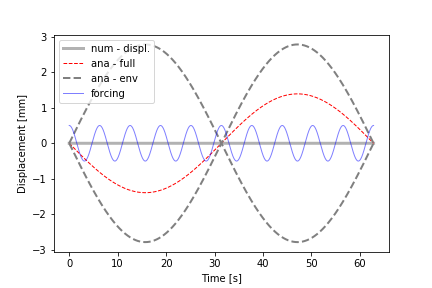
William Dean

EART 119 Hw #5 Undamped Forced Oscillator

Problem 1:



﻿from \_\_future\_\_ import division

import os

import matplotlib as mpl

import matplotlib.pyplot as plt

import numpy as np

#-------------------------------0----------------------------------------

# fct defintion

#------------------------------------------------------------------------

#import ODE.ode\_utils as utils

#:TODO - create a python method that solves the ODE system using Runge-Kutta Method

def num\_sol( at, y0, par):

"""

- solve second order ODE for forced, undamped oscillation by solving two first order ODEs

ODE: y''(t) + ky(t) = f(t)

f(t) = F\*cos(w\*t)

:param y0: - IC

:param at : - time vector

:param par : - dictionary with fct parameters

:return: ay\_hat - forward Euler

"""

nSteps = at.shape[0]

# create vectors for displacement and velocity

au\_hat = np.zeros( nSteps)

av\_hat = np.zeros( at.shape[0])

#:TODO set the initial conditions

au\_hat[0] = y0[0]

av\_hat[0] = y0[1]

for i in range( nSteps-1):

# TODO: slope at previous time step, i. RHS of ODE system

fn1 = av\_hat[i]

fn2 = -dPar['w0']\*\*2\*au\_hat[i]

# TODO: - Euler formula: y[n+1] = y[n] + fn\*h

au\_hat[i+1] = au\_hat[i] + fn1\*par['h']

av\_hat[i+1] = av\_hat[i] + fn2\*par['h']

#alternatively use Euler-Cromer: see Langtangen & Linge, p 129, eq: 4.49 - 4.52

#av\_hat[i+1] = av\_hat[i] + fn2\*par['h']

#au\_hat[i+1] = au\_hat[i] + av\_hat[i+1]\*par['h']

return au\_hat, av\_hat

#-------------------------------1----------------------------------------

# params, files, dir

#------------------------------------------------------------------------

dPar = { #frequencies

'w0' : .8, #= sqrt(k/m) --> natural frequency

'w' : 1, # --> frequency of forcing function

# forcing function amplitude

'F' : 0.5, #20,

# initial conditions for displ. and velocity

'y01' : 0, 'y02' : 0,

# time stepping

'h' : 1e-2,

'tStart' : 0,

'tStop' : 20\*np.pi,}

#a\_t = np.arange( dPar['tStart'], dPar['tStop'], dPar['h'])

#--------------------------------2---------------------------------------

# analytical solution

#------------------------------------------------------------------------

at = np.arange( dPar['tStart'], dPar['tStop']+dPar['h'], dPar['h'])

# forcing function

fct\_t = dPar['F']\*np.cos( dPar['w']\*at)

# TODO: write down the analytical solution

preFacAna = dPar['F']/(dPar['w0']\*\*2 - dPar['w']\*\*2)

ay\_ana1 = preFacAna \* ( np.sin( (dPar['w']-dPar['w0'])/2\*at))

#:TODO rewrite the analytical solution as product of sines

#ay\_ana1 = -preFacAna \* ( np.sin((( dPar['w']) \* np.sin( -dPar['w0']))/2\*at))

# envelope - slow frequency

ay\_ana2 = -2\*preFacAna\* ( np.sin( (dPar['w']-dPar['w0'])/2\*at))

#--------------------------------3---------------------------------------

# numerical solutions

#------------------------------------------------------------------------

aU\_num, aV\_num = num\_sol( at, [dPar['y01'], dPar['y02']], dPar)

#--------------------------------4---------------------------------------

# plots

#------------------------------------------------------------------------

plt.figure(1)

ax = plt.subplot( 111) #plt.axes( [.12, .12, .83, .83])

ax.plot( at, aU\_num, 'k-', lw = 3, alpha = .3, label = 'num - displ.')

ax.plot( at, ay\_ana1, 'r--', lw = 1, label = 'ana - full')

ax.plot( at, ay\_ana2, '--', color = '.5', lw = 2, label = 'ana - env')

ax.plot( at, -ay\_ana2, '--', color = '.5', lw = 2)

ax.plot( at, fct\_t, 'b-', lw = 1, alpha = .5, label ='forcing')

ax.set\_xlabel( 'Time [s]')

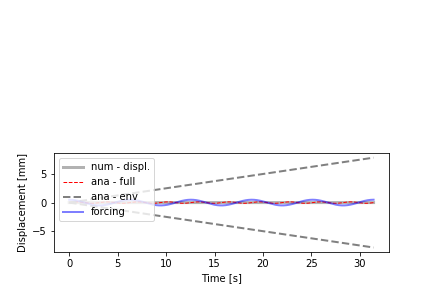
ax.set\_ylabel( 'Displacement [mm]')

ax.legend( loc = 'upper left')

plt.show()

I observed the displacement to be 0 overall. The forcing function has the largest frequency. The slow frequency made a full figure 8 back and forth and the analytical solution followed a sinusoidal curve. Below 1e-1 the forward Euler approximation begins to provide more accurate solutions.

Problem 2:



﻿from \_\_future\_\_ import division

import matplotlib.pyplot as plt

import numpy as np

#-------------------------------0----------------------------------------

# fct. defintion

#------------------------------------------------------------------------

def num\_sol( at, y0, par):

"""

- solve second order ODE for forced, undamped oscillation by solving two first order ODEs

ODE: y''(t) + ky(t) = f(t)

f(t) = F\*cos(w\*t)

:param y0: - IC

:param at : - time vector

:param par : - dictionary with fct parameters

:return: ay\_hat - forward Euler

ay\_hat\_rk = 4th order runge kutta

"""

nSteps = at.shape[0]

# create vectors for displacement and velocity

au\_hat = np.zeros( nSteps)

av\_hat = np.zeros( at.shape[0])

# set initial conditions

au\_hat[0] = y0[0]

av\_hat[0] = y0[1]

for i in range( nSteps-1):

# forward Euler or Runge Kutta

fn1 = av\_hat[i]

fn2 = -par['w0']\*\*2\*au\_hat[i]

au\_hat[i+1] = au\_hat[i] + fn1\*par['h']

av\_hat[i+1] = av\_hat[i] + fn2\*par['h']

return au\_hat, av\_hat

#-------------------------------1----------------------------------------

# params, files, dir

#------------------------------------------------------------------------

dPar = {#frequencies

'w0' : 1, #= sqrt(k/m) --> natural frequency

'w' : 1, # --> frequency of forcing function

# forcing function

'F' : 0.5, #20,

# initial conditions for displ. and velocity

'y01' : 0, 'y02' : 0,

# time stepping

'h' : 1e-2,

'tStart' : 0,

'tStop' : 10\*np.pi,}

#--------------------------------2---------------------------------------

# analytical solution

#------------------------------------------------------------------------

at = np.arange( dPar['tStart'], dPar['tStop']+dPar['h'], dPar['h'])

# forcing function

fct\_t = dPar['F']\*np.cos( dPar['w0']\*at)

# :TODO - write down the analyitcal soltuion

preA = (dPar['F']/(2\*dPar['w0']\*\*2))

ay\_ana1 = fct\_t/(2\*dPar['w0']\*\*2)\*(np.sin(dPar['w0']\*at))

#ay\_ana1 = -preA \* ( np.sin((( dPar['w']) \* np.sin( -dPar['w0']))/2\*at))

# envelope

ay\_ana2 = preA \* at

#--------------------------------3---------------------------------------

# numerical solutions

#------------------------------------------------------------------------

aU\_num, aV\_num = num\_sol( at, [dPar['y01'], dPar['y02']], dPar)

#--------------------------------4---------------------------------------

# plots

#------------------------------------------------------------------------

plt.figure(1)

ax = plt.subplot( 212) #plt.axes( [.12, .12, .83, .83])

# plot the numerical approximation

ax.plot( at, aU\_num, 'k-', lw = 3, alpha = .3, label = 'num - displ.')

# analytical solution and envelope fct.

ax.plot( at, ay\_ana1, 'r--', lw = 1, label = 'ana - full')

ax.plot( at, ay\_ana2, '--', color = '.5', lw = 2, label = 'ana - env')

ax.plot( at, -ay\_ana2, '--', color = '.5', lw = 2)

ax.plot( at, fct\_t, 'b-', lw = 2, alpha = .5, label ='forcing')

ax.set\_xlabel( 'Time [s]')

ax.set\_ylabel( 'Displacement [mm]')

ax.legend( loc = 'upper left')

plt.show()

It isn't completely realistic when w=w0...the computer will get as close as it can to w0/w but won't ever get there since they are limits. In reality I think it would oscillate infinitely(standing resonance wave).