Astro 410 HomeWork 3

Swapnil Dubey

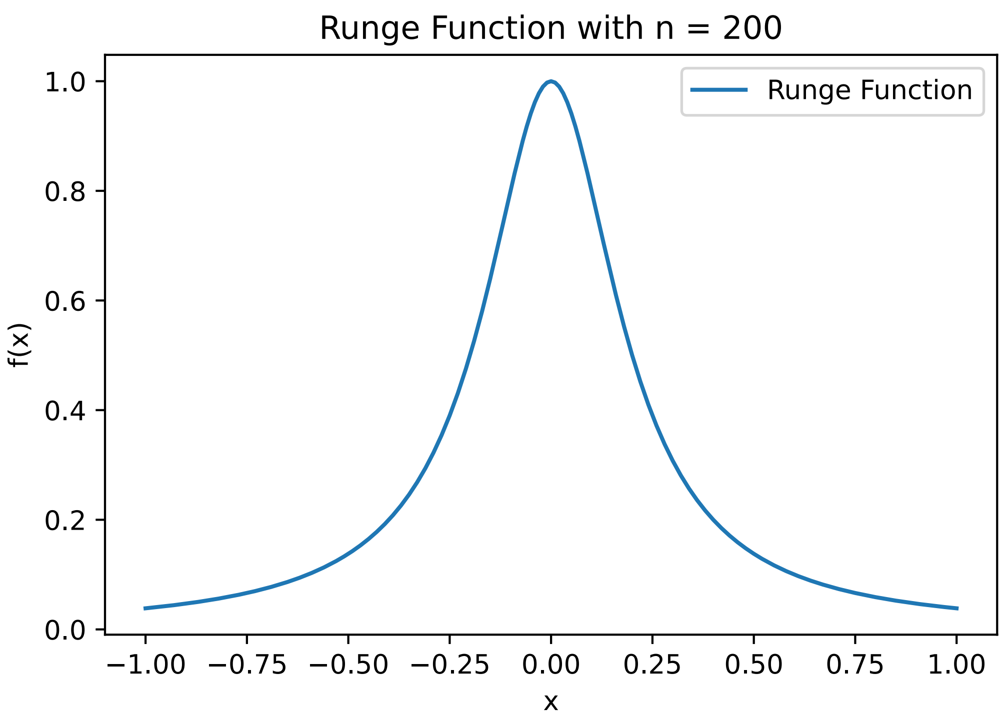
List of included files:

- main.ipynb
- main.py (Python code not as a Jupyter Notebook)
- hw3-dubey.pdf (contains printed pdf of html file)
- hw3-dubey.html (contains html version of Jupyter Notebook for better readability)

The following report attempts to compare Lagrangian and Cubic Natural Spline interpolation functions.

Question 1 Part 1

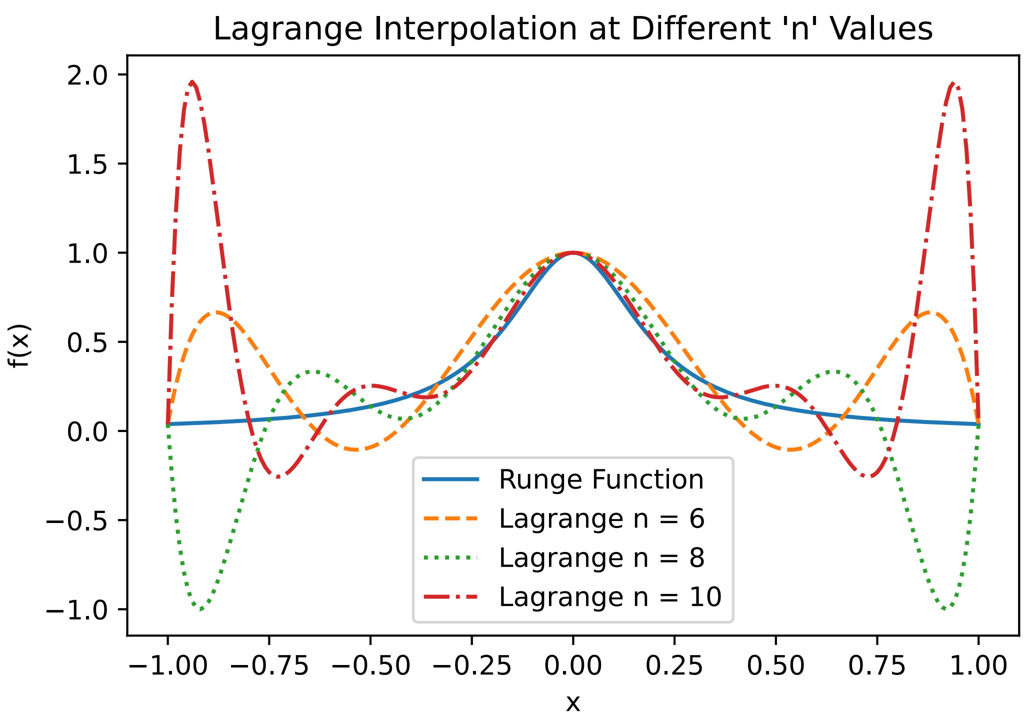
In [1]:
```
from math import *
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import trapezoid
from scipy.interpolate import lagrange
from scipy.interpolate import CubicSpline
from numpy.polynomial.polynomial import Polynomial

def xrng(n):
    x = np.linspace(-1, 1, n + 1)
    return x

## Q1 part 1
def rungeFunction(x):
    return 1 / ((25 * (x**2)) + 1)

y = []
for n in xrng(200):
    y.append(rungeFunction(n))

plt.plot(xrng(200), y)
plt.legend(['Runge Function'])
plt.title("Runge Function with n = 200")
plt.ylabel("f(x)")
plt.xlabel("x")
plt.show()
```



Question 1 Part 2

In [2]:
```
## Q1 b) Lagrange interpolation Formula & plot

plt.plot(xrng(200), rungeFunction(xrng(200)))


poly = lagrange(xrng(6), rungeFunction(xrng(6)))
plt.plot(xrng(200), poly(xrng(200)), '--')


poly = lagrange(xrng(8), rungeFunction(xrng(8)))
plt.plot(xrng(200), poly(xrng(200)), ':')


poly = lagrange(xrng(10), rungeFunction(xrng(10)))
plt.plot(xrng(200), poly(xrng(200)), '-.')


plt.legend(['Runge Function', 'Lagrange n = 6', 'Lagrange n = 8', "Lagrange n = 10"])
plt.title("Lagrange Interpolation at Different 'n' Values")
plt.ylabel("f(x)")
plt.xlabel("x")
```

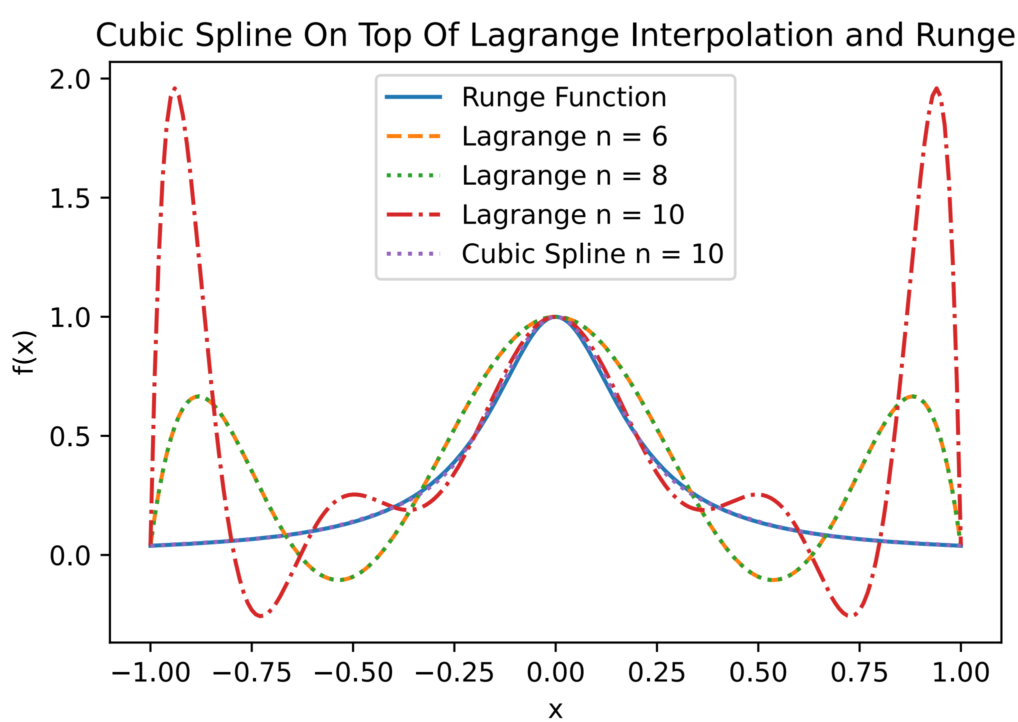
Out[2]:
```
Text(0.5, 0, 'x')
```



Question 1 Part 3

In [3]:
```
## Q1 c) Lagrange interpolation Formula & plot

def xrng(n):
    x = np.linspace(-1, 1, n + 1)
    return x

plt.plot(xrng(200), rungeFunction(xrng(200)))

poly = lagrange(xrng(6), rungeFunction(xrng(6)))
plt.plot(xrng(200), poly(xrng(200)), '--')


poly = lagrange(xrng(6), rungeFunction(xrng(6)))
plt.plot(xrng(200), poly(xrng(200)), ':')


poly = lagrange(xrng(10), rungeFunction(xrng(10)))
plt.plot(xrng(200), poly(xrng(200)), '-.')


# Question 1 part 3
cspline = CubicSpline(xrng(10), rungeFunction(xrng(10)))
plt.plot(xrng(200), cspline(xrng(200)), ':')

plt.legend(['Runge Function', 'Lagrange n = 6', 'Lagrange n = 8', "Lagrange n
 = 10", 'Cubic Spline n = 10'])
plt.title("Cubic Spline On Top Of Lagrange Interpolation and Runge")
plt.ylabel("f(x)")
plt.xlabel("x")
```

Out[3]:
```
Text(0.5, 0, 'x')
```



Question 1 Part 4

From the above plot, we can observe the purple cubic spline line follow the Runge function almost exactly while all the Lagrange functions stray off the the function at the ends and are not a good fit in the middle either. Thus we can conclude that cubic spline is a better interpolation function than lagrange interpolation

Question 2 Part 1

In [4]:
```
from math import exp
def trapezoid(f_x, a, b, n):
    h = (b-a)/float(n)
    integral = (f_x(a) + f_x(b))/2
    for i in range(1,n,1):
        a +=h
        integral += f_x(a)
        integral *= h;
        return integral

def g(t):
    return exp(-t**2)

a = -2;  b = 2
n = 1000
result = trapezoid(g, a, b, n)
print(result)
```

```
0.00014770554872304612
```

Question 2 Part 2

In [5]:
```
from math import *
from pylab import *
def simpson(f_x, a, b, n):
    h = (b-a) / n
    integral=0.0 # initialize
    for i in range(1, int(n/2)): #even points
        a += 2 * h
        integral += 4 * f_x(a)

    for i in range(2, int(n/2)-1): #odd points
        a += 2 * h
        integral += 2 * f_x(a)

    integral += f_x(a) + f_x(b)
    integral *= h / 3

    return integral

def function(x):
    return x

print(simpson(function, 0.0, 1.0, 100))
```

```
0.793866666666667
```

Question 2 Part 3

For the given function S (functionS), we get the following values for both our integration methods.

In [6]:
```
def functionS(x):
    return exp(-((x-1)**2) / 2) / (2*pi)**0.5

print(f"Simpson Integration: {simpson(functionS, -100.0, 100.0, 100)}")
print(f"Trapezoid Integration: {trapezoid(functionS, -100.0, 100.0, 100)}")
```

```
Simpson Integration: 0.6570774924259488
Trapezoid Integration: 0.0
```

Since our trapezoidal integration gives a value of 0, we can conclude that **Simpson** integration is better for this function.