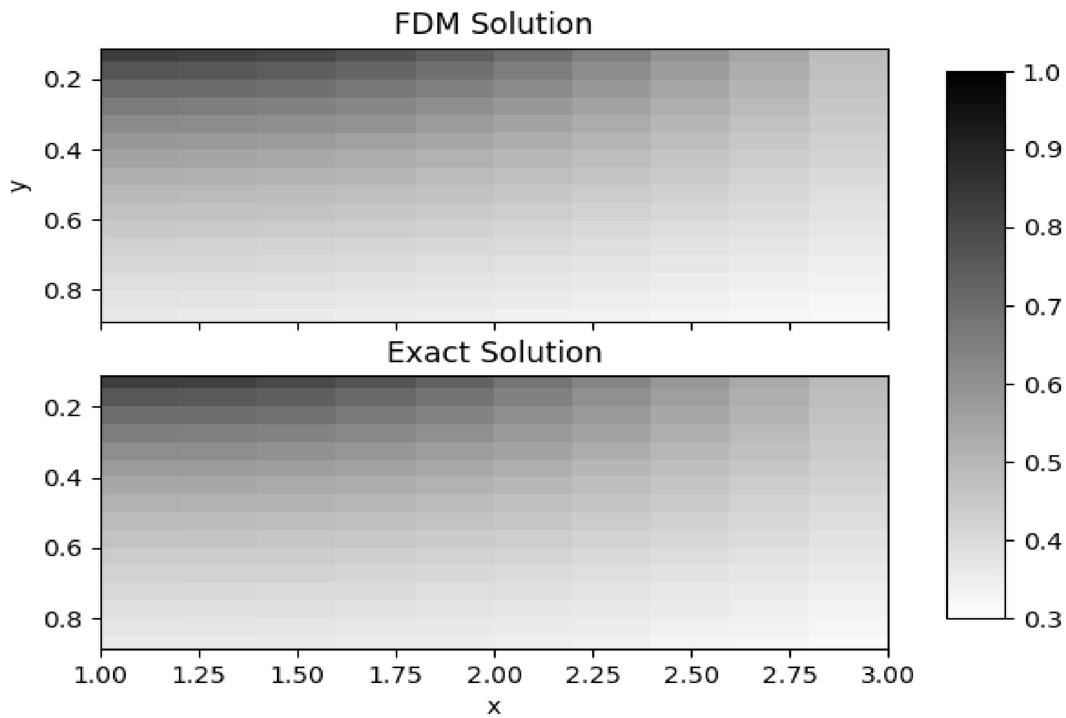
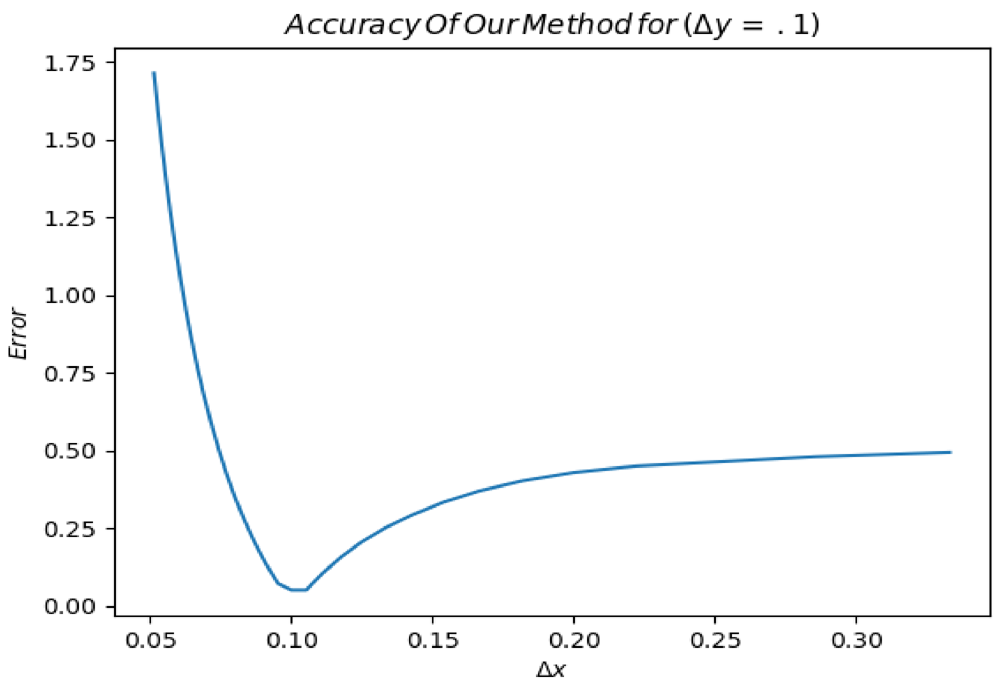
1B and 1C


FDM Solution
Exact Solution
y
x
0.2
0.4
0.6
0.8
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3

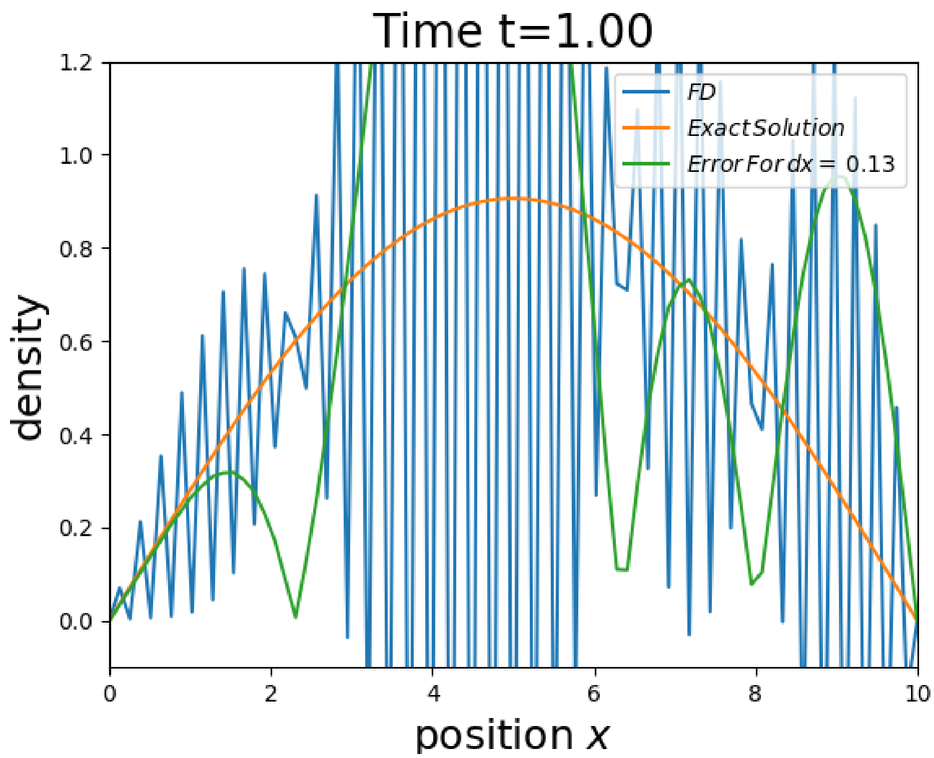

$Accuracy\,Of\,Our\,Method\,for\,(\Delta y=.1)$
$Error$
Δx
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
0.05
0.10
0.15
0.20
0.25
0.30

```
#!/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:
```

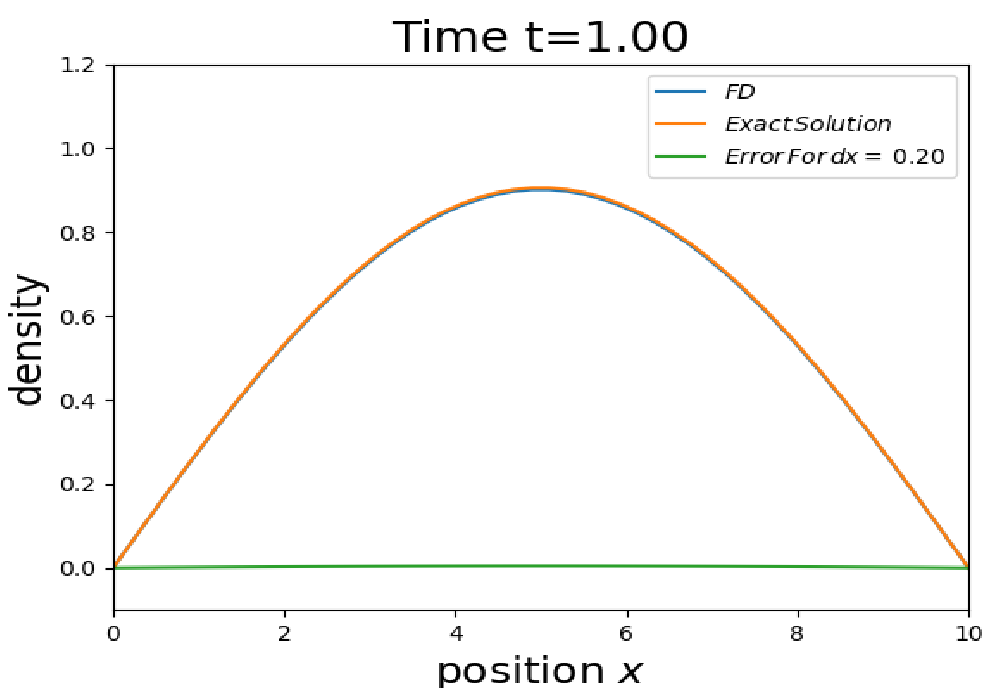
```
                A[l, l_jm] = dx**2 / dy**2
            else:
                B[l] -= g_y0(x_i)
            if j != Ny-1:
                A[l, l_jp] = dx**2 / dy**2
            else:
                B[l] -= g_y1(x_i)

    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')
```

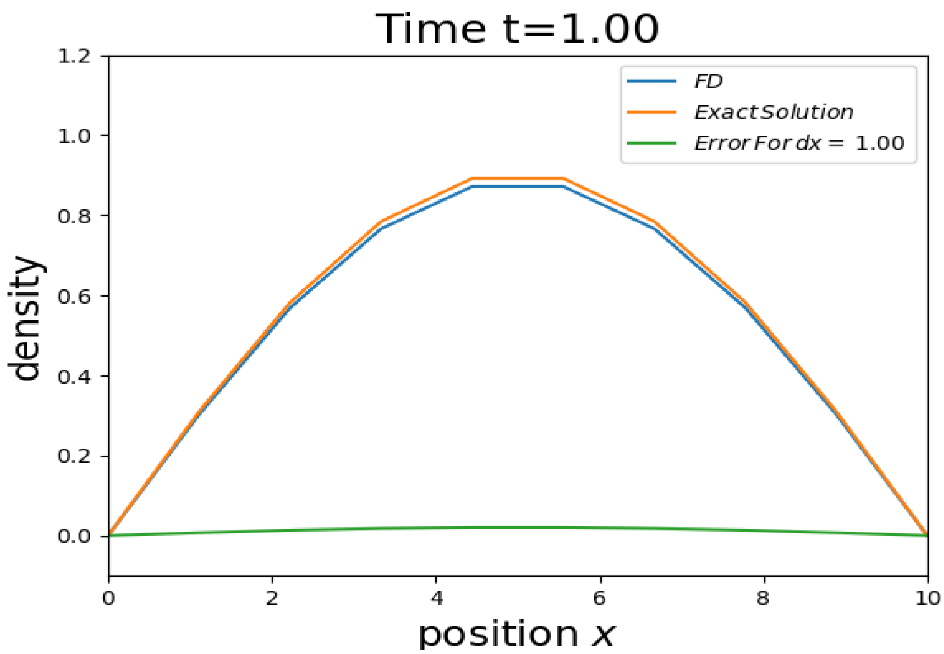
```
2B and 2C)
dx = .13
See how the vibrations overcome the error? Cool, but odd.
```



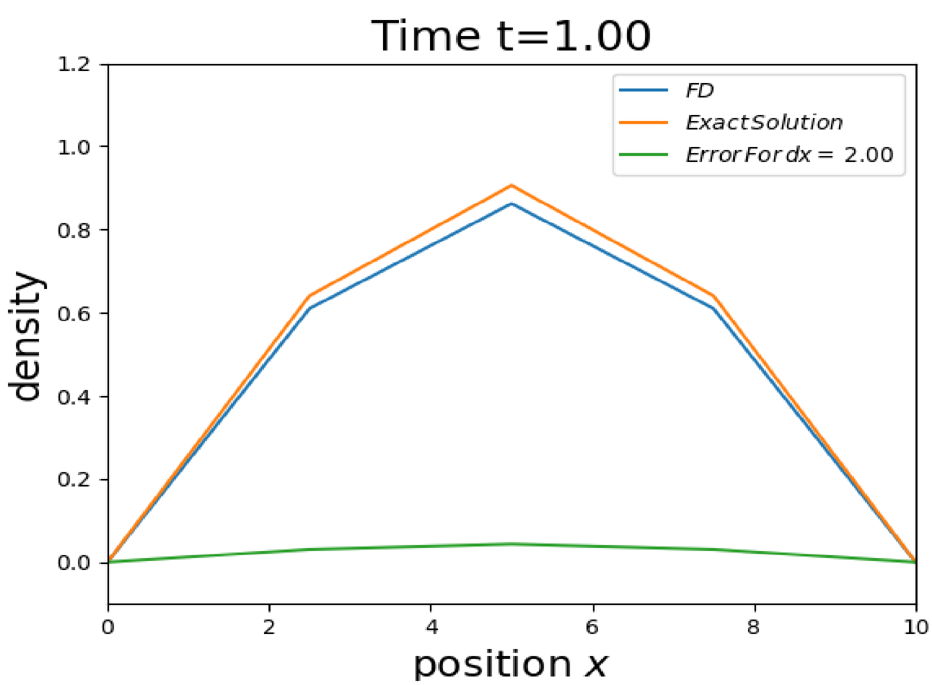

```
dx = .2
```



dx = 1


Time t=1.00
FD
ExactSolution
ErrorFordx = 1.00
density
position x

dx = 2


Time t=1.00
FD
ExactSolution
ErrorFordx = 2.00
density
position x

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))
```