Section2

Elliott Ashby

October 17, 2022

1 q1

If fabs is ommitted there may arise a situation where total is negative and therefore if $1.0e^{-13}$ is added to it would have the opposite effect on the inequality therefore not making the desired outcome. Essentially, if term and total are negative, the wrong stopping condition is applied resulting in undesired results. (See Figure 1)

2 q2

If $1e^{-26}$ used our relative stopping condition would be more accurate than our absolute one, at $1.0e^{-13}$. The reason, that $1e^{-3}$ isnt used is that, while using a less accurate stopping condition is faster, π is defined to a much greater degree than 3 decimal places.

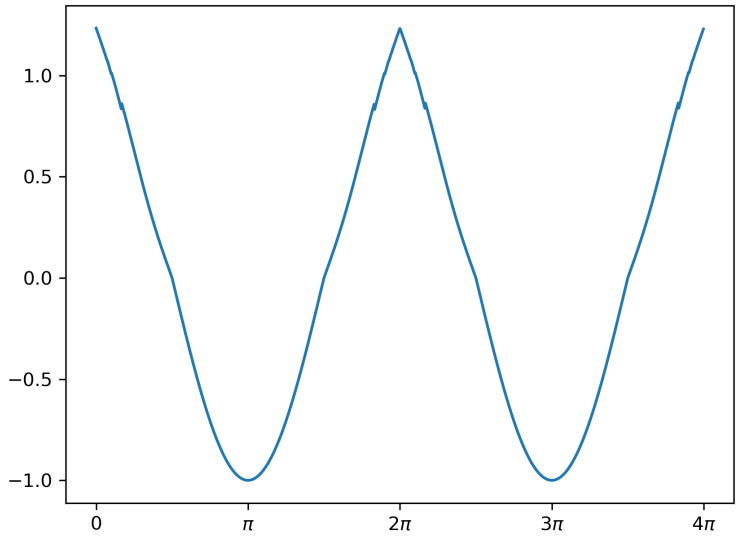
3 q3

```
from math import cos, fabs, pi
import matplotlib.pyplot as plt
import numpy as np


def g(x):
    n = 1
    total = term = cos(x)
    while fabs(term) > (1.0e-7 * fabs(total) + 1.0e-13):
        n += 2
        term = cos(n * x) / (n * n)
        total += term
    return total


x = np.arange(0, 4*pi, 0.01)
y = [g(i) for i in x]
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('g(x)')
```

Figure 1: Incorrect stopping condition:




```
# plt.xticks(ticks=[0, pi, 2 * pi, 3 * pi, 4 * pi], labels=['0',
#                                 '$\pi$', '2$\pi$', '3$\pi$', '4$\pi$'])
plt.savefig('q2_3.png', dpi=300)
```

The script here performs the function g(x) over the range 0 to 4π and saves it as a graph which can be seen as Figure 2.

4 q4

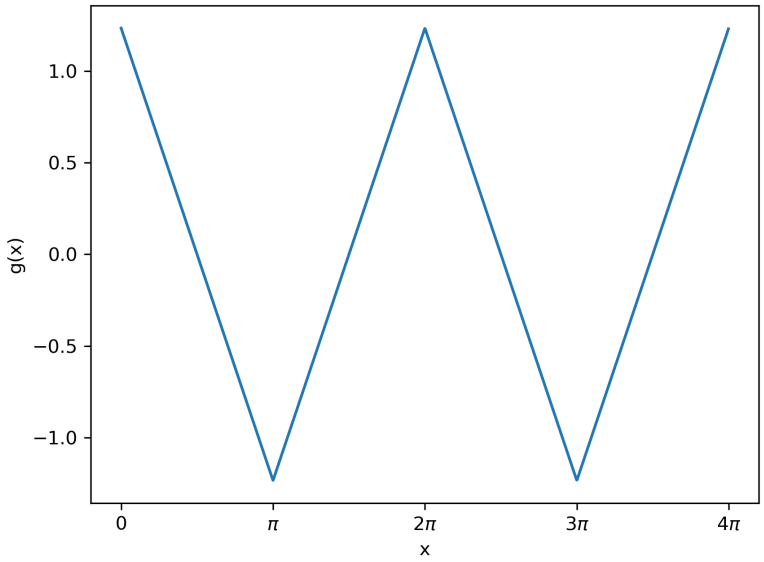
```
import matplotlib.pyplot as plt
import math
import numpy as np


r = 100.0
s = 10000.0
n = math.inf
a = [r + s]
while a[-1] > (1.0e-3 * sum(a) + 1.0e-13):
    a.append(r + s * a[-1] / (s + a[-1]))

print(a[-1])

b = [r + s]
for i in range(50):
    b.append(r + s * b[-1] / (s + b[-1]))
```

Figure 2: Correct stopping conditions and correct output of Q2.3:

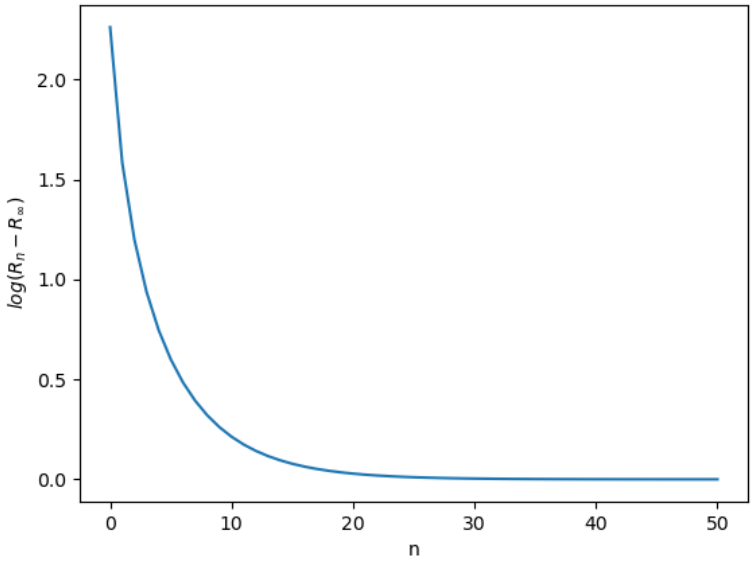



```
c = []
if len(a) >= len(b):
    for i in range(len(b)):
        c.append(math.log(b[i]) - math.log(a[-1]))
        an = np.arange(len(b))
else:
    for i in range(len(a)):
        c.append(math.log(b[i]) - math.log(a[-1]))
        an = np.arange(0, len(a))


plt.plot(an, c)
plt.ylabel('$log(R_n - R_\infty)$')
plt.xlabel('n')
plt.savefig('q2_4.png')
```

Here, we use a stopping condition to acquire R_∞ and print it. We then generate a list of this approximation as it increases in order to compare the two. The if else statement just allows me to change how many entries b has without needing to worry about the length of either list.
The script then saves the plot comparing the difference of R_n and R_∞ at different

Figure 3: $log(R_n - R_\infty)$



values of R_n.

5 q5

$R_n = R_\infty + Ae^{-Bn}$ is a good approximation since R_∞ is a constant and as n increases the Ae^{-Bn} term of the right side decreases in size exponentially, meaning that the value of R_n and R_∞ get exponentially closer together until Ae^{-Bn} is negligible at which point R_n and R_∞ are equal (and hence when taken away from one another equate to 0)