**The assignment asks us to compute the summation of the series:**

1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + ……… + 1/9999 – 1/10000**, in 8 different ways – that is in 4 ways using float and the other four ways in double.** The results of the different summations in 30 digits should be approximately: 0.6930971830599452969117232371458

**Observation on the accuracy of the result:**

**In general, the computations done with double is more accurate compared to those done with the float numbers. Amongst the computation done with doubles the one that computes from right to left is the most accurate followed by the one that computes separately from right to left.**

**Analysis:**

**Double vs float:**

To represent floating point numbers, we use **float**, **double** and **long double**. **The byte size for float is 4 while the byte size for double is 8. Float** is a 32-bit single precision **f**loating **p**oint **n**umber**, which has** 1 bit for the sign, 8 bits for the exponent, and 23 **bits** for the **fractional part**. **Double** is a 64-bit double precision **f**loating **p**oint **n**umber **which has** 1 bit for the sign, 11 bits for the exponent, and 52 bits for the **fractional part**. **F**loat has 7 decimal digits of precision **and** double has 15 decimal digits of precision. That is using double has twice the precision compared to float numbers, therefore using double is better than using double numbers.

**Truncation and Rounding:**

**Floating point numbers on most computers use binary representation, and we know that in the binary numeral system all real numbers that are fractions on the form a/b, with ‘a’ an integer and b = 2k** **for some positive integer k. This means that fractions with the denominator equal to 2k is represented exactly but the fractions with the denominator not equal to 2k is not represented exactly. Even before we start doing arithmetic, we therefore have the challenge of finding good approximations to these numbers being used. There are two different ways to do this, truncation and rounding. A number is said to be truncated to m digits when each digit except the m leftmost ones is replaced by 0. Truncation is a very simple way to convert any real number to a floating-point number. We just start computing the digits of the number and stop as soon as we have all the required digits. Rounding is an alternative to truncation that in general will make the error smaller but is more complicated. A number is said to be rounded to m digits when it is replaced by the nearest number with the property that all digits beyond position m is 0.** Floating point numbers have finite number of bits for representing the mantissa, so we need to approximate the infinitely many real numbers into a finite number, which requires an approximate representation. Most programs can store the results of integer computations in 32 or 64 bits. Given any fixed number of bits, most calculations with real numbers will produce quantities that cannot be exactly represented using a large number of bits. Therefore, the result of a floating-point calculation must often be rounded off to fit into a **finite** representation, and the rest part of the number is truncated. Thus, this causes rounding error, which is a characteristic feature of floating-point computation.

**Accurate summations:**

**The most accurate result by adding from right to left. While adding from right to left we are adding very small values together and letting them to at least add to something so that when it reaches the bigger values will have less difference. But while adding from the left to right we end up adding a very large number to a very small number, which will result in the smaller value being ignored and thus increases the error.** Therefore, adding two values of similar magnitude is more accurate than adding two values of very different magnitudes, since you "discard" fewer bits of precision in the smaller value that way. The second most accurate one is by adding positive and negative terms separately from right to left and then subtracting them. This method is less accurate than the first method because here the difference in the magnitude being added is bigger which may cause a little bit of inaccuracy. Also, that in this case at the end we are subtracting two numbers of similar magnitudes, which results in loss of significant digits. This phenomenon is called catastrophic cancellation in which subtracting good approximations to two nearby numbers may yield a very bad approximation to the difference of the original numbers.

**REFERENCES:**

<https://www.geeksforgeeks.org/difference-float-double-c-cpp/>

<https://www.soa.org/news-and-publications/newsletters/compact/2014/may/com-2014-iss51/losing-my-precision-tips-for-handling-tricky-floating-point-arithmetic/>

**chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.uio.no/studier/emner/matnat/math/MAT-INF1100/h10/kompendiet/kap5.pdf**

**Output Reference:**

**Text

Description automatically generated with medium confidence**