KONG_Python_tutorial

September 4, 2022

[65]:

```
# Problem 1
def nth_Leibniz_prob1(n):
    res = 0
    for k in range(n):
        res += ((-1)**k)/(2*k+1)
    return res
```

[66]:

```
# Problem 2

# for-loop with if-statement
def nth_Leibniz_prob2a(n):
    res = 0
    for k in range(n):
        res += 1/(2*k+1) if k%2 == 0 else -1/(2*k+1)
    return res

# for-loop with ** - same as the result of Problem 1
def nth_Leibniz_prob2b(n):
    res = 0
    for k in range(n):
        res += ((-1)**k)/(2*k+1)
    return res

# Python list with sum
def nth_Leibniz_prob2c(n):
    reslist = [((-1)**k)/(2*k+1) for k in range(n)]
    return sum(reslist)

# Python set with sum
def nth_Leibniz_prob2d(n):
    reslist = {((-1)**k)/(2*k+1) for k in range(n)}
    return sum(reslist)

# Python dictionary with sum
def nth_Leibniz_prob2e(n):
    reslist = {k:((-1)**k)/(2*k+1) for k in range(n)}
    return sum(reslist.values())
```

```
# NumPy array with sum
import numpy as np
def nth_Leibniz_prob2f(n):
    reslist = np.array([((-1)**k)/(2*k+1) for k in range(n)])
    return np.sum(reslist)

# NumPy array with sum of positives add sum of negatives
def nth_Leibniz_prob2g(n):
    reslist = np.array([((-1)**k)/(2*k+1) for k in range(n)])
    sumpos = np.sum(reslist[reslist > 0])
    sumneg = np.sum(reslist[reslist < 0])
    return sumpos + sumneg

# Sum of 1st-2nd, 3rd-4th, ...
def nth_Leibniz_prob2j(n):
    res = 0
    for k in range(0,n//2*2,2):
        # 1/(2k+1) - 1/[2(k+1)+1] = 2/[(2k+1)(2k+3)]
        res += 2/((2*k+1)*(2*k+3))
    if n%2 == 1:
        res += 1/(2*n-1)
    return res
```

[68]:

```
# Problem 3
import time
PI = np.pi
N = 10**5

def analysis(func, funcname):
    startTime = time.time()
    leibnizPI = func(N) * 4
    runTime = time.time() - startTime
    err = abs(PI-leibnizPI)

    res = '%s has an error of %.10e and a running time of %.6f␣
 ↪seconds'%(funcname, err, runTime)

    return res

print('With n = %d'%N)
print('\t',analysis(nth_Leibniz_prob2a, 'prob2a'))
print('\t',analysis(nth_Leibniz_prob2b, 'prob2b'))
print('\t',analysis(nth_Leibniz_prob2c, 'prob2c'))
print('\t',analysis(nth_Leibniz_prob2d, 'prob2d'))
print('\t',analysis(nth_Leibniz_prob2e, 'prob2e'))
print('\t',analysis(nth_Leibniz_prob2f, 'prob2f'))
print('\t',analysis(nth_Leibniz_prob2g, 'prob2g'))
print('\t',analysis(nth_Leibniz_prob2j, 'prob2j'))
```

```
With n = 100000
         prob2a has an error of 1.0000000073e-05 and a running time of 0.031913
seconds
         prob2b has an error of 1.0000000073e-05 and a running time of 0.056848
seconds
         prob2c has an error of 1.0000000073e-05 and a running time of 0.046875
seconds
         prob2d has an error of 1.0000000001e-05 and a running time of 0.097740
seconds
         prob2e has an error of 1.0000000073e-05 and a running time of 0.051861
seconds
         prob2f has an error of 1.0000000001e-05 and a running time of 0.043882
seconds
         prob2g has an error of 1.0000000003e-05 and a running time of 0.046877
seconds
         prob2j has an error of 9.9999999708e-06 and a running time of 0.009971
seconds
```

Problem 3 Analysis From the results, `2j` is the fastest and the most accurate implementation. From my view, 2b is the clearest implementation, which directly reflects the mathematical function. I will use `2j` to calculate because of its accuracy and efficiency.

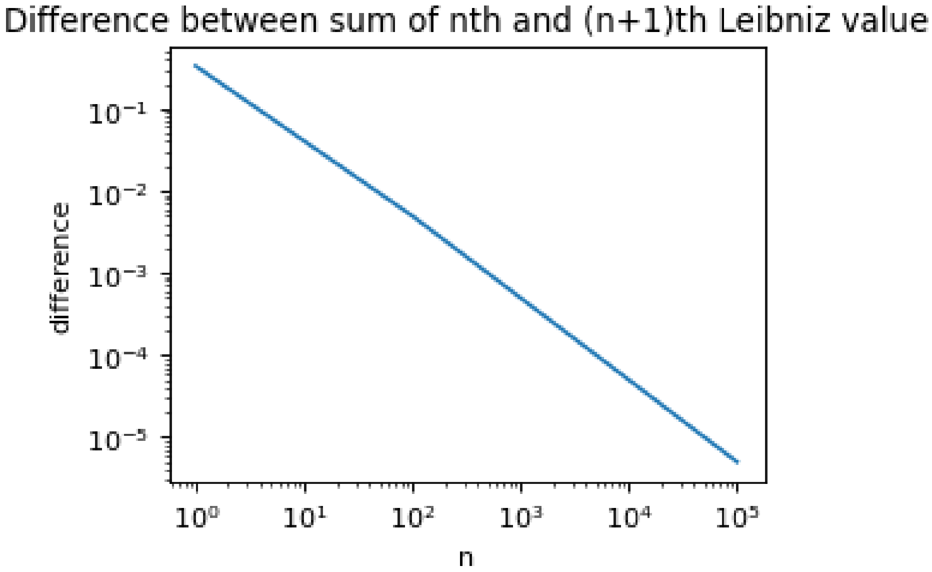
[86]:

```
# Problem 4
import matplotlib.pyplot as plt
import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"

ns = list(range(1,10**5,100))
diffs = []

for n in ns:
    diffs.append(abs(nth_Leibniz_prob2j(n) - nth_Leibniz_prob2j(n+1)))

plt.figure(figsize=(4,3), dpi = 80)
plt.loglog(ns,diffs)
plt.xlabel('n')
plt.ylabel('difference')
plt.title('Difference between sum of nth and (n+1)th Leibniz value')
plt.show()
```



Problem 5 If implementing with Matlab, I will not use array to store the values, because appending elements into an array requires extra running times in Matlab. Instead, I will only use a for-loop to calculate, even though the accuracy drops.