Numerical Analysis HW 1

Michael Vu

August 31, 2023

Problem 1. In the case of the divergent series, $\sum_{i=1}^{\infty} \frac{1}{i}$, my plan is to write a loop to compute this series, however, since computers tend to disagree with infinity, I will choose an arbitrary upper limit. I am going to assume, because this is a divergent series, I am only going to need to compute a relatively small amount of terms to see discrepancies compared to computing the arithmetic by hand. My source code follows:

```
program pr1
real :: s
integer :: n, i

s = 0.0
i = 1
n = 5

do while (i <= n)
s = s + 1.0/float(i)
print *, i, s
i = i + 1
end do

end program pr1
```

Compiling and running this program created what appears to be rounding errors when compared to manual computations. Here are the computations as $i = 1 \rightarrow 5$:

```
1   1.00000000
2   1.50000000
3   1.83333337
4   2.08333349
5   2.28333354
```

Here we find that when the value should be a decimal with 3 repeating, the program rounds off in a seemingly incomprehensible way.

Problem 2. In this problem, my goal is to create a program that establishes two points with arbitrary distance, compute the mid point, establish the mid point as the new left or right limit, compute the new mid point, and continue this loop until the program says otherwise. I will keep things consistent and create two scenarios, one where the mid point becomes the new right limit, and the other where the mid point becomes the new left limit. The code is as follows:

```
program pr2
real :: a, b, m
integer :: i

a = 0.0
b = 10.0
i = 0

print *, "(a < m)"
do while (a < m)
i = i + 1
m = (a + b)/2.0
b = m
print *, i, m
end do

a = 0.0
b = 10.0
i = 0

print *, "(b > m)"
```

```
do while (b > m)
i = i + 1
m = (a + b)/2.0
a = m
print *, i, m
end do

end program pr2
```

After compiling and running the program, we find both scenarios end in similar fashion. I'll display the first and last five lines to each scenario:

```
(a < m) (this is when midpoint becomes new right side)
1   5.00000000
2   2.50000000
3   1.25000000
4   0.625000000
5   0.312500000

149   1.40129846E-44
150   7.00649232E-45
151   2.80259693E-45
152   1.40129846E-45
153   0.00000000

(b > m) (this is when midpoint becomes new left side)
1   5.00000000
2   7.50000000
3   8.75000000
4   9.37500000
5   9.68750000

20   9.99999046
21   9.99999523
22   9.99999809
23   9.99999905
24   10.0000000
```

We find that the program can only compute to a certain precision before it cannot distinguish between m and the other limit that m is getting closer to. I also found that if the outer limit is any number other than 0, the program averages 29 computations before it terminates, whereas the average is significantly higher for an outer limit of 0, with 154 computations.