Computational Physics (SS 2020)
Assignment 1

Maximilian Kruse
332357

Sankarasubramanian Ragunathan
389851

15th April 2020

Question 1) Derivative Formula

We are tasked with calculating a 5-point stencil for the second derivative of a function $f(x)$ and also show that the leading order error term is $\mathcal{O}\left(h^4\right)$

Consider the *Taylor expansion* of the function $f(x)$ about the point a given as:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!} (x-a)^n \tag{1}$$

Now consider a uniform grid of points with the grid spacing h. Applying *Taylor expansion* to the function $f(x)$ at the points $x-2h, x-h, x+h$ and $x+2h$, we get:

$$f(x+2h) = f(x) + 2hf^{(1)}(x) + 2h^2 f^{(2)}(x) + \frac{4}{3}h^3 f^{(3)}(x) + \frac{2}{3}h^4 f^{(4)}(x) + \frac{4}{15}h^5 f^{(5)}(x) + \mathcal{O}\left(h^6\right) \tag{2a}$$

$$f(x+h) = f(x) + hf^{(1)}(x) + \frac{1}{2}h^2 f^{(2)}(x) + \frac{1}{6}h^3 f^{(3)}(x) + \frac{1}{24}h^4 f^{(4)}(x) + \frac{1}{120}h^5 f^{(5)}(x) + \mathcal{O}\left(h^6\right) \tag{2b}$$

$$f(x-h) = f(x) - hf^{(1)}(x) + \frac{1}{2}h^2 f^{(2)}(x) - \frac{1}{6}h^3 f^{(3)}(x) + \frac{1}{24}h^4 f^{(4)}(x) - \frac{1}{120}h^5 f^{(5)}(x) + \mathcal{O}\left(h^6\right) \tag{2c}$$

$$f(x-2h) = f(x) - 2hf^{(1)}(x) + 2h^2 f^{(2)}(x) - \frac{4}{3}h^3 f^{(3)}(x) + \frac{2}{3}h^4 f^{(4)}(x) - \frac{4}{15}h^5 f^{(5)}(x) + \mathcal{O}\left(h^6\right) \tag{2d}$$

We have to eliminate derivative terms other than $f^{(2)}(x)$ in order to obtain the required 5-point stencil. Adding (2a) to (2d) and similarly adding (2b) to (2c), we get

$$f(x+2h) + f(x-2h) = 2f(x) + 4h^2 f^{(2)}(x) + \frac{4}{3}h^4 f^{(4)}(x) + \mathcal{O}\left(h^6\right) \tag{3a}$$

$$f(x+h) + f(x-h) = 2f(x) + h^2 f^{(2)}(x) + \frac{1}{12}h^4 f^{(4)}(x) + \mathcal{O}\left(h^6\right) \tag{3b}$$

Now by multiplying (3b) by 16 and then subtracting (3a) from that, we can remove the 4th-order derivative of the function $f(x)$ leading to the stencil:

$$f^{(2)}(x) = \frac{-\frac{1}{12}f(x+2h) + \frac{4}{3}f(x+h) - \frac{5}{2}f(x) + \frac{4}{3}f(x-h) - \frac{1}{12}f(x-2h)}{h^2} + \mathcal{O}\left(h^4\right) \tag{4}$$

Question 2) Simpson Rule

Approximation of function $f(x)$ using quadratic interpolation function, we get:

$$f(x) \approx \frac{x-x_{k-1}}{x_k-x_{k-1}}\frac{x-x_{k+1}}{x_k-x_{k+1}}f(x_k) + \frac{x-x_k}{x_{k-1}-x_k}\frac{x-x_{k+1}}{x_{k-1}-x_{k+1}}f(x_{k-1}) + \frac{x-x_k}{x_{k+1}-x_k}\frac{x-x_{k-1}}{x_{k+1}-x_{k-1}}f(x_{k+1}) \tag{5}$$

Integrating the approximated quadratic interpolation function from x_{k-1} to x_{k+1}, we get the Simpson Rule for integration given as:

$$\int_{x_{k-1}}^{x_{k+1}} f(x)dx \approx \frac{h}{3}\left[f(x_{k-1}) + 4f(x_k) + f(x_{k+1})\right] \tag{6}$$

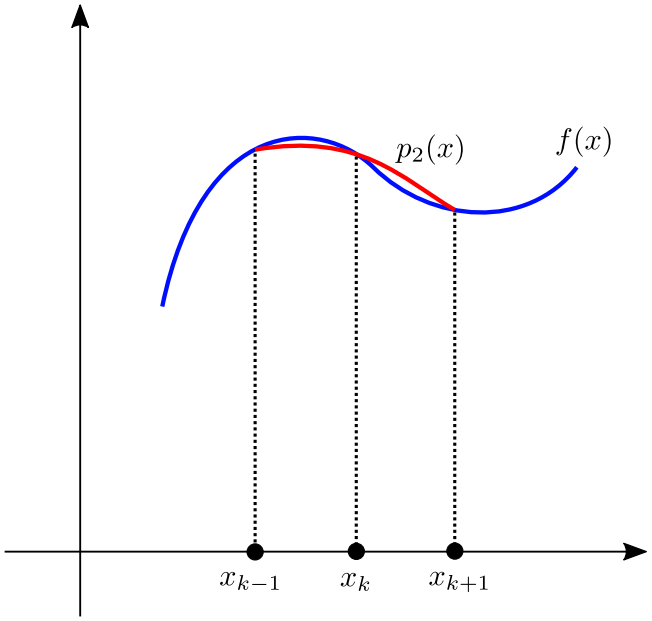

Figure 1: Approximation of the function $f(x)$ using a quadratic *Lagrange* polynomial

The integral of the function $f(x) = \sin(x)$ is calculated over the interval $[0, \pi/2]$ using the Simpson's Rule. The error in the measurement with respect to the analytical solution is recorded along with the theoretical discretization error given by:

$$S - S_S(h) = \frac{1}{180}(b-a)h^4 f^{(4)}(\xi), \quad \xi \in [a,b]$$

for varying grid spacing, h and is as plotted in fig. 2. It can be observed from fig. 2 that for grid spacing around the neighbourhood of $h = 10^{-4}$, we see that the error does not reduce any further and fluctuates around 10^{-15}. This is because the error is close to the ϵ precision of the computer and there will be inherent truncation errors involved on reaching values close to machine precision.

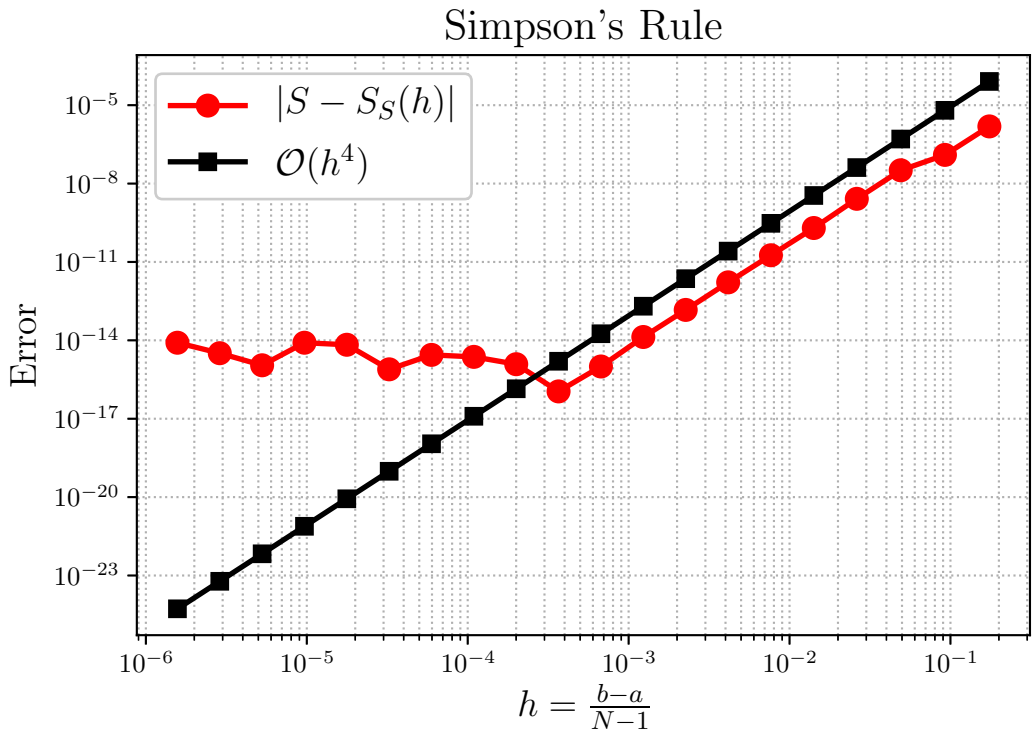

Figure 2: Deviation from the exact solution and the theoretical discretization error plotted against the grid spacing, h

Question 3) Romberg integration

The first part of this task is to show the equivalence of the $R_{n,1}$ column of the Neville scheme and the integration relying on the Simpson rule. It should be noted that since the Romberg integration method utilizes a series of grids with $N_i = 2^i + 1$ grid points, we only need to consider the Simpson integral for the case of an odd number of grid points. The latter is given as follows:

$$\int_{x_{k-1}}^{x_{k+1}} f(x)dx \approx \frac{h}{3}\left[f(x_{k-1}) + 4f(x_k) + f(x_{k+1})\right] \tag{7}$$

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2(f_{x_2}) + 4f(x_3) + \ldots + 2f(x_{N-3}) + 4f(x_{N-2}) + f(x_{N-1})\right] \tag{8}$$

The Romberg integration method and its realization through the Neville scheme rely on the trapezoidal rule:

$$\int_{x_{k-1}}^{x_{k+1}} f(x)dx \approx \frac{h}{2}\left[f(x_k) + f(x_{k+1})\right] \tag{9}$$

$$S_T(h) = \frac{h}{2}\sum_{k=0}^{N-2}\left[f(x_k) + f(x_{k+1})\right] = h\left[\frac{1}{2}f(x_0) + f(x_1) + \ldots + f(x_{N-2}) + \frac{1}{2}f(x_{N-1})\right] \tag{10}$$

Finer approximations can then be determined by a simple recursion relation:

$$R_{n,0} = S_T(h_n) \tag{11}$$

$$R_{n,j} = R_{n,j-1} + \frac{1}{4^j - 1}\left[R_{n,j-1} - R_{n-1,j-1}\right] \tag{12}$$

When considering these terms for a given $h_n = h$, $h_{n-1} = 2h$ and $N = 2^n + 1$, we can expand to:

$$\begin{aligned}
\int_a^b f(x)dx &\approx R_{n,1} \\
&= \frac{4}{3}S_T(h) - \frac{1}{3}S_T(2h) \\
&= \frac{4}{3}h\left[\frac{1}{2}f(a) + f(a+h) + ... + f(b-h) + \frac{1}{2}f(b)\right] \\
&\quad - \frac{1}{3}2h\left[\frac{1}{2}f(a) + f(a+2h) + f(a+4h) + ... + f(b-2h) + \frac{1}{2}f(b)\right] \\
&= \frac{h}{3}[f(a)(2-1) + f(a+h)\cdot 4 + f(a+2h)(4-2) + ... \\
&\quad + f(b-2h)(4-2) + f(b-h)\cdot 4 + f(b)(2-1)] \\
&= \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + ... + 2f(x_{N-3}) + 4f(x_{N-2}) + f(x_{N-1})\right]
\end{aligned} \tag{13}$$

This exactly corresponds to a computation according to the Simpson rule with a level-n grid consisting of $2^n + 1$ nodes.

The second part of this task consists of an implementation of the Romberg integration method. The corresponding source code can be found below. Additionally, the integral of three different test functions is computed:

$$(i)\int_0^1 e^x dx \qquad (ii)\int_0^{2\pi} \sin^4(8x)dx \qquad (iii)\int_0^1 \sqrt{x}dx$$

The resulting values are compared to the corresponding analytical solutions and the absolute deviation is plotted over the number of levels employed in the scheme (c.f. fig. 3). It can be concluded that the integral of the exponential function converges quite rapidly to the analytical solution. An accuracy in the order of the machine precision is achieved after only four levels. Similar behaviour can be observed for the sine function, which converges after eight levels. However, the initial error reduction is much slower. This behaviour might be due to the fact that a minimum number of polynomial degree is necessary to adequately resemble trigonometric functions (as known from Fourier transformation). The square root function integral converges most slowly, achieving an accuracy of 10^{-10} after twenty levels. Nevertheless, this is a favorable result and one should also consider the fact that the scale in the results graph is logarithmic.

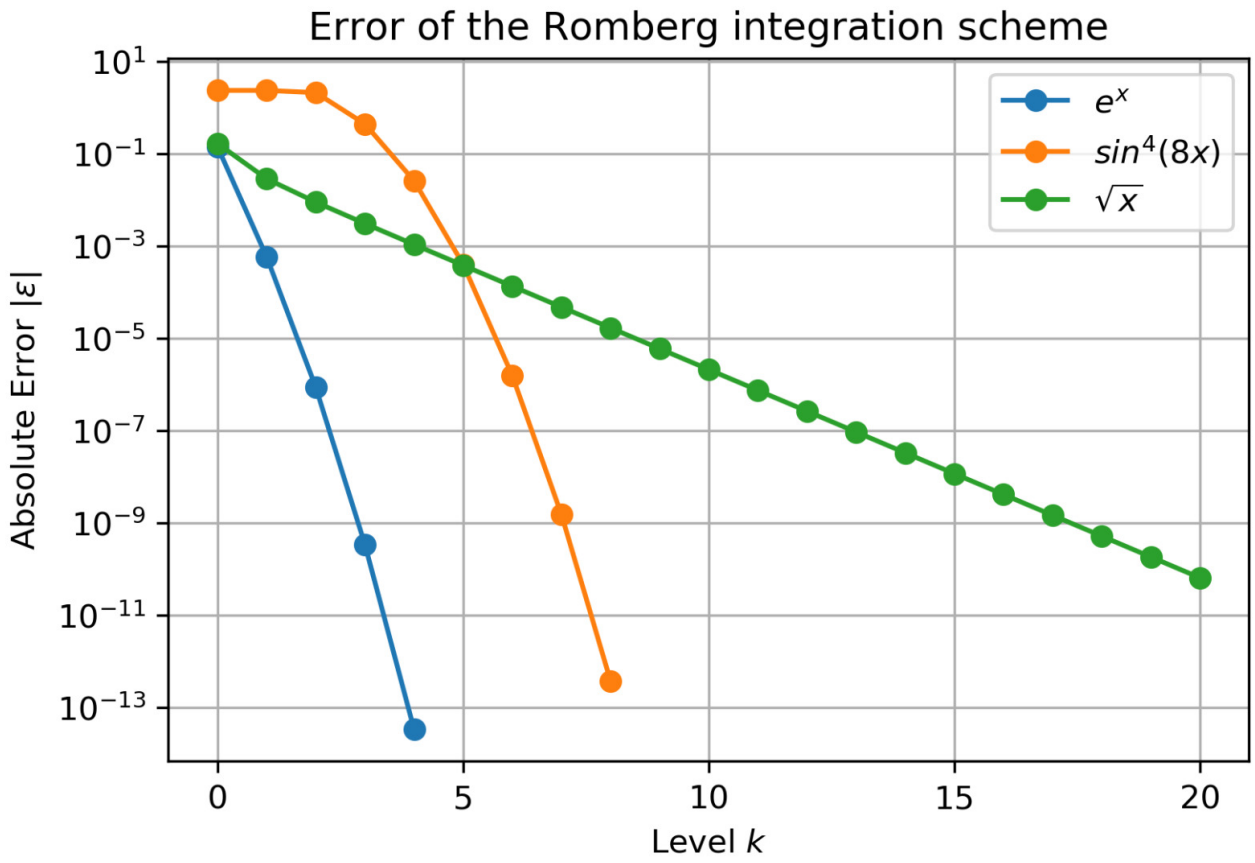

Figure 3: Romberg integration error obtained over varying levels of refinement k

Appendix

Simpson's Rule

```
#!/usr/bin/env python3

"""

@file simpsons.py
@brief Simpson's Rule for integration of a function

This file contains a basic 1D setup for the Simpson's Rule for integration of a function. This
method relies on approximation of a given function in terms of Lagrange polynomials and
then calculating the area under the approximated function.

================================================================================

@author Sankarasubramanian Ragunathan
@date 14.04.2020

"""

# Importing packages required for numerical calculations and plotting
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import rc
rc('text', usetex=True)
rc('font', family='serif')
```

```
# Function definition to perform Simpson's Integration
def simpsons(a,b,N) :

    # Grid spacing for the Uniform grid of points
    h = (b-a)/(N-1)
    # Area obtained from integrating f(x) using Simpson's Rule
    area = 0

    # Checking if the number of slices are even or odd (necessary for the implementation of Simpson's Rule)
    if (N%2 == 1) :
        # Looping through all the points on the grid
        for k in range(1,N-1,2) :
                # Point x_k to evaluate the function value f(x_k)
            xk = a+k*h
            # Calculating the area under the sin(x) curve using Simpson's Rule
            area += (h/3)*(np.sin(xk-h) + \
                           4*np.sin(xk) +  \
                           np.sin(xk+h))
    else :
        # Looping through all the points on the grid
        for k in range(1,N-2,2) :
                # Point x_k to evaluate the function value f(x_k)
            xk = a+k*h
            # Calculating the area under the sin(x) curve using Simpson's Rule
            area += (h/3)*(np.sin(xk-h) + \
                           4*np.sin(xk) + \
                           np.sin(xk+h))
        # Special treatment of the boundary term for even number of points in the grid
        xk = a + (N-2)*h
        # Calculating the area under the sin(x) curve using Simpson's Rule
        area += (h/12)*(-np.sin(xk-h) + \
                        8*np.sin(xk) + \
                        5*np.sin(xk+h))
    return area

# Function defined to calculate the Analytical error and Theoretical error
def errorFunc(a,b,N,area,errOpt) :
        # Grid spacing for the uniform grid
    h = (b-a)/(N-1)
    # Actual solution obtained from integrating the function
    actualSoln = np.cos(a) - np.cos(b)
    if (errOpt == 0) :
            # Analytical error calculation
        error = np.fabs(actualSoln-area)
    elif (errOpt == 1) :
            # Theoretical discretization error
        constFac = 10
        error = (1/180)*(b-a)*(h**4)*constFac
    return (h,error)

""" Main Program """

# No. of terms to use in the generation of array of log-spaced grid points
nPoints = 20
# Array containing the integral calculated using Simpson's Rule for different no. of grid points
area = np.zeros(nPoints)
# log-spaced values for the no. of gridpoints in the uniform mesh
N = np.logspace(1,6,num=nPoints,endpoint=True,dtype=np.int64)
for index in range(0,nPoints) :
    area[index] = simpsons(0, np.pi/2, N[index])

(h,analyticalerror) = errorFunc(0,np.pi/2,N,area,0)
(h,theoreticalerror) = errorFunc(0,np.pi/2,N,area,1)

# Plotting the error vs. grid spacing (log-log plot)

plt.loglog(h,analyticalerror,label=r"$|S-S_{S}(h)|$",linewidth=2,color="r",marker="o",markersize=8)
plt.loglog(h,theoreticalerror,label=r"$\mathcal{O}(h^4)$",linewidth=2,color="k",marker="s",markersize=6)
plt.grid(True,which="both",axis="both",linestyle=":")
plt.title(r"Simpson's Rule",fontsize=16)
plt.xlabel(r'$h = \frac{b-a}{N-1}$',fontsize=14)
plt.ylabel(r'Error',fontsize=14)
```

```
plt.legend(loc=2,fontsize=14)
plt.savefig('./simpsons.eps',format='eps')
```

Romberg Scheme

```
"""

@file Romberg_integration.py
@brief Romberg integrator incorporating Neville scheme

This file contains a basic 1D setup for the Romberg integration scheme. This
method relies on a Richardson extrapolation of integrals calculated with the
trapezoidal rule on a series of successively refined grids.
The practical implementation relies on the Neville scheme.  Three test
functions are provided, of which the error is visualized over the number of
levels in the Romberg integration procedure.

================================================================================

@author Maximilian Kruse
@date 14.04.2020

"""

#============================== Import Libraries ==============================
import numpy as np
import matplotlib.pyplot as plt


#========================== Integration Functions ===========================

#--------------------------------------------------------------------------------
"""
Calculate Trapezoidal integral

@param[in]: function handle, number of grid points, integration domain bounds
@param[out]: Value of the definite Trapezoidal integral
"""
def trapezoidal_integral(funcHandle, numPoints, lb, ub):
    # Calculate function values on grid
    discrSpacing = (ub-lb)/(numPoints-1)
    discrPoints = [lb+(ub-lb)*i/(numPoints-1) for i in range(numPoints)]
    funcVals = funcHandle(discrPoints)

    # Accumulate according to trapezoidal rule
    integralValue = 0;
    for i  in range(numPoints-1):
        integralValue += funcVals[i] + funcVals[i+1]
    integralValue *= discrSpacing/2;

    return integralValue

#--------------------------------------------------------------------------------
"""
Calculate Romberg integral

@param[in]: function handle, number levels, integration domain bounds
@param[out]: Value of the definite Romberg integral
Note: This function calls the trapezoidal_integral function for intrinsic
      calculations
"""
def romberg_integral(funcHandle, levels, lb, ub):
    # Define initial column of neville scheme from trapezoidal rule
    schemeArray = [trapezoidal_integral(funcHandle, pow(2,i)+1, lb, ub)
                   for i in range(levels+1)]

    # Successively compute new columns of the Neville scheme. This routine
    # overwrites "non-diagonal" elements for reduced storage requirements
    for level in range(1, levels+1):
        for i in range (levels,level-1,-1):
            schemeArray[i] = schemeArray[i] + 1/(pow(4,level)-1)*\
```

```
                            (schemeArray[i]-schemeArray[i-1])

    return np.asarray(schemeArray)


#============================== Test Functions ================================

# Function handles
expFunc = lambda x:np.exp(x)
sineFunc = lambda x:np.power(np.sin(8*x),4)
sqrtFunc = lambda x:np.sqrt(x)

# Analytical solutions
solExp = np.exp(1)-1
solSine = 3./4.*np.pi;
solSqrt = 2./3.


#============================= Main Computation ===============================

# Set levels for the test integrals
levelExp = 4
levelSine = 8
levelSqrt = 20

# Compute Romberg integrals for the test functions
xValsExp = np.arange(0, levelExp+1)
xValsSine = np.arange(0, levelSine+1)
xValsSqrt = np.arange(0, levelSqrt+1)
yValsExp = np.absolute(romberg_integral(expFunc, levelExp, 0, 1)-solExp)
yValsSine = np.absolute(romberg_integral(sineFunc, levelSine, 0, 2*np.pi)-solSine)
yValsSqrt = np.absolute(romberg_integral(sqrtFunc, levelSqrt, 0, 1)-solSqrt)

# Plot results
plt.title('Error of the Romberg integration scheme')
plt.xlabel(r'Level $k$')
plt.ylabel(r'Absolute Error $|\epsilon|$')
plt.xticks(np.arange(0, 21, step=5))
plt.grid('on')
plt.semilogy(xValsExp,yValsExp,label=r'$e^x$',marker='o',color='tab:blue',)
plt.semilogy(xValsSine,yValsSine,label=r'$sin^4(8x)$',marker='o',color='tab:orange')
plt.semilogy(xValsSqrt,yValsSqrt,label=r'$\sqrt{x}$',marker='o',color='tab:green')
plt.legend()
plt.show()
```