Computational Astrophysics Exercises — Assignments Set 1

1. Floating Point Arithmetic

Write a program to calculate the sum

$$s_k = \sum_{n=1}^{k} \frac{1}{n^2}.$$

a. Check your results for various different values for $k = 10^3, 10^6, 10^7, 10^8$. What do you find for $k \to \infty$? Compare your results for single and double precision. Compare to the analytical limit $s_\infty = \pi^2/6$ by plotting $|s_k - s_\infty| / s_\infty$. Print out the the runtime of your program.
b. Calculate the sums from a. backwards starting with the smallest summand. Check your results for $k = 10^3, 10^6, 10^7, 10^8$ and compare with your results from a. Explain your findings!

Note: If you use `C`, you can easily switch between the different data types by declaration. In `python`, you have to take care that the interpreter does not change the type dynamically, use `numpy` for the different data types.

```python
import numpy as np
single_precision_one = np.float32(1.0) # 4 bytes long
double_precision_one = np.float64(1.0) # 8 bytes long
also_double_precision_one = float(1.0) # 8 bytes long, built-in default
```

(0b11 points)

2. Machine epsilon/unit roundoff

The machine epsilon is a measure for rounding errors that occur in floating point arithmetic. In practice, the machine epsilon is defined as the smallest value ε which fulfills

$$1 + \varepsilon > 1.$$

Write a computer program to calculate the machine epsilon of your computer. What is the machine epsilon for single and double precision? Compare your result with the information provided by libraries on your system, e.g., `np.finfo()` if you use python or `limits.h` if you're sailing with `C`.

(0b10 points)

3. Nifty tricks - rescaling

Consider the following equation

$$c = \sqrt{a^2 + b^2}.$$

Using single precision data types (aka binary32 [1]) in the following `C`-snippet, you get the results `inf` for $a = 1e30$ and $b = 1$.

[1] see `https://en.wikipedia.org/wiki/Single-precision_floating-point_format`

```c
/* snippet.c, calculate square root of binary32 */
int main (int argc, char **argv)
{
    float a;
    float b;
    float res = 0.0;

    a = 1e30;
    b = 1.0;

    res = sqrt(a*a + b*b);
    fprintf(stdout, "standard method res: %e \n", res);

    return 0;
}
```

```
    $ clang snippet.c
    $ ./a.out
standard method res: inf
```

Explain why you obtain this result. Which nifty trick can you apply to avoid this problem assuming you cannot use binary64? Note: If you do not want to use `C`, you can also use `python` or any other programming language of your choice.

(0b10 points)

4. Old greek guy got pie

In 250 BC the Greek mathematician Archimedes estimated the number π as follows. He considered a circle with radius $\frac{1}{2}$, i.e. circumference π, and inscribed a square (see fig. 1). The perimeter of the square is smaller than the circumference of the circle, and hence a lower bound for π. By using regular n-polygons, he obtained better estimates for π. The n-polygon has the perimeter

$$U_n = n \sin\left(\frac{\pi}{n}\right), \tag{1}$$

with the limit

$$\lim_{n\to\infty} U_n = \pi. \tag{2}$$

Let's look at the square first. The perimeter is given by $A_n = U_{2^n}$ with $n = 2$

$$A_2 = U_4 = 4a = 4\sqrt{r^2 + r^2} = 2\sqrt{2}. \tag{3}$$

Without proof, the recursion formula to calculate the perimeter of a regular n-polyon is given by

$$A_{n+1} = 2^n \sqrt{2\left(1 - \sqrt{1 - \left(\frac{A_n}{2^n}\right)^2}\right)}. \tag{4}$$

Exercise:

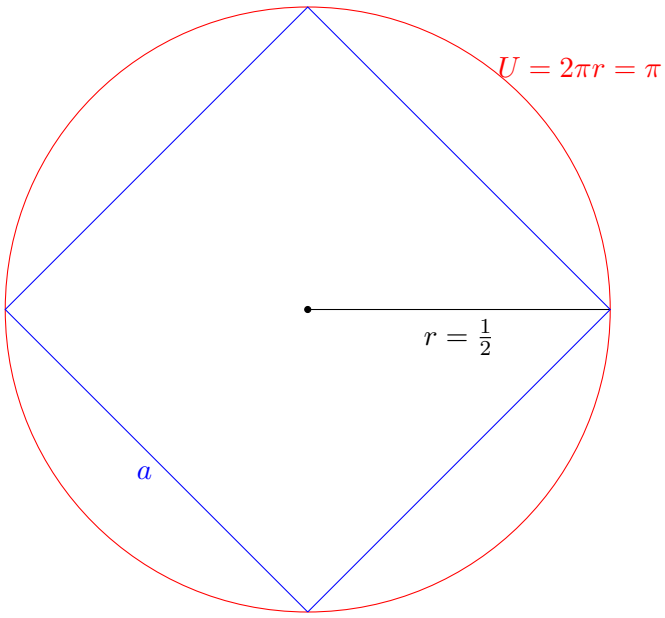

Figure 1: Use polygons to calculate π using the idea of Archimedes from Syrakus (287-212 BC). Here, the perimeter of the square gives a lower limit to π of $2\sqrt{2}$.

1. Write a program that calculates π using eq. (4). You will find a behaviour as shown in fig. 2.

2. Explain why the error gets larger after iteration no. 15, although we would expect an accuracy of 10^{-15} using double precision.

3. Kahan[2] suggested a slightly different algorithm using the iteration formula

$$A_{n+1} = 2^n\sqrt{R_n}, \quad \text{with} \quad R_n = 4\frac{1-\sqrt{1-\left(\frac{A_n}{2^n}\right)^2}}{2}, \tag{5}$$

and define $Z_n = R_n/4$, which can be rewritten to (using $(a+b)(a-b) = a^2 - b^2$)

$$Z_n = \frac{2\left(\frac{A_n}{2^{n+1}}\right)^2}{1+\sqrt{1-\left(\frac{A_n}{2^n}\right)^2}}, \qquad A_{n+1} = 2^n\sqrt{4Z_n}. \tag{6}$$

Now, use this alternative form to calculate π and explain your findings.

(0b11 points)

in total 0b1010 points.

Deadline: 4 May 2025 at noon, via email (including code) to `ch.schaefer@uni-tuebingen.de`.

[2] `https://en.wikipedia.org/wiki/William_Kahan`

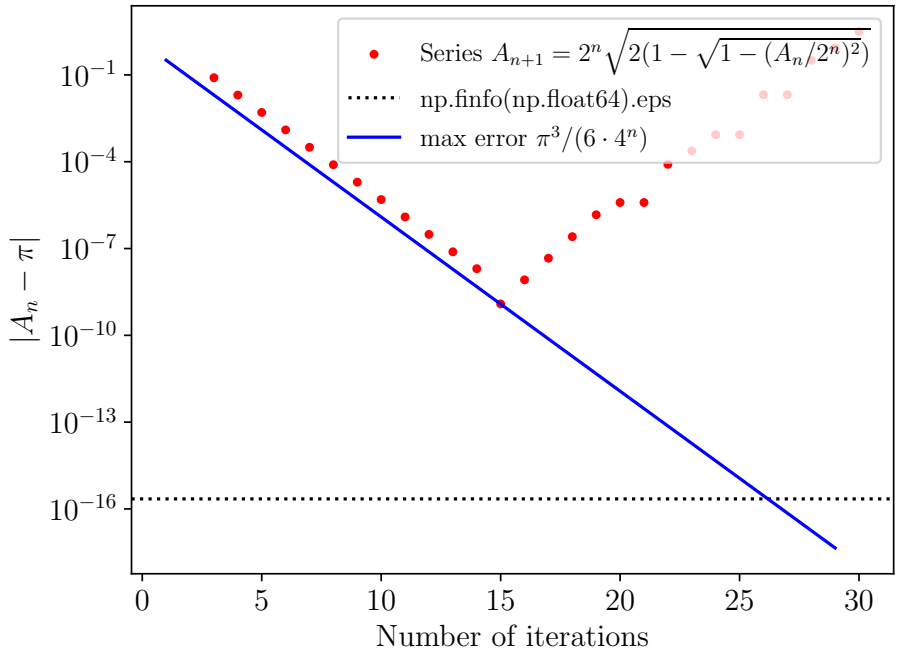

Figure 2: Error for the result obtained with eq. (4) as a function of the number of iterations. Note the divergence after iteration no. 15.