Numerical Analysis Homework 4

Lucas MacQuarrie (20234554)

March 26, 2025

1)

By calculating $B_{i,3}(x)$, show explicitly that if two of the nodes in $\{x_i, x_{i+1}, x_{i+2}, x_{i+3}\}$ coincide, then $B_{i,3}(x)$ is continuous but not differentiable at the coincident node. Similarly, if three of them coincide, show that $B_{i,3}(x)$ is discontinuous at that point.

Solution:

I'll first solve the problem when 2 nodes coincide.

W.L.O.G I'll let the nodes be $x_0, \ldots, x_3$ and suppose $x_j = x_{j+1}$ for some $j \in 0, 1, 2$. A simple calculation shows

$$\begin{aligned}
B_{i,3}(x) &= f_x^2[x_0, \ldots, x_3] \\
&= (x_3 - x_0)\frac{f_x^2[x_1, \ldots, x_3] - f_x^2[x_0, \ldots, x_2]}{x_3 - x_0} \\
&= f_x^2[x_1, x_2, x_3] - f_x^2[x_0, x_1, x_2] \\
&= \frac{f_x^2[x_2, x_3] - f_x^2[x_1, x_2]}{x_3 - x_1} - \frac{f_x^2[x_1, x_2] - f_x^2[x_0, x_1]}{x_2 - x_0}.
\end{aligned}$$

Now for the nodes that do not coincide say, x_i and x_{i+1}, we have that

$$f_x^2[x_i, x_{i+1}] = \frac{f_x^2(x_{i+1}) + f_x^2(x_i)}{x_{i+1} - x_i} = \frac{\max(x - x_{i+1}, 0)^2 + \max(x - x_i, 0)^2}{x_{i+1} - x_i}$$

which is continuous at the coincident node since both $\max(x - x_{i+1}, 0)^2$ and $\max(x - x_i, 0)^2$ are. Now for the nodes that *do* coincide we have that

$$f_x^2[x_j, x_{j+1}] = \frac{\partial}{\partial x} f_x^2(x_j) = \frac{\partial}{\partial x}\max(x - x_j, 0)^2 = 2 \cdot \max(x - x_j, 0)$$

which is continuous but not differentiable at $x = x_j$ as required.

Now suppose three nodes coincide. A similar calculation as before shows

$$\begin{aligned}
B_{i,3}(x) &= f_x^2[x_0, \ldots, x_3] \\
&= (x_3 - x_0)\frac{f_x^2[x_1, \ldots, x_3] - f_x^2[x_0, \ldots, x_2]}{x_3 - x_0} \\
&= f_x^2[x_1, x_2, x_3] - f_x^2[x_0, x_1, x_2]
\end{aligned}$$

and W.L.O.G we assume $x_1 = x_2 = x_3$. We have that

$$f_x^2[x_1, x_2, x_3] = \frac{\partial^2}{\partial x^2} f_x^2(x_1) = \frac{\partial^2}{\partial x^2}\max(x - x_1, 0)^2 = 2 \cdot H(x - x_0)$$

where $H(x) = \begin{cases} 0, & x < 0 \\ 1, & x > 0 \end{cases}$ is the Heaviside step function which best describes the derivative of this function. This is an abuse of notation, because of course the derivative of $\max(x - x_1, 0)$ at x_1 is undefined due to the disagreement of the left and right limits. Irregardless, we have shown that $B_{i,3}(x)$ is discontinuous at this point x_j.

2)

We consider the spline $S(x)$ that is a linear combination of the cubic B-spline $B_{i,4}(x)$ with knots given by:

$$t = \{t_i \mid t_i = 0.1 + 0.2i, \quad i = -5, \ldots, -1, 0, 1, \ldots, 9\}.$$

The spline interpolates Runge's function:

$$f(x) = \frac{1}{1 + 25x^2},$$

on a set of 11 regularly spaced knots:

$$x_k = -1 + \frac{2k}{10}, \quad k = 0, 1, \ldots, 10.$$

We compute the interpolation spline $S(x)$ and the error $f(x) - S(x)$ at 41 regularly spaced points. The error can be visualized either through a plot or a table. Finally, we compare the results with those from the Computer Assignment in Assignment 3.

Solution:

The code to generate an interpolating B-Spline on the runge function over the specified knots and interpolating points is given below and a plot comparing the Runge function to its interpolated function is given in figure 2. Since the error is simply $\sum_i |y_i - \hat{y}_i|$ there is nothing to plot and we report it's value as ≈ 0.165. Comparing the results with Assignment 3 is seen in figure 1.

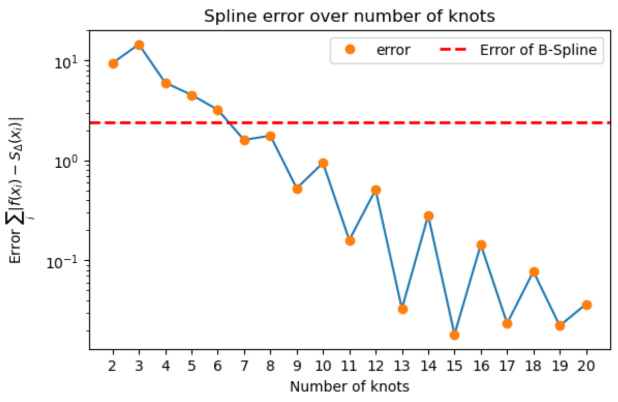

Figure 1: Error of the splines from Assignment 3. The error of the B-spline in assignment 4 is plotted as the dashed red line.

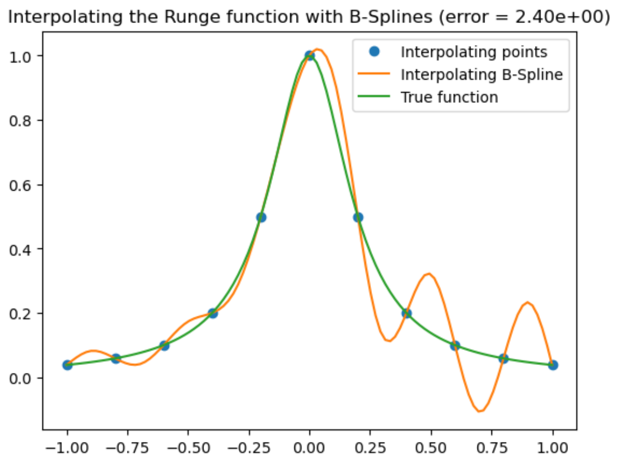

Figure 2: Plot comparing the interpolating B-Spline with the true Runge function.

```
###This is for figure 1###
import numpy as np
from scipy.interpolate import BSpline
from scipy.linalg import solve
import matplotlib.pyplot as plt


def runge(x: float) -> float:
    return 1 / (1 + 25 * x**2)


"""
Evaluate a B-spline basis function over the knots t at a point x
"""
def B(x, k, i, t):
    if k == 0:
        return 1.0 if t[i] <= x < t[i + 1] else 0.0
    if t[i + k] == t[i]:
        c1 = 0.0
    else:
        c1 = (x - t[i]) / (t[i + k] - t[i]) * B(x, k - 1, i, t)
    if t[i + k + 1] == t[i + 1]:
        c2 = 0.0
    else:
        c2 = (t[i + k + 1] - x) / (t[i + k + 1] - t[i + 1]) * B(x, k - 1, i + 1, t)
    return c1 + c2

# Degree of the B-spline
k = 3
# Knots
ts = [-0.1 + 0.2 * k for k in range(-5, 9, 1)]
# Pad the knots with the same initial and final values
ts = [ts[0]] * (k-2) + ts + [ts[-1]] * (k-2)
# Interpolating points
xs = [-1 + 0.2 * k for k in range(11)]
ys = [runge(x) for x in xs]

# Number of basis functions
n_basis = len(xs)
```

```
# Initialize the matrix
# Initialize the B-spline basis matrix
B_matrix = np.zeros((n_basis, n_basis))

# Compute the matrix
for i, x in enumerate(xs):
    for j in range(n_basis):
        B_matrix[i, j] = B(x, k, j, ts)

# Display the B-spline basis matrix in a readable format
print("B-spline Basis Matrix:")
print(np.array2string(B_matrix, formatter={'float_kind': lambda x: f"{x:6.4f}"}))

c = solve(B_matrix, ys)

def bspline(x, t, c, k):
   n = len(c)
   #assert (n >= k+1) and (len(c) >= n)
   return sum(c[i] * B(x, k, i, t) for i in range(n))
# Print the resulting coefficients
print("Coefficients:")
print(c)

# Calculate error
xbars = np.linspace(-1, 1, 41)
ybars = [bspline(x, ts, c, k) for x in xbars]
ytrue = [runge(x) for x in xbars]
error = np.sum(np.abs(np.array(ybars) - np.array(ytrue)))
# Plotting
x_plot = np.linspace(min(xs), max(xs), 100)
y_plot = [bspline(x, ts, c, k) for x in x_plot]
plt.plot(xs, ys, "o", label="Interpolating points")
plt.plot(x_plot, y_plot, label="Interpolating B-Spline")
plt.plot(x_plot, [runge(x) for x in x_plot], label="True function")
plt.legend(loc="best")
plt.title(f"Interpolating the Runge function with B-Splines (error = {error:.2e})")

print(error)

###This is for figure 2###

import numpy as np
from scipy.interpolate import CubicSpline
import matplotlib.pyplot as plt

def runge(x: float) -> float:
    return 1 / (1 + 25 * x**2)

errors = []
steps = range(2,21)
for n in steps:
    #First fit the points
    x = np.linspace(-1,1, n)
    y = runge(x)
    cs = CubicSpline(x, y,bc_type=((2, 0.0), (2, 0.0)))
    #Then calculate the error
    xs = np.linspace(-1, 1, 41)
    ybars= cs(xs)
    error = np.sum(np.abs(ybars - runge(xs)))
    errors.append(error)

print(steps)
```

```
fig, ax = plt.subplots(figsize=(6.5, 4))
ax.plot(steps, errors)
ax.plot(steps, errors, 'o', label='error')

ax.set_title('Spline error over number of knots')
ax.axhline(2.4,linestyle='--', color='red', lw=2, label='Error of B-Spline')
ax.set_xlabel('Number of knots')
ax.set_ylabel(r'Error $\sum_i|f(x_i) - S_\Delta(x_i)|$')

ax.set_yscale('log')
ax.set_xticks(steps)
ax.legend(loc='best', ncol=2)

plt.show()

for i,e in enumerate(errors):
    print(f"n={steps[i]}: {e}")
```