hw9_code

November 4, 2021

Michael Goforth CAAM 550 HW 9 Due 11/05/2021

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
```

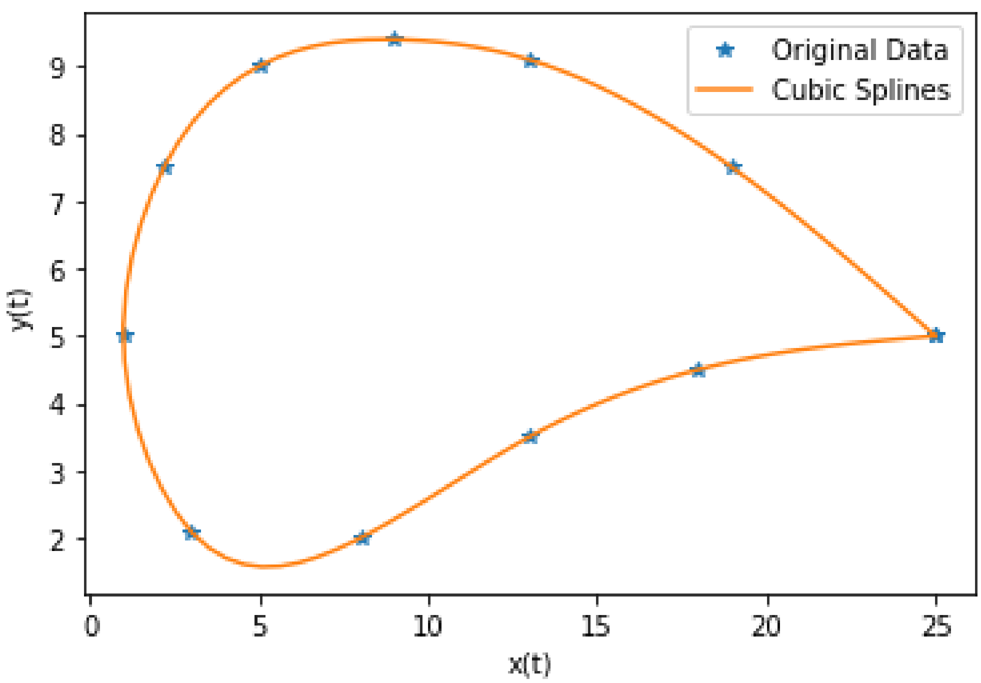
Problem 1.

[2]:
```
def free_cubic_spline(x, f):
    '''Function to calculate cubic spline formulas given a vector of x and f␣
↪values.

    Parameters
    ----------
    x : np.array
        vector of interpolation nodes
    f : np.array
        vector of function values corresponding to interpolation nodes above

    Returns
    -------
    A : np.array
        matrix of coefficients of the free cubic splines

    Michael Goforth
    CAAM 550
    Rice University
    November 5, 2021
    '''
    n = x.size - 1
    A = np.zeros([n + 1, 4])
    A[:, 0] = f

    T = np.zeros([n - 1, n - 1])
    a = np.zeros([n - 1])

    for i in range(n - 1):
```

```python
        h0 = x[i + 1] - x[i]
        h1 = x[i + 2] - x[i + 1]
        if i != 0:
            T[i, i - 1] = h0
        T[i, i] = 2 * (h0 + h1)
        if i != n - 2:
            T[i, i + 1] = h1
        a[i] = 3 / h1 * (A[i + 2, 0] - A[i + 1, 0]) - 3 / h0 * (A[i + 1, 0] - 
 ↪A[i, 0])
    A[1:-1, 2] = np.linalg.solve(T, a)
    for i in range(n):
        hi = x[i+1] - x[i]
        A[i, 1] = 1 / hi * (A[i+1, 0] - A[i, 0]) - hi / 3 * (2 * A[i, 2] +
 ↪A[i+1, 2])
        A[i, 3] = 1 / (3 * hi) * (A[i+1, 2] - A[i, 2])
    return A


def eval_cubic_spline(xin, A, xout):
    '''Function to evalutate cubic spline at a given value of x.

    Parameters
    ----------
    xin : np.array
          vector of interpolation nodes
    A : np.array
        matrix of coefficients of the free cubic splines
    xout: value
          value at which cubic splines will be evaluated

    Returns
    -------
    y : value
        value of cubic spline interpolation evaluated at xout

    Michael Goforth
    CAAM 550
    Rice University
    November 5, 2021
    '''

    for i in range(x.size-1):
        if xout <= xin[i + 1]:
            xi = xin[i]
            return A[i, 0] + A[i, 1] * (xout - xi) + A[i, 2] * (xout - xi)**2 +
 ↪A[i, 3] * (xout - xi)**3
```

[3]:
```python
x = np.array([25, 19, 13, 9, 5, 2.2, 1, 3, 8, 13, 18, 25])
y = np.array([5, 7.5, 9.1, 9.4, 9.0, 7.5, 5, 2.1, 2, 3.5, 4.5, 5.0])
t = np.zeros([x.size])
for i in range(1, x.size):
    t[i] = t[i-1] + ((x[i] - x[i-1])**2 + (y[i] - y[i-1])**2)**.5
A1 = free_cubic_spline(t, x)
A2 = free_cubic_spline(t, y)
tfinal = t[-1]
tplot = np.linspace(0, tfinal, 100)
xplot = np.zeros([100])
yplot = np.zeros([100])
for i in range(tplot.size):
    xplot[i] = eval_cubic_spline(t, A1, tplot[i])
    yplot[i] = eval_cubic_spline(t, A2, tplot[i])
plt.plot(x, y, '*', label='Original Data')
plt.plot(xplot, yplot, label='Cubic Splines')
plt.legend()
plt.xlabel('x(t)')
plt.ylabel('y(t)')
```

[3]: Text(0, 0.5, 'y(t)')



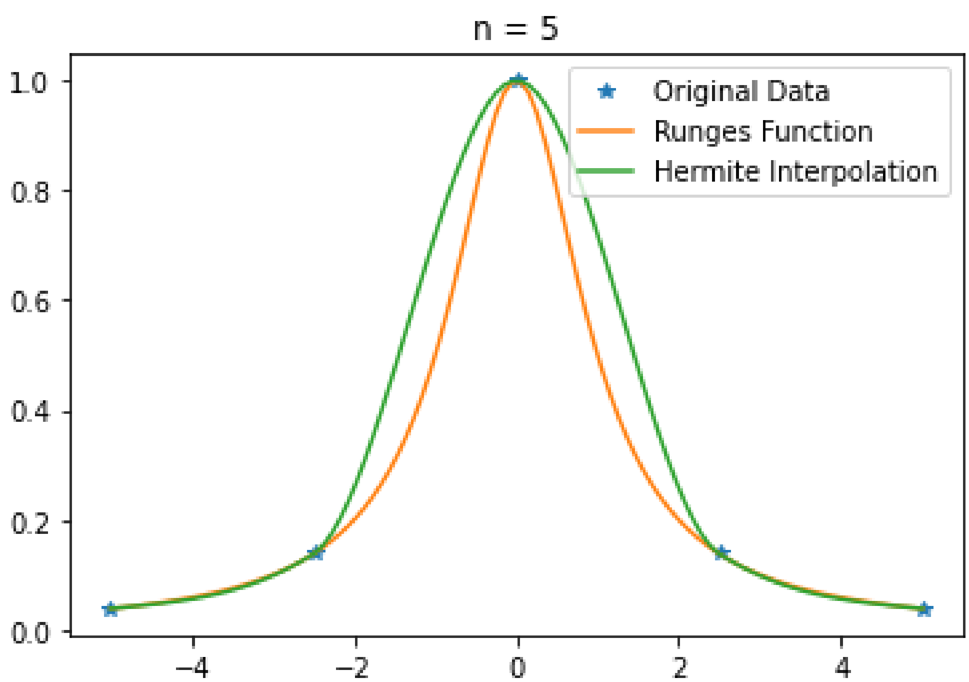
Problem 2

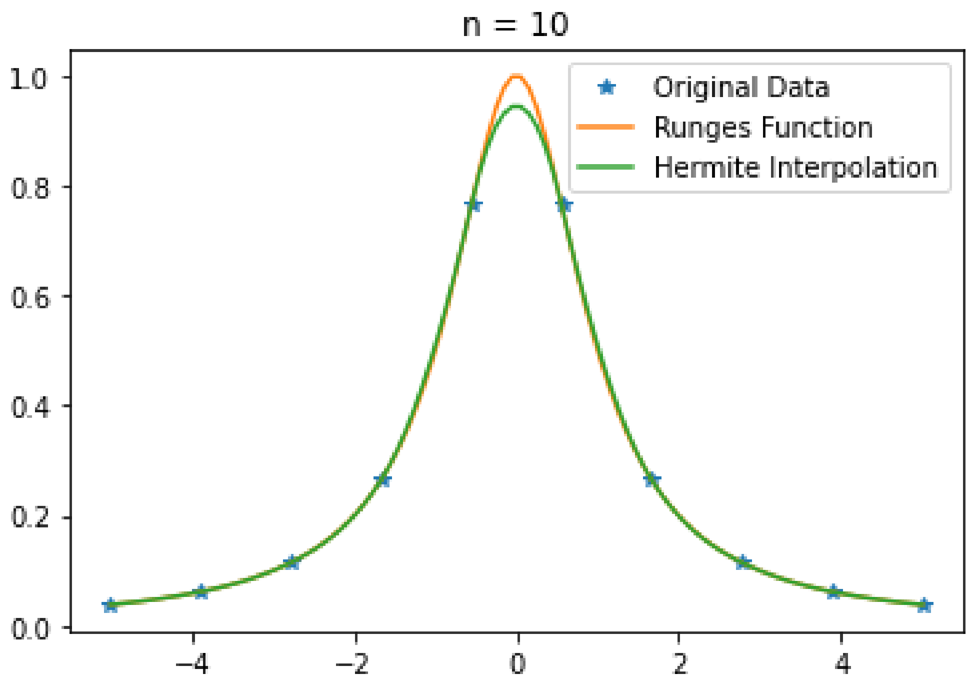
Part b

```
[4]: def runge(x):
        '''Evaluates Runge's function at given values of x.

        Parameters
        ----------
        x : np.array
            vector of values to evaluate function at

        Returns
        -------
        y, yp : tuple of np.array
                tuple of vector of values of Runge's function and derivative of␣
     ↪Runge's
                function corresponding to the points in x

        Michael Goforth
        CAAM 550
        Rice University
        November 5, 2021
        '''
        y = 1 / (1 + np.power(x, 2))
        yp = -1 * (1 + np.power(x, 2))**-2 * 2 * x
        return y, yp


    def hermite_spline(xin, yin, ypin):
        '''Function to calculate Hermite spline formulas given a vector of xin and␣
     ↪yin values.

        Parameters
        ----------
        xin : np.array
              vector of interpolation nodes
        yin : np.array
              vector of function values corresponding to interpolation nodes above
        ypin : np.array
               vector of derivatives corresponding to interpolation nodes above

        Returns
        -------
        A : np.array
            matrix of coefficients of the Hermite polynomials

        Michael Goforth
        CAAM 550
        Rice University
        November 5, 2021
```

```
    '''
    n = xin.size - 1
    A = np.zeros([n, 4])
    for i in range(n):
        hi = xin[i+1] - xin[i]
        A[i, 0] = yin[i]
        A[i, 1] = yin[i + 1]
        A[i, 2] = hi * ypin[i]
        A[i, 3] = hi * ypin[i+1]
    return A


def eval_hermite_spline(xin, A, xout):
    '''Function to evalutate Hermite spline at a given value of x.

    Parameters
    ----------
    xin : np.array
          vector of interpolation nodes
    A : np.array
        matrix of coefficients of the Hermite polynomials
    xout: value
          value at which Hermite splines will be evaluated

    Returns
    -------
    y : value
        value of Hermite spline interpolation evaluated at xout

    Michael Goforth
    CAAM 550
    Rice University
    November 5, 2021
    '''

    for i in range(xin.size-1):
        if xout <= xin[i + 1]:
            xi = xin[i]
            hi = xin[i+1] - xin[i]
            xhat = (xout - xi) / hi
            H0 = (1 - xhat)**2 * (1 + 2 * xhat)
            H1 = (xhat**2) * (3 - 2 * xhat)
            h0 = xhat * (1 - xhat)**2
            h1 = xhat**2 * (xhat - 1)
            return A[i, 0] * H0 + A[i, 1] * H1 + A[i, 2] * h0 + A[i, 3] * h1
```

```
a = -5
b = 5

nvec = [5, 10]
for n in nvec:
    xin = np.linspace(a, b, n)
    yin, ypin = runge(xin)
    A = hermite_spline(xin, yin, ypin)
    xplot = np.linspace(a, b, 100)
    yplot = np.zeros([100])
    for i in range(100):
        yplot[i] = eval_hermite_spline(xin, A, xplot[i])
    plt.plot(xin, yin, '*', label='Original Data')
    yrunge = runge(xplot)
    plt.plot(xplot, yrunge[0], label='Runges Function')
    plt.plot(xplot, yplot, label='Hermite Interpolation')
    plt.legend()
    plt.title('n = ' + str(n))
    plt.show()
```





Problem 4

Part i.

```
[5]: def Trap_Integration(f, a, b, tol):
         '''Function to calculate integral of a function using the trapezoidal rule.␣
     ↪

         Parameters
         ----------
         f : function
             function to be integrated. For a value, x, f(x) should return y s.t.␣
     ↪f(x)=y
         a : value
             lower bound of integration
         b : value
             upper bound of integration
         tol : value
               tolerance of solution

         Returns
         -------
         results : pd.Dataframe
                   Dataframe with columns containing the following data: h/2, T(h/
     ↪2), and
```

```
            |T(h) - T(h/2)|/|T(h/2)|

    Michael Goforth
    CAAM 550
    Rice University
    November 5, 2021
    '''

    h = b - a
    xknown = [a, b]
    yknown = [f(a), f(b)]
    diff = tol + 1
    Th = h/2 * (f(a) + f(b))

    columnnames = ["h/2", "T(h/2)", "|T(h) - T(h/2)|/|T(h/2)|", "Function␣
↪Evals"]
    results = pd.DataFrame(columns = columnnames)

    while diff > tol:
        h = h / 2
        xnew = [xknown[0] + h] + [x + h for x in xknown[2:]]
        ynew = [f(y) for y in xnew]
        xknown = xknown + xnew
        yknown = yknown + ynew
        Th2 = h / 2 * (yknown[0] + yknown[1]) + h * sum(yknown[2:])
        diff = abs(Th - Th2) / abs(Th2)
        results = results.append({"h/2": h, "T(h/2)": Th2, "|T(h) - T(h/2)|/
↪|T(h/2)|": diff,
                                  "Function Evals": len(yknown)},␣
↪ignore_index=True)
        Th = Th2

    return results
```

Part ii.

```
[6]: def Simpson_Integration(f, a, b, tol):
    '''Function to calculate integral of a function using the Simpson's rule.

    Parameters
    ----------
    f : function
        function to be integrated. For a value, x, f(x) should return y s.t.␣
↪f(x)=y
    a : value
        lower bound of integration
```

```
    b : value
        upper bound of integration
    tol : value
          tolerance of solution

    Returns
    -------
    results : pd.Dataframe
              Dataframe with columns containing the following data: h/2, T(h/
↪2), and
              |T(h) - T(h/2)|/|T(h/2)|

    Michael Goforth
    CAAM 550
    Rice University
    November 5, 2021
    '''

    h = b - a
    xknown = [a, b]
    yknown = [f(a), f(b)]
    diff = tol + 1
    Th = h / 6 * (yknown[0] + yknown[1])

    columnnames = ["h/2", "T(h/2)", "|T(h) - T(h/2)|/|T(h/2)|", "Function␣
↪Evals"]
    results = pd.DataFrame(columns = columnnames)

    while diff > tol:
        h = h / 2
        xnew = [xknown[0] + h] + [x + h for x in xknown[2:]]
        ynew = [f(y) for y in xnew]
        Th2 = h / 3 * (yknown[0] + yknown[1]) + 2 * h / 3 * (sum(yknown[2:]) +␣
↪2 * sum(ynew))
        xknown = xknown + xnew
        #print(xknown)
        #print(yknown)
        yknown = yknown + ynew
        diff = abs(Th - Th2) / abs(Th2)
        results = results.append({"h/2": h, "T(h/2)": Th2, "|T(h) - T(h/2)|/
↪|T(h/2)|": diff,
                                  "Function Evals": int(len(yknown))},␣
↪ignore_index=True)
        Th = Th2

    return results
```

Part iv.

[7]:
```python
def f(x):
    return x / (1 + x**2)

results1 = Trap_Integration(f, 0, 3, 1e-6)
truth = math.log(10) * .5
app = results1["T(h/2)"].iloc[-1]
abserr = abs(app - truth)
relerr = abs(abserr / truth)
evals = int(results1["Function Evals"].iloc[-1])
print("f(x) = x/(1+x^2), [a,b] = [0,3]")
print("Composite Trapezoidal rule")
print("Approximate value of the integral = " + str(app))
print("Absolute error                    = " + str(abserr))
print("Relative error                    = " + str(relerr))
print("Number of f evaluations required  = " + str(evals))
print()
results2 = Simpson_Integration(f, 0, 3, 1e-6)
app = results2["T(h/2)"].iloc[-1]
abserr = abs(app - truth)
relerr = abs(abserr / truth)
evals = int(results2["Function Evals"].iloc[-1])
print("Composite Simpson rule")
print("Approximate value of the integral = " + str(app))
print("Absolute error                    = " + str(abserr))
print("Relative error                    = " + str(relerr))
print("Number of f evaluations required  = " + str(evals))
```

```
f(x) = x/(1+x^2), [a,b] = [0,3]
Composite Trapezoidal rule
Approximate value of the integral = 1.151292353377935
Absolute error                    = 1.9311908805441647e-07
Relative error                    = 1.6774110858444253e-07
Number of f evaluations required  = 2049

Composite Simpson rule
Approximate value of the integral = 1.1512925565403263
Absolute error                    = 1.0043303300122375e-08
Relative error                    = 8.72350240664773e-09
Number of f evaluations required  = 129
```

[8]:
```python
def f(x):
    return 1 / (1 - x)

results1 = Trap_Integration(f, 0, .95, 1e-6)
truth = math.log(20)
app = results1["T(h/2)"].iloc[-1]
```

```
abserr = abs(app - truth)
relerr = abs(abserr / truth)
evals = int(results1["Function Evals"].iloc[-1])
print("f(x) = 1/(1-x), [a,b] = [0,.95]")
print("Composite Trapezoidal rule")
print("Approximate value of the integral = " + str(app))
print("Absolute error                    = " + str(abserr))
print("Relative error                    = " + str(relerr))
print("Number of f evaluations required  = " + str(evals))
print()
results2 = Simpson_Integration(f, 0, .95, 1e-6)
app = results2["T(h/2)"].iloc[-1]
abserr = abs(app - truth)
relerr = abs(abserr / truth)
evals = int(results2["Function Evals"].iloc[-1])
print("Composite Simpson rule")
print("Approximate value of the integral = " + str(app))
print("Absolute error                    = " + str(abserr))
print("Relative error                    = " + str(relerr))
print("Number of f evaluations required  = " + str(evals))
```

```
f(x) = 1/(1-x), [a,b] = [0,.95]
Composite Trapezoidal rule
Approximate value of the integral = 2.995732720709648
Absolute error                    = 4.4715565694630754e-07
Relative error                    = 1.4926422527598697e-07
Number of f evaluations required  = 8193

Composite Simpson rule
Approximate value of the integral = 2.9957323365615767
Absolute error                    = 6.300758581545551e-08
Relative error                    = 2.1032448851214057e-08
Number of f evaluations required  = 513
```

[9]:

```
def f(x):
    return 1 / (1 - .5 * math.sin(x)**2)**.5
b = math.pi/2
results1 = Trap_Integration(f, 0, b, 1e-6)
app = results1["T(h/2)"].iloc[-1]
evals = int(results1["Function Evals"].iloc[-1])
print("f(x) = 1/(1-.5*sin^2(x))^.5, [a,b] = [0,pi/2]")
print("Composite Trapezoidal rule")
print("Approximate value of the integral = " + str(app))
print("Number of f evaluations required  = " + str(evals))
print()
results2 = Simpson_Integration(f, 0, b, 1e-6)
app = results2["T(h/2)"].iloc[-1]
```

```
abserr = abs(app - truth)
relerr = abs(abserr / truth)
evals = int(results2["Function Evals"].iloc[-1])
print("Composite Simpson rule")
print("Approximate value of the integral = " + str(app))
print("Number of f evaluations required  = " + str(evals))
```

```
f(x) = 1/(1-.5*sin^2(x))^.5, [a,b] = [0,pi/2]
Composite Trapezoidal rule
Approximate value of the integral = 1.8540746773016665
Number of f evaluations required  = 9

Composite Simpson rule
Approximate value of the integral = 1.8540746773012737
Number of f evaluations required  = 17
```

[10]:
```
def f(x):
    return 1 / (1 - .8 * math.sin(x)**2)**.5
b = math.pi/2
results1 = Trap_Integration(f, 0, b, 1e-6)
app = results1["T(h/2)"].iloc[-1]
evals = int(results1["Function Evals"].iloc[-1])
print("f(x) = 1/(1-.8*sin^2(x))^.5, [a,b] = [0,pi/2]")
print("Composite Trapezoidal rule")
print("Approximate value of the integral = " + str(app))
print("Number of f evaluations required  = " + str(evals))
print()
results2 = Simpson_Integration(f, 0, b, 1e-6)
app = results2["T(h/2)"].iloc[-1]
abserr = abs(app - truth)
relerr = abs(abserr / truth)
evals = int(results2["Function Evals"].iloc[-1])
print("Composite Simpson rule")
print("Approximate value of the integral = " + str(app))
print("Number of f evaluations required  = " + str(evals))
```

```
f(x) = 1/(1-.8*sin^2(x))^.5, [a,b] = [0,pi/2]
Composite Trapezoidal rule
Approximate value of the integral = 2.2572053268208734
Number of f evaluations required  = 17

Composite Simpson rule
Approximate value of the integral = 2.257205326820847
Number of f evaluations required  = 33
```

[11]:
```
def f(x):
    return 1 / (1 - .95 * math.sin(x)**2)**.5
b = math.pi/2
```

```
results1 = Trap_Integration(f, 0, b, 1e-6)
app = results1["T(h/2)"].iloc[-1]
abserr = abs(app - truth)
evals = int(results1["Function Evals"].iloc[-1])
print("f(x) = 1/(1-.95*sin^2(x))^.5, [a,b] = [0,pi/2]")
print("Composite Trapezoidal rule")
print("Approximate value of the integral = " + str(app))
print("Absolute error                    = " + str(abserr))
print("Number of f evaluations required  = " + str(evals))
print()
results2 = Simpson_Integration(f, 0, b, 1e-6)
app = results2["T(h/2)"].iloc[-1]
abserr = abs(app - truth)
relerr = abs(abserr / truth)
evals = int(results2["Function Evals"].iloc[-1])
print("Composite Simpson rule")
print("Approximate value of the integral = " + str(app))
print("Absolute error                    = " + str(abserr))
print("Number of f evaluations required  = " + str(evals))
```

```
f(x) = 1/(1-.95*sin^2(x))^.5, [a,b] = [0,pi/2]
Composite Trapezoidal rule
Approximate value of the integral = 2.9083372484446572
Absolute error                    = 0.08739502510933361
Number of f evaluations required  = 33

Composite Simpson rule
Approximate value of the integral = 2.9083372484445156
Absolute error                    = 0.08739502510947528
Number of f evaluations required  = 65
```

Problem 5.

Part ii.

[12]:
```
def f(x):
    return 1 / (1 + np.power(x, 2))

nvec = [5, 10, 15]
a = -5
b = 5
for n in nvec:
    xnodes = np.array([-5 + i * 10/n for i in range(n + 1)])
    ynodes = f(xnodes)
    A = np.ones([n+1, n+1])
    B = np.zeros([n+1])
    for i in range(n+1):
        A[i, :] = np.power(xnodes, i)
```

```
        B[i] = 1 / (i + 1) * (b**(i + 1) - a**(i + 1))
    W = np.linalg.solve(A, B)
    app = sum(W * ynodes)
    print('n = ' + str(n))
    print('Nodes: ' + str(xnodes))
    print('Weights: ' + str(W))
    print('Computed Approximation: ' + str(app))
```

```
n = 5
Nodes: [-5. -3. -1.  1.  3.  5.]
Weights: [0.65972222 2.60416667 1.73611111 1.73611111 2.60416667 0.65972222]
Computed Approximation: 2.307692307692309
n = 10
Nodes: [-5. -4. -3. -2. -1.  0.  1.  2.  3.  4.  5.]
Weights: [ 0.26834148  1.77535941 -0.81043571  4.54946288 -4.35155123  7.1376463
 -4.35155123  4.54946288 -0.81043571  1.77535941  0.26834148]
Computed Approximation: 4.673300555670889
n = 15
Nodes: [-5.         -4.33333333 -3.66666667 -3.         -2.33333333 -1.66666667
 -1.         -0.33333333  0.33333333  1.          1.66666667  2.33333333
  3.          3.66666667  4.33333333  5.        ]
Weights: [ 0.170873    1.28507379 -1.12722905  5.07042708 -7.56293114
11.91360348
 -9.68005208  4.93023492  4.93023496 -9.68005211 11.91360349 -7.56293115
  5.07042708 -1.12722905  1.28507379  0.170873  ]
Computed Approximation: 4.155558988017933
```

Part iii.

[13]:
```
# Using Chebyshev Nodes
nvec = [5, 10, 15]
a = -5
b = 5
for n in nvec:
    xnodes = np.array([5 * math.cos((2 * i + 1) * math.pi / (2 * n + 2)) for i␣
 ↪in range(n + 1)])
    ynodes = f(xnodes)
    A = np.ones([n+1, n+1])
    B = np.zeros([n+1])
    for i in range(n+1):
        A[i, :] = np.power(xnodes, i)
        B[i] = 1 / (i + 1) * (b**(i + 1) - a**(i + 1))
    W = np.linalg.solve(A, B)
    app = sum(W * ynodes)
    print('n = ' + str(n))
    print('Nodes: ' + str(xnodes))
    print('Weights: ' + str(W))
    print('Computed Approximation: ' + str(app))
```

```
n = 5
Nodes: [ 4.82962913  3.53553391  1.29409523 -1.29409523 -3.53553391 -4.82962913]
Weights: [0.59330511 1.88888889 2.517806   2.517806   1.88888889 0.59330511]
Computed Approximation: 2.2113115356791346
n = 10
Nodes: [ 4.94910721e+00  4.54815998e+00  3.77874787e+00  2.70320409e+00
  1.40866278e+00  1.41638472e-15 -1.40866278e+00 -2.70320409e+00
 -3.77874787e+00 -4.54815998e+00 -4.94910721e+00]
Weights: [0.17698858 0.60847767 0.92441624 1.20994808 1.36247298 1.43539289
 1.36247298 1.20994808 0.92441624 0.60847767 0.17698858]
Computed Approximation: 2.8307823662995624
n = 15
Nodes: [ 4.97592363  4.78470168  4.40960632  3.86505227  3.17196642  2.35698368
  1.45142339  0.4900857  -0.4900857  -1.45142339 -2.35698368 -3.17196642
 -3.86505227 -4.40960632 -4.78470168 -4.97592363]
Weights: [0.08401378 0.29168232 0.45859157 0.62564809 0.75696231 0.86709706
 0.93874864 0.97725624 0.97725624 0.93874864 0.86709706 0.75696231
 0.62564809 0.45859157 0.29168232 0.08401378]
Computed Approximation: 2.736056218945736
```

Part iv.

[14]:

```
results1 = Trap_Integration(f, -5, 5, 1e-4)
app = results1["T(h/2)"].iloc[-1]
evals = int(results1["Function Evals"].iloc[-1])
print("f(x) = x/(1+x^2), [a,b] = [-5,5]")
print("Composite Trapezoidal rule")
print("Approximate value of the integral = " + str(app))
print("Number of f evaluations required  = " + str(evals))
print()
results2 = Simpson_Integration(f, -5, 5, 1e-4)
app = results2["T(h/2)"].iloc[-1]
abserr = abs(app - truth)
relerr = abs(abserr / truth)
evals = int(results2["Function Evals"].iloc[-1])
print("Composite Simpson rule")
print("Approximate value of the integral = " + str(app))
print("Number of f evaluations required  = " + str(evals))
```

```
f(x) = x/(1+x^2), [a,b] = [-5,5]
Composite Trapezoidal rule
Approximate value of the integral = 2.7467413518567882
Number of f evaluations required  = 65

Composite Simpson rule
Approximate value of the integral = 2.7468014883907834
Number of f evaluations required  = 65
```

[ ]:

```

```