Assignment

Exercise sheet 6 - Integration

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: 2. Dezember 2024

Inhaltsverzeichnis

1 Problem I

We want to combine Simpsons 1/3 rule with Romberg therefore we need Simpsons with step size h and $\frac{h}{2}$.

$$
\begin{aligned}
I_1 &= \frac{h_1}{3}(f_0 + 4f_2 + f_4) - \frac{1}{180}h_1^4 f^{(4)}(\xi_1) \\
I_2 &= \frac{h_1}{6}(f_0 + 4f_1 + 2f_2 + 4f_3 + f_4) - \frac{1}{2880}h_1^4 f^{(4)}(\xi_2) \\
h_1 &= 2h
\end{aligned}
$$

Where h is the step size for five points. We can plug those in Romberg with $k = \frac{1}{2}$ and $O(h^4)$:

$$A = I_2 + \frac{I_2 - I_1}{2^4 - 1}$$

Than we get:

$$
\begin{aligned}
A = &\left[\frac{h_1}{6}(f_0 + 4f_1 + 2f_2 + 4f_3 + f_4) - \frac{1}{2880}h_1^4 f^{(4)}(\xi_1)\right] \\
&+ \frac{h_1}{90} \cdot \frac{-f_0 + 4f_1 - 6f_2 + 4f_3 - f_4}{15} + \frac{1}{2800}h_1^4 f^{(4)}(\xi_3)
\end{aligned}
$$

Which we can combine to :

$$A = \frac{h_1}{45}\left(7f_0 + 32f_1 + 12f_2 + 32f_3 + 7f_4\right) + O(h^7)$$

Finally, we replaced h_1 with h:

$$A = \frac{2h_1}{45}\left(7f_0 + 32f_1 + 12f_2 + 32f_3 + 7f_4\right) + \frac{8}{945}h^7 f^{(6)}(\xi)$$

As we see the result is the fourth formula in table 1. Calculating the error term is quite hard, because the higher order errors ar not defined directly. Because of that we used the Error from the lecture sheet.

2 Problem II

In the following problem we try to calculate the integral of the following function with different methods:

$$f(x) = \frac{2^x sin(x)}{x}$$

Because of no limits are given, we will calculate the integral from 1 to π.

2.1 Simpson 1/3 Rule

To calculate the an integral via Simpson 1/3 method, we have to calculate the value of the functions at different x_i. The formula for Simpson 1/3 method is the following, if we divide the set [1,b] in n subsets:

$$\begin{aligned}\int_a^b f(x)\,dx &= \frac{b-a}{2\cdot 3}\cdot(f(x_0)+4f(x_1)+2f(x_2)+...+f(x_n)) \\ &= \frac{h}{3}\cdot(f_0+4f_1+...+f_n)\end{aligned}$$

The code for this method is the following:

```
import numpy as np


def func(x): # Define function
    return 2**x*np.sin(x)/x



def Simpson(h,values,x): #Simpson Rule 1/3
    sum = 0
    sum = values[0]+values[-1] # Get the sum of the values of
                                the integral limits
```

```
    for i in range(1,len(x)): # Sum all other values based on
                                 the number of steps

        if i % 2 == 0: # Multiply all odd index values with 2
                                    and the others with 4
                                    and calculate the sum
            sum += 4*values[i]

        else:
            sum += 2*values[i]

    result = h/3*sum # Get the result

    return result

sol = {}


indices = [300, 600]


for i, index in enumerate(indices):
    x = np.linspace(1, np.pi, index)
    f_x = [func(x_i) for x_i in x]
    h = (x[-1] - x[0]) / (len(x) - 1)  # Calculate step size
                                 for our iteration
    sol[i] = Simpson(h, f_x, x)



err = np.abs(sol[1]-sol[0]) #Calculate the error

print("Solution of the integral:\n ", sol[0])
print("Error:\n ", err)
```

This gives us the following result and error:

$$\int_1^{\pi} f(x)\,dx \approx 3.128827562$$
$$\epsilon \approx 0.00201599277$$

2.2 Romberg with Simpson 1/3

We can also calculate an integral using Romberg combined with Simpson 1/3. To get the value of the Integral we first calculate the value of the integral numerically via Simpson 1/3 with two different step sizes (h and kh) . The true value of the Integral would be in this case:

$$A = I_1 + ch^4 \quad or$$
$$A = I_2 + c(kh)^4$$

If we now solve this system of equation with $k = \frac{1}{2}$ and an error term $O(h^4)$ of the Simpson method, we get:

$$A = I_2 + \frac{I_2 - I_1}{2^4 - 1}$$

The result is the following code:

```
import numpy as np

# Define function
def func(x):
    return 2**x * np.sin(x) / x



# Simpson 1/3 Rule
def Simpson(h, values):
    sum = values[0] + values[-1]

    # Apply Simpson's rule for odd and even indices
    for i in range(1, len(values) - 1): # Multiply values
                                          with to if they ar odd,
                                          else with four and
                                          calculate sum
        if i % 2 == 0:
            sum += 4 * values[i]
        else:
            sum += 2 * values[i]

    result = h / 3 * sum  # Calculate result
    return result
```

```python
# Romberg Integration
def Romberg(I_1, I_2, n=4): # Error h^4 because we use
                              Simpson 1/3
    return I_2 + (I_2 - I_1) / (2**n - 1)


indices = [300,600, 400, 800]
sol = {}

for i, index in enumerate(indices): # Calculate the result
                              for different steps
    x = np.linspace(1, np.pi, index)
    f_x = [func(x_i) for x_i in x]
    h = (x[-1] - x[0]) / (len(x) - 1)
    sol[i] = Simpson(h,f_x)



# Apply Romberg method for higher accuracy
result_1 = Romberg(sol[0], sol[1])
result_2 = Romberg(sol[0], sol[1])

err = np.abs(result_2-result_1)

print(f"Final result:",result_1)
print(f"Error:",err) # Must be zero, just for own knowledge
```

This gives us the following result and error:

$$\int_1^{\pi} f(x)\, dx \approx 3.130977954804248$$

$$\epsilon = 0.0$$

2.3 Gauss-Legendre with 4 points

For the Gauss-Legendre method we need the nodes and weights to calculate the integral. We will come back on this later. How does the Gauss-Legendre method works?

With the Gauss-Legendre method we can calculate the following integral:

$$\int_{-1}^{1} f(x)\, dx$$

To evaluate the integral, we combine weight constants (A_i) with the real value of the used points:

$$\int_{-1}^{1} f(x)\, dx \approx \sum_{i=1}^{n} A_i f(x_i)$$

Where A_i are given weight constants, x_i are the nodes and n is the number of used points. The weight factor can be calculated the following:

$$A_i = \frac{2(1-x^2)}{n^2[L_{n-1}(x_i)]^2}$$

$L_{n-1}(x_i)$ is the the $n-1$ Legendre polynom.

If we don't have an integral with bounds -1 and 1 we have to transform our x_i via the following equation:

$$x_{new} = \frac{b-a}{2} \cdot x_i + \frac{b+a}{2}$$

b and a are our bounds.

In the following we try to implement a Gauss-Legendre integration with four points. In this case our A_i and x_i are:

$$A_i = [0.3478548451, 0.6521451549, 0.3478548451, 0.6521451549]$$
$$x_i = [0.8611363116, 0.3394810436, -0.8611363116, -0.3394810436]$$

As a result we get the following code:

```
import numpy as np

# Define Function
def func(x):
    return 2**x*np.sin(x)/x

# Transformation for our used Intervall
def transform(x, b, a):
    return 0.5*(b - a) * x + 0.5*(b + a)

# Gauss-Legendre weights and nodes
A_i = [0.3478548451, 0.6521451549, 0.3478548451, 0.6521451549
    ]
x_i = [0.8611363116, 0.3394810436, -0.8611363116, -0.
    3394810436]

# Gau -Legendre function
def Gauss_legendre(b, a, x_v, A):
    sum = 0
    x_transformed = [transform(r, b, a) for r in x_v]  #
        Transform nodes for use

    for i in range(len(x_transformed)):
        sum += A[i] * func(x_transformed[i])  # Calculate the
            values with the
            weigths and nodes

    return 0.5 * (b - a) * sum  # Multiply by the scaling
        factor
```

```
result = Gauss_legendre(np.pi, 1, x_i, A_i) #Calculate the
                                   integral

new_int = (np.pi+1) / 2
result_2 = Gauss_legendre(new_int, 1, x_i, A_i) +
                                   Gauss_legendre(np.pi, new_int,
                                    x_i, A_i) #Calculate the
                                   integral with finer steps for
                                   error calculation

err = abs(result_2 - result) #Calculate the error

print("Solution of the integral:\n ", result)
print("Error:\n ", err)
```

This gives us the following result and error:

$$\int_1^{\pi} f(x)\,dx \approx 3.133309412$$

$$\epsilon \approx 0.00034966$$

3 Problem III

In the following section we will calculate the integrals via Simpson 1/3, which is:

$$\int_a^b f(x)\,dx = \frac{h}{3} \cdot (f_0 + 4f_1 + ... + f_n)$$

If we want to define multiply integrals we have to use this Simpson method for both integrals separately. As a result we get:

$$\int_a^b \left(\int_c^d f(x)\,dx \right) dy = \frac{h_1 h_2}{6} \cdot ((f_{11} + 2f_{21} + f_{31} + f_{41}) + 4 \cdot (f_{12} + 2f_{22} + f_{32} + f_{42}) + ...)$$

Let us start with the first integral:

$$\int_0^1 \left(\int_0^2 xy^2\,dx \right) dy$$

In the following we will use the bibliography sympy to calculate multiply integrals. The could is given by:

```
import numpy as np
import sympy as sp

# Define variables
y = sp.Symbol('y')
x = sp.Symbol('x')

# Define function
def func(x):
    return x * y**2

# Define Simpson 1/3
def Simpson(h, values):
    sum_result = values[0] + values[-1]  # Get the sum of the
                                            values of the integral
                                           limits
```

```
    for i in range(1, len(values)):  # Sum all other values
                                       based on the number of
                                       steps
        if i % 2 == 0:  # Multiply all odd index values with
                                           2 and the others with
                                           4 and calculate the
                                           sum
            sum_result += 4 * values[i]
        else:
            sum_result += 2 * values[i]

    result = h / 3 * sum_result  # Get the result
    return result


sol = {}


indices = [400, 600]


for i, index in enumerate(indices):
    h_1 = 2/index #Define the steps for both integrals
    h_2 = 1/index

    x_vals = [j * h_1 for j in range(index+ 1)]

    # Calculate values
    f_x = [func(xi) for xi in x_vals]

    # Calculate integral 1
    f_y = Simpson(h_1, f_x)
    x_vals = [j * h_2 for j in range(index+ 1)]
    result_func = sp.lambdify(y, f_y, 'sympy')

    # Calculate values
    f = [result_func(xi) for xi in x_vals]
    sol[i] = Simpson(h_2, f)
```

```
err = np.abs(sol[1]-sol[0])


for i in sol:
    print(f"Ergebnis f r {indices[i]} Intervalle: {sp.
                                  simplify(sol[i])}")
print(f"Error:", err)
```

As a result we get:

$$\int_0^1 \left(\int_0^2 xy^2 \, dx \right) dy \approx 0.672235191358026$$

$$\epsilon \approx 0.00279399$$

Let's continue with the next integral:

$$\int_0^1 \left(\int_{2y}^2 xy^2 \, dx \right) dy$$

We also use the code from above. But now we have to put variables in the bounds. So in this case we get the following result:

```
import numpy as np
import sympy as sp

# Define variables
y = sp.Symbol('y')
x = sp.Symbol('x')

# Define function
def func(x):
    return x * y**2

# Define Simpson 1/3
def Simpson(h, values):
    sum_result = values[0] + values[-1]  # Get the sum of the
                                          values of the integral
                                         limits

    for i in range(1, len(values)):  # Sum all other values
                                     based on the number of
                                     steps
        if i % 2 == 0:  # Multiply all odd index values with
                             2 and the others with
                             4 and calculate the
                             sum
            sum_result += 4 * values[i]
        else:
            sum_result += 2 * values[i]

    result = h / 3 * sum_result  # Get the result
    return result
```

```
sol = {}


indices = [400, 600]


for i, index in enumerate(indices):
    h_1 = (2-2*y)/index #Define the steps for both integrals
    h_2 = 1/index

    x_vals = [2*y + j * h_1 for j in range(index+ 1)]

    # Calculate values
    f_x = [func(xi) for xi in x_vals]

    # Calculate integral 1
    f_y = Simpson(h_1, f_x)
    x_vals = [j * h_2 for j in range(index+ 1)]
    result_func = sp.lambdify(y, f_y, 'sympy')

    # Calculate values
    f = [result_func(xi) for xi in x_vals]
    sol[i] = Simpson(h_2, f)

err = np.abs(sol[1]-sol[0])


for i in sol:
    print(f"Ergebnis f r {indices[i]} Intervalle: {sp.
                                simplify(sol[i])}")
print(f"Error:", err)
```

As a result we get:

$$\int_0^1 \left(\int_{2y}^2 xy^2 \, dx \right) dy \approx 0.267109257204729$$

$$\epsilon \approx 0.0002199025$$

Which is approximately the given value $\frac{4}{15}$.

Now let's move on to the last integral:

$$\int_0^2 \left(\int_0^{\frac{x}{2}} xy^2 \, dy \right) \, dx$$

We will also solve the integral step by step like before. The result is:

$$\int_0^2 \left(\int_0^{\frac{x}{2}} xy^2 \, dy \right) \, dx \approx 0.270236561733186$$
$$\epsilon \approx 0.001795723$$