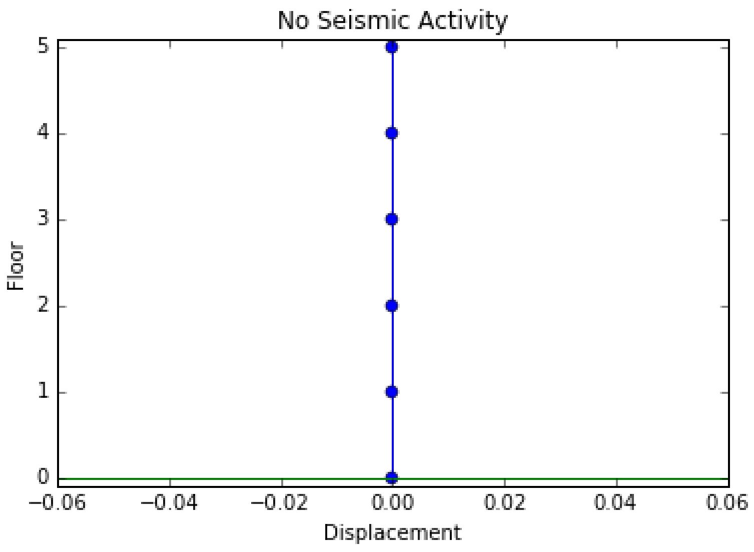
Simulating a Building in an Earthquake

12 points

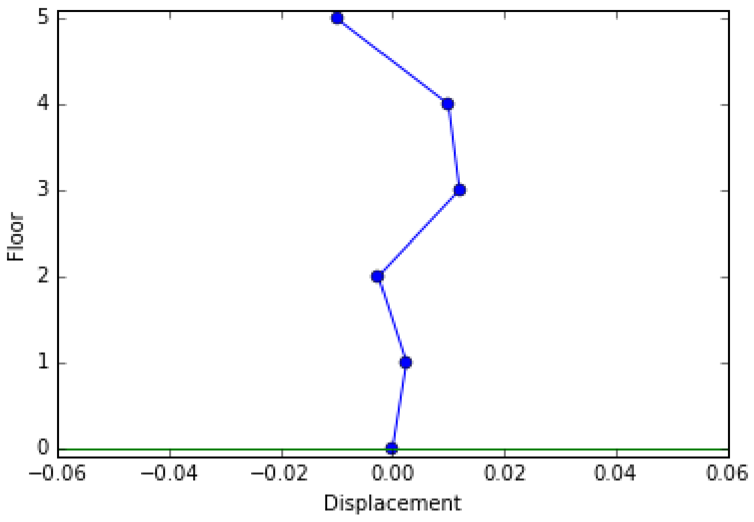
Overview (click to view)

Suppose we have a building with n floors. In the case of an earthquake, it is important to be able to predict the lateral displacement ("side-to-side" movement) of each floor. Specifically, let x_i denote the lateral displacement of the roof of floor i. The following is a plot of a 5-story building when all displacements are zero (i.e., no seismic activity):



There are 6 points here: the bottom is the relative displacement of the ground (always zero), while every other point represents the displacement of the roof of its respective floor. Since there is no displacement, all points are perfectly aligned vertically.

If we apply a (large) force $F(t)$, each floor will be affected, shifting its center of mass horizontally, so that the displacement becomes a function of time, $x_i(t)$. This is illustrated here:



Here, we see lateral displacements of each floor. The location of the top of floor 1 is slightly to the right of its original location, so $x_1 > 0$. The location of the top of floor 2 is slightly to the left of its orginal location, so $x_2 < 0$. If you would like a more artistic depiction of the problem, see the figure here (http://www.shodor.org/~wmyers/curric/workshops/lessons/ise/renee/weave/module1/general.html).

Mathematical Formulation (click to view)

Denote the mass of each floor as m_i and the restoring force of each floor (Hooke's force) k_i. The displacement of floor j is governed by the following differential equation:

$$m_j x_j''(t) = k_j x_{j-1}(t) - (k_j + k_{j+1}) x_j(t) + k_{j+1} x_{j+1}(t) + m_j F(t) \qquad (*)$$

Where $F(t)$ is the forcing function. If we are at the first floor ($j = 1$) we omit the first term in the right hand side, since the foundation of the building has no displacement (relative to itself). If we are at the top floor ($j = n$), we omit x_{j+1} term since there is no floor above it. We also set $k_{j+1} = 0$ for the top floor, as there is no "restoring force" for the sky above.

If we collect all these equations together, we can write the system of equations as:

$$\boldsymbol{M}\boldsymbol{x}''(t) = \boldsymbol{K}\boldsymbol{x}(t) + \boldsymbol{H}(t)$$

where $\boldsymbol{M}$ is a diagonal matrix with the masses of each floor m_i placed along diagonal, $\boldsymbol{K}$ is a tridiagonal matrix that can be constructed from equation $(*)$ above, and $\boldsymbol{H}(t)$ is a column vector of the m_j's, multiplied by the forcing function $F(t)$.

To use the methods you will learn in class and in the text, we require a first-order system of equations; this is second order. If we introduce an intermediate variable $\boldsymbol{v}(t) = \boldsymbol{x}'(t)$, we can rewrite this as a first-order system:

$$\begin{aligned} \boldsymbol{x}'(t) &= \boldsymbol{v}(t) \\ \boldsymbol{M}\boldsymbol{v}'(t) &= \boldsymbol{K}\boldsymbol{x}(t) + \boldsymbol{H}(t) \end{aligned}$$

Or written more compactly:

$$\begin{bmatrix} \boldsymbol{I} & \\ & \boldsymbol{M} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}'(t) \\ \boldsymbol{v}'(t) \end{bmatrix} = \begin{bmatrix} & \boldsymbol{I} \\ \boldsymbol{K} & \end{bmatrix} \begin{bmatrix} \boldsymbol{x}(t) \\ \boldsymbol{v}(t) \end{bmatrix} + \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{H}(t) \end{bmatrix}$$

If we let $\boldsymbol{y}(t) = \begin{bmatrix} \boldsymbol{x}(t) & \boldsymbol{v}(t) \end{bmatrix}^T$, we can express our equation in the form $\boldsymbol{y}' = f(t, \boldsymbol{y})$ **after** inverting the matrix on the left hand side.

What You Need to Do (click to view)

We will consider the following forcing function induced by the earthquake:

$$F(t) = -\omega^2 F_0 cos(\omega t)$$

Starting with a building initially at rest (i.e. $x_i(0) = v_i(0) = 0$ for all i), integrate this differential equation forward in time using three-step Adams-Bashforth,

$$\boldsymbol{y}_{k+3} = \boldsymbol{y}_{k+2} + \frac{h}{12}\Big(23f(t_{k+2}, \boldsymbol{y}_{k+2}) - 16f(t_{k+1}, \boldsymbol{y}_{k+1}) + 5f(t_k, \boldsymbol{y}_k)\Big)$$

Use the fourth-order Runge-Kutta method (RK4) to initialize Adams-Bashforth.

You'll be given `n` , the number of floors, you may take each m_i to be equal to 10, and each k_i to be 1000.

You should integrate this forward in time until $t = 4$, using 400 time steps, **including** the initial time. I.e you should consider t values in np.linspace(0,4,400). Set $F_0 = 0.05$ and consider two cases: `w1` and `w2` which will be given to you. Save an array of displacements for each floor after $t = 4$ corresponding to `w1` and another for `w2` . You are provided a function to plot the array of displacements (`plot(x)`)to help understand visually what is occurring.

Inputs:

- `n` : the number of floors
- `w1` : an instance of `np.float` the first frequency of the forcing function.
- `w2` : an instance of `np.float` the second frequency of the forcing function.

Outputs:

- `x1_initial` : a numpy array of length n corresponding to the displacements after the first RK4 step used in initializing AB3 for `w1`
- `x2_initial` : a numpy array of length n corresponding to the displacements after the first RK4 step used in initializing AB3 for `w2`
- `x1` : a numpy array of length n corresponding to the displacements at $t = 4$ corresponding to `w1`
- `x2` : a numpy array of length n corresponding to the displacements at $t = 4$ corresponding to `w2`

Problem set-up code (click to view)

Starter code (click to view)

Answer*

```
tlist = np.linspace(0,4,400)
y0 = np.zeros(2*n)
y1a = RK(tlist[0],y0,w1)
y1b = RK(tlist[0],y0,w2)
x1_initial = y1a[:n]
x2_initial = y1b[:n]
y2a = RK(tlist[1],y1a,w1)

y2b = RK(tlist[1],y1b,w2)
y0a = y0.copy()
y0b = y0.copy()
for i in range(2,399):
    y3a = y2a + h*(23*ff(tlist[i],y2a,w1) - 16*ff(tlist[i-1],y1a,w1) + 5*
    y0a = y1a.copy()
    y1a = y2a.copy()
    y2a = y3a.copy()
    y3b = y2b + h*(23*ff(tlist[i],y2b,w2) - 16*ff(tlist[i-1],y1b,w2) + 5*
    y0b = y1b.copy()
    y1b = y2b.copy()
    y2b = y3b.copy()
x1 = y3a[:n]
x2 = y3b[:n]
plot(y3a[:n])
```

Press F9 to toggle full-screen mode. Set editor mode in user profile (/profile/).

Your answer is correct.

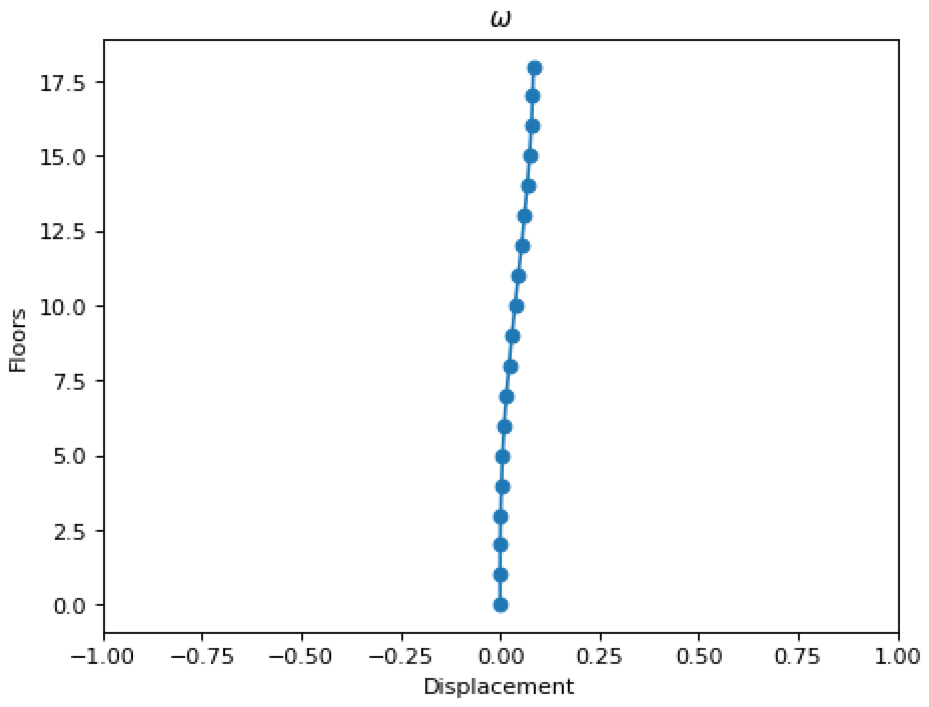
Here is some feedback on your code:

- 'x1_initial' looks good
- 'x2_initial' looks good
- 'x1' looks good
- 'x2' looks good
- Execution time: 1.2 s -- Time limit: 20.0 s

Your code ran on relate-02.cs.illinois.edu.

Your code produced the following plots:

Figure1



The following code is a valid answer:

```
import numpy as np
import matplotlib.pyplot as plt

F_0 = 0.05

def F(t, w):
    return - (w ** 2.0) * F_0 * np.cos(w * t) * np.ones((n,))

k = np.ones(n + 1) * 1000.0
m = np.ones(n) * 10.0
k[-1] = 0.0
K = np.diag(-k[0:-1] - k[1:]) + np.diag(k[1:-1], 1) + np.diag(k[1:-1], -1)

def plot(x):
    plt.figure()
    xplot = np.insert(x,0,0)
    plt.plot(xplot,np.arange(0,n+1),'o-')
    plt.xlim([-1,1])
    plt.title('$\omega$')
    plt.xlabel('Displacement')
    plt.ylabel('Floors')
    plt.show()

def f(y, t, w):
    m = int(len(y) / 2)
    x = y[:m]
    v = y[m:]
    z = (K @ x) / 10.0 + F(t, w)
    ynew = np.hstack((v, z)).T
    return ynew

def rk4(ts, y, w):
    for t in ts[:2]:
        b1 = f(y[-1], t, w)
        b2 = f(y[-1] + (dt / 2.0) * b1, t + dt / 2.0, w)
        b3 = f(y[-1] + (dt / 2.0) * b2, t + dt / 2.0, w)
        b4 = f(y[-1] + dt * b3, t + dt, w)
        y.append(y[-1] +  (dt / 6.0) * (b1 + 2.0 * b2 + 2.0 * b3 + b4))

def ab3(ts,y,w):
    for i in range(2, ts.shape[0] - 1):
        t = ts[i]
        b1 = f(y[-1], ts[i], w)
        b2 = f(y[-2], ts[i - 1], w)
        b3 = f(y[-3], ts[i - 2], w)
        y.append(y[-1] + dt * (23.0 / 12.0 * b1 - 4.0 / 3.0 * b2 + 5.0 / 12.0 * b3)

T = 4.0
steps = 400
tvec = np.linspace(0.0, T, steps)
dt = tvec[1] - tvec[0]
test = tvec[-2]
y1 = [np.zeros(2 * n)]
y2 = [np.zeros(2 * n)]

# Time integration
rk4(tvec, y1, w1)
```

```
x1_initial = y1[1][:n]
ab3(tvec, y1, w1)

rk4(tvec, y2, w2)
x2_initial = y2[1][:n]
ab3(tvec, y2, w2)

x1 = y1[-1][:n]
x2 = y2[-1][:n]
```