Textbook Exercises

December 10, 2024

1 Chapter 2: Python Programming for Physicists

1.0.1 Exercise 1

[1]:
```
import numpy as np

def timeToFall(h):
    g = 9.81
    return np.sqrt((2 * h) / g)

timeToFall(100)
```

[1]:
```
4.515236409857309
```

1.0.2 Exercise 2

b) Write a program that asks the user to enter the desired value of T and then calculates and prints out the correct altitude in meters.

[2]:
```
def altitude(T):
    G = 6.67e-11
    M = 5.97e24
    R = 6371e3
    return ((G * M * T**2) / (4 * np.pi**2))**(1/3) - R
```

c) Use your program to calculate the altitudes of satellites that orbit the Earth once a day (so-called "geosynchronous" orbit), once every 90 minutes, and once every 45 minutes. What do you conclude from the last of these calculations?

[3]:
```
T_geosynch = (24 * 60 * 60)
T_90 = (90 * 60)
T_45 = (45 * 60)

altitude(T_geosynch), altitude(T_90), altitude(T_45)
```

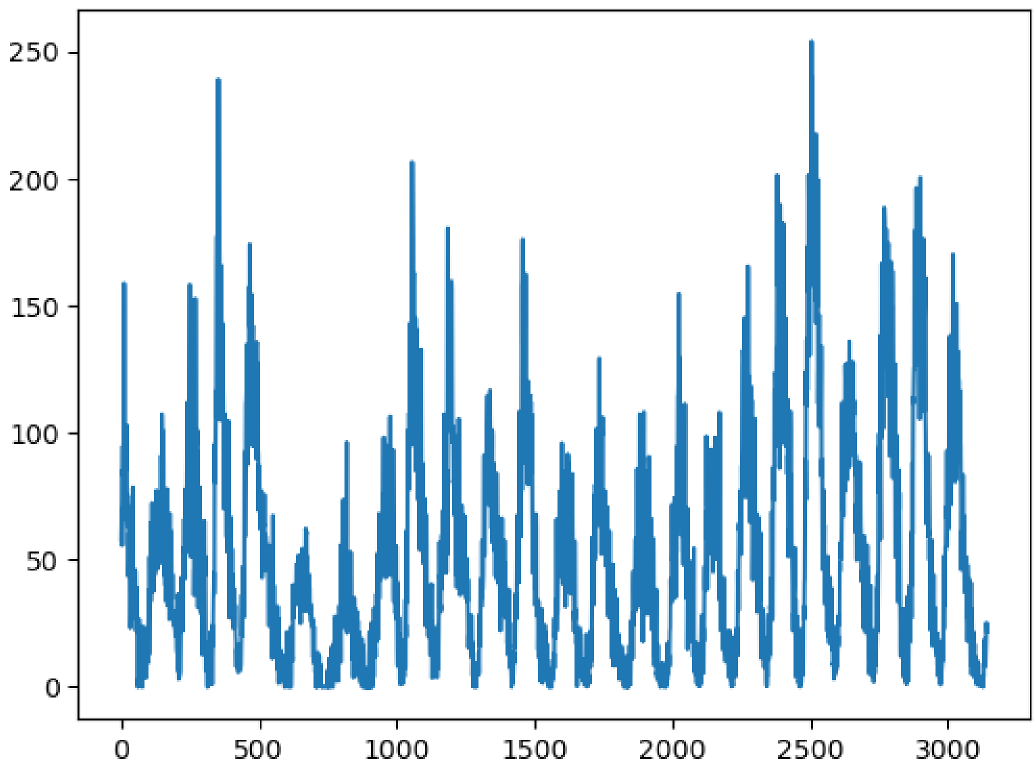
[3]:
```
(35855910.176174976, 279321.6253728606, -2181559.8978108233)
```

2 Chapter 3: Graphics and Visualization

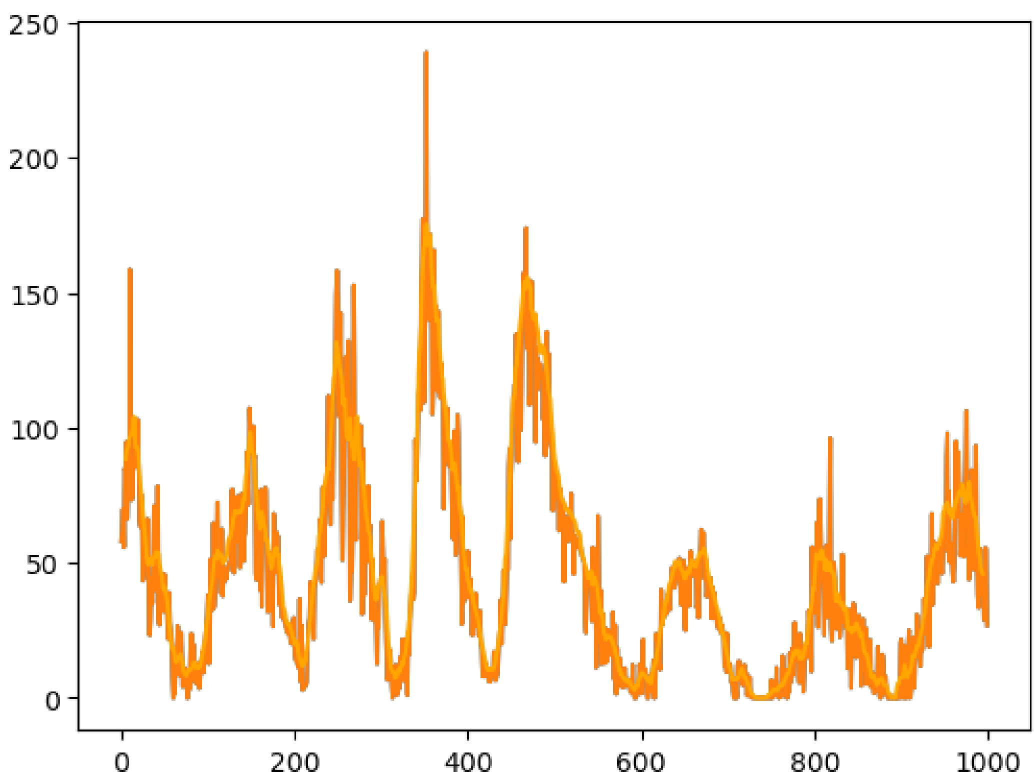
2.0.1 Exercise 1

[4]:
```
from matplotlib import pyplot as plt

# Part a
months, num = np.loadtxt("sunspots.txt", unpack=True)
plt.plot(months, num)

# Part b
plt.figure()
plt.plot(months[:1000], num[:1000])

# Part c

# For r < k < n - r
def runnin_avg(r, data):

    run_avgs = []
    for k in range(r, len(data) - r):
        avg = 0
        for m in range(-r, r + 1):
            avg += data[k + m]
        run_avgs.append(avg / (2 * r))
    return run_avgs

k = np.arange(5,956)
r = 5
run_avg = runnin_avg(r, num)

plt.plot(months[:1000], num[:1000], label="Original Data")
plt.plot(months[r:1000 - r], run_avg[:1000 - 2 * r], label="Running Average",␣
 ↪color="orange")
```

[4]:
```
[<matplotlib.lines.Line2D at 0x7f0c24390dd0>]
```



250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000

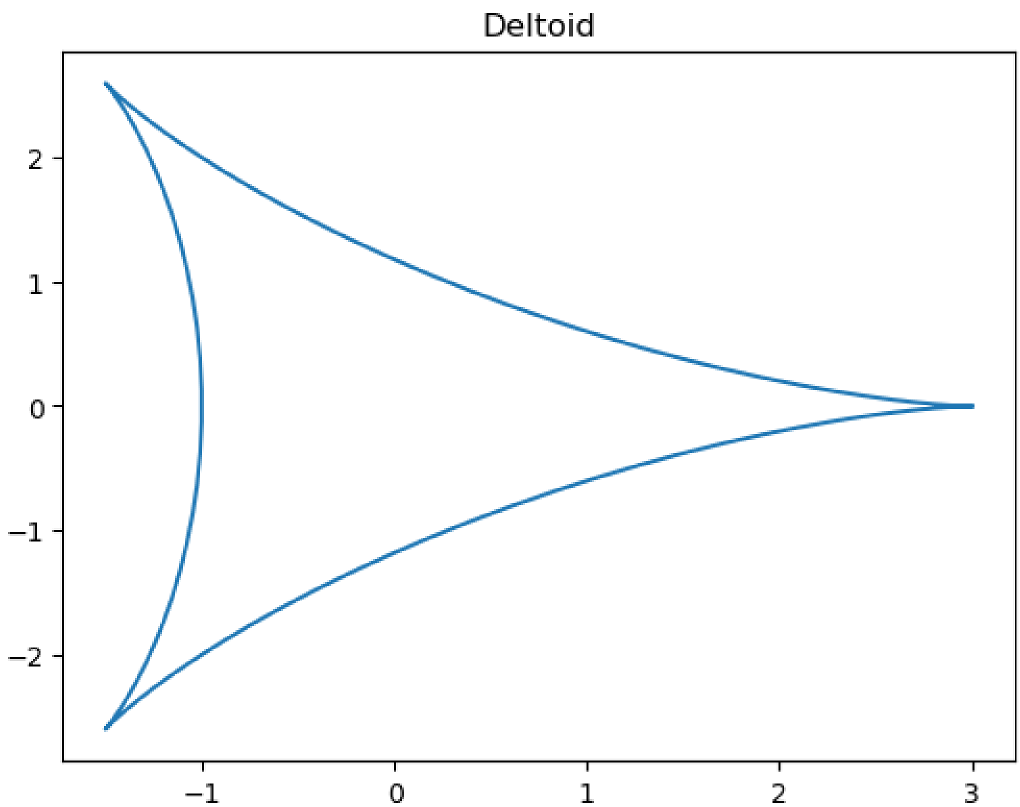


250
200
150
100
50
0
0
200
400
600
800
1000

2.0.2 Exercise 2

[5]:
```
# Part a

def deltoid(thetas):
    x_vals, y_vals = [], []
    for theta in thetas:
        x = 2 * np.cos(theta) + np.cos(2 * theta)
        y = 2 * np.sin(theta) - np.sin(2 * theta)
        x_vals.append(x)
        y_vals.append(y)
    return x_vals, y_vals

N = 100
thetas = np.linspace(0, 2 * np.pi, N)
x_vals, y_vals = deltoid(thetas)
plt.title("Deltoid")
plt.plot(x_vals, y_vals)
```

[5]:
```
[<matplotlib.lines.Line2D at 0x7f0c243cc2d0>]
```

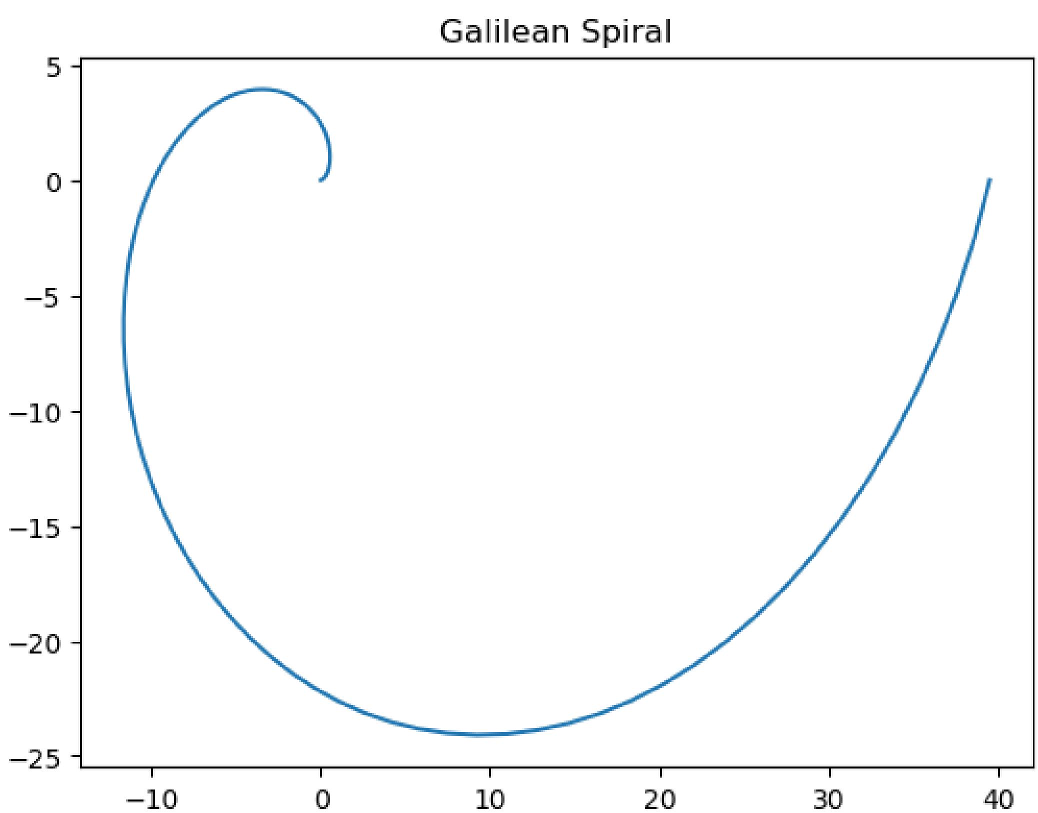


[6]:

```python
# Part b

def galileanSpiral(thetas):
    x_vals, y_vals = [], []
    for theta in thetas:
        x = theta**2 * np.cos(theta)
        y = theta**2 * np.sin(theta)
        x_vals.append(x)
        y_vals.append(y)
    return x_vals, y_vals

N = 100
thetas = np.linspace(0, 2 * np.pi, N)
x_vals, y_vals = galileanSpiral(thetas)
plt.title("Galilean Spiral")
plt.plot(x_vals, y_vals)
```

[6]:

```
[<matplotlib.lines.Line2D at 0x7f0c24463d90>]
```

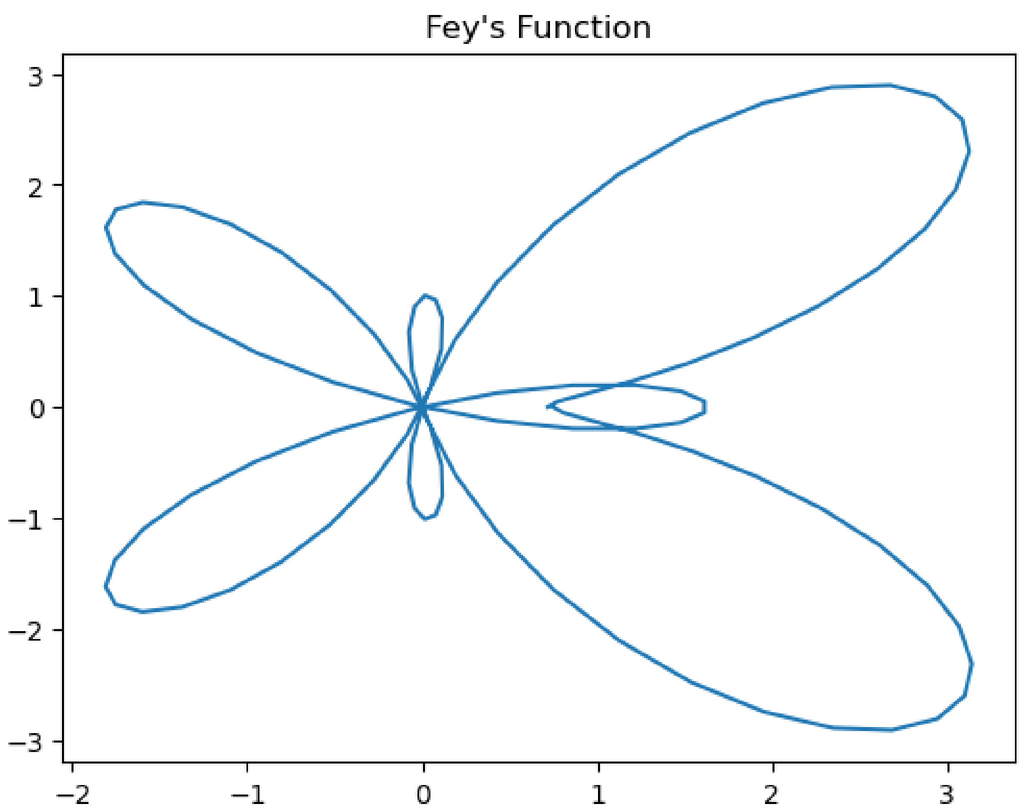


[7]:

```python
# Part c

def feys(thetas):
    x_vals, y_vals = [], []
    for theta in thetas:
        r = np.exp(np.cos(theta)) - 2 * np.cos(4 * theta) + \
            (np.sin(theta/12))**5
        x = r * np.cos(theta)
        y = r * np.sin(theta)
        x_vals.append(x)
        y_vals.append(y)
    return x_vals, y_vals

N = 100
thetas = np.linspace(0, 2 * np.pi, N)
x_vals, y_vals = feys(thetas)
plt.title("Fey's Function")
plt.plot(x_vals, y_vals)
```

[7]:

```
[<matplotlib.lines.Line2D at 0x7f0c8940c410>]
```



3 Chapter 4: Accuracy and Speed

3.0.1 Exercise 1

[8]:
```python
import numpy as np


def factorial_int(n):
    result = 1
    for k in range(1,int(n+1)):
        result *= k
    return result

def factorial_float(n):
    result = 1.
    for k in range(1,int(n+1)):
        result *= float(k)
    return result


n = 200
n_float = 200.

# print(f"{factorial_int(n):e}")
# print(f"{factorial_float(n_float):e}")

# The resulting integer 200! is too large to be represented in a float
# It exceeds the e+308 max precision that floats have in 64-bit machines
```

[9]:
```python
from numpy import finfo, iinfo, float64, float32, int32, int64

# For floating-point types
print("Maximum numbers in 64-bit and 32-bit precision (float):",
      finfo(float32).max, finfo(float64).max)

# For integer types
print("Maximum and minimum numbers in 32-bit and 64-bit precision (int):",
      iinfo(int32).max, iinfo(int32).min,
      iinfo(int64).max, iinfo(int64).min)
```

```
Maximum numbers in 64-bit and 32-bit precision (float): 3.4028235e+38
1.7976931348623157e+308
Maximum and minimum numbers in 32-bit and 64-bit precision (int): 2147483647
-2147483648 9223372036854775807 -9223372036854775808
```

3.0.2 Exercise 2

[10]:
```
def solveQuadratic(a, b, c):
    root1 = (-b + np.sqrt(b**2 - (4 * a * c))) / (2 * a)
    root2 = (-b - np.sqrt(b**2 - (4 * a * c))) / (2 * a)
    return root1, root2

a = 0.001
b = 1000
c = 0.001

solveQuadratic(a, b, c)
```

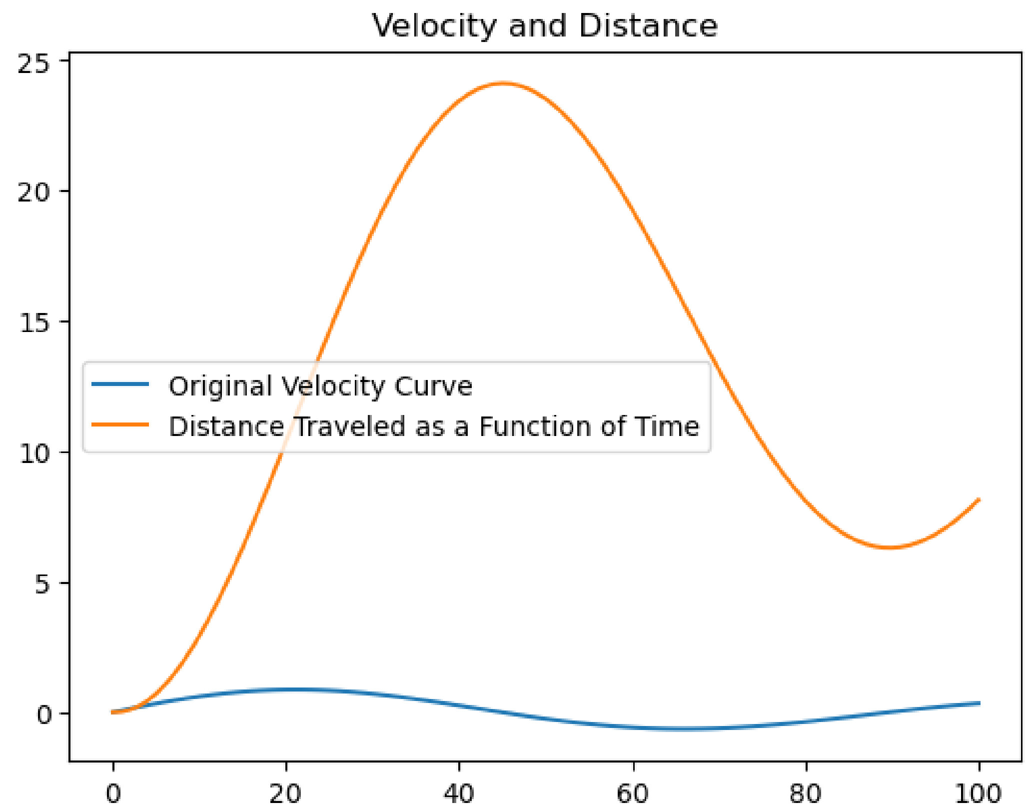
[10]:
```
(-9.999894245993346e-07, -999999.999999)
```

4 Chapter 5: Integrals and Derivatives

4.0.1 Exercise 1

[11]:
```
import numpy as np
from matplotlib import pyplot as plt

time, vel_x = np.loadtxt("velocities.txt", unpack="true")

def trapezoidal(f, a, b, N):
    h = (b - a) / N
    integral = 0.5 * (f(a) + f(b))
    for k in range(1, N):
        integral += f(a + k * h)
    return h * integral

# Calc the distance --> integral of vel w.r.t time

def f(i):
    return vel_x[int(i)]

a = 0
b = 100
N = 100

distances = []
for t in time:
    dist = trapezoidal(f, a, t, int(t) + 1)
    distances.append(dist)

plt.title("Velocity and Distance")
plt.plot(time, vel_x, label="Original Velocity Curve")
```

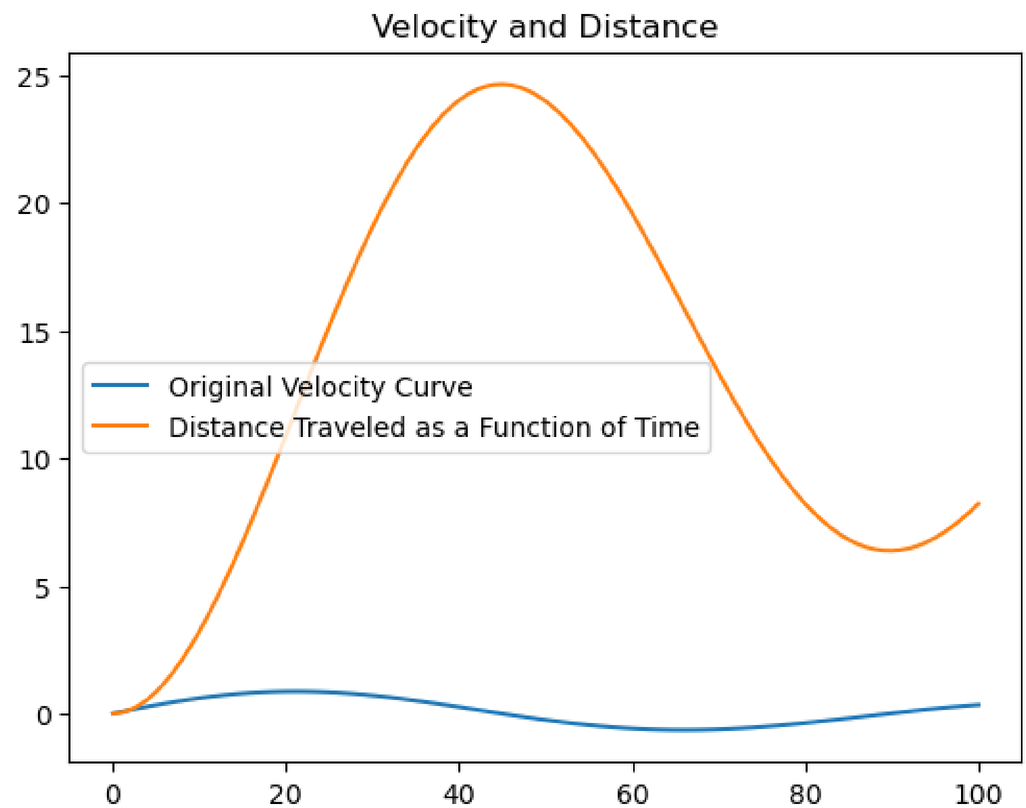
```
plt.plot(time, distances, label="Distance Traveled as a Function of Time")
plt.legend()
plt.show()
```



Iterative Attempt At Trapezoid Rule (More Efficient)

[12]:
```
time, vel_x = np.loadtxt("velocities.txt", unpack="true")

a = 0
b = 100
N = (b - a)


distances = [0]
for i in range(1, N + 1):
    h = time[i] - time[i - 1]

    # Add the area of the trapezoid to the previous distance
    new_dist = distances[-1] + 0.5 * h * (vel_x[i] + vel_x[i - 1])

    # Add new dist
```

```
    distances.append(new_dist)

plt.title("Velocity and Distance")
plt.plot(time, vel_x, label="Original Velocity Curve")
plt.plot(time, distances, label="Distance Traveled as a Function of Time")
plt.legend()
plt.show()
```



4.0.2 Exercise 2

[13]:
```
def f(x):
    return x**4 - (2 * x) + 1

def simpsons(f, a, b, N):
    if (N % 2 != 0):
        print("N must be even")
        return None
    h = (b - a) / N # dx

    # Start the integral
    integral = (f(a) + f(b))
```

```
    # Add all the even components
    for k in range(2, N, 2):
        integral += 4 * f(a + k * h)

    # Add all the odd components
    for k in range(1, N, 2):
        integral += 2 * f(a + k * h)

    return (h/3) * integral

print(simpsons(f, 0, 2, 1000))
```

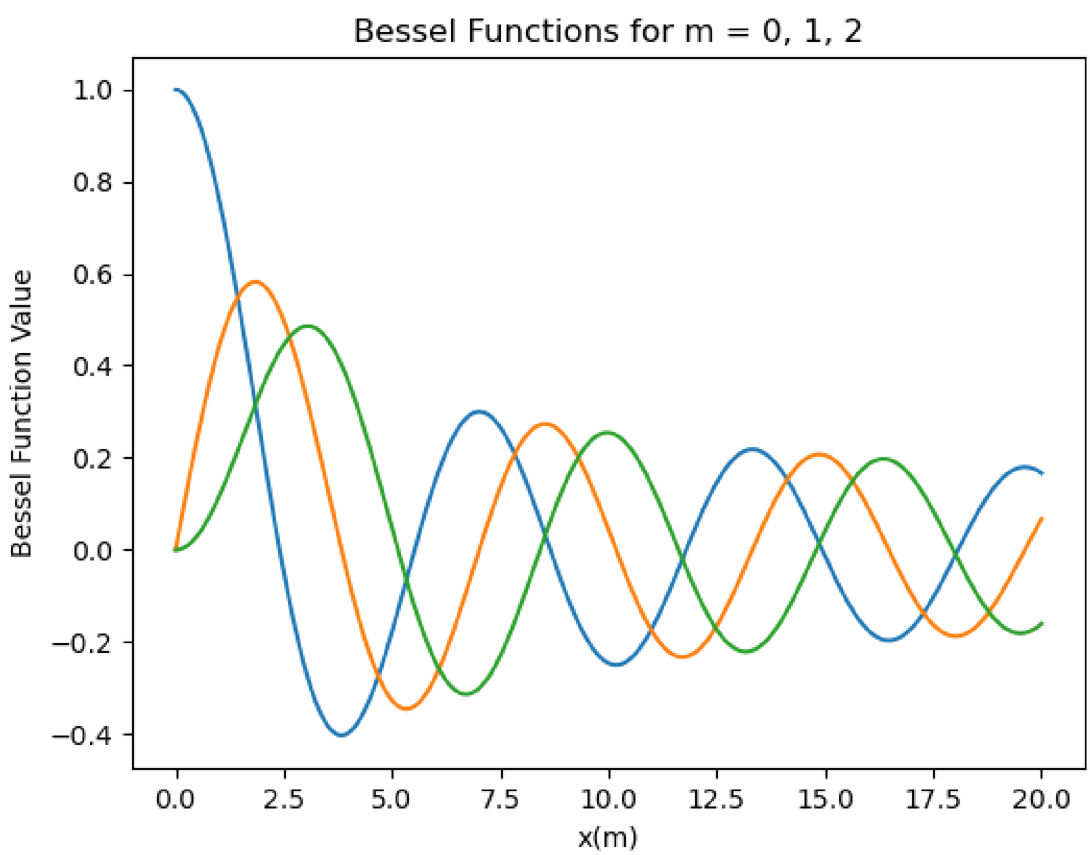
```
4.390687999993602
```

4.0.3 Exercise 4: Diffraction Limit of a Telescope

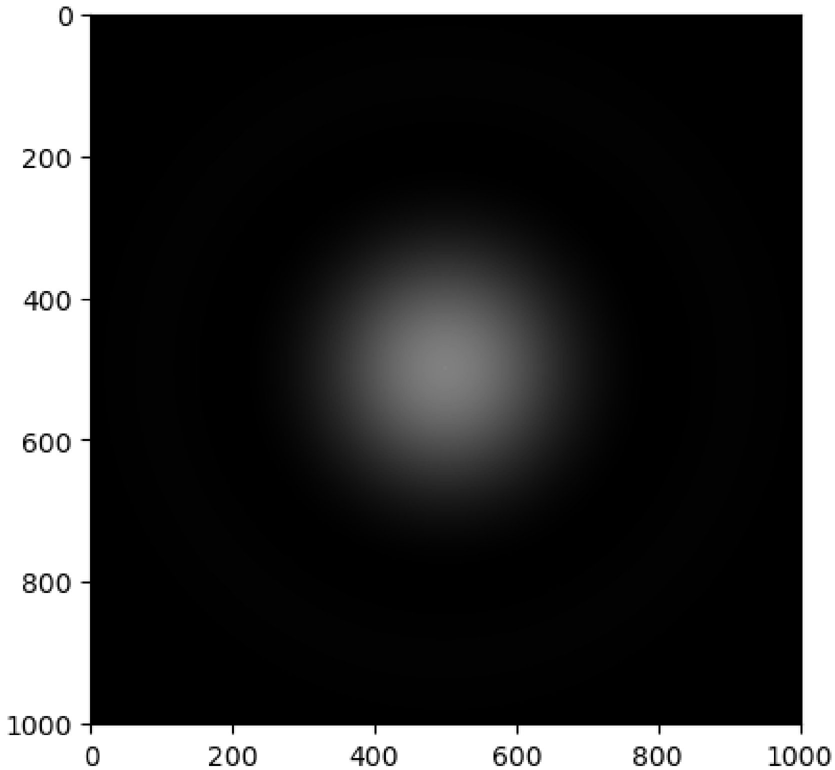
[14]:

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

# Part a

N = 1000

def f(x, m, theta):
    return np.cos(m * theta - x * np.sin(theta))

def simpsons_bessel(f, theta_i, theta_f, N, m, x):
    if (N % 2 != 0):
        print("N must be even")
        return None

    h = (theta_f - theta_i) / N

    # Start the integral
    integral = f(x, m, theta_i) + f(x, m, theta_f)

    # Add the even components
    for k in range(2, N, 2):
        integral += 4 * f(x, m, (theta_i + k * h))

    # Add the odd components
    for k in range(1, N, 2):
        integral += 2 * f(x, m, (theta_i + k * h))

    return integral * (h / 3)
```

```
def bessel(x, m):
    integral = simpsons_bessel(f, 0, np.pi, N, m, x)
    return integral / np.pi

m_vals = [0, 1, 2]
x_range = np.linspace(0, 20, N)

plt.title("Bessel Functions for m = 0, 1, 2")
plt.xlabel("x(m)")
plt.ylabel("Bessel Function Value")

for m in m_vals:
    bessel_m = []
    for x in x_range:
        bessel_m.append(bessel(x, m))
    plt.plot(x_range, bessel_m, label=f"J_{m}")
```



[42]:
```python
# Part b
import numpy as np
from pylab import imshow,gray,show
from numpy import empty,zeros,max

N = 1000

# Simulate the square focal plane
I = zeros([N+1,N+1], float)

radii = np.arange(0, 1e-6, N)
m = 1

wavelength = 500e-9

def intensity(r):
    if r == 0:
        return 1/2

    k = (2 * np.pi) / wavelength
    return (bessel((k * r), m) / (k * r))**2


x_offset, y_offset = N/2, N/2

for x in range(N+1):
    for y in range(N+1):

        x_phys = (x-x_offset) * (1e-6/N)
        y_phys = (y-y_offset) * (1e-6/N)

        r = np.sqrt(x_phys**2 + y_phys**2)
        I[x,y] = intensity(r)

imshow(I)
show()
```



5 Chapter 6

```
[ ]:
```