SEMINAR IV HOMEWORK

7. [Python] Show numerically that the alternating sum of((-1)^(n+1))/n converges to ln2. Change the order of summation in this series – for example by first adding p positive terms, then q negative terms, and so on – and show numerically that the rearrangement gives a different sum (depending on p, q).

Mathematically we have demonstrated that the alternating sum of (-1) ^(n+1) / n converges to ln (2). However, we know that this series is only conditionally (or semi) convergent.

In this program we will test the Riemann Series Theorem (also known as the Riemann Rearrangement Theorem), which states as follows:

If an infinite series is conditionally convergent, then its terms can be arranged in a permutation so that the series converges to any given value, or even diverges.

We are unable to test on an infinite set of numbers, but we will use sets of >10.000 numbers in order to approximate the sum of the series arranged in some different configurations:

By Hus Lucian-Antonio, Group 914, FMI, UBB

CONTROL CASES

1. Default Configuration – One Positive, One Negative

```
Default Configuration: One positive - One Negative
    50.000 Elements: 0.6931371802599592
   250.000 Elements: 0.6931451805479898
   500.000 Elements: 0.6931461805570046
 1.000.000 Elements: 0.6931466805592525
 2.500.000 Elements: 0.6931469805599338
 5.000.000 Elements: 0.6931470805600276
10.000.000 Elements: 0.6931471305600962
```

```
Official ln(2) approximation:  0.6931471805599453
```

Hence, we have demonstrated numerically that the alternating sum of ((-1) ^(n+1))/n converges to ln2

2. Slightly Modified Configuration – Two Positive, Two Negative

```
Notice that nothing has changed in this variation
    50.000 Elements: 0.6931371798599193
   250.000 Elements: 0.6931451805319899
   500.000 Elements: 0.6931461805530049
 1.000.000 Elements: 0.6931466805582531
 2.500.000 Elements: 0.6931469805597748
 5.000.000 Elements: 0.6931470805599885
10.000.000 Elements: 0.6931471305600869
```

Obs. It seems that nothing changes when we keep the ratio of added positive / negative elements equal.

MODIFIED CONFIGURATIONS

1. One Positive / Two Negative and vice versa (Two Positive / One Negative)

```
Modified Configuration: Two positive - One Negative
    50.000 Elements: 1.0397007711399044
   250.000 Elements: 1.039716770851867
   500.000 Elements: 1.039718770842846
 1.000.000 Elements: 1.039719770840546
 2.500.000 Elements: 1.039720370839835
 5.000.000 Elements: 1.039720570839577
10.000.000 Elements: 1.0397206708394047

Modified Configuration: One positive - Two Negative
    50.000 Elements: 0.34656358967993034
   250.000 Elements: 0.346571590255956
   500.000 Elements: 0.34657259027395343
 1.000.000 Elements: 0.34657309027843697
 2.500.000 Elements: 0.3465733902796769
 5.000.000 Elements: 0.34657349027983925
10.000.000 Elements: 0.3465735402798763
```

We notice that in the case of more positive terms added first, our sum tends to slightly increase as we will see later on, whilst in the case of more negative terms added first, our sum will half.

2. One Positive / Three Negative and vice versa (Three Positive / One Negative)

```
Modified Configuration: Three positive - One Negative
    50.000 Elements: 1.2424233243273155
   250.000 Elements: 1.2424473248232877
   500.000 Elements: 1.2424503248882772
 1.000.000 Elements: 1.2424518248894623
 2.500.000 Elements: 1.24245272489313
 5.000.000 Elements: 1.2424530248935786
10.000.000 Elements: 1.242453174893539

Modified Configuration: One positive - Three Negative
    50.000 Elements: 0.14383103619257195
   250.000 Elements: 0.14383903620855956
   500.000 Elements: 0.14384003622556182
 1.000.000 Elements: 0.14384053622482018
 2.500.000 Elements: 0.14384083622573274
 5.000.000 Elements: 0.14384093622591937
10.000.000 Elements: 0.14384098622590635
```

MODIFIED CONFIGURATIONS

3. One Positive / Five Negative and vice versa (Five Positive / One Negative)

```
Modified Configuration: Five positive - One Negative
    50.000 Elements: 1.497816138476976
   250.000 Elements: 1.4978561368449534
   500.000 Elements: 1.4978611367939498
 1.000.000 Elements: 1.4978636367811402
 2.500.000 Elements: 1.4978651367774671
 5.000.000 Elements: 1.4978656367769254
10.000.000 Elements: 1.4978658867766022

Modified Configuration: One positive - Five Negative
    50.000 Elements: -0.11158177675714658
   250.000 Elements: -0.11157377570110891
   500.000 Elements: -0.11157277566811039
 1.000.000 Elements: -0.1115722756598621
 2.500.000 Elements: -0.1115719756575503
 5.000.000 Elements: -0.11157187565721942
10.000.000 Elements: -0.11157182565714062
```

4. One Positive / Twenty-Five Negative and vice versa (Twenty-Five Positive / One Negative)

```
Modified Configuration: Twenty-Five positive - One Negative
    50.000 Elements: 2.30233513469 4056
   250.000 Elements: 2.302535094662115
   500.000 Elements: 2.302560093411072
 1.000.000 Elements: 2.302572593098507
 2.500.000 Elements: 2.302580093011015
 5.000.000 Elements: 2.3025825929987467
10.000.000 Elements: 2.302583842995694

Modified Configuration: One positive - Twenty-Five Negative
    50.000 Elements: -0.916300720942298
   250.000 Elements: -0.9162927314378785
   500.000 Elements: -0.9162917317650974
 1.000.000 Elements: -0.9162912318468784
 2.500.000 Elements: -0.9162909318697336
 5.000.000 Elements: -0.9162908318730104
10.000.000 Elements: -0.9162907818738254
```

MODIFIED CONFIGURATIONS

5. One Positive / A Hundred Negative and vice versa (A Hundred Positive / One Negative)

```
Modified Configuration: A Hundred Positive - One Negative
    50.000 Elements: 2.994732940252938
   250.000 Elements: 2.995532300222066
   500.000 Elements: 2.9956322802210047
 1.000.000 Elements: 2.9956822752209344
 2.500.000 Elements: 2.99571227382097
 5.000.000 Elements: 2.9957222736212263
10.000.000 Elements: 2.995727273571223

Modified Configuration: One Positive - A Hundred Negative
    50.000 Elements: -1.6094476165123976
   250.000 Elements: -1.6094399006780442
   500.000 Elements: -1.6094389094976318
 1.000.000 Elements: -1.6094384117002791
 2.500.000 Elements: -1.6094381123168486
 5.000.000 Elements: -1.6094380124050032
10.000.000 Elements: -1.609437962427081
```

6. Theoretical case for infinite terms

Following the numerical computations, we can notice that introducing more positive terms has the effect of increasing the total sum, while including more negative terms has the opposite effect, reducing the sum.

This behaviour conforms to the expected characteristics of conditionally convergent series.

Whilst we are technologically limited to working with finite sums, we can theorize that the more positive terms we add first, our sum will be ever closer to infinity, and the more negative terms we add first, our sum will be ever close to minus infinity.

So, the case that reaches these endpoints of the completed real line are:

1. *All Positive Terms First – Then the negative ones => Infinity*
2. *All Negative Terms First – Then the positive ones => -Infinity*

PYTHON IMPLEMENTATION

For an easier understanding of the python implementation of this program I have sectioned the code in 3 segments: Imports, Back-End, Front End.

Imports: Using ‘math’ library for calculating the actual value of ln (2).

Back-End:

- Positive / Negative Terms Lists Generator (I decided to go with lists for ease of computations later on despite the obvious space complexity drawbacks)
- Arrangement Generators

Front-End:

- Result Table Generator
- Main GUI (in main function)

Here’s a quick glance at the implementation. For the full code, which I have also attached in the assignment please visit my repository on GitHub. [Ctrl+Click to redirect].

CODE SEGMENTS:

```python
def lists_generator(limit: int) -> (list, list):

    v_pos = []
    v_neg = []
    for i in range(1, limit, 2):
        v_pos.append(1/i)
    for j in range(2, limit, 2):
        v_neg.append(-1*(1/j))

    v_pos.reverse()
    v_neg.reverse()
    return v_pos, v_neg

```

List Generator Function

```python
def one_positive_one_negative_arrangement(limit: int) -> float:

    v_pos, v_neg = lists_generator(limit)

    s = 0.0
    while v_pos != [] and v_neg != []:
        s += v_pos[-1]
        s += v_neg[-1]
        v_pos.pop()
        v_neg.pop()
    return s

```

One Positive – One Negative Arrangement Generator

```python
def hun_positive_one_negative_arrangement(limit: int) -> float:

    v_pos, v_neg = lists_generator(limit)

    s = 0.0
    while len(v_pos) >= 100 and v_neg != []:

        times = 1
        while times <= 100:
            times = times + 1
            s += v_pos[-1]
            v_pos.pop()

        s += v_neg[-1]
        v_neg.pop()

    return s
```

One Hundred Positive – One Negative Arrangement Generator

```python
    if configuration == -100:
        print()
        print("Modified Configuration: One Positive - A Hundred Negative")

        result1 = one_positive_hun_negative_arrangement(50000)
        result2 = one_positive_hun_negative_arrangement(250000)
        result3 = one_positive_hun_negative_arrangement(500000)
        result4 = one_positive_hun_negative_arrangement(1000000)
        result5 = one_positive_hun_negative_arrangement(2500000)
        result6 = one_positive_hun_negative_arrangement(5000000)
        result7 = one_positive_hun_negative_arrangement(10000000)

        print("    50.000 Elements:", result1)
        print("   250.000 Elements:", result2)
        print("   500.000 Elements:", result3)
        print(" 1.000.000 Elements:", result4)
        print(" 2.500.000 Elements:", result5)
        print(" 5.000.000 Elements:", result6)
        print("10.000.000 Elements:", result7)
```

One Hundred Negative – One Positive Result Table Generator

POSSIBLE CODE OPTIMIZATIONS:

Due to the limited time I had to spend on this assignment the code is in its current form a bit bare, and can be improved as following:

Space Complexity-Wise:

- Dynamically Generate the arrangements – no longer need to generate lists

Cleaner Code:

- In the front-end section avoid the repeated explicit function calls, rather use a for loop to iterate through the result cases
- More Documentation