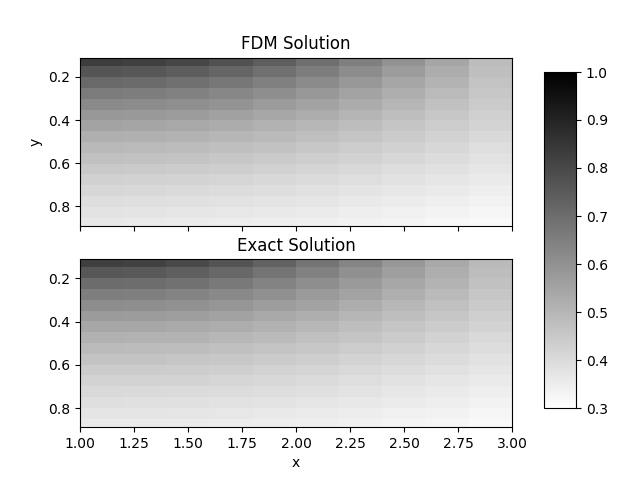
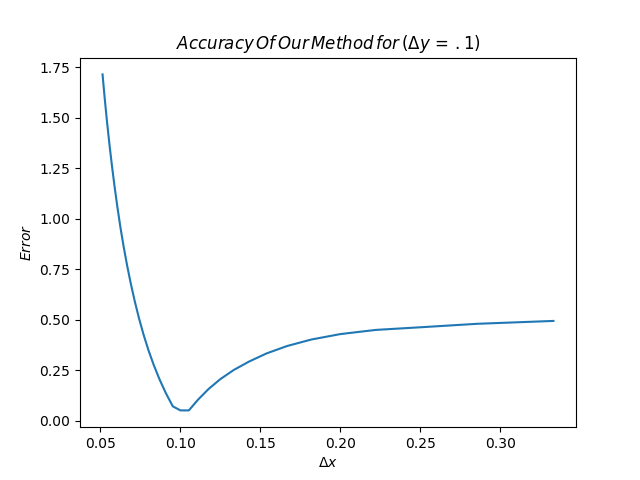
1B and 1C





#!/usr/bin/env python3

import numpy as np

from matplotlib import pyplot as plt

import time

# Initial conditions

x0, x1 = 1, 3

y0, y1 = 0, 1

Ny = 9

error = []

x\_range = []

for intX in range(5, 39):

dx = (x1 - x0) / (intX + 1)

dy = (y1 - y0) / (Ny + 1)

l\_max = (intX-1)\*(Ny-1)

def f(x, y):

# u\_xx + u\_yy = 0

return 0

#conditions given in the problem

def g\_x0(y):

return 1/(1 + y\*\*2)

def g\_x1(y):

return 3/(9 + y\*\*2)

def g\_y0(x):

return 1/x

def g\_y1(x):

return x/(x\*\*2 + 1)

def ij\_l(i\_x, i\_y):

#making our matrix for l

return i\_x + (Ny - i\_y - 1) \* (intX - 1) - 1

#Setting the initial conditions for our matrix

A = np.zeros((l\_max, l\_max))

B = np.zeros(l\_max)

start\_time = time.time()

for i in range(1, intX):

for j in range(1, Ny):

l = ij\_l(i, j)

x\_i = x0 + i\*dx

y\_i = y0 + j\*dy

l\_ip = ij\_l(i+1, j)

l\_im = ij\_l(i-1, j)

l\_jp = ij\_l(i, j+1)

l\_jm = ij\_l(i, j-1)

A[l, l] = -2 \* (1 + dx\*\*2 / dy\*\*2)

B[l] = dx\*\*2 \* f(x\_i, y\_i)

#enforce BC

if i != 1:

A[l\_im, l] = 1

else:

B[l] -= g\_x0(y\_i)

if i != intX-1:

A[l\_ip, l] = 1

else:

B[l] -= g\_x1(y\_i)

if j != 1:

A[l, l\_jm] = dx\*\*2 / dy\*\*2

else:

B[l] -= g\_y0(x\_i)

if j != Ny-1:

A[l, l\_jp] = dx\*\*2 / dy\*\*2

else:

B[l] -= g\_y1(x\_i)

u\_vec = np.linalg.solve(A, B)

u\_solution = np.zeros((intX+1, Ny+1))

# Turn u\_solution back into a 2d matrix

for i in range(intX+1):

for j in range(Ny+1):

# At border. These values are not in u\_vec

if i==0 or i==intX or j==0 or j==Ny:

x\_i = x0 + i\*dx

y\_i = y0 + j\*dy

if i==0:

u\_solution[i, j] = g\_x0(y\_i)

elif i==intX:

u\_solution[i, j] = g\_x1(y\_i)

if j==0:

u\_solution[i, j] = g\_y0(x\_i)

elif j==Ny:

u\_solution[i, j] = g\_y1(x\_i)

continue

l = ij\_l(i, j)

u\_solution[i, j] = u\_vec[l]

def solution(x, y):

return x / (x\*\*2 + y\*\*2)

u\_exact\_soln = np.zeros((intX+1, Ny+1))

for i, x in enumerate(np.linspace(x0, x1, intX+1)):

for j, y in enumerate(np.linspace(y0, y1, Ny+1)):

u\_exact\_soln[i, j] = solution(x, y)

if round(dx, 3) == .1:

#FDM Solution

cm = plt.get\_cmap('binary')

fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True)

ax1.imshow(u\_solution, extent=[x0, x1, y1, y0], cmap=cm)

ax1.set\_title('FDM Solution')

ax1.set\_ylabel('y')

#Exact Solution Figure

im2 = ax2.imshow(u\_exact\_soln, extent=[x0, x1, y1, y0], cmap=cm)

ax2.set\_title('Exact Solution')

ax2.set\_xlabel('x')

#Our graph locations and plots

fig.subplots\_adjust(right=.8)

ax3 = fig.add\_axes([.85, .15, .05, .7])

fig.colorbar(im2, cax=ax3)

plt.savefig('FDMvsExact.png')

#appending our error so we can graph it later on another graph

error.append(np.max(np.abs(u\_exact\_soln - u\_solution)))

x\_range.append(dx)

plt.figure()

plt.plot(x\_range, error)

plt.xlabel(r'$\Delta x$')

plt.ylabel(r'$Error$')

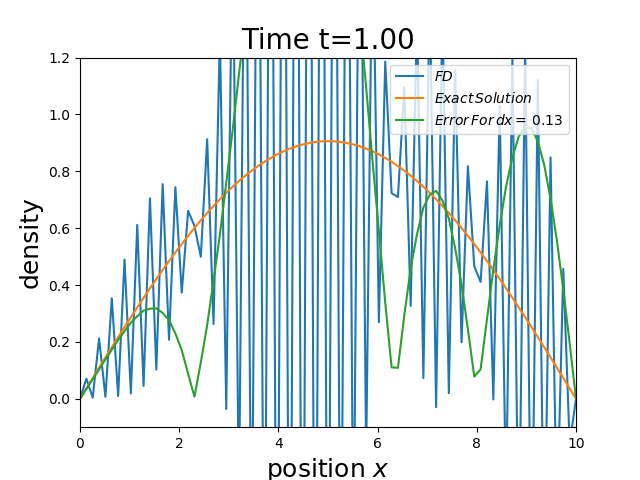
plt.title(r'$Accuracy\/Of\/Our\/Method\/for\/(\Delta y\/=\/.1)$')

plt.savefig('Error\_Plot\_Problem\_1.png')

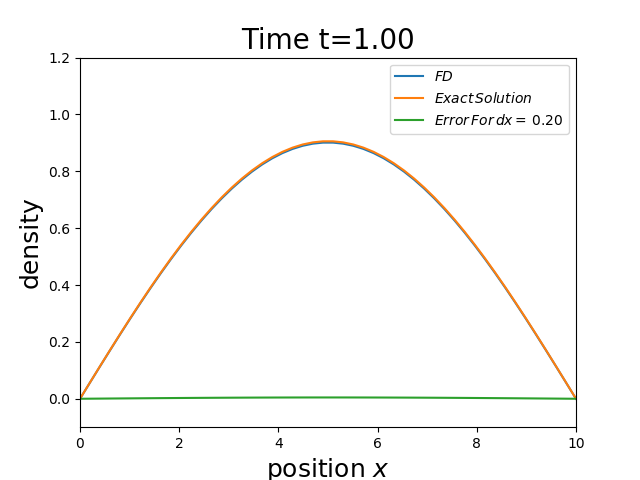
2B and 2C)

dx = .13

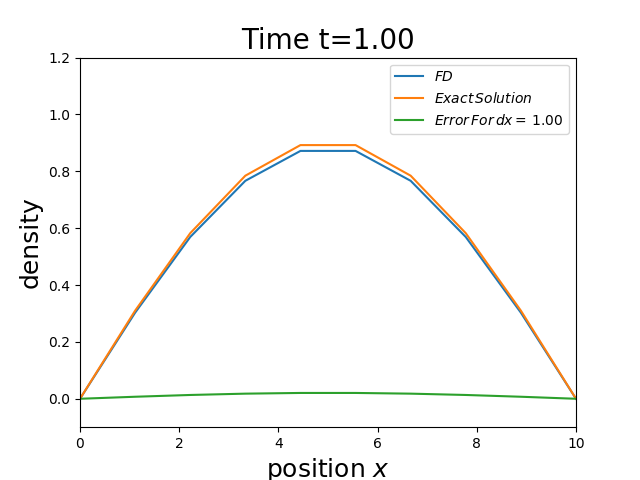
See how the vibrations overcome the error? Cool, but odd.



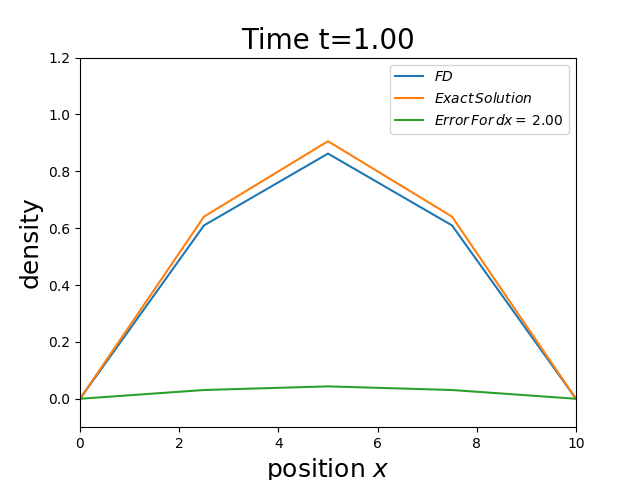
dx = .2



dx = 1



dx = 2



CODE

#using the labsolutions as my barebones code. Otherwise I have no clue how to make the graph move

#in time

import numpy as np

import matplotlib.pyplot as plt

D = 1

l = 10

T = 1

def u0(x):

return np.sin(np.pi \* x / l)

dx = 2

dt = 10\*\*(-2)

ld = D/dx\*\*2\*dt

nTime = int(T/dt+.5) + 1

nX = int(l/dx+.5)

intX = np.linspace(0,l,nX)

intT = np.linspace(dt,T)

u = u0(intX)

utrue = u0(intX)

error = np.abs(utrue-u)

plt.ion()

plt.clf()

pltU = plt.plot(intX, u,label = r'$FD$')[0]

pltUtrue = plt.plot(intX,u,label = r'$Exact\/Solution$')[0]

plterror = plt.plot(intX,error,label = r'$Error\/For\/dx=\/ $''%-.2f'%dx)[0]

plt.legend(loc = 1)

plt.axis([0, l, -.1, 1.2])

plt.xlabel(r'position $x$', fontsize=18)

plt.ylabel('density', fontsize=18)

theTitle = plt.title('Solution at T=' + '{:04.2f}, dx = %-.2f'.format(0\*dt),

horizontalalignment='center', fontsize=20)

for n in range(nTime):

u\_true = np.exp(-1\*n\*dt\*np.pi\*\*(2)/l\*\*(2))\*utrue

u = (1-2\*ld)\*u + ld\*(np.append(u[1:], 0) + np.append(0, u[:-1]))

u[0] = 0

u[-1] = 0

plterror.set\_ydata(np.abs(u\_true - u))

pltU.set\_ydata(u)

pltUtrue.set\_ydata(u\_true)

theTitle.set\_text('Time t=' + '{:04.2f}'.format(n\*dt))

plt.pause(.001)

if n ==0:

plt.savefig("FD\_dx\_%-.2f\_t\_%-.1f.png"%(dx,n\*dt))

if n ==50:

plt.savefig("FD\_dx\_%-.2f\_t\_%-.1f.png"%(dx,n\*dt))

if n ==100:

plt.savefig("FD\_dx\_%-.2f\_t\_%-.1f.png"%(dx,n\*dt))