Kiel University
Christian-Albrechts-Universität zu Kiel

Utku Kaya, Msc.
Deparment of Mathematics,
Ludewig-Meyn-Str. 4,
D-24118 Kiel, Germany
kaya@math.uni-kiel.de

Exercise Sheet 2
Introduction to Scientific Programming
with Python

27.02.2020

The aim of this exercise sheet is to provide guidance for implementing several quadrature formulas for numerical integration. We are interested in approximating

$$\int_a^b f(x)dx$$

where $a, b \in \mathbb{R}$. For a given $N \in \mathbb{N}_{>0}$, $h = (b-a)/N$ be the width of subintervals. We call $x_i = a + hi$ the nodal points.

Exercise 1 Open an empty file and name it as quadrature.py. Create a class named Integrator.

As first step prepare an __init__() function that receives arguments f, a, b and N. Assign following expressions to the attributes in the constructor:

(a) An array $[x_0, x_1, \cdots, x_N]$ by using np.linspace
(b) The subinterval size $h = (b-a)/N$.
(c) The function f.

```
import numpy as np

class Integrator():
  """A class to integrate functions numerically.
  Create an object by Integrator(f, a, b, N)
  f = Integrand function
  a = Interval lowerbound
  b = Interval upperbound
  N = Number of subintervals
  """
  def __init__(self, f, a, b, N):
    """ (Integrator, f, a, b, N) -> NoneType
    Create an Integrator with N intervals.
    """
    self.a = a
    self.b = b
    self.N = N
    self.x = 0  # TODO: Part (a)
    self.h = 0  # TODO: Part (b)
    self.f = f  # TODO: Part (c)
```

This file will be extended in Exercises 3 and 4.

Exercise 2 On the Python shell import the quadrature module from Exercise 1 by typing

```
>>> import quadrature
```

```
>>> help(quadrature)
```

Now we can create an object quad as an instance of Integrator class. Type

```
>>> quad = quadrature.Integrator(0, 0, 1, 10)
```

This creates an object, that receives arguments $f = 0,\ a = 0,\ b = 1$, and $N = 10$. Type the following lines in order to check if your implementation from Exercise 1 is correct.

```
>>> print('The␣interval␣({0},{1})␣is␣divided␣into␣{2}␣'
... 'subintervals.␣\nThe␣nodal␣points␣are␣{3}␣'.format(
... quad.a, quad.b, quad.N, quad.x ))
```

If the output is correct continue with Exercise 3. Otherwise, check your implementation from Exercise 1.

Exercise 3 Reopen the quadrature.py file from Exercise 1. Define a new function named midpoint in order to implement the summed midpoint rule. Recall that:

$$M_h(f) = h \sum_{i=1}^{N} f\left(\frac{x_i + x_{i-1}}{2}\right).$$

(a) Create an array $[m_1, m_2, \cdots, m_N]$ where $m_i = \frac{x_i + x_{i-1}}{2},\ i = 1,\ \cdots,\ N$,
(b) Create an array $[y_1, y_2, \cdots, y_N]$ where $y_i = f(m_i),\ i = 1,\ \cdots,\ N$,
(c) Sum all elements of y, multiply with h and save as T.

```
>>> def midpoint(self):
...     """ (Integrator) -> float
...     Numerical integration by midpoint rule
...     """
...     s =   # TODO: Part (a)
...     y =   # TODO: Part (b)
...     T =   # TODO: Part (c)
...     return T
```

Exercise 4 Since the quadrature.py file is modified in Exercise 3, restart your bash and import the quadrature module by typing

```
>>> import quadrature
```

Now create an object, that is integrator for exponential function in the interval $(0, 1)$ using 10 subintervals.

```
>>> quad = quadrature.Integrator(np.exp, 0, 1, 10)
```

Note that, in order to be able to call the exponential function (`np.exp`), we need to `import numpy`. Type

```
>>> import numpy as np
```

Finally, we can compute and print the approximate integral by typing

```
>>> result = quad.midpoint()
>>> print(result)
```

Exercise 5 Implement the trapezodial and Simpson's rules similarly as in Exercise 3. Finally, we want to consider the error behaviours for increasing numbers of subintervals. Create a new file named testquad.py.

(a) Create an array called errmidpoint with 5 zero elements.
(b) Calculate the numerical integral for exponential function over the interval $(0, 1)$ for $N \in \{10, 20, 40, 80, 160\}$ and assign the errors to each array component. For example, for `N = 10`:

```
>>> exactintegral = np.exp(1)-1.
>>> quad = quadrature.Integrator(np.exp, 0, 1, 10)
>>> errmidpoint[0] = fabs(exactintegral - quad.midpoint())
>>> print(result)
```

Place the above lines in a for loop, so that the calculations for different N will be performed sequentially and saved in the array errmidpoint.

(d) Save the errors from Trapezoidal and Simpson's rules in errtrapez, errsimpson analogously to part (c).
(e) Calculate the convergence rates. For example:

```
>>> convratemid = np.log(errmid[:-1] / errmid[1:])/np.log(2)
>>> print(convratemid)
```

Exercise 6 Define a function $f = x * \exp(x)$ and integrate over $(0, 1)$ numerically. Report your results as in Exercise 5.