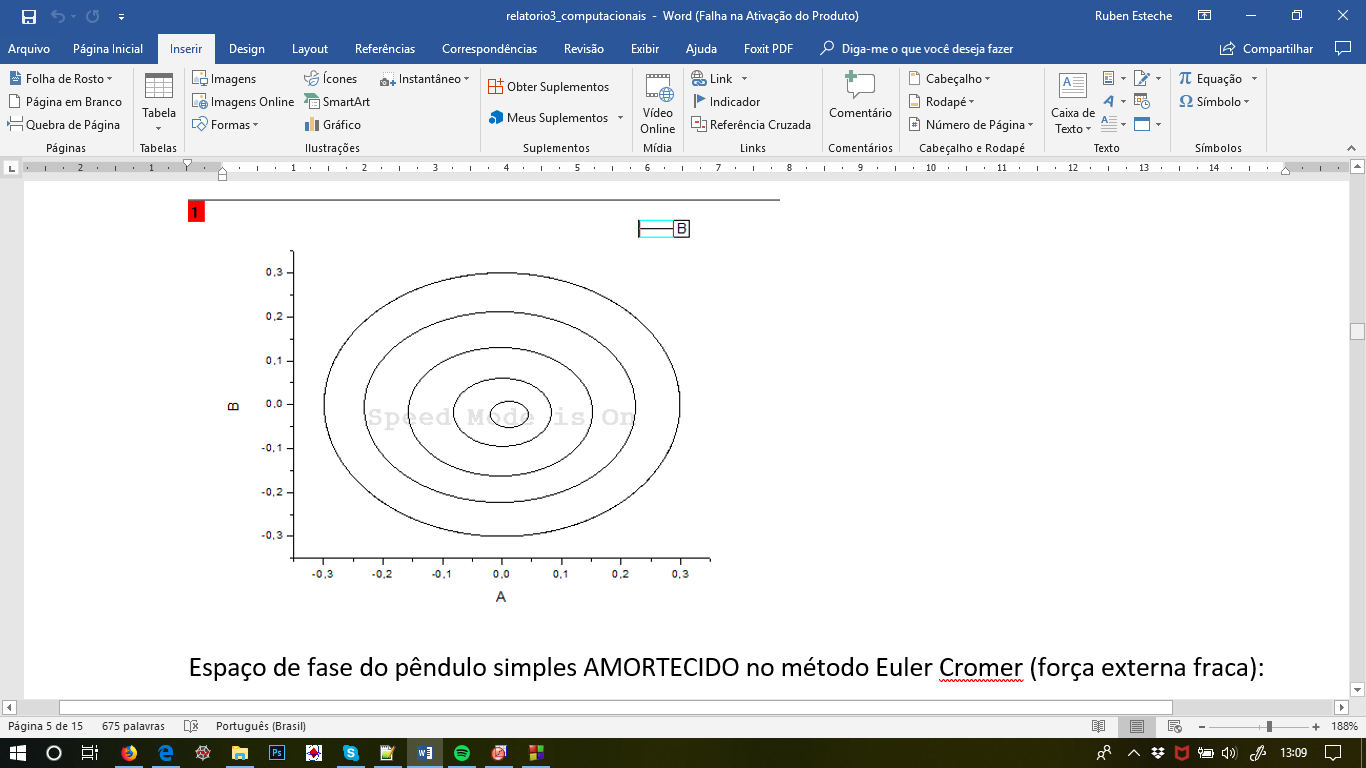
}

printf("seu arquivo foi criado com sucesso");

return 0;

}

O programa em questão se mostrou funcional. Como podemos observar no gráfico a seguir, diversos estados iniciais puderam ser simulados no estado de fase, onde os resultados batem com o resultado teórico esperado; Todavia, encontrei muita dificuldade para reorganizar o arquivo onde as interações estavam sendo salvas e com o software que estava utilizando para plotar os gráficos (em questão, o Origin) visto que cada linha do arquivo continha a informação de fase de um condição inicial diferente, e elas evoluiíam no arquivo horizontalmente, enquanto o Origin somente lê informação verticalmente, o que tornou o trabalho para plotar um gráfico simples muito custoso.



Decidi então migrar para a linguagem Python, por ser de alto nível (e, sinceramente, facilitar bastante a minha vida) e já permitir para o usuário em questão plotar gráficos diretamente da interface do compilador. Os programas que desenvolvi para simular os sistemas do pêndulo e da OHS podem ser conferidos a seguir, com seus respectivos gráficos que atestam sua eficácia dados resultados teóricos já conhecidos por nós.

**Programa OHS:**

# -\*- coding: utf-8 -\*-

"""

Created on Thu Sep 20 13:58:15 2018

@author: ruben

"""

import numpy as np

from math import \*

import matplotlib.pyplot as plt

def simpleOHSSimulation(pos0,vel0,tau,m,k,w0,numSteps,plotFlag):

# initialize vectors

time\_vec = [0]\*numSteps

pos\_vec = [0]\*numSteps

vel\_vec = [0]\*numSteps

KE\_vec = [0]\*numSteps

PE\_vec = [0]\*numSteps

# set initial conditions

pos = pos0

vel = vel0

time = 0

# begin time stepping

for i in range(0,numSteps):

vel\_old = vel

pos\_old = pos

# update the values

vel = vel\_old - (w0\*w0\*pos)\*tau

pos = pos\_old + vel\*tau

# record the values

time\_vec[i] = tau\*i

pos\_vec[i] = pos

vel\_vec[i] = vel

KE\_vec[i] = (1/2)\*m\*vel\*vel

PE\_vec[i] = (1/2)\*k\*pos\*pos

TE\_vec = np.add(KE\_vec,PE\_vec)

# make graphs

if plotFlag == 1:

plt.figure(0)

plt.plot(time\_vec,pos\_vec)

plt.xlabel('Tempo (s)')

plt.ylabel('Posição (rad)')

plt.savefig('plot1.png', bbox\_inches='tight')

plt.figure(1)

plt.plot(time\_vec,KE\_vec,label='Energia cinética')

plt.plot(time\_vec,PE\_vec,label='energia potencial')

plt.plot(time\_vec,TE\_vec,label='energia total')

plt.legend(loc='upper left')

plt.xlabel('Tempo (s)')

plt.ylabel('Energia (J)')

plt.savefig('plot2.png', bbox\_inches='tight')

plt.figure(2)

plt.plot(pos\_vec,vel\_vec)

plt.xlabel('posição (rad)')

plt.ylabel('Velocidade (rad/s)')

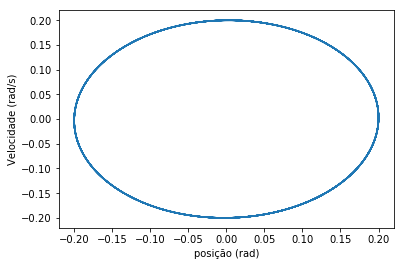
plt.savefig('plot3.png', bbox\_inches='tight')

plt.show()

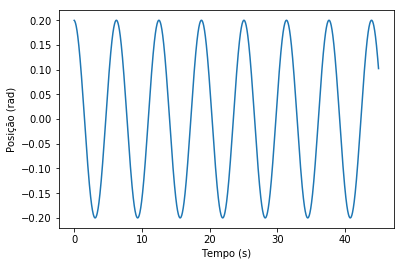
# return the vectors

simpleOHSSimulation(0.2,0,0.03,1,1,1,1500,1)

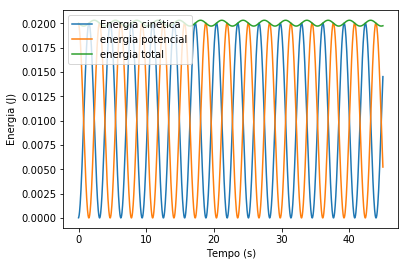
Espaço de fase:



Posição:



Energia:



**Programa Pêndulo:**

# -\*- coding: utf-8 -\*-

"""

Created on Wed Sep 19 18:17:57 2018

@author: ruben

"""

import numpy as np

from math import \*

import matplotlib.pyplot as plt

def simplePendulumSimulation(theta0,omega0,tau,m,g,l,numSteps,plotFlag):

# initialize vectors

time\_vec = [0]\*numSteps

theta\_vec = [0]\*numSteps

omega\_vec = [0]\*numSteps

KE\_vec = [0]\*numSteps

PE\_vec = [0]\*numSteps

# set initial conditions

theta = theta0

omega = omega0

time = 0

# begin time stepping

for i in range(0,numSteps):

omega\_old = omega

theta\_old = theta

# update the values

omega = omega\_old - (g/l)\*sin(theta\_old)\*tau

theta = theta\_old + omega\*tau

# record the values

time\_vec[i] = tau\*i

theta\_vec[i] = theta

omega\_vec[i] = omega

KE\_vec[i] = (1/2)\*m\*l\*\*2\*omega\*\*2

PE\_vec[i] = m\*g\*l\*(1-cos(theta))

TE\_vec = np.add(KE\_vec,PE\_vec)

# make graphs

if plotFlag == 1:

plt.figure(0)

plt.plot(time\_vec,theta\_vec)

plt.xlabel('Tempo (s)')

plt.ylabel('Posição (rad)')

plt.savefig('plot1.png', bbox\_inches='tight')

plt.figure(1)

plt.plot(time\_vec,KE\_vec,label='Energia cinética')

plt.plot(time\_vec,PE\_vec,label='energia potencial')

plt.plot(time\_vec,TE\_vec,label='energia total')

plt.legend(loc='upper left')

plt.xlabel('Tempo (s)')

plt.ylabel('Energia (J)')

plt.savefig('plot2.png', bbox\_inches='tight')

plt.figure(2)

plt.plot(theta\_vec,omega\_vec)

plt.xlabel('posição (rad)')

plt.ylabel('Velocidade (rad/s)')

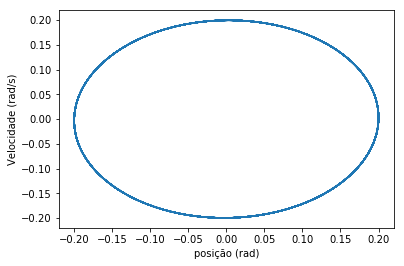
plt.savefig('plot3.png', bbox\_inches='tight')

plt.show()

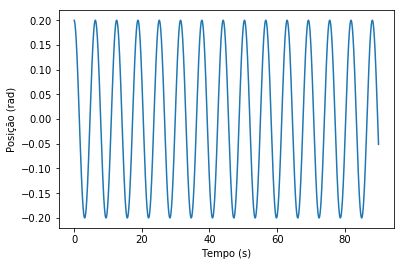
# return the vectors

simplePendulumSimulation(0.2,0,0.03,1,1,1,3000,1)

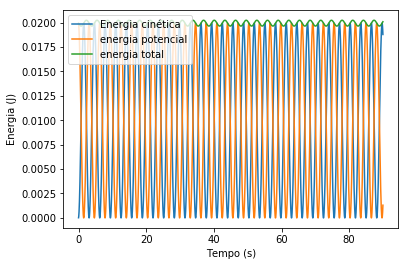
Espaço de fase:



Posição:



Energia:



Agora, verificando a funcionalidade do novo desenvolvimento em Python para a tarefa especifica demandada, tenho aqui o exemplo de um gráfico simples de um tempo (inicial) com os arrays mostrando o espaço de fase do sistema dadas várias condições iniciais diferentes:

