Assignment

Numerical Differentiation

Name TN 1: Tim Peinkofer

tim.peinkofer@student.uni-tuebingen.de

Name TN 2: Fabian Kostow

fabian.kostow@student.uni-tuebingen.de

Tutor: Jose Carlos Olvera Meneses

Date: 29. Januar 2025

Inhaltsverzeichnis

1 Problem III - Differentiation

1.1 Part A

In the following we want to calculate the derivative of

$$f(x) = arctan(x)$$

at $x = 0$ via Richardson extrapolation and Central Differences method.
Let us first take a look at the Richardson extrapolation:

With the Richardson extraploation we can calculate the value of the derivative at $x = x_0$ using following equation:

$$\begin{aligned} y'(x_0) &= \Psi(h) = \frac{1}{3}(4F(h/2) - F(h)) \\ E(x_0) &= 4\Psi(h/2) - \Psi(h) \\ F(h) &= \frac{1}{2h} \cdot (y(x_0 + h) - y(x_0 - h)) \end{aligned}$$

where h is our step and E our error.

So in conclusion we get the following code:

```
import numpy as np

def func(x):
    return np.arctan(x)

# Approximation of the function values
def F(h, x):
    return 1/(2*h)*(func(x + h) - func(x - h))

# Define function for Richardson extrapolation
def psi(h, x):
    return (4 * F(h / 2, x) - F(h, x)) / 3

# Richardson Extrapolation
def Richardson(x, h):
    value = psi(h, x)
    error = abs(value - psi(h / 2, x))  # Estimate error of
                                         our calculation
    return value, error

# Calculate derivative via Richradson extrapolation
result, error = Richardson(0, np.pi / 6)
print(f"Richardson Extrapolated Value: {result}")
print(f"Estimated Error: {error}")
```

And the results:

$$y'(x_0) \approx 0.99699383$$
$$E(x_0) \approx 0.00278489$$

We now try to calculate the same derivative with the central differnces method to compare both results.

From the lecture sheet we know that the formula for the first derivative is:

$$y'(x_0) = \frac{-y_2 + 8y_1 + 8y_{-1} - y_{-2}}{12h}$$

The code for this method is:

```python
import numpy as np

def func(x):
    return np.arctan(x)

def Central_diff_first_deri(x, h):
    # Get the value of the function
    y1 = func(x + h)
    y_1 = func(x - h)
    y_2 = func(x - 2*h)
    y2 = func(x + 2*h)

    # Apply the formula for central differnces
    value = (y_2 - y2 + 8*(y1-y_1)) / (12 * h)
    return value

# Calculating error and result
result = Central_diff_first_deri(0, np.pi / 6)
error = abs(result - Central_diff_first_deri(0,1e-5))
print(f"Derivatibve value: {result}")
print(f"Error: {error}")
```

And the results:

$$y'(x_0) \approx 0.97095161$$
$$E(x_0) \approx 0.02904838$$

As we see the result of the Central Differnce method is less accurate then of the Richardson extrapolation. The reason is that the error of the Central Differnces method ist of order h^2 while the Richardson is of order h^4.

1.2 Part B

In the next step we want to calculate the same derivative using spline interpolation. The basic formula for a cubic spline is:

$$y_i(x) = a_i \cdot (x - x_i)^3 + b_i \cdot (x - x_i)^2 + c_i \cdot (x - x_i) + d_i$$

If we calculate the derivative for two differnt x_i and simplify the equations, we get:

$$\begin{pmatrix} \ddots & & & & & \\ h_0 & 2(h_0+h_1) & h_1 & & & \\ & h_1 & 2(h_1+h_2) & h_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ & & & & & \ddots \end{pmatrix} \begin{pmatrix} S_0 \\ S_1 \\ S_2 \\ \vdots \\ S_n \end{pmatrix} = 6 \begin{pmatrix} \frac{y_1-y_0}{h_0} \\ \frac{y_2-y_1}{h_1} \\ \frac{y_3-y_2}{h_2} \\ \vdots \\ \frac{y_{n-1}-y_{n-2}}{h_{n-2}} \\ \frac{y_n-y_{n-1}}{h_{n-1}} \end{pmatrix}$$

which gives us the following for each interval i:

$$\begin{aligned} a_i &= \frac{S_{i+1} - S_i}{6h_i} \\ b_i &= \frac{S_i}{2} \\ c_i &= \frac{y_{i+1} - y_i}{h_i} - \frac{h_i S_{i+1} + 2h_i S_i}{6} \\ d_i &= y_i \end{aligned}$$

We will now use the boundary condition for linear extrapolation, which is:

$$\begin{aligned} S_0 &= \frac{(h_0 + h_1)S_1 - h_0 S_2}{h_1} \\ S_n &= \frac{(h_{n-2} + h_{n-1})S_{n-1} - h_{n-1} S_{n-2}}{h_{n-2}} \end{aligned}$$

If we combine all we get the following code:

```
import numpy as np

def func(x):
    return np.arctan(x)  # Funktion zur Auswertung

x = np.array([-6,-5,-4,-3,-2, -1, 0, 1, 2,3,4,5,6])  # Array
                               mit St tzstellen
values = func(x)  # Funktionswerte
n = len(x) - 1  # Anzahl der Intervalle f r die
                               Interpolation

def h(i):
    return x[i + 1] - x[i]  # Schrittweite zwischen den
                                   Punkten

def cubic_splines():
    if n < 2:
        print("Nicht gen gend Punkte f r Splines")
        return None, None, None

    matrix = np.zeros((n - 1, n - 1), dtype=float)
    vec = np.zeros(n - 1, dtype=float)

    matrix[0, 0] = (h(0) + h(1)) * (h(0) + 2 * h(1)) / h(1)
    if n > 2:
        matrix[0, 1] = (h(1)**2 - h(0)**2) / h(1)

    vec[0] = (values[2] - values[1]) / h(1) - (values[1] -
                                   values[0]) / h(0)

    for i in range(1, n - 2):
        matrix[i, i - 1] = h(i)
        matrix[i, i] = 2 * (h(i) + h(i + 1))
        matrix[i, i + 1] = h(i + 1)
        vec[i] = (values[i + 2] - values[i + 1]) / h(i + 1) -
                                        (values[i + 1] -
                                       values[i]) / h(i)

    if n > 2:
        matrix[n - 2, n - 3] = (h(n - 3)**2 - h(n - 2)**2) /
                                       h(n - 3)
        matrix[n - 2, n - 2] = (h(n - 2) + h(n - 1)) * (h(n -
                                        2) + 2 * h(n - 1)) /
                                       h(n - 1)
    vec[n - 2] = (values[n] - values[n - 1]) / h(n - 1) - (
                                              values[n - 1] - values[n -
                                              2]) / h(n - 2)
    vec *= 6

    if np.linalg.cond(matrix) > 1e12:
        print("Matrix ist schlecht konditioniert, L sung
                                               k nnte ungenau sein."
                                               )
        return matrix, vec, None

    try:
        solution = np.linalg.solve(matrix, vec)
        S_0 = ((h(0) + h(1)) * solution[0] - h(0) * solution[
                                               1]) / h(1)
        S_n = ((h(n - 2) + h(n - 1)) * solution[n - 2] - h(n
                                               - 1) * solution[n - 3]
                                               ) / h(n - 2)

        solution = np.append(solution, S_n)
        solution = np.insert(solution, 0, S_0)
        return matrix, vec, solution
    except np.linalg.LinAlgError:
        print("Keine L sung vorhanden")
        return matrix, vec, None

m, v, sol = cubic_splines()

if sol is not None:
    def spline_derivative(i, x_val):
        if x[i] <= x_val <= x[i + 1]:
            a = (sol[i + 1] - sol[i]) / (6 * h(i))
            b = sol[i] / 2
            c = (values[i + 1] - values[i]) / h(i) - h(i) / 6
                                                    * (2 * sol[i] +
                                                   sol[i + 1])
            return 3 * a * (x_val - x[i])**2 + 2 * b * (x_val
                                                    - x[i]) + c
        else:
            print(f"x_val = {x_val} liegt au erhalb des
                                                   Intervalls [{x[i]}
                                                   , {x[i+1]}]")
            return None

    derivative_at_0 = spline_derivative(5, 0)  # Index
                                              angepasst f r x=0
    if derivative_at_0 is not None:
        print(f"Approx. Ableitung von arctan(x) bei x=0: {
                                           derivative_at_0}")
        print(f"Exakte Ableitung: {1 / (1 + 0**2)}")
        print(f"Error: {1-derivative_at_0}")
```

The result is the following:

$$y'(x_0) \approx 0.897839940$$
$$E(x_0) \approx 0.102160059$$