PHY1112 Lab 7

Measurement Errors, Linear Regression, and Vectors – Uncertain Observations, Finding Trends, and Shifting Perspectives

February 27th, 2024

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Part | 1 | 2 | 3 | Total |
| Points | 5 | 5 | 12 | 22 |
| Score |  |  |  |  |

Objectives

1. Add measurement error to a data set and show how this looks when plotting.
2. Do a regression of a data set to plot and observe the trend.
3. Create a system which can be observed from multiple reference frames.

Part 1: Visualizing measurement uncertainty

Let’s return once again to the temperature data for Ottawa from previous labs. Now, we will consider that each temperature measurement has a measurement error.

Consider the script below that was used to plot the high/low temperatures in Ottawa in Sept 2023, along with shaded regions that indicate the errors (see Figure 1). It assumes an error of for each temperature measurement, and calls a function, plot\_errors() that has not been provided.

* 1. (5 points) Write the plot\_errors() function.

Then, use a modified version of the given script and your plot\_errors() function to plot the high and low temperatures with errors for Ottawa in March 2022 (days 59 to 89, inclusive), Assume a measurement error of .

Take a snapshot of your figure and include the code that you wrote

# Part 1

# import given plot program as a function and make changes for March

*def* temperature\_plot():

    '''

    () -> None

    This function plots the high and low temperatures for Ottawa in March 2023.

    Preconditions: high\_temperatures and low\_temperatures data exist and are both np.arrays of the same length

    '''

    #initialize the figure

    plt.figure(1)

    # specify the month and the days of the month

    month = "March 2023"

    day = range(1,32)

    measurement\_error = 1.5 # +/- in degrees Celsius

    # specify the high and low temperatures so that they 'exist'

    high\_temperatures = (load\_data()[:,0])[59:90]

    low\_temperatures = (load\_data()[:,1])[59:90]

    plt.plot(day, high\_temperatures, 'r', *label*="Highs")

    plt.plot(day, low\_temperatures,  'b', *label*="Lows")

    # the plot\_errors function called below is to be defined by the student

    plot\_errors(day, high\_temperatures, measurement\_error, 'mistyrose', "Highs Uncertainty")

    plot\_errors(day, low\_temperatures, measurement\_error, 'lightblue', "Lows Uncertainty")

    plt.title(*f*"High and Low Temperatures for Ottawa in {month}")

    plt.xlabel("Day of the Month")

    plt.ylabel("Temperature (Celcius)")

    plt.xlim(1,day[-1])

    plt.grid()

    plt.legend()

    plt.show()

# define a function to load the necessary data

*def* load\_data():

    '''

    () -> np.array

    This function loads the high and low temperatures for Ottawa in September 2023.

    Preconditions: None

    '''

    return np.genfromtxt('weather\_data\_lab5.csv', *usecols*=(9,11), *skip\_header*=1, *delimiter*=',', *dtype*=*float*)

# define a function to plot the errors

*def* plot\_errors(*day*, *temperature*, *measurement\_error*, *color*, *label*):

    '''

    (range, np.array, float) -> None

    This function plots the errors as a shaded region for each day in the given temperature data.

    Preconditions: day and error are numbers, temperature is an np.array.

    '''

    # calculate the upper and lower bounds of the error

    upper\_bound = temperature + measurement\_error

    lower\_bound = temperature - measurement\_error

    # fill the area between the upper and lower bounds

    plt.fill\_between(day, upper\_bound, lower\_bound, *color*=color, *label*=label)

A graph of high and low temperatures

Description automatically generated

Figure 1: Plot of high/low temperature data for the month of March 2023, where the shaded regions represent the uncertainty of corresponding high or low temperature data.

# assume high\_temperatures and low\_temperatures data exist

month = "September 2023"

day = range(1,31)

measurement\_error = 2.0 # +/- in degrees Celsius

plt.plot(day, high\_temperatures, 'r', label="Highs")

plt.plot(day, low\_temperatures,  'b', label="Lows")

# the plot\_errors function called below is to be defined by the student

plot\_errors(day, high\_temperatures, measurement\_error,

            'r:', 'mistyrose', "Highs Uncertainty")

plot\_errors(day, low\_temperatures, measurement\_error,

            'b:', 'lightblue', "Lows Uncertainty")

plt.title(f"High and Low Temperatures for Ottawa in {month}")

plt.xlabel("Day of the Month")

plt.ylabel("Temperature (Celcius)")

plt.xlim(1,day[-1])

plt.grid()

plt.legend()

plt.show()

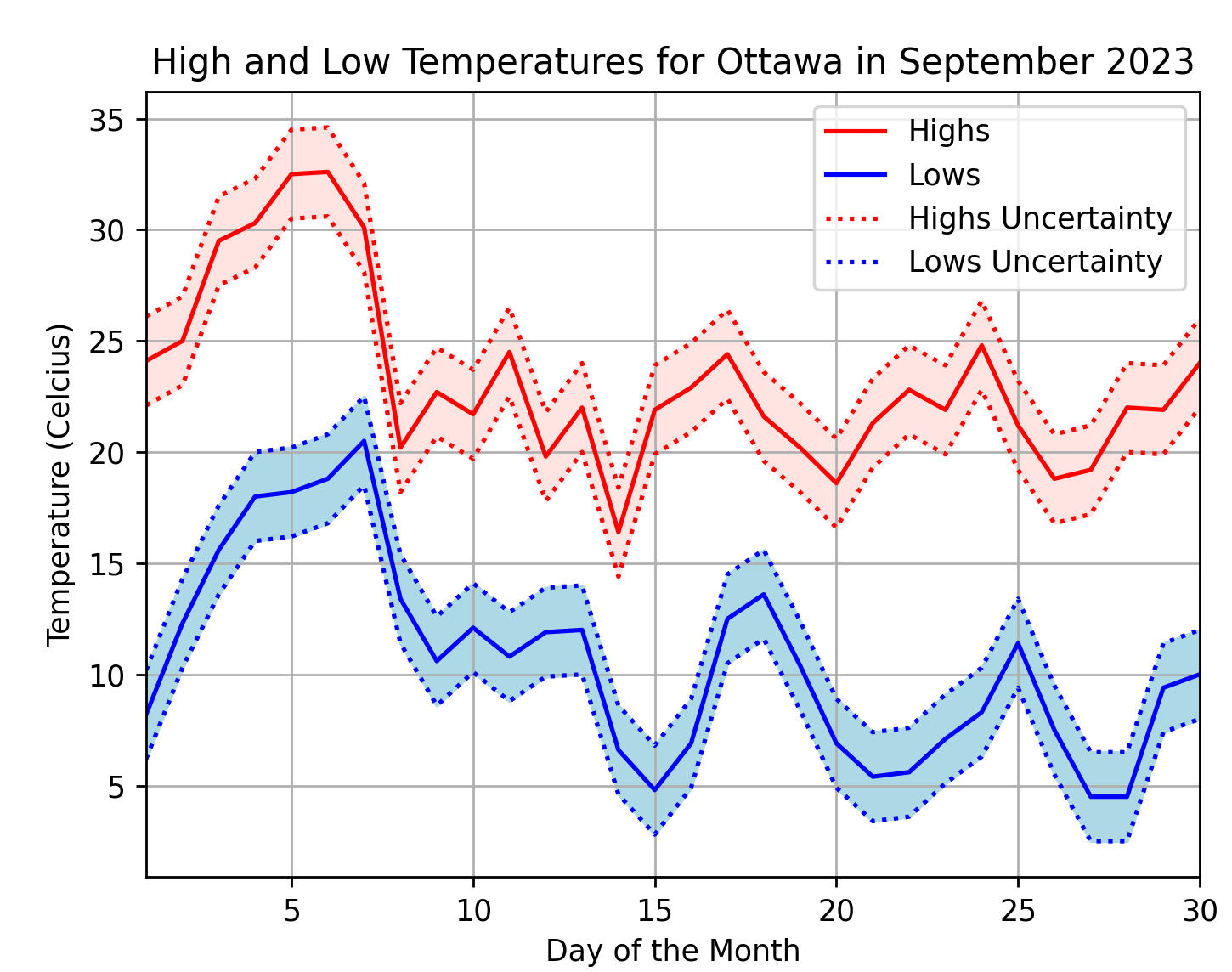


Figure 1: Plot of high/low daily temperatures in Ottawa in September 2023, where the shaded regions represent the uncertainty in the measurements.

Part 2: Fitting to a line – How does temperature change over a month?

Last time we calculated the mean and standard deviation of high and low temperatures over a month. While these are interesting statistics, we know that, depending on the time of year, the temperature can either increase, decrease, or stay the same over the course of a month.

Consider the September 2023 Ottawa high/low temperatures plotted in Figure 1. Qualitatively, we would say that it was warmer in the beginning of the month, and cooler near the end.

To evaluate this more quantitatively, we will fit the data to a line and determine its slope, which tells us the average rate of temperature change over the month.

Below find an example script where we do this for the September 2023 data. The plot it generates is shown in Figure 2. Our fits indicate that the temperature is dropping on average over the month of September (negative slopes), and that the lows are dropping slightly faster than the highs.

1. (5 points) Use np.polynomial.polynomial.polyfit() to fit the high and low temperature data from March 2022 in Ottawa to straight lines (polynomials of order 1). Report the slope and intercept values for each fit.

Highs: m = 0.29, b = -1.49

Lows: m = 0.31, b = -11.52

Plot the high and low temperatures, along with your fits, in a similar manner to the September 2023 data in Figure 2. Take a snapshot of your figure and include it here.

A graph of high and low temperatures

Description automatically generated

Figure 2: Plot of high/low temperatures in the month of March 2022. Shaded regions represent uncertainty measurements and dotted lines represent linear fits of each set of data.

What claims can be made about the temperature change over the month?

Over the month the temperature has been increasing, the lows more so than the highs.

# insert this code snippet into the script above

def line(x, intercept, slope):

    return slope\*x + intercept

b\_highs, m\_highs = np.polynomial.polynomial.polyfit( day, high\_temperatures, 1 )

b\_lows,  m\_lows  = np.polynomial.polynomial.polyfit( day, low\_temperatures,  1 )

# plot the fits

plt.plot(day, line(day, b\_highs, m\_highs), "r--",

         label=f"Highs fit to a line with m={m\_highs:.2f} and b={b\_highs:.2f}")

plt.plot(day, line(day, b\_lows,  m\_lows),  "b--",

         label=f"Lows fit to a line with m={m\_lows:.2f} and b={b\_lows:.2f}")

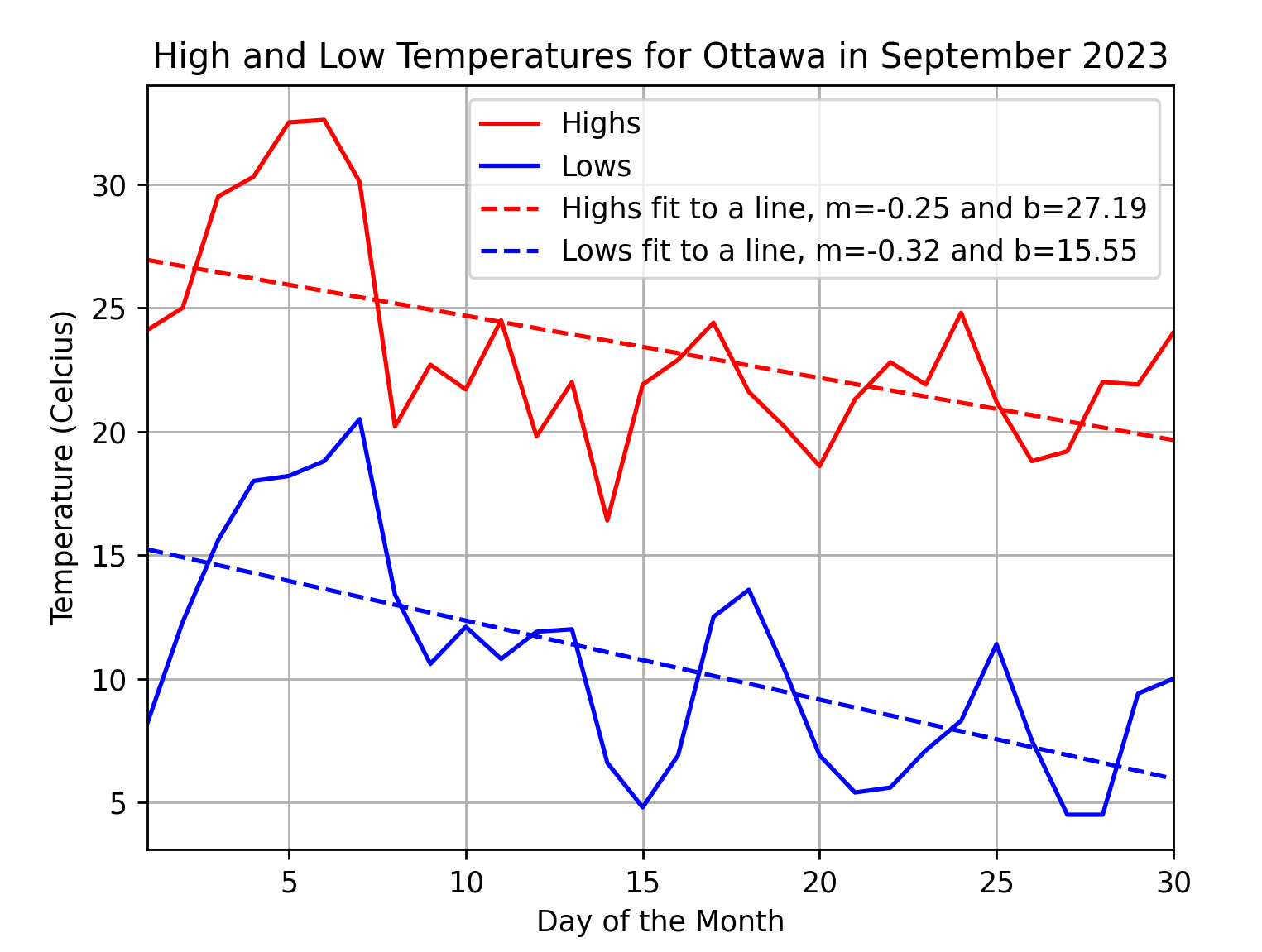


Figure 2: Plot of high/low daily temperatures in Ottawa in September 2023, where the dashed lines are fits of the data to straight lines.

Part 3: Ramping Down

1. (12 points) Consider a box at the top of a ramp. The situation appears as follows:

m

**rise**

**run**

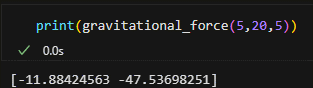
+x

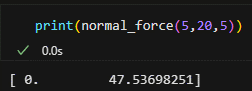
+y

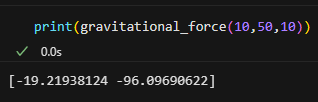
In this question, complete parts ‘a’ through ‘d’ for the following values:

|  |  |  |
| --- | --- | --- |
| **m (kg)** | **rise (m)** | **run (m)** |
| 5 | 5 | 20 |
| 10 | 10 | 50 |
| 10 | 50 | 500 |

1. (3 points) Consider the x-direction to be up the incline of the ramp, and the y-direction perpendicular to the incline of the ramp, as indicated in the diagram above. Define vectors for the gravitational and normal forces as NumPy arrays with length 2. Print your results to the console and include a snapshot here. Assume g = 9.8 m/s2

****

****

****

**A screen shot of a computer

Description automatically generated**

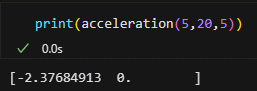
**A screenshot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

1. (3 points) Assuming no friction, determine the acceleration vector for the box. Print your results to the console and include a snapshot here.

****

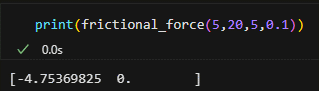
**A screen shot of a computer

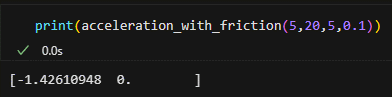
Description automatically generated**

**A screen shot of a computer

Description automatically generated**

1. (3 points) Assuming a dynamic coefficient of friction of 0.1, define a vector for your friction force and determine a new acceleration vector. Print your results to the console and include a shapshot here.

****

****

**A screenshot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

1. (3 points) For the situation described in part ‘c’, what is the velocity vector after 10 seconds? Print your results to the console and include a snapshot here.

**A screen shot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**CODE:**

'''

Filename:       lab7.py

Author:         Patrick Geraghty

Date Created:   2024-02-27

Date Modified:  2024-02-27

Description:

'''

import numpy as np

import matplotlib.pyplot as plt

# Part 1

# import given plot program as a function and make changes for March

*def* temperature\_plot():

    '''

    () -> None

    This function plots the high and low temperatures for Ottawa in March 2022.

    Preconditions: high\_temperatures and low\_temperatures data exist and are both np.arrays of the same length

    '''

    #initialize the figure

    plt.figure(1)

    # specify the month and the days of the month

    month = "March 2022"

    day = range(1,32)

    measurement\_error = 1.5 # +/- in degrees Celsius

    # specify the high and low temperatures so that they 'exist'

    high\_temperatures = (load\_data()[:,0])[59:90]

    low\_temperatures = (load\_data()[:,1])[59:90]

    plt.plot(day, high\_temperatures, 'r', *label*="Highs")

    plt.plot(day, low\_temperatures,  'b', *label*="Lows")

    # the plot\_errors function called below is to be defined by the student

    plot\_errors(day, high\_temperatures, measurement\_error, 'mistyrose', "Highs Uncertainty")

    plot\_errors(day, low\_temperatures, measurement\_error, 'lightblue', "Lows Uncertainty")

    plt.title(*f*"High and Low Temperatures for Ottawa in {month}")

    plt.xlabel("Day of the Month")

    plt.ylabel("Temperature (Celcius)")

    plt.xlim(1,day[-1])

    plt.grid()

    plt.legend()

    plt.show()

# define a function to load the necessary data

*def* load\_data():

    '''

    () -> np.array

    This function loads the high and low temperatures for Ottawa in 2022.

    Preconditions: None

    '''

    return np.genfromtxt('weather\_data\_lab5.csv', *usecols*=(9,11), *skip\_header*=1, *delimiter*=',', *dtype*=*float*)

# define a function to plot the errors

*def* plot\_errors(*day*, *temperature*, *measurement\_error*, *color*, *label*):

    '''

    (range, np.array, float) -> None

    This function plots the errors as a shaded region for each day in the given temperature data.

    Preconditions: day and error are numbers, temperature is an np.array.

    '''

    # calculate the upper and lower bounds of the error

    upper\_bound = temperature + measurement\_error

    lower\_bound = temperature - measurement\_error

    # fill the area between the upper and lower bounds

    plt.fill\_between(day, upper\_bound, lower\_bound, *color*=color, *label*=label)

# Part 2

# import code from lab worksheet

*def* line(*x*, *intercept*, *slope*):

    return slope\*x + intercept

# define a function to plot the linear fits

*def* linear\_fit(*day*, *b*, *m*, *label*, *color*):

    # plot the fits

    plt.plot(day, line(day, b, m), color,

*label*=*f*"{label} fit to a line with m={m*:.2f*} and b={b*:.2f*}")

# clone temperature\_plot() and edit to account for linear fits

*def* temperature\_plot\_with\_fits():

    '''

    () -> None

    This function plots the high and low temperatures for Ottawa in March 2022.

    Preconditions: high\_temperatures and low\_temperatures data exist and are both np.arrays of the same length

    '''

    #initialize the figure

    plt.figure(2)

    # specify the month and the days of the month

    month = "March 2022"

    day = range(1,32)

    measurement\_error = 1.5 # +/- in degrees Celsius

    # specify the high and low temperatures so that they 'exist'

    high\_temperatures = (load\_data()[:,0])[59:90]

    low\_temperatures = (load\_data()[:,1])[59:90]

    plt.plot(day, high\_temperatures, 'r', *label*="Highs")

    plt.plot(day, low\_temperatures,  'b', *label*="Lows")

    # the plot\_errors function called below is to be defined by the student

    plot\_errors(day, high\_temperatures, measurement\_error, 'mistyrose', "Highs Uncertainty")

    plot\_errors(day, low\_temperatures, measurement\_error, 'lightblue', "Lows Uncertainty")

    # define the linear fits

    b\_highs, m\_highs = np.polynomial.polynomial.polyfit( day, high\_temperatures, 1 )

    b\_lows,  m\_lows  = np.polynomial.polynomial.polyfit( day, low\_temperatures,  1 )

    # plot the linear fits

    linear\_fit(day, b\_highs, m\_highs, 'Highs', 'r--')

    linear\_fit(day, b\_lows, m\_lows, 'Lows', 'b--')

    plt.title(*f*"High and Low Temperatures for Ottawa in {month}")

    plt.xlabel("Day of the Month")

    plt.ylabel("Temperature (Celcius)")

    plt.xlim(1,day[-1])

    plt.grid()

    plt.legend()

    plt.show()

# Part 3

# define functions to calculate gravitational force and normal force on an incline

*def* gravitational\_force(*mass*, *base*, *height*):

    '''

    (int, int, int) -> np.array

    Calculates the gravitational force vector acting on an object on an inclined surface given the objects mass, and the base and height lengths oh the inclined surface. The x and y vector coordinates are given with the surface representing the positive x-axis and perpendicular to the surface representing the positive y-axis.

    Preconditions: mass, base, and height are all numbers, the mass is positioned at the topmost point of the surface.

    '''

    angle = np.arctan(height/base)

    return mass\*9.8\*np.array([-np.sin(angle), -np.cos(angle)])

# define a function to calculate the normal force

*def* normal\_force(*mass*, *base*, *height*):

    '''

    (int, int, int) -> np.array

    Calculates the normal force vector acting on an object on an inclined surface given the objects mass, and the base and height lengths oh the inclined surface.

    Preconditions: mass, base, and height are all numbers, the mass is positioned at the topmost point of the surface.

    '''

    # use standard equation for normal force Fn = mg\*cos(angle) where g is the acceleration due to gravity (9.8 m/s^2)

    # note that the x component is 0 because the normal force is perpendicular to the surface (solely the y-axis)

    angle = np.arctan(height/base)

    return mass\*9.8\*np.array([0, np.cos(angle)])

# define a function to calculate the acceleration of the mass

*def* acceleration(*mass*, *base*, *height*):

    '''

    (int, int, int) -> np.array

    Calculates the acceleration vector acting on an object on an inclined surface given the objects mass, and the base and height lengths oh the inclined surface. The x and y vector coordinates are given with the surface representing the positive x-axis and perpendicular to the surface representing the positive y-axis.

    Preconditions: mass, base, and height are all numbers, the mass is positioned at the topmost point of the surface.

    '''

    # use standard equation for acceleration a = (Fg + Fn)/m

    return (gravitational\_force(mass, base, height) + normal\_force(mass, base, height))/mass

# define a function to calculate the frictional force vector

*def* frictional\_force(*mass*, *base*, *height*, *friction\_coefficient*):

    '''

    (int, int, int, float) -> np.array

    Calculates the frictional force vector acting on an object on an inclined surface given the objects mass, and the base and height lengths oh the inclined surface. The x and y vector coordinates are given with the surface representing the positive x-axis and perpendicular to the surface representing the positive y-axis.

    Preconditions: mass, base, and height are all numbers, the mass is positioned at the topmost point of the surface.

    '''

    # use standard equation for frictional force Ff = -u\*|Fn|, use np.linalg.norm to calculate the magnitude of the normal force

    return np.array([-friction\_coefficient\*np.linalg.norm(normal\_force(mass, base, height)), 0])

# define a function to calculate acceleration of the mass taking friction into consideration

*def* acceleration\_with\_friction(*mass*, *base*, *height*, *friction\_coefficient*):

    '''

    (int, int, int, float) -> np.array

    Calculates the acceleration vector acting on an object on an inclined surface given the objects mass, and the base and height lengths oh the inclined surface. The x and y vector coordinates are given with the surface representing the positive x-axis and perpendicular to the surface representing the positive y-axis.

    Preconditions: mass, base, and height are all numbers, the mass is positioned at the topmost point of the surface.

    '''

    # use standard equation for acceleration with friction a = (Fg + Fn - Ff)/m

    return (gravitational\_force(mass, base, height) + normal\_force(mass, base, height))/mass - frictional\_force(mass, base, height, friction\_coefficient)/mass

# define a function to calculate the velocity of the mass after t seconds

*def* velocity\_wrt\_time(*mass*, *base*, *height*, *friction\_coefficient*, *t*):

    '''

    (int, int, int, float, int) -> np.array

    Calculates the velocity vector acting on an object on an inclined surface given the objects mass, and the base and height lengths oh the inclined surface. The x and y vector coordinates are given with the surface representing the positive x-axis and perpendicular to the surface representing the positive y-axis.

    Preconditions: mass, base, and height are all numbers, the mass is positioned at the topmost point of the surface.

    '''

    # use standard equation for velocity v = at

    return acceleration\_with\_friction(mass, base, height, friction\_coefficient)\*t