Computational Physics Exercise 1

Berat Ertural 406055

16.04.20

1 Derivative Formula

Let h be constant and let $f(x-2h), f(x-h), f(x+h), f(x+2h)$ contain the Taylor expansion up to order 4 with the error term in $O(h^5)$. We first construct the following;

$$f(x+h) + f(x-h) = 2f(x) + h^2 f^{''}(x) + \frac{h^4}{12} f^{''''}(x) + O(h^6) \qquad (1)$$

Obtaining an error bound in $O(h^6)$ since derivatives of uneven order cancel out. Similarily we get;

$$f(x+2h) + f(x-2h) = 2f(x) + 4h^2 f^{''}(x) + \frac{16h^4}{12} f^{''''}(x) + O(h^6) \qquad (2)$$

Multplying equation (1) by 16 and subtracting equation (2) we get the desired result;

$$-f(x+2h)+16f(x+h)+16f(x-h)-f(x-2h) = 30f(x)+12h^2 f^{''}(x)+O(h^6)$$
$$\Rightarrow f^{''}(x) = \frac{1}{h^2}[-\frac{1}{12}f(x+2h)+\frac{4}{3}f(x+h)-\frac{5}{2}f(x)+\frac{4}{3}f(x-h)-\frac{1}{12}f(x-2h)]+O(h^4) \qquad (3)$$

Which is a **central 5-point formula** for the second derivative of **4-th order accuracy**.

2 Simpson Rule

2.1 Simpson integration in Python

The following function evaluates the integral of a given function $f(x)$ on a finite interval $x \in [a, b]$ using a set of n equidistant sample points.

```
import math

def simpson(f, a, b, n):
    """Approximate the Integral of f in the interval
    between a and b using n equidistant sample points using
    Simpsons rule. The number of sample points can be even or odd."""
    # Enforce oddity
    n = max (n, 2)
    h = (b-a)/(n-1)
    odd_n = n -1 + n%2 # Enforce oddity
    S = f(a) + f(a+(odd_n-1)*h)
    # Integrate over alternating coefficients
    for i in range(1, n-1, 2):
S += 4*f(a + i * h)
    for i in range(2, n-2, 2):
S += 2*f(a + i * h)
    S *= h/3
    # Case n even
    if (n%2 == 0):
S += (h/12)* (5*f(b) + 8*f(b-h) - f(b-2*h))
    return S

# Lets output a test value
simpson(math.sin, 0, math.pi/2, 8)
```

```
# Out[49]:
: 0.9999906618321359
```

2.2 Example

2.2.1 Integrating $sin(x)$ with Simpsons rule

Now let us consider $sin(x)$ over the interval $[0, \frac{\pi}{2}]$. For the integral we get $S = \int_0^{\frac{\pi}{2}} sin(x)dx = 1$. So the deviation from the analytical result for our example is given by $\Delta S = 1 - S_s(h)$, where $S_s(h)$ is our Simpson integral of step-size h. The theoretical discretization error bound for the Simpson rule is given by;

$$\Delta S = S - S_s(h) \leq \frac{\gamma}{180}(b-a)h^4 \tag{4}$$

where $\gamma \geq |f''''(x)|$ is usually the smallest possible value greater than or equal to the$(n+1)$-th differential of $f(x)$ between the upper and lower interval bound. In our case $\gamma = 1$.

2.2.2 Plotting the values

The following code demonstrates the Simpson integration function with the discussed values.

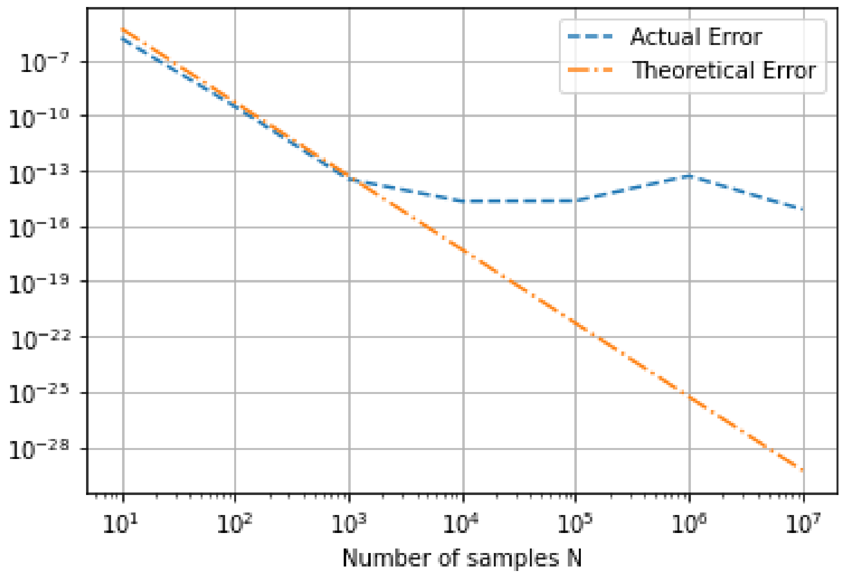
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

# Theoretical error bound
def deltaSimpson(a, b, n, k):
    return k*((b-a)**5)/(180*(n**4))

# Collect Simpson integrals and errors for different n in [0, pi/2]
n = []
S = []
ErrBound = []  # theoretical error
# Interation Number, Number of Sample Points, S_T(h_n), Delta S
print("{:10}| {:10}| {:20}| {:20}".format("Iteration", "N", "S_s", "Delta S"))
for k in range(1, 8):
    n.append(10**k)
    S.append(simpson(math.sin, 0, math.pi/2, n[k-1]))
    ErrBound.append(deltaSimpson(0, math.pi/2, n[k-1], 1))
    print("{:10}| {:10}| {:20}| {:20}".format(k, n[k-1], S[k-1], ErrBound[k-1]))
```

```
Iteration | N         | S_s                 | Delta S
         1|         10|   0.9999984600259557| 5.312841749744469e-06
         2|        100|   1.0000000003102127| 5.312841749744469e-10
         3|       1000|   1.0000000000000344| 5.312841749744469e-14
         4|      10000|   0.9999999999999978| 5.312841749744469e-18
         5|     100000|   0.9999999999999977| 5.312841749744469e-22
         6|    1000000|   1.0000000000000517| 5.312841749744469e-26
         7|   10000000|   0.9999999999999992| 5.312841749744469e-30

# Actual error is 1 - S_T(h)
```

```
Deviation = [] # actual error
for i in S:
    print("1 - {} = {}".format(i, abs(1-i)))
    Deviation.append(abs(1-i))
```

```
1 - 0.9999984600259557 = 1.53997404428452e-06
1 - 1.0000000003102127 = 3.102127443810332e-10
1 - 1.0000000000000344 = 3.441691376337985e-14
1 - 0.9999999999999978 = 2.220446049250313e-15
1 - 0.9999999999999977 = 2.3314683517128287e-15
1 - 1.0000000000000517 = 5.1736392947532295e-14
1 - 0.9999999999999992 = 7.771561172376096e-16
```

```
line1, = plt.plot(n, Deviation, '--', label='Actual Error')
line2, = plt.plot(n, ErrBound, '-.', label='Theoretical Error')
# Log-Log plot of actual error versus theoretical error
plt.grid()
plt.xscale("log")
plt.yscale("log")
plt.xlabel("Number of samples N")
plt.legend(handles=[line1, line2], loc='best')
plt.show()
```



2.2.3 The discretization error

Above is the log-log plot of the actual deviation from the analytical function and the theoretical error bound of the Simpson rule. The actual error (blue) shows artifacts around 10^4 sample points due to precission errors in h accumulating in the integration function. This is due to the finite internal represention of floating point units in the underlying architecture and the resulting round-off errors for very small fractions.

3 Romberg Integration

3.1 Neville scheme

Let $R_{n,0}$ be $S_T(h_n)$ as the first column (column number 0) of the Neville scheme, with $S_T(h_n)$ being the n-th order integral using the trapezoidal rule. The recursion relation for column j and row n for the Neville scheme is given by;

$$R_{n,j} = R_{n,j-1} + \frac{1}{4^j - 1}[R_{n,j-1} - R_{n-1,j-1}]. \tag{5}$$

From this follows;

$$R_{n,1} = S_T(h_n) + \frac{1}{3}[S_T(h_n) - S_T(h_{n-1})] \tag{6}$$

3.2 Correlation of $R_{n,1}$ to Simpsons rule

To demonstrate the connection of the Neville scheme to the Simpson rule, let us first consider the case for $R_{1,1}$;

$$
\begin{aligned}
R_{1,1} &= R_{1,0} + \frac{1}{3}(R_{1,0} - R_{0,0}) \\
&= S_T(h_1) + \frac{1}{3}(S_T(h_1) - S_T(h_0)) \\
&= \frac{1}{2}S_T(h_0) + h_1 f(a+h_1) - \frac{1}{6}S_T(h_0) + \frac{1}{3}h_1 f(a+h_1) \\
&= \frac{1}{3}S_T(h_0) + \frac{4}{3}h_1 f(a+h_1) \\
&= \frac{1}{3}(\frac{1}{2}(b-a)(f(a)+f(b))) + \frac{4}{3}h_1 f(a+h_1)
\end{aligned}
\tag{7}
$$

Here we can use the fact that $h_{i+1} = \frac{1}{2}h_i$ and $h_0 = b - a$.

$$
\begin{aligned}
R_{1,1} &= \frac{1}{3}(h_1(f(a)+f(b))) + \frac{4}{3}h_1 f(a+h_1) \\
&= \frac{1}{3}h_1(f(a) + 4f(a+h_1) + f(b)
\end{aligned}
\tag{8}
$$

Which is the integral of the parabola approximating f at the endpoints a, $a+h$ and b. This is the Simpson rule in its most basic form.

Now let us expand $R_{n,1}$ for any n;

$$
\begin{aligned}
R_{n,1} &= \frac{1}{3}[S_T(h_{n-1}) + 4h_n \sum_{i=1}^{2^{n-1}} f(a+(2i-1)h_n)] \\
&= \frac{1}{3}[(\frac{1}{2}S_T(h_{n-2}) + 2h_n \sum_{i=1}^{2^{n-2}} f(a+(2i-1)2h_n)) + 4h_n \sum_{i=1}^{2^{n-1}} f(a+(2i-i)h_n)] \\
&= \frac{1}{3}[(\frac{1}{2}(\frac{1}{2}S_T(h_{n-3}) + 4h_n \sum_{i=1}^{2^{n-3}} f(a+(2i-1)4h_n))) + 2h_n \sum_{i=1}^{2^{n-2}} f(a+(2i-1)2h_n)) \\
&\qquad + 4h_n \sum_{i=1}^{2^{n-1}} f(a+(2i-1)h_n)]
\end{aligned}
\tag{9}
$$

After n iterations, we end up with the following;

$$R_{n,1} = \frac{1}{3}h_n[\frac{1}{2^n}S_T(h_0)+2\sum_{j=1}^{n-2}\sum_{i=1}^{2^j} f(a+(2i-1)2^{n-1-j}h_n)+4\sum_{i=1}^{2^{n-1}} f(a+(2i-1)h_n)]$$

$$= \frac{1}{3}h_n[f(a)+f(b)+2\sum_{j=1}^{n-2}\sum_{i=1}^{2^j} f(a+(2i-1)2^{n-1-j}h_n)+4\sum_{i=1}^{2^{n-1}} f(a+(2i-1)h_n)]$$

$$= \frac{1}{3}h_n[f(a) + f(b) + 2\sum_{i=1}^{2^{n-1}} f(a + (2i)h_n) + 4\sum_{i=1}^{2^{n-1}} f(a + (2i-1)h_n)] \quad (10)$$

Therefore we can see, that the second column of the Neville scheme corresponds to the Simpson rule, QED.

3.3 Romberg integration in Python

The following code implements the Romberg integration of the function f in the interval $[a, b]$ using n steps of the Neville scheme.

```
def romberg(f, a, b, n):
    """Romberg integration of f in [a,b] with n levels."""
    h = b-a # Interval size
    Table = np.zeros((n, n))
    Table[0][0] = ((h/2) * (f(a) + f(b))) # Insert S_T(h_0)
    # Trapezoidal rule
    for k in range(1,n):
h /= 2
# Collect new sample points
new_samples = 0
for i in range(1, 2**k, 2):
    new_samples += f(a+i*h)
Table[k][0] = (Table[k-1][0]/2) + h*new_samples

    # Richardson extrapolation
    for c in range(1, n):
pow4 = (4**c) - 1
for r in range(c, n):
    Table[r][c] = Table[r][c-1] + ((Table[r][c-1] - Table[r-1][c-1])/pow4)

    return Table[n-1][n-1]
```

```
# Test value, 10 steps should get us around 20 digits of accurary
romberg(math.sin, 0, math.pi/2, 10)
```

```
# Out[53]:
: 1.0000000000000002
```

3.4 Examples

3.4.1 Integrating various functions with Romberg integration

We will now examine a few examples.

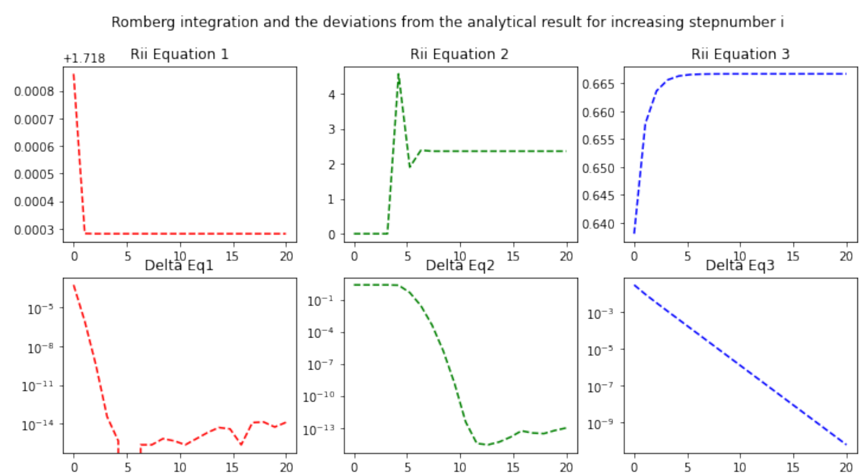
$$\int_0^1 e^x dx \approx 1,718281828 \tag{11}$$

$$\int_0^{2\pi} sin^4(8x)dx = \frac{3}{4}\pi \tag{12}$$

$$\int_0^1 x^{\frac{1}{2}} = \frac{2}{3} \tag{13}$$

```
k = 22
# Tabulate the R_ii for the three test functions for increasing values of i
Rii = np.zeros((3, k-2))

# go get a coffee
for i in range(2, k):
    Rii[0][i-2] = romberg((lambda x: math.exp(x)),      0, 1,         i)
    Rii[1][i-2] = romberg((lambda x: math.sin(8*x)**4), 0, 2*math.pi, i)
    Rii[2][i-2] = romberg((lambda x: math.sqrt(x)),     0, 1,         i)

# calculate deviation from analytical result
Delta = np.zeros((3, k-2))
for i in range(0, k-2):
    Delta[0][i] = abs((math.exp(1)-1) - abs(Rii[0][i]))
    Delta[1][i] = abs(3*math.pi/4 - abs(Rii[1][i]))
    Delta[2][i] = abs((2/3) - abs(Rii[2][i]))

x = np.linspace(0,k-2,k-2)

fig, ((ax11, ax12, ax13), (ax21, ax22, ax23)) = plt.subplots(2, 3)
```

```
fig.set_size_inches((12, 6))
fig.suptitle('Romberg integration and the deviations from the analytical result for inc

ax11.plot(x,Rii[0:1].T,'r--')
ax11.set_title("Rii Equation 1")
ax12.plot(x,Rii[1:2].T,'g--')
ax12.set_title("Rii Equation 2")
ax13.plot(x,Rii[2:3].T,'b--')
ax13.set_title("Rii Equation 3")
ax21.plot(x,Delta[0:1].T,'r--')
ax21.set_title("Delta Eq1")
ax22.plot(x,Delta[1:2].T,'g--')
ax22.set_title("Delta Eq2")
ax23.plot(x,Delta[2:3].T,'b--')
ax23.set_title("Delta Eq3")
plt.setp((ax21, ax22, ax23), yscale="log")
plt.show()
```



3.5 Results

In the top row of the above graph the $R_{i,i}$ of equations (1), (2) and (3) is plotted together with the deviation from the respective analytical result shown in the row below. We can see, that all $R_{i,i}$ converge around $i = 10$. It does seem like we have achieved the best possible accuracy at this point. With the Richardson extrapolation, a small improvement in the accuracy of our computations is gained with increasing stepsize. The underlying architecture however does not support any more accuracy in its floating point representations beyond stepsizes of $i = 10$. Compared to our previous observations on the Simpson rule example, the resulting stepsizes at around $i = 10$ roughly corresponds to the range of n in $[10^3, 10^4]$, where the deviation of the Simpson integral showed artifacts. A similar behaviour can be observed on the bottom row of the graph. The respective deviations show slight irregularities after a certain iteration number.