Numerics 1: HW 11

Cooper Simpson

November 20, 2020

Numerics 1: Homework 11

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.integrate as spi

sns.set()
```

Problem 1

We investigate composite Trapezoidal and composite Simpson's rules for numerical quadrature.

a)

We begin by writing functions for applying our two rules on an arbitrary interval for an arbitrary function – we assume the interval and function are valid.

[2]:
```
'''
Function: compTrap -> Composite Trapezoidal rule
    Uses the composite Trapezoidal rule to compute an approximation
    to the integral of a function on an interval. Assumes that the
    function and interval constitute a valid integral.
Arguments:
    func: The callable function handle
    a: Left hand endpoint
    b: Right hand endpoint
    n: Number of subintervals greater than 0
    info: Return number of function evaluations (optional)
'''
def compTrap(func, a, b, n, info=False):
    if not callable(func) or n<=0:
        raise ValueError('Function not callable, or non-positive number of␣
 ↪subintervals.')

    x = np.linspace(a, b, n+1) #Evaluation locations
    h = x[1] - x[0] #Spacing

    f = func(x) #Evaluate the function at all locations
    f[0], f[-1] = 0.5*f[0], 0.5*f[-1] #Weight the endpoints
```

```
    if info:
        return h*np.sum(f), len(x)
    else:
        return h*np.sum(f)

'''
function: compSimp -> Composite Simpson's rule
    Uses the composite Simpson's rule to compute an approximation
    to the integral of a function on an interval. Assumes that the
    function and interval constitute a valid integral.
Arguments:
    func: The callable function handle
    a: Left hand endpoint
    b: Right hand endpoint
    n: Number of subintervals, is greater than 2 and even
    info: Return number of function evaluations (optional)
'''
def compSimp(func, a, b, n, info=False):
    if not callable(func) or n<=0:
        raise ValueError('Function not callable, or non-positive number of␣
 ↪subintervals.')

    x = np.linspace(a, b, n+1) #Evaluation locations
    h = x[1] - x[0] #Spacing

    f = func(x) #Evaluate the function at all locations
    f[2:n-1:2] *= 2 #All even terms sans endpoint
    f[1:n:2] *= 4 #All odd terms sans endpoint

    if info:
        return (h/3)*np.sum(f), len(x)
    else:
        return (h/3)*np.sum(f)
```

We are considering the integral given below.

$$\int_{-5}^{5} \frac{1}{1+s^2} ds$$

We will try out our functions to make sure we get a reasonable answer free of any errors, but we will not do any analysis yet.

[3]:
```
a = -5
b = 5

def f(s):
    return 1/(1+s**2)
```

[4]:
```
n = 100

T = compTrap(f, a, b, n)
S = compSimp(f, a, b, n)

print(T,S)
```

```
2.7467768808078956 2.746801526895768
```

Seems like we are good to go.

b).

Using the error estimates for the composite Trapezoidal and composite Simpson's rules we can choose the number of subintervals (n) to satisfy the following:

$$E_T(n) = \left| \int_{-5}^{5} \frac{1}{1+s^2} ds - T_n \right| < 10^{-4} \text{ and } E_S(n) = \left| \int_{-5}^{5} \frac{1}{1+s^2} ds - S_n \right| < 10^{-4}$$

Where T_n and S_n are the approximations using the Trapezoidal and Simpson's methods with respectively.

Starting with the Trapezoidal method we know that the absolute value of the error is given by the following formula:

$$E_T(n) = \frac{h^2}{12}(b-a)|f''(\eta)|,\ \eta \in [a,b]$$

We note that in our case $a = -5$ and $b = 5$. To use this we will need the second derivative of our function which is given as...

$$f''(s) = \frac{8s^2}{(1+s^2)^3} - \frac{2}{(1+s^2)^2}$$

We will maximize the derivative magnitude to ensure that we choose an appropriate number of subintervals. It is clear to see that this will occur at $s = 0$, and result in $f''(0) = -2$. Plugging this into our error equation along with a, b, and the definition for h, we have the following:

$$E_T(n) \leq \frac{2(10)^3}{12n^2}$$

We want $E_T(n) < 10^{-4}$, so we solve the resulting inequality for n and obtain...

$$\sqrt{\frac{10^7}{6}} < n$$

Moving on to Simpson's method we have the following formula for the absolute error:

$$\frac{h^4}{180}(b-a)f^{(4)}(\eta),\ \eta \in [a,b]$$

Again, we note that in our case $a = -5$ and $b = 5$. The fourth derivative of our function is given as ...

$$\frac{384s^4}{(1+s^2)^5} - \frac{288s^2}{(1+s^2)^4} + \frac{24}{(1+s^2)^3}$$

To maximize our derivative we take $s = 0$ which gives $f^{(4)}(0) = 24$. We then get the following inequality for the error:

$$E_S(n) \leq \frac{24(10)^5}{180n^4}$$

Again, we want $E_S(n) < 10^{-4}$ which gives the following:

$$\left(\frac{24 \cdot 10^9}{180}\right)^{1/4} < n$$

We need to subintervals to be integers (and in the case of Simpson's, even), so we will use our bounds to adjust.

[5]:
```
((10e7)/6)**(1/2)
```

[5]:
```
4082.48290463863
```

[6]:
```
((10e9)*24/180)**(1/4)
```

[6]:
```
191.08855844087336
```

So we take our critical values of n as follows:

$$\boxed{n_T = 4083,\ n_S = 192}$$

c).

We now compare our methods defined above using the values of n obtained from the theory above to a built in adaptive quadrature (Python equivalent of *quad*).

[7]:
```
exact = 2*np.arctan(5) #Exact value of the integral
print(exact)
```

```
2.746801533890032
```

We can see the exact analytic value of the integral given above for reference.

[8]:
```
n_T = 4083
n_S = 192

T, evalT = compTrap(f, a, b, n_T, info=True)
S, evalS = compSimp(f, a, b, n_S, info=True)
```

[9]:
```
print('Trapezoidal -> Result: {:f}, Error: {:.8f}, Evaluations: {}'.format(T, np.
 ↪abs(T-exact), evalT))
print("Simpson's -> Result: {:f}, Error: {:.9f}, Evaluations: {}".format(S, np.
 ↪abs(S-exact), evalS))
```

```
Trapezoidal -> Result: 2.746802, Error: 0.00000001, Evaluations: 4084
Simpson's -> Result: 2.746802, Error: 0.000000001, Evaluations: 193
```

[10]:
```
tol_1 = 10e-4
tol_2 = 10e-6

q1, _, info1 = spi.quad(f, a, b, epsabs=tol_1, full_output=1)
q2, _, info2 = spi.quad(f, a, b, epsabs=tol_2, full_output=1)
```

[11]:
```
print('quad(10e-4) -> Result: {:f}, Error: {:.11f}, Evaluations: {}'.format(q1,␣
 ↪np.abs(q1-exact), info1['neval']))
print('quad(10e-6) -> Result: {:f}, Error: {:.11f}, Evaluations: {}'.format(q2,␣
 ↪np.abs(q2-exact), info2['neval']))
```

```
quad(10e-4) -> Result: 2.746802, Error: 0.00000000002, Evaluations: 63
quad(10e-6) -> Result: 2.746802, Error: 0.00000000001, Evaluations: 105
```

We can see the results of all of our quadrature given above. Importantly, we note that the adaptive quadrature (even with a smaller tolerance) requires far fewer function evaluations than the Trapezoidal or Simpson's methods. In particular, to achieve the same accuracy (10^{-4}), the adaptive quadrature requires only 63 evaluations while Simpson's requires 193, and Trapezoidal a crazy 4084. Clearly whatever is happening inside the adaptive quadrature is much more efficient than our basic Newton-Cotes formulae.

Problem 3

We apply the midpoint rule, composite Trapezoidal rule, and the composite Simpson's rule to the following integral:

$$-4\int_0^1 xln(x)dx = 1$$

We use $n = 2, 4, 8, 16, \ldots, 512$.

We begin by writing code to execute the midpoint rule for an arbitrary function on an arbitrary interval, assuming they are valid.

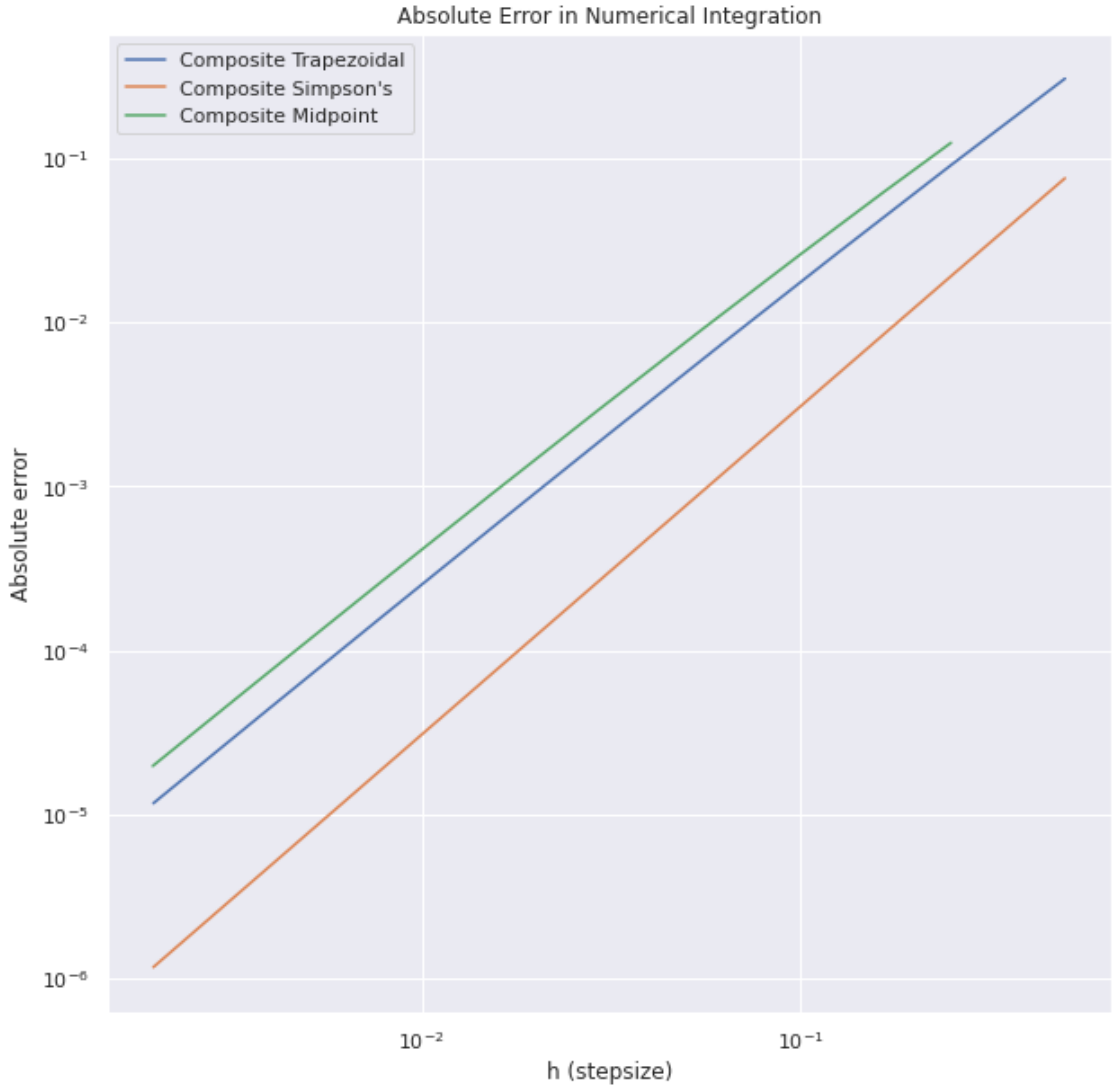
[12]:
```
'''
function: midpoint -> Midpoint Riemman sum
    Uses the composite Midpoint rule to compute an approximation
    to the integral of a function on an interval. Assumes that the
    function and interval constitute a valid integral.
Arguments:
    func: The callable function handle
    a: Left hand endpoint
    b: Right hand endpoint
    n: Number of subintervals, is greater than 0 and even
'''
def compMid(func, a, b, n):
    if not callable(func) or n<=0:
        raise ValueError('Function not callable, or non-positive number of␣
 ↪subintervals.')

    x = np.linspace(a, b, n+3) #Evaluation locations
    h = x[1]-x[0] #Spacing

    f = func(x) #Evaluate the function at all locations

    return 2*h*np.sum(f[1:-1:2]) #Only sum even points
```

We note that when we define our function below we account for the fact that $ln(0)$ is undefined (or limits to $-\infty$) even though the function $xln(x)$ evaluated at zero is zero.

[13]:
```
a = 0 #Left hand endpoint
b = 1 #Right hand endpoint

exact = 1 #Exact value of integral
n_vec = np.array([2**i for i in range(1,10)]) #Subintervals

#Define our function
def f(x):
    result = np.zeros(len(x))
    if x[0] == 0 and x[-1] == 1:
        result[1:-1] = x[1:-1]*np.log(x[1:-1])
    else:
```

```
        result = x*np.log(x)

    return result
```

[14]:
```
fig, ax = plt.subplots(1, 1, figsize=(10,10))
F = np.zeros((3,len(n_vec)))

#For each n use all three methods
for i,n in enumerate(n_vec):
    F[0,i] = np.abs(-4*compTrap(f, a, b, n) - 1)
    F[1,i] = np.abs(-4*compSimp(f, a, b, n) - 1)
    F[2,i] = np.abs(-4*compMid(f, a, b, n) - 1)

ax.loglog(1.0/n_vec, F[0,:])
ax.loglog(1.0/n_vec, F[1,:])
ax.loglog(1.0/(n_vec+2), F[2,:])

ax.set_title('Absolute Error in Numerical Integration')
ax.set_xlabel('h (stepsize)')
ax.set_ylabel('Absolute error')
ax.legend(['Composite Trapezoidal', "Composite Simpson's", 'Composite␣
 ↪Midpoint']);
```



In the composite Trapezoidal and composite Midpoint rules we have an error term that is $\mathcal{O}(h^2)$, and for composite Simpson's it is $\mathcal{O}(h^4)$. With this we expect to see similar error for Trapezoidal and Midpoint with a slope of about 2, and smaller error (with a slope of about 4) for Simpson's.

This is indeed what we find in the figure above. We see that overall Simpson's rule out performs the other two methods. As well, Midpoint and Trapezoidal have roughly the same error, and we note that the slopes appear to match expectations as well. Lastly, we note that the stepsize for Midpoint is actually $\frac{b-a}{n+2}$ which is why the line does not extend as far as the other two.