Ex

Exercise 6.1

- 9, 11, 17, 21, 26, 27

Exercise 6.2

- 4, 8, 17, 23, 27, 37, 41

Exercise 6.3

- 3, 6, 7, 21, 24, 26, 27, 29, 34, 37, 38

Com Ex

Computer Exercise 6.1

- 4

Computer Exercise 6.2

- 4, 11

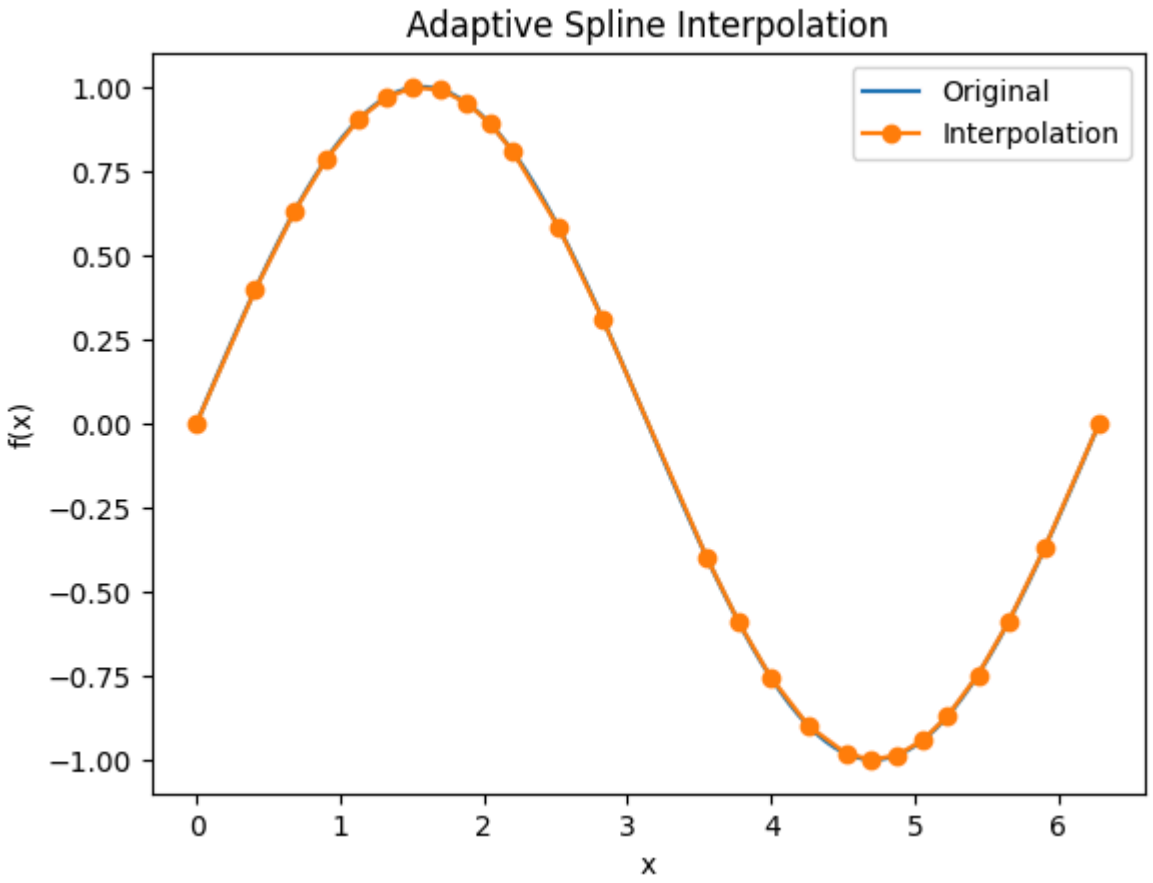
Computer Exercise 6.3

- 7, 19

In [34]:
```
import cv2
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
```

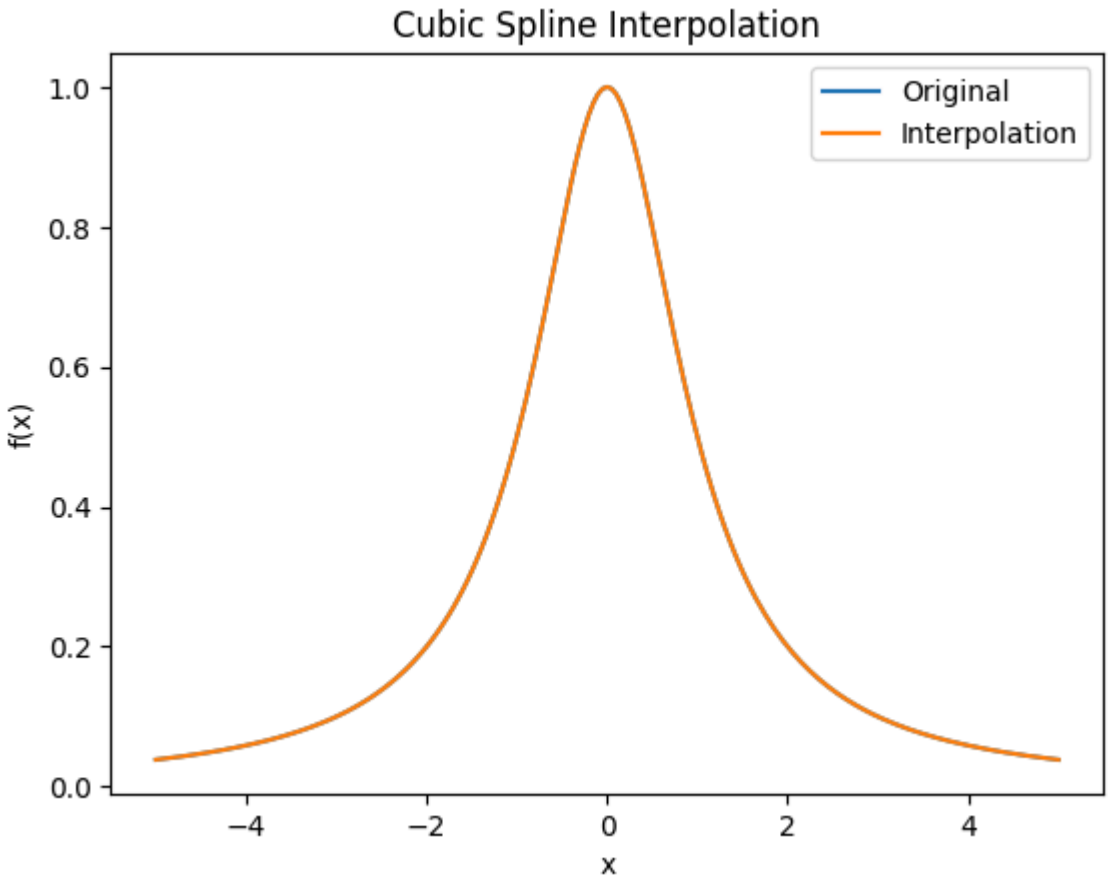
In [35]:
```
# 6.1 - 4

def adaptive_spline_interpolation(f, a, b, tol=0.01):
    nodes = [a, b]

    done = False
    while not done:
        done = True
        new_nodes = []
        nodes.sort()

        for i in range(len(nodes) - 1):
            t0 = nodes[i]
            t1 = nodes[i + 1]
            sample_points = [t0 + j * (t1 - t0) / 10 for j in range(10)]
            max_error = 0
```

```
            max_x = None

            for x in sample_points:
                Sx = f(t0) + (f(t1) - f(t0)) * (x - t0) / (t1 - t0)
                err = abs(Sx - f(x))

                if err > max_error:
                    max_error = err
                    max_x = x

            if max_error > tol:
                new_nodes.append(max_x)
                done = False

        if new_nodes:
            nodes.extend(new_nodes)

    nodes.sort()
    return nodes, [f(x) for x in nodes]

f = np.sin
a = 0
b = 2 * np.pi

x, y = adaptive_spline_interpolation(f, a, b, tol=0.01)
plt.plot(np.linspace(a, b, 100), f(np.linspace(a, b, 100)), label='Original')
plt.plot(x, y, 'o-', label='Interpolation')
plt.title('Adaptive Spline Interpolation')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.legend()
plt.show()
```



In [36]:
```
# 6.2 - 4

def cubic_spline_interpolation_coef(f, a, b, n, y = None):
    x = np.linspace(a, b, n)
    if y is None:
        y = f(x)
    h = np.diff(x)
    b = np.diff(y) / h

    u = np.zeros(n)
    v = np.zeros(n)
    z = np.zeros(n)

    u[0] = 1
    u[-1] = 1
    v[0] = 0
    v[-1] = 0

    for i in range(1, n - 1):
        if i == 1:
            u[i] = 2 * (h[i - 1] + h[i])
            v[i] = 6 * (b[i] - b[i - 1])
        else:
            u[i] = 2 * (h[i - 1] + h[i]) - h[i - 1]**2 / u[i - 1]
            v[i] = 6 * (b[i] - b[i - 1]) - h[i - 1] * v[i - 1] / u[i - 1]

    for i in range(n - 2, 0, -1):
        z[i] = (v[i] - h[i] * z[i + 1]) / u[i]

    return x, y, z

def cubic_spline_interpolation_eval(f, a, b, n, x_eval, y=None):
    x, y, z = cubic_spline_interpolation_coef(f, a, b, n, y)
    h = np.diff(x)

    S = np.zeros_like(x_eval)

    for i, xi in enumerate(x_eval):
        j = np.searchsorted(x, xi) - 1
        if j < 0:
            j = 0
        elif j >= len(h):
            j = len(h) - 1

        dx = xi - x[j]
        b_coef = (y[j+1] - y[j]) / h[j] - h[j] * (2 * z[j] + z[j+1]) / 6
        S[i] = y[j] + b_coef * dx + (z[j] / 2) * dx**2 + ((z[j+1] - z[j]) / (6 *

    return S


f = lambda x: 1 / (1 + x ** 2)
a = -5
b = 5
n = 41
x_eval = np.linspace(a, b, 201)

S = cubic_spline_interpolation_eval(f, a, b, n, x_eval)
plt.plot(x_eval, f(x_eval), label='Original')
```

```
plt.plot(x_eval, S, label='Interpolation')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Cubic Spline Interpolation')
plt.legend()
plt.show()

for x in x_eval:
    if x >= 0:
        print(f"S({x}) = {S[np.searchsorted(x_eval, x)]}, f({x}) = {f(x)}, error
```



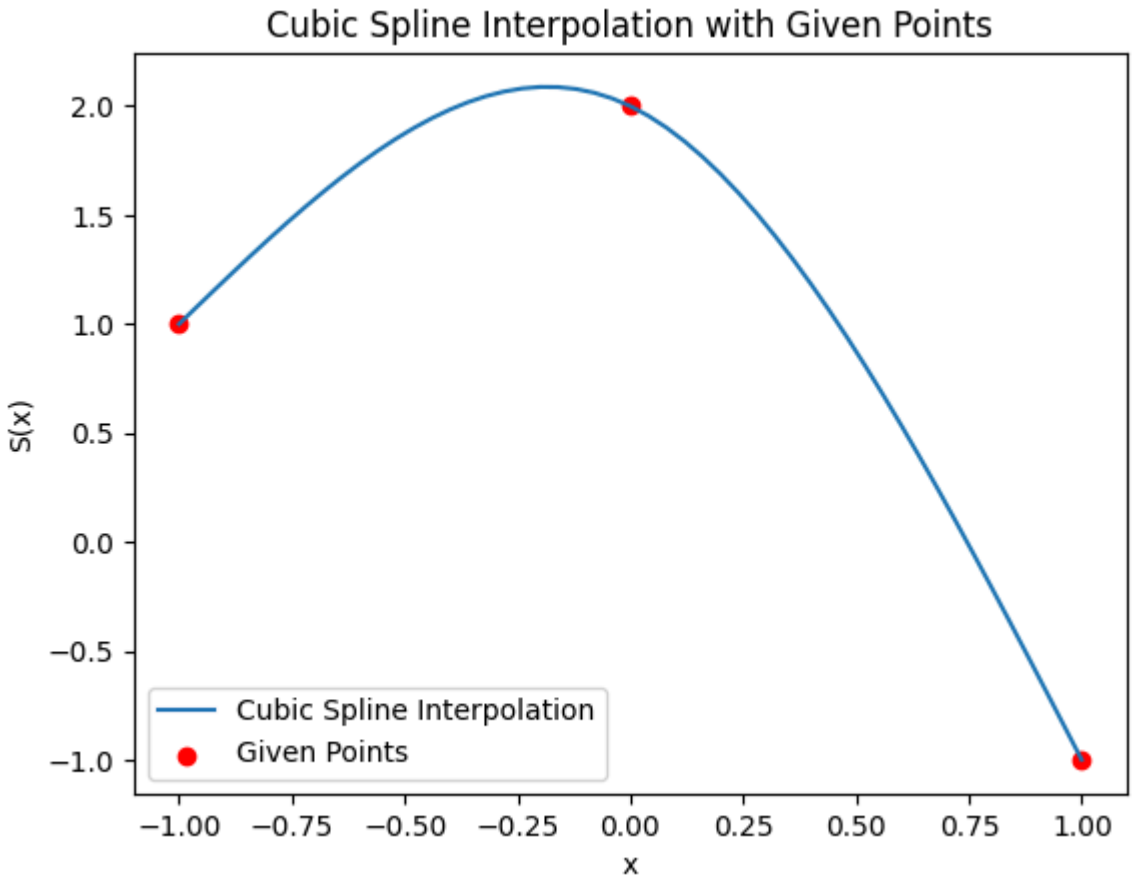
```
S(0.0) = 0.9999999999999999, f(0.0) = 1.0, error = 1.1102230246251565e-16
S(0.0500000000000071) = 0.9974028120835318, f(0.0500000000000071) = 0.997506234
4139649, error = 0.00010342233043314497
S(0.1000000000000053) = 0.9898554950741247, f(0.1000000000000053) = 0.990099009
9009901, error = 0.0002435148268653542
S(0.1500000000000036) = 0.9777244190817755, f(0.1500000000000036) = 0.977995110
0244497, error = 0.0002706909426741788
S(0.2000000000000018) = 0.9613759542164801, f(0.2000000000000018) = 0.961538461
5384615, error = 0.0001625073219813089
S(0.25) = 0.9411764705882353, f(0.25) = 0.9411764705882353, error = 0.0
S(0.3000000000000007) = 0.9175304445082155, f(0.3000000000000007) = 0.91743119266
05501, error = 9.925184766546025e-05
S(0.3500000000000053) = 0.8909947770923127, f(0.3500000000000053) = 0.890868596
8819597, error = 0.00012618021035293303
S(0.4000000000000036) = 0.8621644756575959, f(0.4000000000000036) = 0.862068965
5172411, error = 9.551014035480421e-05
S(0.4500000000000002) = 0.8316345475211351, f(0.4500000000000002) = 0.83160083160
08315, error = 3.371592030354531e-05
S(0.5) = 0.8, f(0.5) = 0.8, error = 0.0
S(0.5500000000000007) = 0.7678071745894864, f(0.5500000000000007) = 0.76775431861
80419, error = 5.285597144455423e-05
S(0.6000000000000005) = 0.7354077494977996, f(0.6000000000000005) = 0.73529411764
70584, error = 0.00011363185074120263
S(0.6500000000000004) = 0.7031047371113692, f(0.6500000000000004) = 0.70298769771
52897, error = 0.0001170393960795435
S(0.7000000000000002) = 0.6712011498166257, f(0.7000000000000002) = 0.67114093959
73154, error = 6.021021931035264e-05
S(0.75) = 0.6400000000000001, f(0.75) = 0.64, error = 1.1102230246251565e-16
S(0.8000000000000007) = 0.6097632100750122, f(0.8000000000000007) = 0.60975609756
09753, error = 7.112514036955453e-06
S(0.8500000000000005) = 0.5805883425635459, f(0.8500000000000005) = 0.58055152394
775, error = 3.681861579585277e-05
S(0.9000000000000004) = 0.5525318700145733, f(0.9000000000000004) = 0.55248618784
53037, error = 4.568216926958968e-05
S(0.9500000000000002) = 0.5256502649770669, f(0.9500000000000002) = 0.52562417871
22207, error = 2.6086264846236773e-05
S(1.0) = 0.4999999999999994, f(1.0) = 0.5, error = 5.551115123125783e-17
S(1.0500000000000007) = 0.4756219363299743, f(1.0500000000000007) = 0.47562425683
70983, error = 2.320507124009552e-06
S(1.1000000000000005) = 0.45249449000411257, f(1.1000000000000005) = 0.4524886877
8280527, error = 5.802221307305011e-06
S(1.1500000000000004) = 0.4305804657571659, f(1.1500000000000004) = 0.43057050592
034435, error = 9.95983682156032e-06
S(1.2000000000000002) = 0.409842668323886, f(1.2000000000000002) = 0.409836065573
77045, error = 6.602750115547451e-06
S(1.25) = 0.3902439024390244, f(1.25) = 0.3902439024390244, error = 0.0
S(1.3000000000000007) = 0.37174424160321246, f(1.3000000000000007) = 0.3717472118
959105, error = 2.9702926980235134e-06
S(1.3500000000000005) = 0.35429283438060283, f(1.3500000000000005) = 0.3542958370
2391476, error = 3.0026433119334506e-06
S(1.4000000000000004) = 0.33783609810122744, f(1.4000000000000004) = 0.3378378378
378377, error = 1.7397366102733791e-06
S(1.4500000000000002) = 0.32232045009511834, f(1.4500000000000002) = 0.3223207091
0556, error = 2.590104416499983e-07
S(1.5) = 0.3076923076923077, f(1.5) = 0.3076923076923077, error = 0.0
S(1.5500000000000007) = 0.29389940870182735, f(1.5500000000000007) = 0.2939015429
831005, error = 2.1342812731584004e-06
S(1.6000000000000005) = 0.28089477284870 96, f(1.6000000000000005) = 0.28089887640
449424, error = 4.103555784618074e-06
S(1.6500000000000004) = 0.26863274033698564, f(1.6500000000000004) = 0.2686366689
```

```
053055, error = 3.9285683198775345e-06
S(1.7000000000000002) = 0.2570676513706873, f(1.7000000000000002) = 0.2570694087403599, error = 1.7573696725614596e-06
S(1.75) = 0.24615384615384617, f(1.75) = 0.24615384615384617, error = 0.0
S(1.8000000000000007) = 0.23584761702287282, f(1.8000000000000007) = 0.23584905660377348, error = 1.4395809006528815e-06
S(1.8500000000000005) = 0.22611306484369506, f(1.8500000000000005) = 0.22611644997173533, error = 3.385128040267471e-06
S(1.9000000000000004) = 0.21691624261461923, f(1.9000000000000004) = 0.21691973963963123, error = 3.4970816930879334e-06
S(1.9500000000000002) = 0.20822320333395206, f(1.9500000000000002) = 0.20822488287350338, error = 1.6795395513247158e-06
S(2.0) = 0.2, f(2.0) = 0.2, error = 0.0
S(2.0500000000000007) = 0.19221432622016196, f(2.0500000000000007) = 0.19221528111148485, error = 9.548946865445274e-07
S(2.1000000000000005) = 0.1848404380382068, f(2.1000000000000005) = 0.18484288354089833, error = 2.445510776494242e-06
S(2.1500000000000004) = 0.1778542321069953, f(2.1500000000000004) = 0.17785682525566915, error = 2.5931486738461906e-06
S(2.2) = 0.1712316050793885, f(2.2) = 0.17123287671232876, error = 1.271632940252898e-06
S(2.25) = 0.16494845360824742, f(2.25) = 0.16494845360824742, error = 0.0
S(2.3000000000000007) = 0.15898187292269764, f(2.3000000000000007) = 0.1589825119236883, error = 6.39000990676486e-07
S(2.3500000000000005) = 0.15331375255692428, f(2.3500000000000005) = 0.15331544653123796, error = 1.6939743136823449e-06
S(2.4000000000000004) = 0.1479271806213769, f(2.4000000000000004) = 0.14792899408284022, error = 1.8134614633014134e-06
S(2.45) = 0.14280524522650512, f(2.45) = 0.14280614066404854, error = 8.954375434189199e-07
S(2.5) = 0.13793103448275862, f(2.5) = 0.13793103448275862, error = 0.0
S(2.5500000000000007) = 0.13328847159475687, f(2.5500000000000007) = 0.1332889036098767, error = 4.3210401012849786e-07
S(2.6000000000000005) = 0.1288648201437993, f(2.6000000000000005) = 0.12886597938144326, error = 1.1592376439606422e-06
S(2.6500000000000004) = 0.12464817880535506, f(2.6500000000000004) = 0.12464942349641632, error = 1.2446910612612383e-06
S(2.7) = 0.1206266462548933, f(2.7) = 0.12062726176115801, error = 6.155062647061893e-07
S(2.75) = 0.11678832116788322, f(2.75) = 0.11678832116788321, error = 1.3877787807814457e-17
S(2.8000000000000007) = 0.11312187612918569, f(2.8000000000000007) = 0.11312217194570132, error = 2.95816515630265e-07
S(2.8500000000000005) = 0.10961827936122887, f(2.8500000000000005) = 0.10961907371882705, error = 7.943575981855799e-07
S(2.9000000000000004) = 0.10626907299583246, f(2.9000000000000004) = 0.10626992561105206, error = 8.526152195931225e-07
S(2.95) = 0.10306579916481627, f(2.95) = 0.10306622004637979, error = 4.208815635 1760553e-07
S(3.0) = 0.1, f(3.0) = 0.1, error = 0.0
S(3.0500000000000007) = 0.09706361273483445, f(3.0500000000000007) = 0.09706381946129576, error = 2.067264613103692e-07
S(3.0999999999999996) = 0.09425015500929457, f(3.0999999999999996) = 0.09425070688030163, error = 5.518710070595567e-07
S(3.1500000000000004) = 0.09155353956498598, f(3.1500000000000004) = 0.09155413138017851, error = 5.918151925327075e-07
S(3.200000000000001) = 0.08896767914351465, f(3.200000000000001) = 0.08896797153024905, error = 2.923867343934683e-07
S(3.25) = 0.08648648648648648, f(3.25) = 0.08648648648648649, error = 1.3877787807814457e-17
```

```
S(3.300000000000007) = 0.08410414803541925, f(3.300000000000007) = 0.0841042893
1875523, error = 1.412833359748511e-07
S(3.3499999999999996) = 0.08181594503147942, f(3.3499999999999996) = 0.0818163223
5631009, error = 3.773248306720989e-07
S(3.400000000000004) = 0.07961743241574519, f(3.400000000000004) = 0.0796178343
9490444, error = 4.01979159250776e-07
S(3.450000000000001) = 0.07750416512929509, f(3.450000000000001) = 0.077504359620
2286, error = 1.9449093351109337e-07
S(3.5) = 0.07547169811320756, f(3.5) = 0.07547169811320754, error = 1.38777878078
14457e-17
S(3.550000000000007) = 0.07351578032030343, f(3.550000000000007) = 0.0735158978
1290202, error = 1.1749259859472616e-07
S(3.5999999999999996) = 0.07163293675037426, f(3.5999999999999996) = 0.0716332378
2234958, error = 3.010719753226976e-07
S(3.650000000000004) = 0.06981988641495392, f(3.650000000000004) = 0.0698202129
5164949, error = 3.2653669557181075e-07
S(3.700000000000001) = 0.06807334832557646, f(3.700000000000001) = 0.068073519400
95299, error = 1.710753765316042e-07
S(3.75) = 0.06639004149377595, f(3.75) = 0.06639004149377593, error = 1.387778780
7814457e-17
S(3.800000000000007) = 0.06476681603356342, f(3.800000000000007) = 0.0647668393
7823832, error = 2.334467490150427e-08
S(3.8499999999999996) = 0.06320104646885895, f(3.8499999999999996) = 0.0632011376
2047717, error = 9.115161822559337e-08
S(3.900000000000004) = 0.061690238426059564, f(3.900000000000004) = 0.061690314
620604564, error = 7.619454499979694e-08
S(3.950000000000001) = 0.060231897531562426, f(3.950000000000001) = 0.06023189278
723082, error = 4.744331608130814e-09
S(4.0) = 0.0588235294117647, f(4.0) = 0.058823529411764705, error = 6.93889390390
7228e-18
S(4.050000000000001) = 0.05746276000535366, f(4.050000000000001) = 0.057463008188
478645, error = 2.48183124988699e-07
S(4.1) = 0.05614769650017755, f(4.1) = 0.05614823133071309, error = 5.34830535539
5988e-07
S(4.15) = 0.05487656639637472, f(4.15) = 0.054877212237618316, error = 6.45841243
5961236e-07
S(4.200000000000001) = 0.053647597194083595, f(4.200000000000001) = 0.05364806866
952788, error = 4.7147544428305377e-07
S(4.25) = 0.05245901639344262, f(4.25) = 0.05245901639344262, error = 0.0
S(4.300000000000001) = 0.05130904287546676, f(4.300000000000001) = 0.051308363263
211886, error = 6.796122548752281e-07
S(4.35) = 0.0501958610446776, f(4.35) = 0.05019450370184466, error = 1.3573428329
430315e-06
S(4.4) = 0.049117646686473254, f(4.4) = 0.04911591355599213, error = 1.7331304811
227244e-06
S(4.450000000000001) = 0.048072575586251914, f(4.450000000000001) = 0.04807114529
503664, error = 1.430291215277768e-06
S(4.5) = 0.047058823529411764, f(4.5) = 0.047058823529411764, error = 0.0
S(4.550000000000001) = 0.04607492083669109, f(4.550000000000001) = 0.046077640824
78975, error = 2.7199880986636393e-06
S(4.600000000000001) = 0.04512081597018902, f(4.600000000000001) = 0.045126353790
61369, error = 5.537820424665607e-06
S(4.65) = 0.04419681192734487, f(4.65) = 0.044203779423140674, error = 6.96749579
5803802e-06
S(4.700000000000001) = 0.04330321170559784, f(4.700000000000001) = 0.043308791684
711974, error = 5.5799791141356025e-06
S(4.75) = 0.04244031830238726, f(4.75) = 0.042440318302387266, error = 6.93889390
3907228e-18
S(4.800000000000001) = 0.04160735055504664, f(4.800000000000001) = 0.041597337770
382686, error = 1.0012784663955554e-05
```

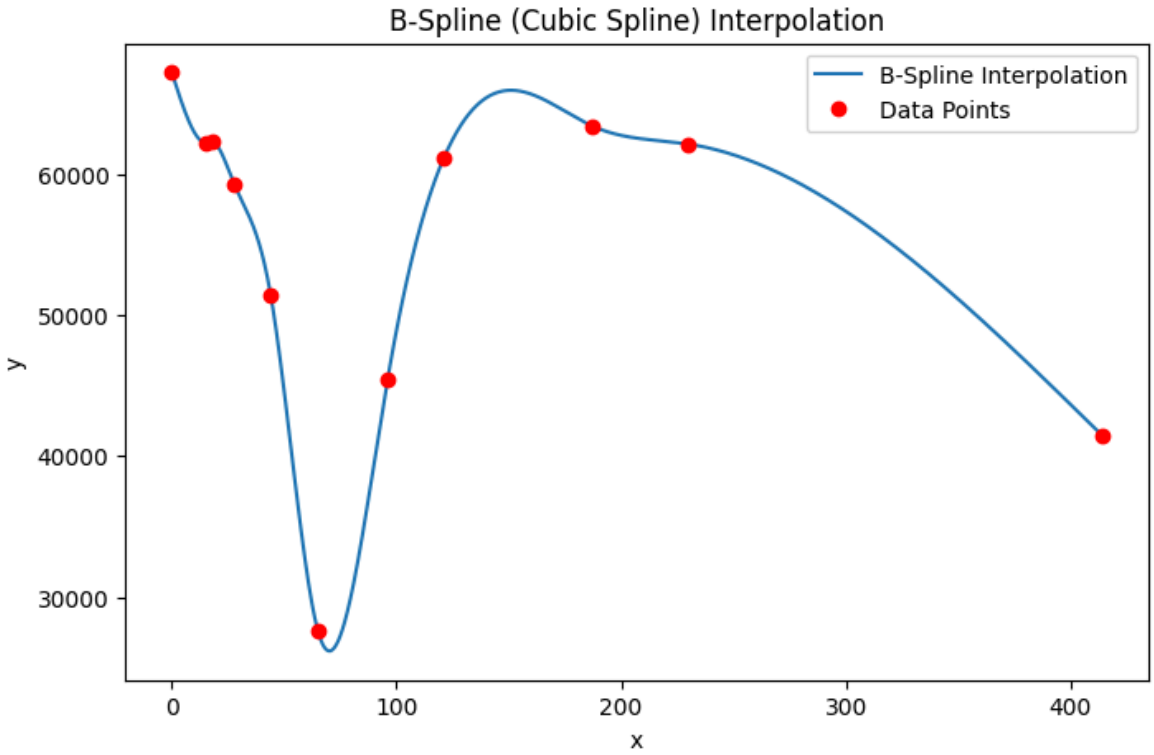
```
S(4.850000000000001) = 0.04079919066048659, f(4.850000000000001) = 0.040778876541
951246, error = 2.0314118535341708e-05
S(4.9) = 0.040009636655511986, f(4.9) = 0.03998400639744101, error = 2.5630258070
973022e-05
S(4.950000000000001) = 0.039232486576927635, f(4.950000000000001) = 0.03921184197
627682, error = 2.0644600650814027e-05
S(5.0) = 0.038461538461538464, f(5.0) = 0.038461538461538464, error = 0.0
```

In [37]:
```python
# 6.2 - 11
x = [-1, 0, 1]
y = [1, 2, -1]
x_eval = np.linspace(-1, 1, 51)

S = cubic_spline_interpolation_eval(x, -1, 1, 3, x_eval, y)

plt.plot(x_eval, S, label='Cubic Spline Interpolation')
plt.scatter(x, y, color='red', label='Given Points')
plt.xlabel('x')
plt.ylabel('S(x)')
plt.title('Cubic Spline Interpolation with Given Points')
plt.legend()
plt.show()
```



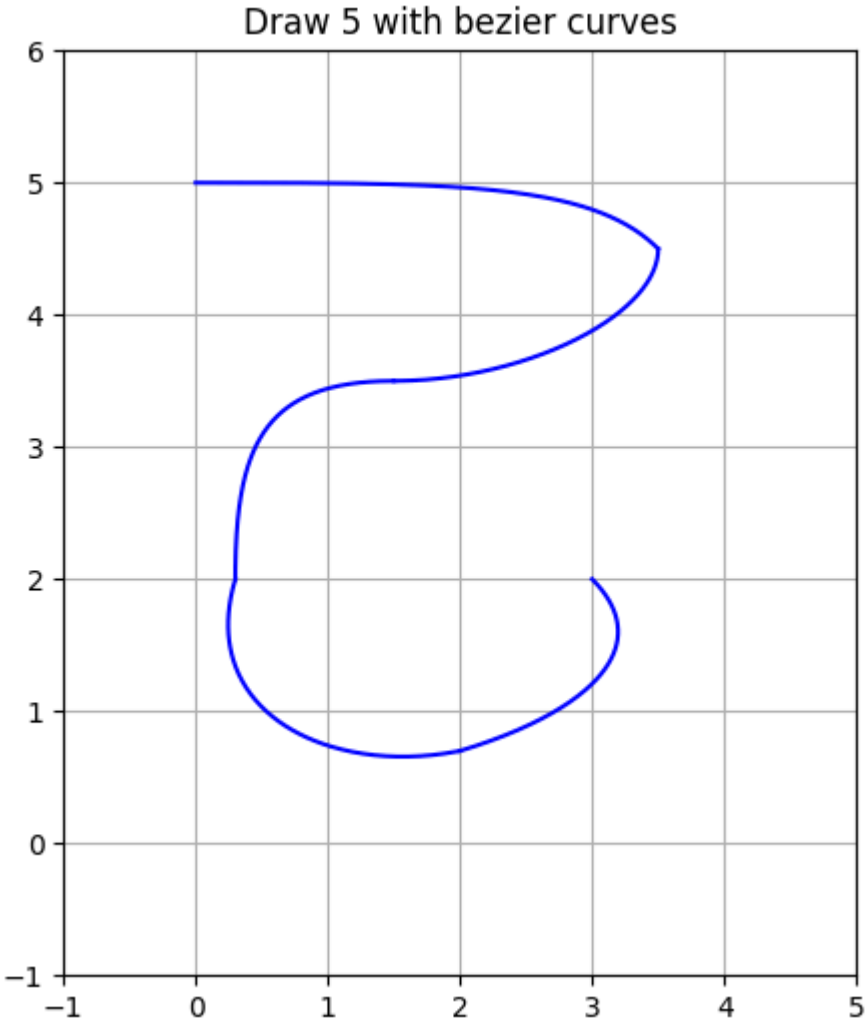
In [38]:
```python
# 6.3 - 7

def simple_trapezoidal(x, y):
    h = np.diff(x)
    ret = 0
    for i in range(len(h)):
        ret += (y[i] + y[i + 1]) * h[i] / 2
    return ret
```

```
def bspline_2nd_coef(f, a, b, n, x=None, y=None):
    if x is None:
        x = np.linspace(a, b, n)
    else:
        n = len(x)
    if y is None:
        y = f(x)

    h = np.diff(x)
    A = np.zeros((n, n))
    b_vec = np.zeros(n)

    A[0, 0] = 1
    A[-1, -1] = 1

    for i in range(1, n-1):
        A[i, i-1] = h[i-1]
        A[i, i]   = 2*(h[i-1] + h[i])
        A[i, i+1] = h[i]
        b_vec[i]  = 6 * ((y[i+1] - y[i]) / h[i] - (y[i] - y[i-1]) / h[i-1])

    M = np.linalg.solve(A, b_vec)
    return M, x

def bspline_2nd_eval(f, a, b, n, x_eval, x=None, y=None):
    M, x = bspline_2nd_coef(f, a, b, n, x, y)
    if y is None:
        y = f(x)

    S = np.zeros_like(x_eval)
    for i, xi in enumerate(x_eval):
        j = np.searchsorted(x, xi) - 1
        if j < 0:
            j = 0
        elif j >= len(x) - 1:
            j = len(x) - 2
        h_j = x[j+1] - x[j]
        A = (x[j+1] - xi) / h_j
        B = (xi - x[j]) / h_j
        S[i] = A * y[j] + B * y[j+1] + ((A**3 - A) * M[j] + (B**3 - B) * M[j+1])
    return S

x_nodes = np.array([0, 15, 18, 28, 44, 65, 96, 121, 187, 230, 414])
y_nodes = np.array([67259, 62280, 62350, 59250, 51457, 27603, 45435, 61162, 6345

x_eval = np.linspace(0, 414, 1000000)
S = bspline_2nd_eval(lambda t: np.sin(t), 0, 230, len(x_nodes), x_eval, x_nodes,

plt.figure(figsize=(8, 5))
plt.plot(x_eval, S, label='B-Spline Interpolation')
plt.plot(x_nodes, y_nodes, 'ro', label='Data Points')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.title('B-Spline (Cubic Spline) Interpolation')
plt.show()

integral = simple_trapezoidal(x_eval, S)
print(f"Integral of B-Spline interpolation: {integral}")
print(f"Average value of B-Spline interpolation: {integral / (x_eval[-1] - x_eva
```




```
Integral of B-Spline interpolation: 22720667.520836458
Average value of B-Spline interpolation: 54880.83942231028
```

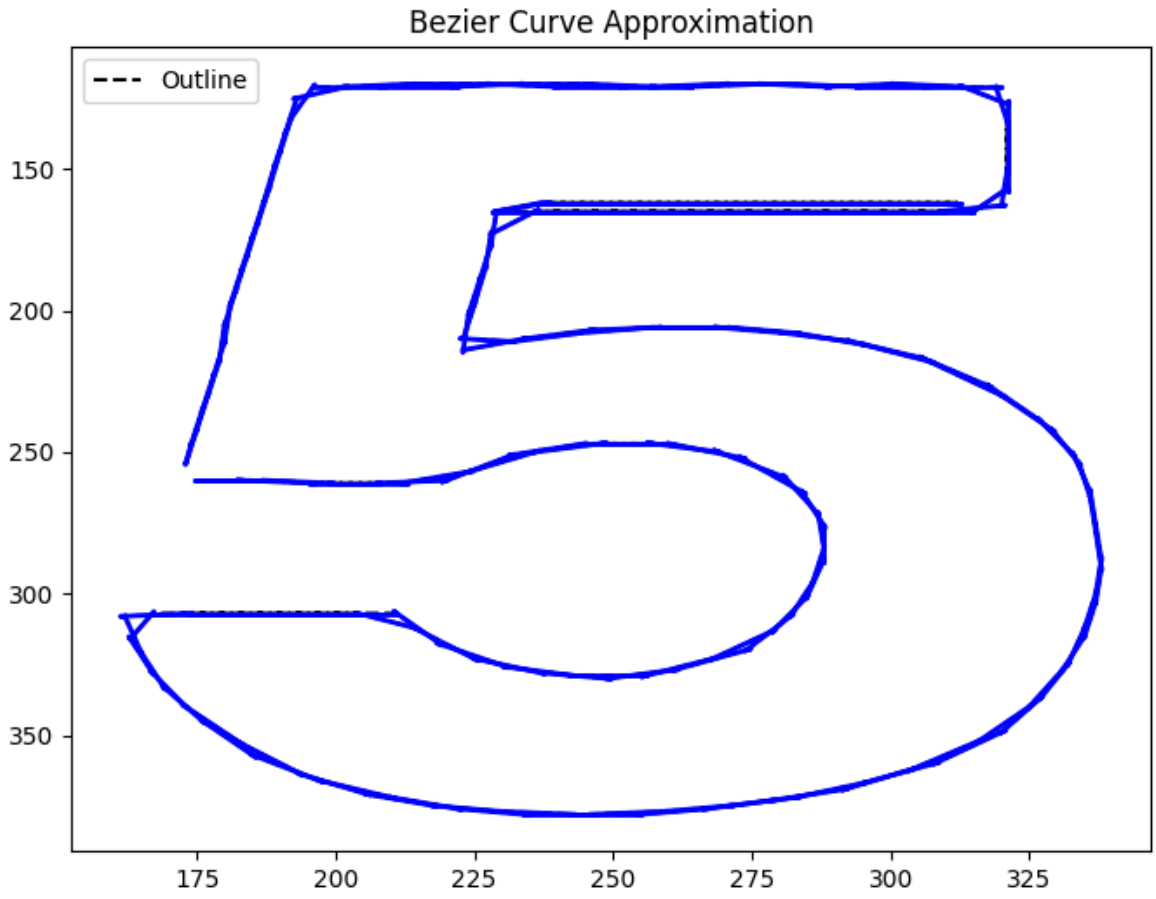
In [ ]:
```python
# 6.3 - 19

def bezier_curve(p0, p1, p2, p3, num_points=100):
    t = np.linspace(0, 1, num_points)
    x = (1 - t)**3 * p0[0] + 3 * (1 - t)**2 * t * p1[0] + 3 * (1 - t) * t**2 * p
    y = (1 - t)**3 * p0[1] + 3 * (1 - t)**2 * t * p1[1] + 3 * (1 - t) * t**2 * p
    return x, y

bezier_segments = [
    [(0, 5), (2, 5), (3, 5), (3.5, 4.5)],
    [(3.5, 4.5), (3.5, 4), (2.5, 3.5), (1.5, 3.5)],
    [(1.5, 3.5), (0.5, 3.5), (0.3, 3), (0.3, 2)],
    [(0.3, 2), (0, 1), (1, 0.5), (2, 0.7)],
    [(2, 0.7), (3, 1), (3.5, 1.5), (3, 2)]
]

plt.figure(figsize=(6, 6))
all_x, all_y = [], []

for seg in bezier_segments:
    p0, p1, p2, p3 = seg
    x_vals, y_vals = bezier_curve(p0, p1, p2, p3, num_points=200)

    plt.plot(x_vals, y_vals, 'b-')

    all_x.extend(x_vals)
    all_y.extend(y_vals)

plt.xlim(-1, 5)
plt.ylim(-1, 6)
plt.gca().set_aspect('equal', adjustable='box')
plt.title("Draw 5 with bezier curves")
plt.grid(True)
plt.show()
```



In [ ]:
```python
def bezier_point(t, P0, P1, P2, P3):
    return (1-t)**3 * P0 + 3*(1-t)**2 * t * P1 + 3*(1-t) * t**2 * P2 + t**3 * P3

def evaluate_bezier(P0, P1, P2, P3, num=100):
    t_vals = np.linspace(0, 1, num)
    curve = np.array([bezier_point(t, P0, P1, P2, P3) for t in t_vals])
    return curve

def chord_length_parameterize(points):
    distances = [0]
    for i in range(1, len(points)):
        distances.append(np.linalg.norm(points[i] - points[i-1]))
    cumulative = np.cumsum(distances)
    total = cumulative[-1]
    if total == 0:
        return [0 for _ in cumulative]
    return cumulative / total

def bezier_curve(points, t_vals, left_tangent, right_tangent):
    n = len(points)
    a = np.zeros((n, 2, 2))
    c = np.zeros((2, 2))
    x = np.zeros(2)

    p0 = points[0]
    p3 = points[-1]

    for i in range(n):
```

```
        t = t_vals[i]
        inv_t = 1 - t
        b0 = inv_t**3
        b1 = 3 * inv_t**2 * t
        b2 = 3 * inv_t * t**2
        b3 = t**3

        a[i][0] = left_tangent * b1
        a[i][1] = right_tangent * b2

        tmp = points[i] - (p0 * b0 + p3 * b3)
        c[0][0] += np.dot(a[i][0], a[i][0])
        c[0][1] += np.dot(a[i][0], a[i][1])
        c[1][0] += np.dot(a[i][0], a[i][1])
        c[1][1] += np.dot(a[i][1], a[i][1])

        x[0] += np.dot(a[i][0], tmp)
        x[1] += np.dot(a[i][1], tmp)

    det_C = c[0][0]*c[1][1] - c[0][1]*c[1][0]
    if abs(det_C) < 1e-12:
        alpha1 = alpha2 = np.linalg.norm(p3 - p0) / 3.0
    else:
        sol = np.linalg.solve(c, x)
        alpha1, alpha2 = sol[0], sol[1]

    seg_length = np.linalg.norm(p3 - p0)
    if alpha1 < 1e-6 or alpha2 < 1e-6:
        alpha1 = alpha2 = seg_length / 3.0

    p1 = p0 + left_tangent * alpha1
    p2 = p3 + right_tangent * alpha2
    return p0, p1, p2, p3

def compute_max_error(points, bezier, t_vals):
    max_error = 0.0
    split_point = len(points) // 2
    for i in range(1, len(points)-1):
        p_curve = bezier_point(t_vals[i], *bezier)
        error = np.linalg.norm(p_curve - points[i])
        if error > max_error:
            max_error = error
            split_point = i
    return max_error, split_point

def tangent(points, index):
    if index == 0:
        tangent = points[1] - points[0]
    elif index == len(points)-1:
        tangent = points[-1] - points[-2]
    else:
        tangent = points[index+1] - points[index-1]
    norm = np.linalg.norm(tangent)
    if norm == 0:
        return tangent
    return tangent / norm

def bezier_curve_fitting(points, error_tolerance):
    t_vals = chord_length_parameterize(points)
    left_tangent = tangent(points, 0)
```

```
    right_tangent = tangent(points, len(points)-1)
    bezier = bezier_curve(points, t_vals, left_tangent, right_tangent)

    max_error, split_point = compute_max_error(points, bezier, t_vals)

    if max_error < error_tolerance:
        return [bezier]
    else:
        left_beziers = bezier_curve_fitting(points[:split_point+1], error_tolera
        right_beziers = bezier_curve_fitting(points[split_point:], error_toleran
        return left_beziers + right_beziers

def detecting_outlines(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    edged = cv2.Canny(blurred, 50, 150)

    contours, _ = cv2.findContours(edged, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NO
    if not contours:
        return None

    largest_contour = max(contours, key=cv2.contourArea)
    return largest_contour

def main(image_path, error_tolerance=4.0):
    image = cv2.imread(image_path)
    if image is None:
        IOError(f"Image not found: {image_path}")
        return
    contour = detecting_outlines(image)
    if contour is None or len(contour) < 2:
        ValueError("Outline Not fuond or too few points")
        return

    points = contour.reshape(-1, 2).astype(np.float32)

    if len(points) > 200:
        indices = np.linspace(0, len(points)-1, 200).astype(int)
        points = points[indices]

    beziers = bezier_curve_fitting(points, error_tolerance)

    plt.figure(figsize=(8, 6))
    plt.plot(points[:,0], points[:,1], 'k--', label="Outline")
    for bezier in beziers:
        curve = evaluate_bezier(*bezier, num=100)
        plt.plot(curve[:,0], curve[:,1], 'b-', linewidth=2)
        # ctrl_points = np.array(bezier)
        # plt.plot(ctrl_points[:,0], ctrl_points[:,1], 'ro--', markersize=4)
    plt.gca().invert_yaxis()
    plt.title("Bezier Curve Approximation")
    plt.legend()
    plt.show()


main("5.jpg", error_tolerance=4.0)
```


Bezier Curve Approximation
Outline
150
200
250
300
350
175
200
225
250
275
300
325