BMI500_PythonQuiz

September 5, 2022

1 BMI 500 Lab1

1.1 Problem 1

[1]:
```
#make for loop for range of n
#use Leibniz formula
def Leibniz_formula(n):
    total = 0
    for k in range(n):
        total += (-1)**k/(2*k+1)
    return total
```

```
0.8666666666666667
```

1.2 Problem 2

[14]:
```
#part a; add if statement with %
def Leibniz_formula_remainder(n):
    total = 0
    for k in range(n):
        value = (-1)/(2*k+1)
        if k%2 == 0:
            value *=-1
        total += value
    return total
```

[15]:
```
#part b; for-loop with the quantity (-1)**n
def Leibniz_formula_npower(n):
    total = 0
    for k in range(n):
        total += (-1)**k/(2*k+1)
    return total
```

[19]:
```
#part c; construct a list
def Leibniz_formula_list(n):
    total_list = []
    for k in range(n):
        value = (-1)/(2*k+1)
        if k%2 == 0:
            value *=-1
        total_list.append(value)
    total = sum(total_list)
    return total
```

[28]:
```
#part d; construct a set
def Leibniz_formula_set(n):
    total_set = set()
    for k in range(n):
        value = (-1)/(2*k+1)
        if k%2 == 0:
            value *=-1
        total_set.add(value)
    total = sum(total_set)
    return total
```

[28]:
```
0.8666666666666667
```

[47]:
```
#part e; construct a dictionary
def Leibniz_formula_dict(n):
    dictionary = {}
    for k in range(n):
        value = (-1)/(2*k+1)
        if k%2 == 0:
            value *=-1
        dictionary[k] = value
    total = sum(dictionary.values())
    return total
```

[22]:
```
#part f; NumPy array
import numpy as np
def Leibniz_formula_np(n):
    arr = np.empty(n)
    for k in range(n):
        value = (-1)/(2*k+1)
        if k%2 == 0:
            value *=-1
        arr[k]=value
    total = np.sum(arr)
    return total
```

```
0.8666666666666667
```

[29]:
```
#part g; NumPy array indexing alternating sums
import numpy as np
def Leibniz_formula_np_alt(n):
    arr = np.empty(n)
    for k in range(n):
        value = (-1)/(2*k+1)
        if k%2 == 0:
            value *=-1
        arr[k]=value
    positive_list = arr[::2]
    negative_list = arr[1::2]
    totalpos = np.sum(positive_list)
    totalneg = np.sum(negative_list)
    return totalpos+totalneg
```

[44]:
```
#part j; non-alternating sums
def Leibniz_formula_nalt(n):
    total_list = []
    for k in range(n):
        value = (-1)/(2*k+1)
        if k%2 == 0:
            value *=-1
        total_list.append(value)

    pair_list = []
    i=0
    if n%2==0:
        while i<len(total_list):
            pair_list.append(total_list[i]+total_list[i+1])
            i+=2
    else:
        while i<len(total_list)-1:
            pair_list.append(total_list[i]+total_list[i+1])
            i+=2
        pair_list.append(total_list[-1])

    total = sum(total_list)
    return total
```

1.3 Problem 3

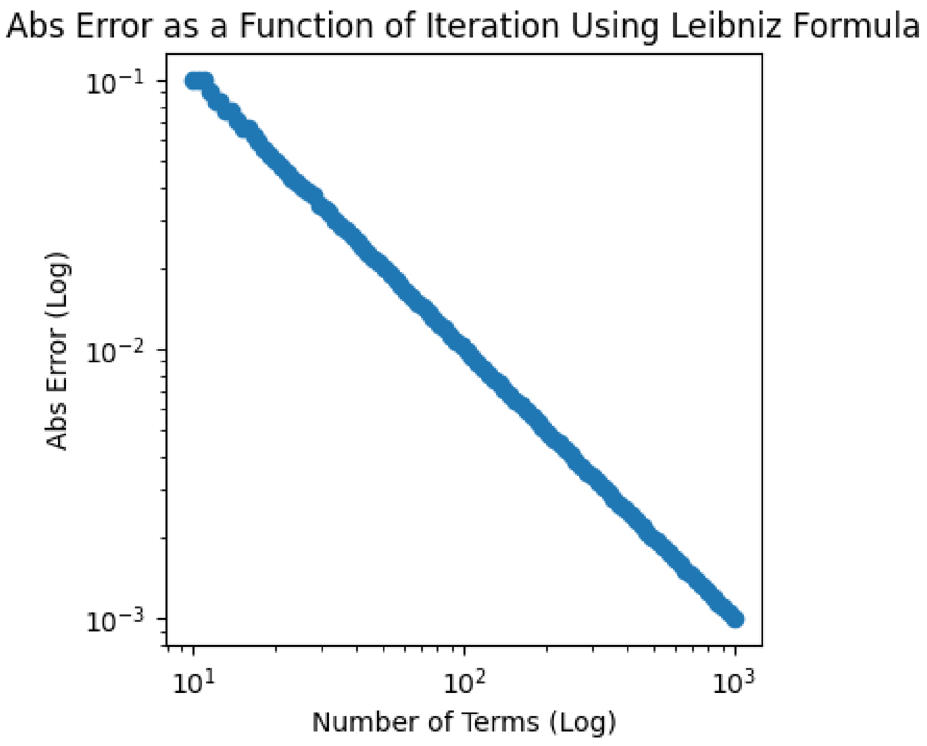
[66]:
```
#Which of these implementations of the Leibniz formula is the most accurate,␣
 ↪fastest, and/or clearest?
import time
import numpy as np
PI = np.pi

def Leibniz_formula_remainder(n):
    total = 0
    for k in range(n):
        value = (-1)/(2*k+1)
        if k%2 == 0:
            value *=-1
        total += value
    return total

def Leibniz_formula_npower(n):
    total = 0
    for k in range(n):
        total += (-1)**k/(2*k+1)
    return total

def Leibniz_formula_set(n):
    total_set = set()
    for k in range(n):
        value = (-1)/(2*k+1)
        if k%2 == 0:
            value *=-1
        total_set.add(value)
    total = sum(total_set)
    return total

def Leibniz_formula_dict(n):
    dictionary = {}
    for k in range(n):
        value = (-1)/(2*k+1)
        if k%2 == 0:
            value *=-1
        dictionary[k] = value
    total = sum(dictionary.values())
    return total

def Leibniz_formula_np(n):
    arr = np.empty(n)
    for k in range(n):
        value = (-1)/(2*k+1)
        if k%2 == 0:
            value *=-1
        arr[k]=value
    total = np.sum(arr)
    return total

def Leibniz_formula_np_alt(n):
    arr = np.empty(n)
    for k in range(n):
        value = (-1)/(2*k+1)
        if k%2 == 0:
            value *=-1
        arr[k]=value
    positive_list = arr[::2]
    negative_list = arr[1::2]
    totalpos = np.sum(positive_list)
    totalneg = np.sum(negative_list)
    return totalpos+totalneg

def Leibniz_formula_alt(n):
    total_list = []
    for k in range(n):
        value = (-1)/(2*k+1)
        if k%2 == 0:
            value *=-1
        total_list.append(value)

    pair_list = []
    i=0
    if n%2==0:
        while i<len(total_list):
            pair_list.append(total_list[i]+total_list[i+1])
            i+=2
    else:
        while i<len(total_list)-1:
            pair_list.append(total_list[i]+total_list[i+1])
            i+=2
        pair_list.append(total_list[-1])

    total = sum(total_list)
    return total

funct_names = [Leibniz_formula_remainder, Leibniz_formula_npower,␣
 ↪Leibniz_formula_set, Leibniz_formula_dict, Leibniz_formula_np,␣
 ↪Leibniz_formula_np_alt, Leibniz_formula_alt]

#making a function to test the parameters for each subsection of problem 2

time_list = []
return_soln = []
accuracy_list = []

def functions_tester():
    n = 999
    for func in funct_names:
        start_time = time.time()
        run_time = time.time() - start_time
        time_list.append(run_time)
        Leibniz_estimate = func(n) * 4
        return_soln.append(Leibniz_estimate)
        accuracy_list.append(abs(PI - Leibniz_estimate))

    return time_list, return_soln, accuracy_list

#running the tester, and printing results
functions_tester()
print("min time : ", min(time_list))
print("best accuracy : ", max(accuracy_list))
print("The method:", funct_names[accuracy_list.index(max(accuracy_list))])
```

```
min time :  0.0
best accuracy :  0.001001000750255443
The method: <function Leibniz_formula_set at 0x7ff28217b1f0>
```

Having made a function to test the prior variations of the Leibniz formula. The quickest version was that of each subsection at 0.0ms, and the most accurate was the version of the Set subsection, when calculating pi.

1.4 Problem 4

[5]:
```
#plot the absolute error in the sum as a function of the number of terms in the␣
↪sum

import matplotlib.pyplot as plt
import numpy as np

n_lst = np.logspace(1, 3, 100)

def Leibniz_formula_set(n):
    total_set = set()
    for k in range(int(n)):
        value = (-1)/(2*k+1)
        if k%2 == 0:
            value *=-1
        total_set.add(value)
    total = sum(total_set)
    return abs(np.pi-4*total)


#using set Leibniz formula with for loop to go through iterations
soln = []
for n in n_lst:
    soln.append(Leibniz_formula_set(n))

#use logarithmic axes/ legible labels for both the x-axis and y-axis
plt.figure(figsize=(4, 4))
plt.title('Abs Error as a Function of Iteration Using Leibniz Formula')
plt.yscale("log")
plt.ylabel('Abs Error (Log)')
plt.xscale("log")
plt.xlabel('Number of Terms (Log)')
plt.plot((n_lst), (soln),'o')


#creating the plot itself
plt.show()
```



1.5 Problem 5

[ ]:
```
#Would you do anything differently for computing   using the Leibniz formula if␣
 ↪you were using Matlab instead of Python?

At this task level there won't be too much of a difference between the Matlab␣
 ↪and Python functions. However, in future use Matlab could be better for␣
 ↪using arrays instead of lists when it's coming to iterations. Especially,␣
 ↪implementing a null matrix to get started instead of an empty list. Matlab␣
 ↪libraries are also better for the plots such as in question 4, to create a␣
 ↪quick understandable visual.
```