Overview

This week we will solve a different kind of differential equation. Instead of solving initial value problems (functions of time), we'll solve a boundary value problem (functions of position). Below you will find some explanations and examples of coding concepts that you will need to solve your problem this week. Specifically, you will learn to:

1. work with multidimensional numpy arrays.
2. plot multidimensional functions.

Working with multi-dimensional arrays

So far, you have mostly worked with one-dimensional lists and arrays. This week it will be essential that you can build and manipulate multi-dimensional arrays. In the code box below you will find a few examples of how to build a multidimensional array.

1. Use print statements to figure out what each line does.
2. Add short explanation comments next to each line.

In [ ]:
```python
from numpy import zeros, ones, ones_like, zeros_like, array, random, any, all

# Creating a 3x3 array with specified values
a = array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Creating a 4x4 array filled with zeros
b = zeros([4, 4])

# Creating an array of zeros with the same shape as array 'a'
c = zeros_like(a)

# Creating a 3x4 array filled with ones
d = ones([3, 4])

# Creating an array of ones with the same shape as array 'b'
e = ones_like(b)

# Creating a 3x3 array filled with random numbers between 0 and 1
f = random.random([3, 3])

a, b, c, d, e, f
```

Out[ ]:
```
(array([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]),
 array([[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]]),
 array([[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]]),
 array([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]),
 array([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]),
 array([[0.2057728 , 0.12260721, 0.19451565],
        [0.6579758 , 0.67621976, 0.32321055],
        [0.72082249, 0.4111903 , 0.77503428]]))
```

There are many useful functions intended to be used with numpy arrays. Below you will find a small sample of them.

1. Use print statements to figure out what each line does.
2. Add short explanation comments next to each line.

In [ ]:
```python
from numpy import random

# Checking if any element in array 'f' is greater than 0.5
g = any(f > .5)

# Checking if all elements in array 'f' are greater than 0.5
h = all(f > .5)

# Getting the shape of array 'a'
i = a.shape

# Getting the number of dimensions in array 'a'
j = a.ndim

# Getting the total number of elements in array 'a'
k = a.size

# Iterating through all elements of array 'a'
for ele in a.flat:
    print(ele)

# Transposing array 'a'
l = a.T

g, h, i, j, k, l
```

```
1
2
3
4
5
6
7
8
9
```

Out[ ]:
```
(True,
 False,
 (3, 3),
 2,
 9,
 array([[1, 4, 7],
        [2, 5, 8],
        [3, 6, 9]]))
```

Often it will be necessary to extract a single element, or several elements, from an array. Extracting a single element is done mostly the same as with lists except that you don't need multiple square brackets but can separate the indices with a comma. (i.e. a[0][2] for lists, but a[0,2] is ok for arrays) When extracting a bigger chunk of a multidimensional array, we call it slicing and use the colon operator (:). Some examples are shown below.

1. Use print statements to figure out what each line does.
2. Add short explanation comments next to each line.

In [ ]:
```python
from numpy import random

# Creating a 5x5 array filled with random numbers between 0 and 1
a = random.random([5, 5])

# Extracting a single element from array 'a'
b = a[0, 2]

# Slicing a portion of array 'a'
c = a[0:2, 2:]

# Slicing a portion of array 'a'
d = a[:3, 1]

# Extracting a column from array 'a'
e = a[:, 1]

# Slicing a portion of array 'a'
f = a[1:2, :]

a, b, c, d, e, f
```

Out[ ]:
```
(array([[0.21773743, 0.72480048, 0.8402193 , 0.58524668, 0.31811648],
        [0.6557365 , 0.30373159, 0.51803989, 0.27619448, 0.15536851],
        [0.63982914, 0.92580307, 0.93507605, 0.88482145, 0.66170663],
        [0.21815481, 0.94630309, 0.66650239, 0.4676395 , 0.98394509],
        [0.6366694 , 0.73495678, 0.12804523, 0.3275577 , 0.3737415 ]]),
 0.8402193042322409,
 array([[0.8402193 , 0.58524668, 0.31811648],
        [0.51803989, 0.27619448, 0.15536851]]),
 array([0.72480048, 0.30373159, 0.92580307]),
 array([0.72480048, 0.30373159, 0.92580307, 0.94630309, 0.73495678]),
 array([[0.6557365 , 0.30373159, 0.51803989, 0.27619448, 0.15536851]]))
```

To practice the skills that you'll need using arrays this week, write a code that does the following:

1. Construct a 6 x 6 array of random numbers between 0 and 10. Call it "a".
2. Construct a loop to modify the entries of "a" to be equal to the average of the entry's neighboring values. (i.e. $V_{i,j} = \frac{V_{i-1,j}+V_{i+1,j}+V_{i,j-1}+V_{i,j+1}}{4}$) Note that the edge values cannot be changed because they don't have four neighbors.
3. Continue updating the entries of the array until no entry changes by more than 1.0×10^{-4}.
4. Print the array before and after the main loop so you can see how it has changed.

Watch out for infinite loops!!

In [ ]:
```python
from numpy import random
import numpy as np

# Construct a 6x6 array of random numbers between 0 and 10
a = random.randint(0, 11, size=(6, 6))

# Print the initial array
print("Initial array:")
print(a)

# Modify the entries of "a" to be the average of neighboring values
# Continue updating until no entry changes by more than 1.0e-4
while True:
    a_new = a.copy()
    max_change = 0.0

    for i in range(1, a.shape[0] - 1):
        for j in range(1, a.shape[1] - 1):
            avg = (a[i - 1, j] + a[i + 1, j] + a[i, j - 1] + a[i, j + 1]) / 4
            max_change = max(max_change, abs(a[i, j] - avg))
            a_new[i, j] = avg

    if max_change <= 1e-4:
        break

    a = a_new

# Print the final array after convergence
print("\nFinal array after convergence:")
print(a)
```

```
Initial array:
[[ 0  2  6  5  4  4]
 [ 5  1  0  3  9  1]
 [ 0 10  3  4  8  5]
 [ 9 10  4  9 10  1]
 [ 1  6  5  7  8  9]
 [ 6  9  3  6  0  9]]
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/var/folders/wt/m3rtv90d1xvfs20h6qn_kfjc0000gn/T/ipykernel_23640/2893035258.py in <module>
     15     max_change = 0.0
     16
---> 17     for i in range(1, a.shape[0] - 1):
     18         for j in range(1, a.shape[1] - 1):
     19             avg = (a[i - 1, j] + a[i + 1, j] + a[i, j - 1] + a[i, j + 1]) / 4

KeyboardInterrupt:
```

Three-Dimensional plots.

Remember that computers don't plot functions, they plot points and connect them. When we plotted one-dimensional functions, we first had to generate an array of densely-spaced points for x and then evaluate the function using this array. Plotting a function of two variable requires a similar process:

1. Generate a list of densely-spaced x-y pairs over the domain of your function. This is done using a function called 'meshgrid' in python. The meshgrid function generates two, 2D arrays: one that holds the x-coordinates and one that hold the y-coordinates. The arrays are ordered so that the same element in each array form the x-y pairs. For example, the (1,1) element of one x array pairs with the (1,1) element of the y array.
2. Evaluate the function of interest over the array of x-y pairs. Since we are working with arrays, evaluating the function can be done quite succinctly; just evaluate the function using the 2D arrays that you generated in step 1.

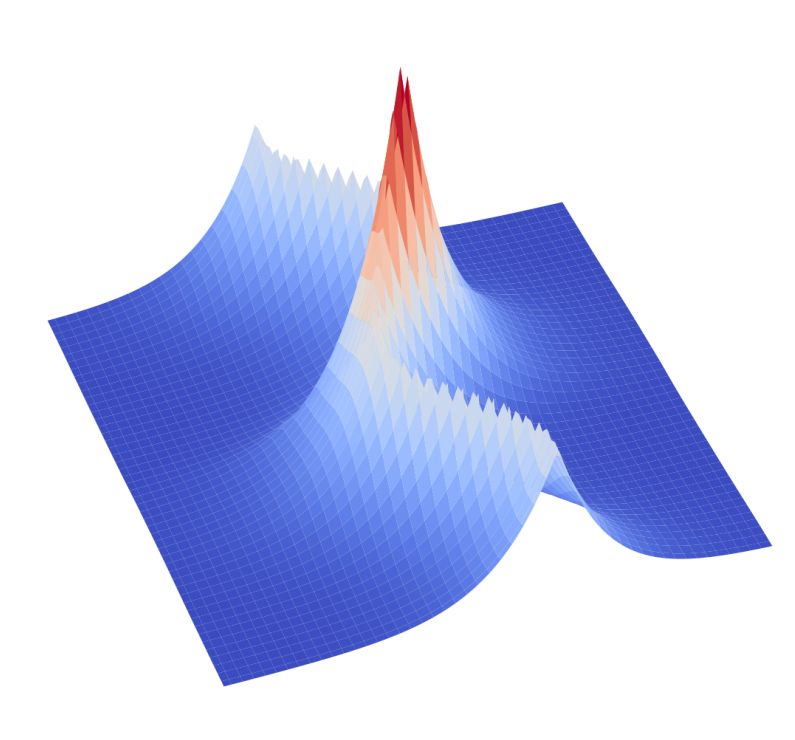
Consider a function of two variables:

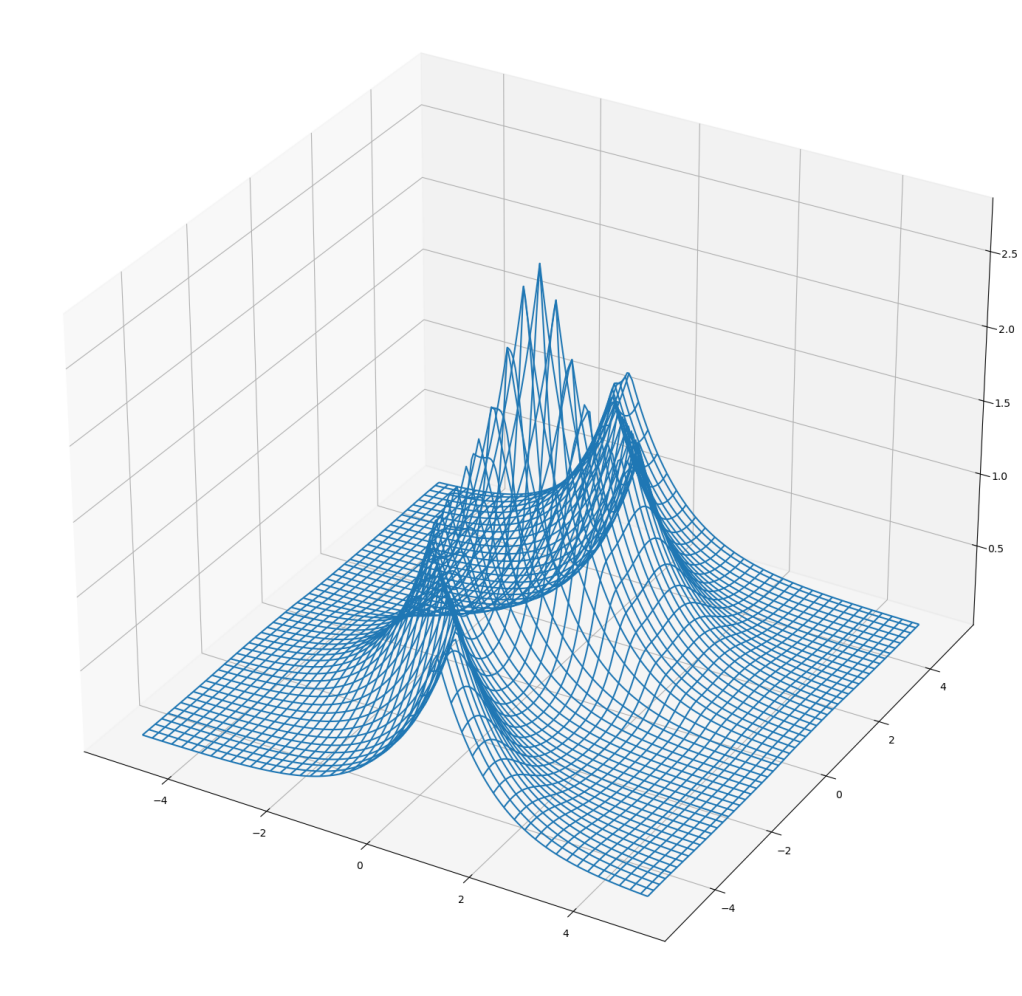
$$f(x, y) = e^{-|x-\sin(y)|}(1 + \frac{1}{5}\cos(\frac{x}{2}))(1 + \frac{4}{3 + 10y^2})$$

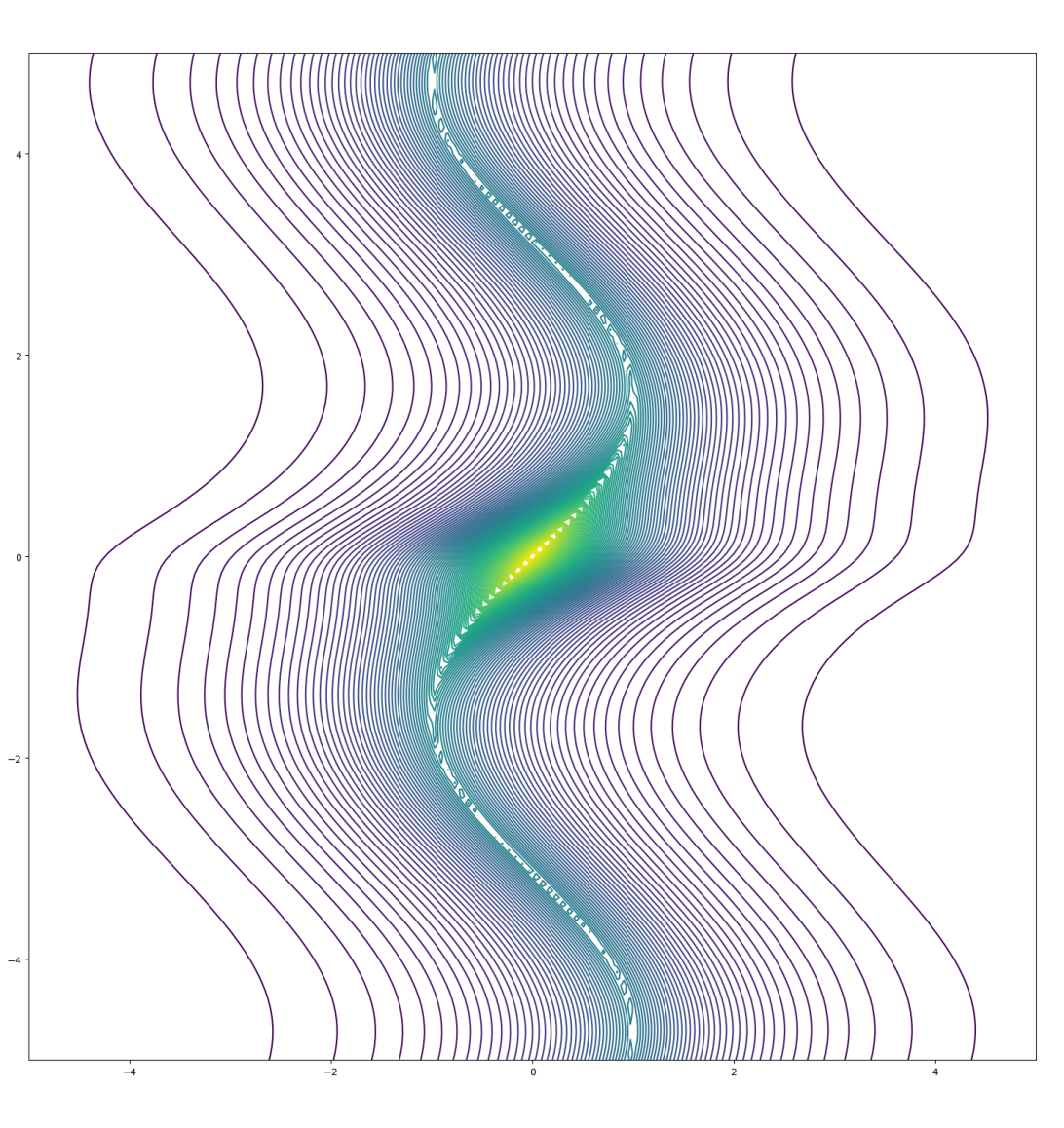
Below you will find some example code for plotting this function and other more advanced plots.

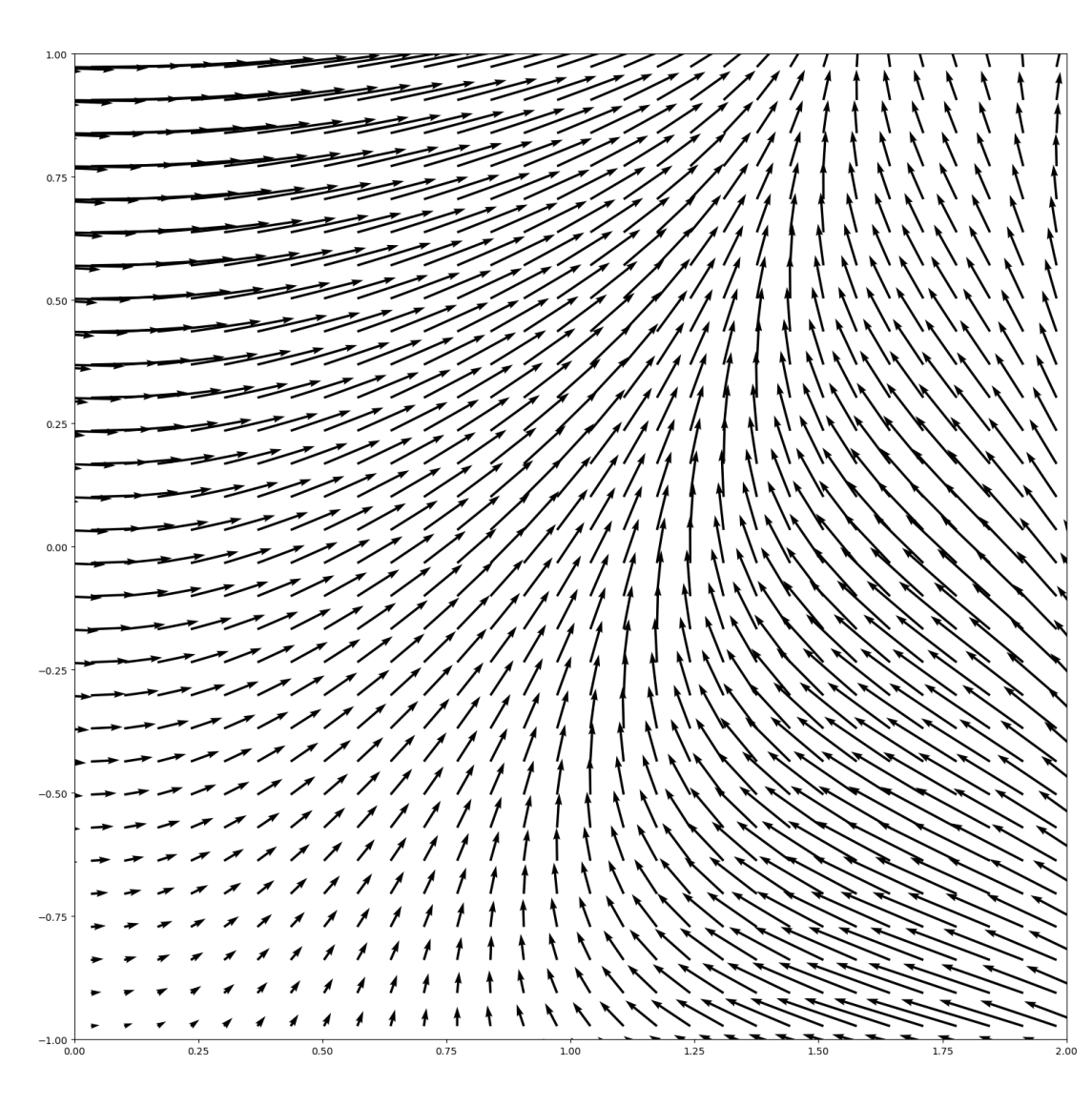
1. Evaluate the code and be impressed by the output.
2. Study the code below, adding print statements and asking questions until you understand.
3. Add comments to help you remember what each line does.

In [ ]:
```python
# Importing necessary libraries
from numpy import meshgrid, linspace, exp, cos, sin
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

# Creating a figure for plotting
fig = plt.figure(figsize=(40, 40))

# Generating x and y values for the surface plot
x = linspace(-5, 5, 150)
y = linspace(-5, 5, 150)
X, Y = meshgrid(x, y)

# Defining the function Z as a combination of exponential, cosine, and sine functions
Z = exp(-abs(X - sin(Y))) * (1 + 1/5 * cos(X/2)) * (1 + 4/(3 + 10 * Y**2))

# Defining electric field components Ex and Ey
Ex = cos(X**2) + sin(Y)
Ey = sin(X) * cos(Y)

# Creating subplots for different types of plots
ax1 = fig.add_subplot(2, 2, 1, projection='3d')  # 3D surface plot
ax1.plot_surface(X, Y, Z, cmap=cm.coolwarm)  # Plotting the surface
ax1.view_init(elev=40, azim=70)  # Setting the view angle
ax1.set_axis_off()  # Turning off the axes

ax2 = fig.add_subplot(2, 2, 2, projection='3d')  # 3D wireframe plot
ax2.plot_wireframe(X, Y, Z)  # Plotting the wireframe

ax3 = fig.add_subplot(2, 2, 3)  # Contour plot
ax3.contour(X, Y, Z, 100)  # Plotting contour lines
ax3.set_aspect(1)  # Setting aspect ratio to 1

ax4 = fig.add_subplot(2, 2, 4)  # Quiver plot
ax4.quiver(X, Y, Ex, Ey, scale=20)  # Plotting vector field
ax4.set_xlim(0, 2)  # Limiting x-axis
ax4.set_ylim(-1, 1)  # Limiting y-axis

# Displaying the plot
plt.show()
```









Numerical Method

From your electrostatics class, you should remember Laplace's equation:

$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} + \frac{\partial^2 V}{\partial z^2} = 0$$

which involves the electrostatic potential, $V(x, y, z$, as a function of position. This is a very different differential equation from what we have solved in the past. Instead of starting with some initial conditions and using the differential equation to find $x(t)$ and $v(t)$, here we will need boundary conditions and then use the differential equation to solve for $V(x, y, z)$. To formulate a numerical recipe for doing that, we first have to think about calculating the derivative of a function on a computer. You'll recall that one way to calculate the slope of a function is:

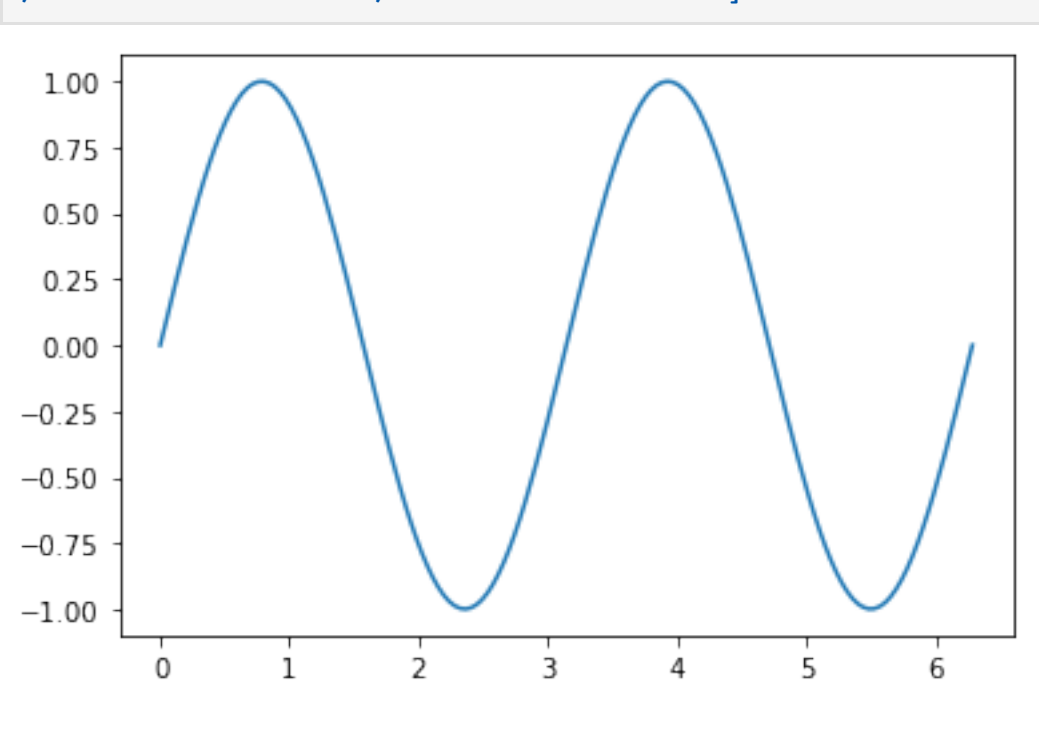
$$\left(\frac{\partial V}{\partial x}\right)_{x-\frac{1}{2}\Delta x} \approx \frac{V(x, y, z) - V(x - \Delta x, y, z)}{\Delta x}$$

The subscript $x - \frac{1}{2}\Delta x$ is to remind us that this formula is calculating the derivative at this location. In other words, the derivative is centered at $x - \frac{1}{2}\Delta x$. Can you figure out what the derivative centered at $x + \frac{1}{2}\Delta x$ would look like?

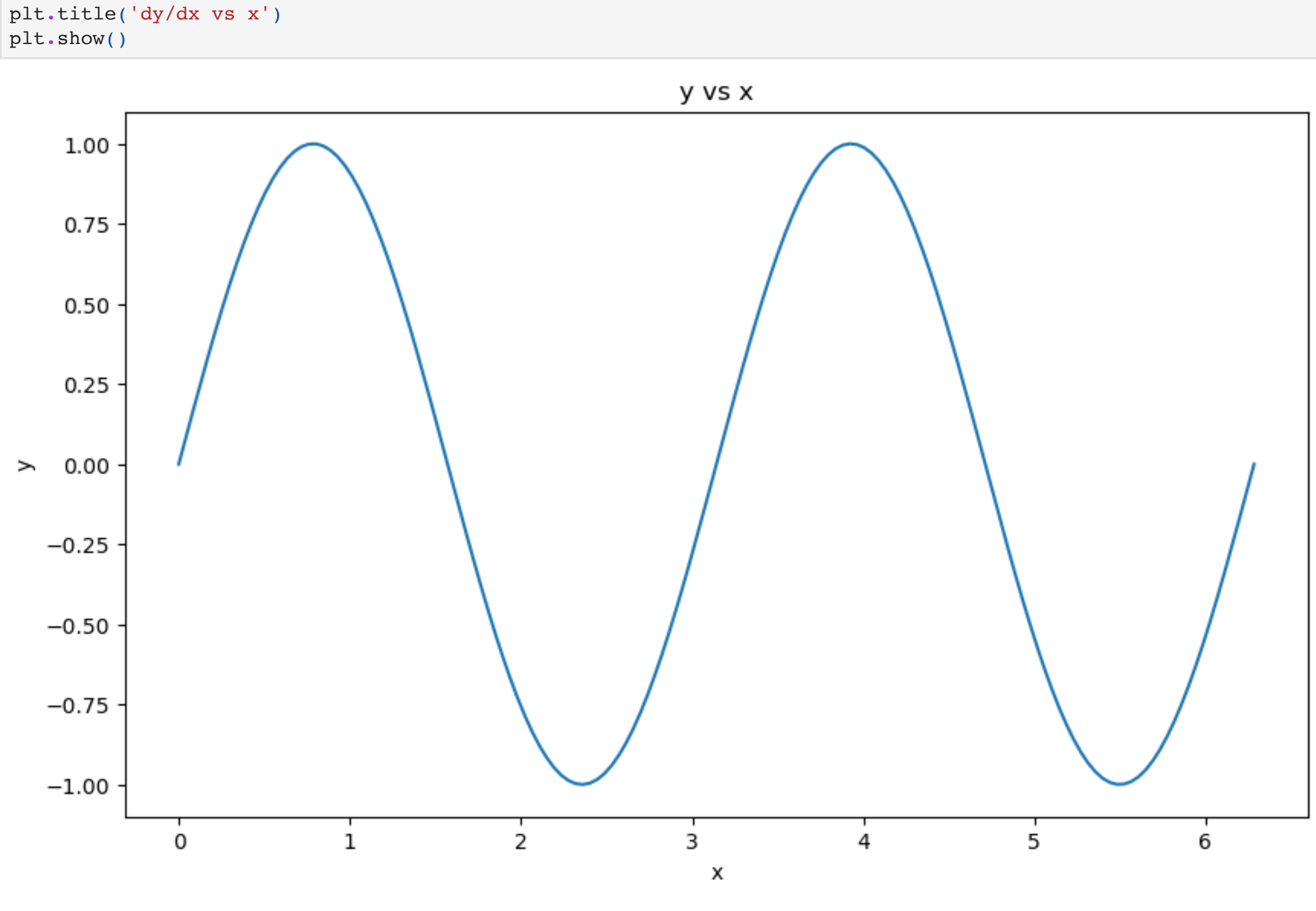
$$\left(\frac{\partial V}{\partial x}\right)_{x+\frac{1}{2}\Delta x} \approx \frac{V(x + \Delta x, y, z) - V(x, y, z)}{\Delta x}$$

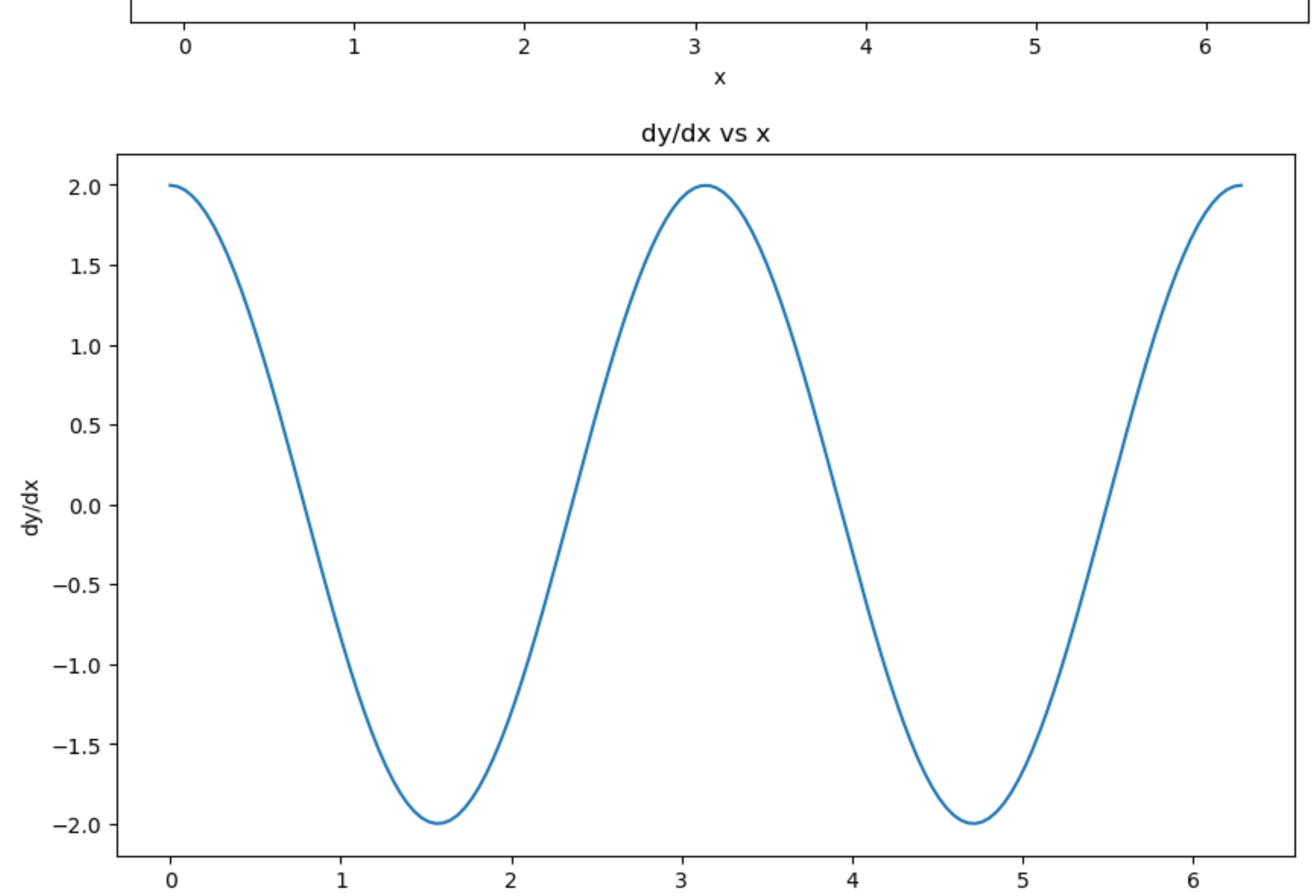
1. In the cell below you will find a dataset. Variable "x" contains the x values, and variable "y" contains the y values. Plot the data.
2. Using the equation above, and **without using a loop**, calculate the derivative of this function and plot it on the same axes as the first plot.
3. Look at the plots and verify that they seem correct.

In [ ]:
```python
x = [0.,0.04216903, 0.08433806, 0.12650709, 0.16867612, 0.21084514
, 0.25301417, 0.2951832,  0.33735223, 0.37952126, 0.42169029, 0.46385932
, 0.50602835, 0.54819738, 0.5903664,  0.63253543, 0.67470446, 0.71687349
, 0.75904252, 0.80121155, 0.84338058, 0.88554961, 0.92771864, 0.96988766
, 1.01205669, 1.05422572, 1.09639475, 1.13856378, 1.18073281, 1.22290184
, 1.26507087, 1.3072399,  1.34940893, 1.39157795, 1.43374698, 1.47591601
, 1.51808504, 1.56025407, 1.6024231,  1.64459213, 1.68676116, 1.72893019
, 1.77109921, 1.81326824, 1.85543727, 1.8976063,  1.93977533, 1.98194436
, 2.02411339, 2.06628242, 2.10845145, 2.15062047, 2.1927895,  2.23495853
, 2.27712756, 2.31929659, 2.36146562, 2.40363465, 2.44580368, 2.48797271
, 2.53014173, 2.57231076, 2.61447979, 2.65664882, 2.69881785, 2.74098688
, 2.78315591, 2.82532494, 2.86749397, 2.90966299, 2.95183202, 2.99400105
, 3.03617008, 3.07833911, 3.12050814, 3.16267717, 3.2048462,  3.24701523
, 3.28918425, 3.33135328, 3.37352231, 3.41569134, 3.45786037, 3.5000294
, 3.54219843, 3.58436746, 3.62653649, 3.66870551, 3.71087454, 3.75304357
, 3.7952126,  3.83738163, 3.87955066, 3.92171969, 3.96388872, 4.00605775
, 4.04822678, 4.0903958,  4.13256483, 4.17473386, 4.21690289, 4.25907192
, 4.30124095, 4.34340998, 4.38557901, 4.42774804, 4.46991706, 4.51208609
, 4.55425512, 4.59642415, 4.63859318, 4.68076221, 4.72293124, 4.76510027
, 4.8072693,  4.84943832, 4.89160735, 4.93377638, 4.97594541, 5.01811444
, 5.06028347, 5.1024525,  5.14462153, 5.18679056, 5.22895958, 5.27112861
, 5.31329764, 5.35546667, 5.3976357,  5.43980473, 5.48197376, 5.52414279
, 5.56631182, 5.60848084, 5.65064987, 5.6928189,  5.73498793, 5.77715696
, 5.81932599, 5.86149502, 5.90366405, 5.94583308, 5.9880021,  6.03017113
, 6.07234016, 6.11450919, 6.15667822, 6.19884725, 6.24101628, 6.28318531]
y = [0.00000000e+00 , 8.42381119e-02,  1.67877401e-01,  2.50323301e-01
,  3.30989730e-01,  4.09303254e-01,  4.84707167e-01,  5.56665446e-01
,  6.24666561e-01,  6.88227114e-01,  7.46895271e-01,  8.00253979e-01
,  8.47923927e-01,  8.89566245e-01,  9.24884909e-01,  9.53628851e-01
,  9.75593738e-01,  9.90623429e-01,  9.98611082e-01,  9.99499915e-01
,  9.93283611e-01,  9.80006358e-01,  9.59762542e-01,  9.32696068e-01
,  8.98999344e-01,  8.58911910e-01,  8.12718735e-01,  7.60748192e-01
,  7.03369724e-01,  6.40991217e-01,  5.74056100e-01,  5.03040194e-01
,  4.28448330e-01,  3.50810759e-01,  2.70679381e-01,  1.88623827e-01
,  1.05227404e-01,  2.10829523e-02, -6.32113721e-02, -1.47056346e-01
, -2.29855942e-01, -3.11021562e-01, -3.89976226e-01, -4.66158669e-01
, -5.39027332e-01, -6.08064216e-01, -6.72778558e-01, -7.32710324e-01
, -7.87433478e-01, -8.36559008e-01, -8.79737698e-01, -9.16662602e-01
, -9.47071234e-01, -9.70747427e-01, -9.87522875e-01, -9.97278325e-01
, -9.99944431e-01, -9.95502239e-01, -9.83983327e-01, -9.65469579e-01
, -9.40092605e-01, -9.08032802e-01, -8.69518072e-01, -8.24822205e-01
, -7.74262931e-01, -7.18199659e-01, -6.57030926e-01, -5.91191562e-01
, -5.21149599e-01, -4.47402944e-01, -3.70475838e-01, -2.90915134e-01
, -2.09286403e-01, -1.26169919e-01, -4.21565323e-02,  4.21565323e-02
,  1.26169919e-01,  2.09286403e-01,  2.90915134e-01,  3.70475838e-01
,  4.47402944e-01,  5.21149599e-01,  5.91191562e-01,  6.57030926e-01
,  7.18199659e-01,  7.74262931e-01,  8.24822205e-01,  8.69518072e-01
,  9.08032802e-01,  9.40092605e-01,  9.65469579e-01,  9.83983327e-01
,  9.95502239e-01,  9.99944431e-01,  9.97278325e-01,  9.87522875e-01
,  9.70747427e-01,  9.47071234e-01,  9.16662602e-01,  8.79737698e-01
,  8.36559008e-01,  7.87433478e-01,  7.32710324e-01,  6.72778558e-01
,  6.08064216e-01,  5.39027332e-01,  4.66158669e-01,  3.89976226e-01
,  3.11021562e-01,  2.29855942e-01,  1.47056346e-01,  6.32113721e-02
, -2.10829523e-02, -1.05227404e-01, -1.88623827e-01, -2.70679381e-01
, -3.50810759e-01, -4.28448330e-01, -5.03040194e-01, -5.74056100e-01
, -6.40991217e-01, -7.03369724e-01, -7.60748192e-01, -8.12718735e-01
, -8.58911910e-01, -8.98999344e-01, -9.32696068e-01, -9.59762542e-01
, -9.80006358e-01, -9.93283611e-01, -9.99499915e-01, -9.98611082e-01
, -9.90623429e-01, -9.75593738e-01, -9.53628851e-01, -9.24884909e-01
, -8.89566245e-01, -8.47923927e-01, -8.00253979e-01, -7.46895271e-01
, -6.88227114e-01, -6.24666561e-01, -5.56665446e-01, -4.84707167e-01
, -4.09303254e-01, -3.30989730e-01, -2.50323301e-01, -1.67877401e-01
, -8.42381119e-02, -4.89858720e-16]
```



In [ ]:
```python
# Importing necessary libraries
import numpy as np
import matplotlib.pyplot as plt

# Define the dataset
x = np.array([0., 0.04216903, 0.08433806, 0.12650709, 0.16867612, 0.21084514, 0.25301417, 0.2951832, 0.33735223
              0.42169029, 0.46385932, 0.50602835, 0.54819738, 0.5903664, 0.63253543, 0.67470446, 0.71687349, 0.
              0.84338058, 0.88554961, 0.92771864, 0.96988766, 1.01205669, 1.05422572, 1.09639475, 1.13856378, 1
              1.26507087, 1.3072399, 1.34940893, 1.39157795, 1.43374698, 1.47591601, 1.51808504, 1.56025407, 1.
              1.68676116, 1.72893019, 1.77109921, 1.81326824, 1.85543727, 1.8976063, 1.93977533, 1.98194436, 2.
              2.10845145, 2.15062047, 2.1927895, 2.23495853, 2.27712756, 2.31929659, 2.36146562, 2.40363465, 2.
              2.53014173, 2.57231076, 2.61447979, 2.65664882, 2.69881785, 2.74098688, 2.78315591, 2.82532494, 2
              2.95183202, 2.99400105, 3.03617008, 3.07833911, 3.12050814, 3.16267717, 3.2048462, 3.24701523, 3.
              3.37352231, 3.41569134, 3.45786037, 3.5000294, 3.54219843, 3.58436746, 3.62653649, 3.66870551, 3.
              3.7952126, 3.83738163, 3.87955066, 3.92171969, 3.96388872, 4.00605775, 4.04822678, 4.0903958, 4.1
              4.21690289, 4.25907192, 4.30124095, 4.34340998, 4.38557901, 4.42774804, 4.46991706, 4.51208609, 4
              4.63859318, 4.68076221, 4.72293124, 4.76510027, 4.8072693, 4.84943832, 4.89160735, 4.93377638, 4.
              5.06028347, 5.1024525, 5.14462153, 5.18679056, 5.22895958, 5.27112861, 5.31329764, 5.35546667, 5.
              5.48197376, 5.52414279, 5.56631182, 5.60848084, 5.65064987, 5.6928189, 5.73498793, 5.77715696, 5.
              5.90366405, 5.94583308, 5.9880021, 6.03017113, 6.07234016, 6.11450919, 6.15667822, 6.19884725, 6.

y = np.array([0.00000000e+00 , 8.42381119e-02,  1.67877401e-01,  2.50323301e-01,  3.30989730e-01,  4.09303254e-
              5.56665446e-01,  6.24666561e-01,  6.88227114e-01,  7.46895271e-01,  8.00253979e-01,  8.47923927e-
              9.24884909e-01,  9.53628851e-01,  9.75593738e-01,  9.90623429e-01,  9.98611082e-01,  9.99499915e-
              9.80006358e-01,  9.59762542e-01,  9.32696068e-01,  8.98999344e-01,  8.58911910e-01,  8.12718735e-
              7.03369724e-01,  6.40991217e-01,  5.74056100e-01,  5.03040194e-01,  4.28448330e-01,  3.50810759e-
              1.88623827e-01,  1.05227404e-01,  2.10829523e-02, -6.32113721e-02, -1.47056346e-01, -2.29855942e-
             -3.89976226e-01, -4.66158669e-01, -5.39027332e-01, -6.08064216e-01, -6.72778558e-01, -7.32710324e
             -8.36559008e-01, -8.79737698e-01, -9.16662602e-01, -9.47071234e-01, -9.70747427e-01, -9.87522875e
             -9.99944431e-01, -9.95502239e-01, -9.83983327e-01, -9.65469579e-01, -9.40092605e-01, -9.08032802e
             -8.24822205e-01, -7.74262931e-01, -7.18199659e-01, -6.57030926e-01, -5.91191562e-01, -5.21149599e
             -3.70475838e-01, -2.90915134e-01, -2.09286403e-01, -1.26169919e-01, -4.21565323e-02,  4.21565323e
              2.09286403e-01,  2.90915134e-01,  3.70475838e-01,  4.47402944e-01,  5.21149599e-01,  5.91191562e-
              7.18199659e-01,  7.74262931e-01,  8.24822205e-01,  8.69518072e-01,  9.08032802e-01,  9.40092605e-
              9.83983327e-01,  9.95502239e-01,  9.99944431e-01,  9.97278325e-01,  9.87522875e-01,  9.70747427e-
              9.16662602e-01,  8.79737698e-01,  8.36559008e-01,  7.87433478e-01,  7.32710324e-01,  6.72778558e-
              5.39027332e-01,  4.66158669e-01,  3.89976226e-01,  3.11021562e-01,  2.29855942e-01,  1.47056346e-
             -2.10829523e-02, -1.05227404e-01, -1.88623827e-01, -2.70679381e-01, -3.50810759e-01, -4.28448330e
             -5.74056100e-01, -6.40991217e-01, -7.03369724e-01, -7.60748192e-01, -8.12718735e-01, -8.58911910e
             -9.32696068e-01, -9.59762542e-01, -9.80006358e-01, -9.93283611e-01, -9.99499915e-01, -9.98611082e
             -9.75593738e-01, -9.53628851e-01, -9.24884909e-01, -8.89566245e-01, -8.47923927e-01, -8.00253979e
             -6.88227114e-01, -6.24666561e-01, -5.56665446e-01, -4.84707167e-01, -4.09303254e-01, -3.30989730e
             -1.67877401e-01, -8.42381119e-02, -4.89858720e-16])

# Plotting the dataset
plt.figure(figsize=(10, 6))
plt.plot(x, y, label='Original Data')
# Adding labels and legend
plt.xlabel('x')
plt.ylabel('y')
plt.title('y vs x')
plt.show()

# Calculating the derivative using numpy gradient function
dy_dx = np.gradient(y, x)

# Plotting the dataset
plt.figure(figsize=(10, 6))
# Plotting the derivative
plt.plot(x, dy_dx, label='Derivative')
# Adding labels and legend
plt.xlabel('x')
plt.ylabel('dy/dx')
plt.title('dy/dx vs x')
plt.show()
```





Now that we understand how to calculate a discrete first derivative, let's see if we can use the expressions for them to calculate a second order derivative. There was a reason we wrote expressions for the first derivative centered at two different points above. If we take the difference of those two expressions and divide by Δx we arrive at a second derivative:

$$\frac{\partial^2 V}{\partial x^2} \approx \frac{1}{\Delta x}\left(\left(\frac{\partial V}{\partial x}\right)_{x+\frac{1}{2}\Delta x} - \left(\frac{\partial V}{\partial x}\right)_{x-\frac{1}{2}\Delta x}\right)$$

After plugging in the expressions above you will arrive at the following equation:

$$\frac{\partial^2 V}{\partial x^2} \approx \frac{V(x + \Delta x, y, z) + V(x - \Delta x, y, z) - 2V(x, y, z)}{(\Delta x)^2}$$

and a similar expression for the derivative with respect to y:

$$\frac{\partial^2 V}{\partial y^2} \approx \frac{V(x, y + \Delta y, z) + V(x, y - \Delta y, z) - 2V(x, y, z)}{(\Delta x)^2}$$

If we insert these expressions into Laplace's equation above and solve for $V(x, y, z)$ we arrive at (for two dimensions, and assuming that $\Delta x = \Delta y$):

$$V(x, y) = \frac{1}{4}\left(V(x + \Delta x, y) + V(x - \Delta x, y) + V(x, y + \Delta y) + V(x, y - \Delta y)\right)$$

This is an important equation for your work this week. It tells you how to update the value of the potential at one point using the value of the potential at it's neighboring points. All of the methods we will use this week will involve this expression so study it until you understand it!