num0_ex08

June 16, 2021

Übungszettel 8	Einführung in die Numerik
Dozent	Prof. Kanschat
Tutoren	S. Meggendorfer und J. Witte
Abgabe	bis 17.06.21 23.15 Uhr
Studierende	Alexander Baucke, Lennart Walter, Rodulf Braun

0.0.1 Aufgabe 1: Summierte Quadratur

Schreiben Sie eine Funktion, die die Funktion $f(x) = \sin(\pi x)$ über das Intervall $[0, 1]$ mit der iterierten Trapezregel integriert. Geben Sie den approximierten Integralwert, sowie den Fehler für verschiedene Schrittweiten $h = 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, ...$ aus.

0.0.2 Aufgabe 2: Romberg-Quadratur

Wenden Sie die Romberg-Quadratur mit der Schrittfolge $h = 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, ...$ auf die iterierte Trapezregel aus Aufgabe 1 an und beobachten Sie die Konvergenz gegen den exakten Integralwert $\frac{2}{\pi}$ für die verschiedenen Spalten im Tableau des Neville-Algorithmus. Lassen Sie sich auch hierfür wieder den Fehler ausgeben und vergleichen Sie den Aufwand der Romberg-Quadratur mit der iterierten Trapezregel.

0.0.3 Bonus: Konvergenzraten

Die Konvergenzrate p eines numerischen Verfahrens, dessen Fehler $e(h)$ eine Abschätzung $e(h) = ch^p + o(h^p)$ erlaubt, kann experimentell bestimmt werden mit der Formel

$$p \approx \log_2(e(h)) - log_2(e(\tfrac{h}{2})).$$

Berechnen Sie mit dieser Formel die Konvergenzraten im Tableau des Neville-Algorithmus aus Aufgabe 2.

Hinweis: Der Logarithmus zur Basis 2 lässt sich in numpy mit dem Befehl `np.log2()` berechnen.

---

[1]:
```
import math
import numpy as np

l = 0
```

```
r = 1

def f(x):
    return math.sin(math.pi*x)

def F(x):
    return -math.cos(math.pi*x)/math.pi

def trapez_integral(func, l:float, r:float, count:int):
    h = (r-l)/(count)
    ysum = 0
    xvals = np.linspace(l, r, count+1)
    yvals = [func(x) for x in xvals]
    for i in range(0,len(xvals)-1):
        yl = yvals[i]
        yr = yvals[i+1]
        ysum += yl + yr
    return ysum * h * .5

def romberg(func, l:float, r:float, count:int):
    result = []
    h = (r-l)
    result.append([(h/2.0)*(func(l)+func(r))])
    for i in range(1,count+1):
        h = h/2.
        sum = 0
        for k in range(1,2**i ,2):
            sum = sum + func(l+k*h)
        rowi = [0.5*result[i-1][0] + sum*h]
        for j in range(1,i+1):
            rij = rowi[j-1] + (rowi[j-1]-result[i-1][j-1])/(4.**j-1.)
            rowi.append(rij)
        result.append(rowi)
    return result

ideal = F(1) - F(0)
print("Ideal:", ideal)

for n in range(1,20):
    integral = trapez_integral(f, l, r, n)
    print("Steps:", n, "\tValue:", integral, "\tError:", abs(ideal - integral))
print("="*20)
for n in range(1,20):
    integral = romberg(f, l, r, n)[-1][-1]
    print("Steps:", n, "\tValue:", integral, "\tError:", abs(ideal - integral))
```

```
#Aufwand ist merklich höher bei Romberg, da schon bei kleinen Zahlen die␣
 ↪Rechenzeit merklich ansteigt.
```

```
Ideal: 0.6366197723675814
Steps: 1        Value: 6.123233995736766e-17    Error: 0.6366197723675813
Steps: 2        Value: 0.5      Error: 0.13661977236758138
Steps: 3        Value: 0.5773502691896257       Error: 0.05926950317795565
Steps: 4        Value: 0.6035533905932737       Error: 0.03306638177430765
Steps: 5        Value: 0.6155367074350507       Error: 0.02108306493253065
Steps: 6        Value: 0.6220084679281462       Error: 0.014611304439435147
Steps: 7        Value: 0.6258980382192605       Error: 0.010721734148320894
Steps: 8        Value: 0.628417436515731        Error: 0.008202335851850373
Steps: 9        Value: 0.6301424244019679       Error: 0.0064773479656135
Steps: 10       Value: 0.6313751514675043       Error: 0.005244620900077046
Steps: 11       Value: 0.6322866156157704       Error: 0.004333156751811007
Steps: 12       Value: 0.6329795093937625       Error: 0.0036402629738189196
Steps: 13       Value: 0.6335185349614225       Error: 0.0031012374061588632
Steps: 14       Value: 0.6339461053549827       Error: 0.002673667012598635
Steps: 15       Value: 0.634290963614839        Error: 0.0023288087527423285
Steps: 16       Value: 0.6345731492255539       Error: 0.002046623142027526
Steps: 17       Value: 0.6348069798389168       Error: 0.0018127925286646196
Steps: 18       Value: 0.6350029057089636       Error: 0.0016168666586178215
Steps: 19       Value: 0.6351686989209344       Error: 0.0014510734466469666
====================
Steps: 1        Value: 0.6666666666666666       Error: 0.030046894299085247
Steps: 2        Value: 0.6361648221771005       Error: 0.0004549501904809183 4
Steps: 3        Value: 0.6366215389809788       Error: 1.7666133974580944e-06
Steps: 4        Value: 0.6366197706446624       Error: 1.7229190296674801e-09
Steps: 5        Value: 0.6366197723680019       Error: 4.205524817280093e-13
Steps: 6        Value: 0.6366197723675812       Error: 2.220446049250313e-16
Steps: 7        Value: 0.6366197723675813       Error: 1.1102230246251565e-16
Steps: 8        Value: 0.6366197723675812       Error: 2.220446049250313e-16
Steps: 9        Value: 0.6366197723675814       Error: 0.0
Steps: 10       Value: 0.6366197723675812       Error: 2.220446049250313e-16
Steps: 11       Value: 0.6366197723675806       Error: 7.771561172376096e-16
Steps: 12       Value: 0.6366197723675819       Error: 5.551115123125783e-16
Steps: 13       Value: 0.6366197723675815       Error: 1.1102230246251565e-16
Steps: 14       Value: 0.6366197723675823       Error: 8.881784197001252e-16
Steps: 15       Value: 0.6366197723675826       Error: 1.2212453270876722e-15
Steps: 16       Value: 0.6366197723675813       Error: 1.1102230246251565e-16
Steps: 17       Value: 0.6366197723675824       Error: 9.992007221626409e-16
Steps: 18       Value: 0.6366197723675728       Error: 8.548717289613705e-15
Steps: 19       Value: 0.6366197723675862       Error: 4.773959005888173e-15
```

[ ]: