Homework 5: Integration and Linear Algebra

Sandy Auttelet

July 26, 2023

1 Integration

1.1

Consider the following code which approximates a Riemann sum for $f(x) = sin(x)$ on the interval $[0, 2\pi]$ with $h = dx$ spacing.

```
import numpy as np

#Homework 5: Integration 1

def f(x):
    return np.sin(x)

F = -np.cos(2*np.pi) + np.cos(0)

Integrals = []
dx = [1,0.1,0.01,0.001,0.0001,0.00001,0.000001]
for i in range(len(dx)):
    x = np.arange(0,2*np.pi,dx[i])
    I = sum(f(x))*dx[i]
    Integrals.append(I)

errors = []
for i in range(len(Integrals)):
    error = np.abs(F - Integrals[i])
    errors.append(error)

for i in range(len(errors)):
    print(errors[i],dx[i],dx[i]**2,dx[i]**3)
```

Use the code to generate a table of the error between I and the true value of the integral against the size of dx as it decreases from $1 \rightarrow 10^{-6}$. What value of n does the algorithm seem to scale with in terms of $O(h^n)$? (You may alter the code to generate the table)

Table 1: Errors vs dx

Errors	dx	dx^2	dx^3
0.103	1	1	1
$6.99 \cdot 10^{-4}$	0.1	0.01	0.001
$1.09 \cdot 10^{-5}$	0.01	0.0001	$1.00 \cdot 10^{-6}$
$7.55 \cdot 10^{-8}$	0.001	$1.00 \cdot 10^{-6}$	$1.00 \cdot 10^{-9}$
$6.27 \cdot 10^{-10}$	0.0001	$1.00 \cdot 10^{-8}$	$1.00 \cdot 10^{-12}$
$1.24 \cdot 10^{-11}$	$1.00 \cdot 10^{-5}$	$1.00 \cdot 10^{-10}$	$1.00 \cdot 10^{-15}$
$1.60 \cdot 10^{-13}$	$1.00 \cdot 10^{-6}$	$1.00 \cdot 10^{-12}$	$1.00 \cdot 10^{-18}$

From Table 1, you can see the error is closest when $n = 2$ so the error is $O(h^2)$.

1.2

Let f be a real valued function that is continuous on the interval $[a, b]$. Then the fundamental theorem of calculus says that for any $x \in [a, b]$:

$$\frac{d}{dx}\left(\int_a^x f(t)dt\right) = f(x)$$

We're going to numerically verify this for $f(x) = e^{-x^2}$ by estimating the derivative of an integral.
(A) Program the function $f(x) = e^{-x^2}$.
(B) Program the function $F(x) = \int_0^x f(x)dx$ using any method of numerical integration you like.
(C) Using a finite difference formula to estimate the derivative, verify that $f(2) \approx F'(2)$.

```
#Homework 5: Integration 2
h = 0.01 #found this to be good enough to pass assertion
def f(x):
    return np.exp(-x**2)

def F(x):
    return sc.integrate.quad(f,0,x)

def edF(x):
    constants = np.array([-1/12,2/3,0,-2/3,1/12])
    funvals = np.array([F(x+2*h),F(x+h),\
                        F(x),F(x-h),F(x-2*h)])
    return (1/h)*np.dot(constants,funvals)

assert np.allclose(f(2),edF(2)[0]) #indexed edF because quad also includes the error.
```

2 Linear Algebra

2.1

The trace of a square $n \times n$ matrix A is defined by the sum of the diagonal entries:

$$tr(A) := \sum_{i=1}^{n} a_{ii}$$

Write a program that calculates the trace of a given matrix.

```
    A = [[1,2,3],[4,5,6],[7,8,9]]
    trace = sum(np.diag(A))
```

2.2

Let e_i denote the column vector in $\mathbb{R}^n$ with a 1 in the ith entry and 0 everywhere else, so that $e_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$. You can find the inverse of a $n \times n$ matrix A by solving $Ax_i = e_i$ for $i = 1, \ldots, n$.
The inverse of the matrix will be:

$$A^{-1} = [x_1 \; x_2 \ldots x_n]$$

Where the x_i's form the columns of A^{-1}.
Write a program to calculate the inverse of a matrix based on any of the matrix solving methods in this chapter. You may find the command np.column stack((col1,col2,...,coln)) useful in forming the final matrix.

```
    A = [[1,2,3],[4,5,6],[7,8,9]]
    n = len(A)

    e = np.eye(n)
    A_inv = np.zeros((n,n))
    for i in range(n):
        A_inv[i] = np.linalg.solve(A,e[i])

    A_inv_true = np.linalg.inv(A)

    assert np.allclose(A_inv,A_inv_true)
```