Exercise 1.

a) Consider the series

$$e^{\tan x} = 1 + x + \frac{x^2}{2!} + \frac{3x^3}{3!} + \frac{9x^4}{4!} + \cdots \quad \left(|x| \le \frac{\pi}{2} \right)$$

Retaining three terms in the series, estimate the remaining series using *o*-notation (little *o*), as $x \to 0$ i.e., find the largest integer *n* such that

$$e^{\tan x} = 1 + x + \frac{x^2}{2!} + o(x^n).$$

b) Repeat the problem using *O*-notation and the series

$$\ln \tan x = \ln x + \frac{x^2}{3} + \frac{7x^4}{90} + \frac{62x^6}{2835} + \cdots \quad \left(0 < |x| < \frac{\pi}{2} \right),$$

i.e. find an integer *n* such that $\ln \tan x = \ln x + \frac{x^2}{3} + \frac{7x^4}{90} + O(x^n)$.

Exercise 2. Let *x* be a real number. Consider a geometric series

$$x_n = 1 + x + \cdots + x^n$$

a) Find the condition when the series converges and its limit.
b) Determine the order and rate of convergence of this series.

Exercise 3. Polynomials appear in many areas of mathematics, physics and generally in science. A polynomial of degree *n* in variable *x* is commonly defined as

$$P(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n = \sum_{k=0}^{n} a_k x^k$$

Write a computer function `poly(a,x)` that receives an array/list of coefficients, *a*, and a number, *x*, and returns the value of the corresponding polynomial, *P*(*x*), using the most straightforward approach based on the direct evaluation of powers x^k, `x^k` in MATLAB and `x**k` in Python.

A British mathematician, William George Horner (1786-1837), described an efficient way to evaluate polynomials. It is based on the following representation.

$$P(x) = \Big(\Big(\cdots \Big(\big(a_n x + a_{n-1} \big) x + a_{n-1} \Big) + \ldots + a_2 \Big) x + a_1 \Big) x + a_0$$

This approach is also known as Horner's method, Horner's rule (chiefly US) or Horner's scheme (chiefly UK). It allows evaluation of a polynomial of degree *n* with only *n* multiplications and *n* additions. This is optimal, since there are polynomials of degree *n* that cannot be evaluated with fewer arithmetic operations. This algorithm is much older than Horner and can be traced back many hundred years to Chinese and Persian mathematicians.

Write another computer function, `nest(a,x)`, with the same arguments to evaluate the same polynomial using Horner's method. Then do the following.

a) Test your functions using coefficients, $a_0 = -1;\ a_1 = 5;\ a_2 = -3;\ a_3 = 3;\ a_4 = 2$ Verify that either function evaluates to 1.25 when $x = 1/2$.

b) Use these functions to evaluate $P(x) = 1 + x + \ldots + x^{50}$ at $x = 1.00001$ (Use built-in function `ones` in MATLAB or `numpy.ones` in Python to save typing). Find the absolute and relative error of the computations by comparing with the equivalent expression, $P(x) = \left(x^{51} - 1\right) \big/ (x - 1)$

c) Use again these functions to evaluate $P(x) = 1 - x + x^2 - x^3 + \ldots + x^{98} - x^{99}$ at $x = 1.00001$ Then find a simpler, equivalent expression, and use it to estimate the error of the nested multiplication.

d) Test the efficiency of the direct and nested methods. To this end, read how to measure the execution time of MATLAB/Python scripts. Then define a polynomial of degree 2000 with unit coefficients and run `poly` and `nest` in two separate loops each time measuring the elapsed time. It is recommended to run 15000 cycles however you may select a different number that is more suitable for your computer system. Report your results.

Exercise 4. What are the limit and the order of convergence of the following expression as $h \to 0$.

$$f(h) = \frac{(1+h) - e^h}{h^2}$$

Express it in the form of big and little o, i.e. $f(h) = L + O(h^n) = L + o(h^m)$.

Exercise 5. A remarkable theorem from calculus is often useful in establishing the convergence of a series and in estimating the error involved in truncating a series. From it, we have the following important principle for alternating series*: If the magnitudes of the terms in an alternating series converge monotonically to zero, then the error in truncating the series is no larger than the magnitude of the first omitted term.*

The alternating series theorem states if $a_1 \geq a_2 \geq \ldots \geq a_n \geq \ldots \geq 0$ for all n and $a_n \to 0$ then the alternating series, $a_1 - a_2 + a_3 - a_4 + \ldots$ converges to a certain number S

$$S_n \equiv \sum_{k=1}^{n} (-1)^{k-1} a_k \xrightarrow[n\to\infty]{} S \equiv \sum_{k=1}^{\infty} (-1)^{k-1} a_k = a_1 - a_2 + a_3 - a_4 + \ldots$$

Moreover, for all n, it holds

$$\left|S - S_n\right| = \left| \sum_{k=n+1}^{\infty} (-1)^{k-1} a_k \right| \leq a_{n+1}$$

Armed with this piece of knowledge, consider a very useful expansion of $\ln(1+x)$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots$$

Then do the following.

a) Establish that the series converges when $|x|<1$. Verify that the series is still convergent when $x=1$ and is divergent when $x=-1$ thus giving the interval of convergence as $-1<x\le 1$

b) When $x = 1$, this expansion gives a logarithmically convergent series for ln2

$$\ln 2 = 1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots$$

Estimate how many terms in the partial sum, S_n, are needed to approximate ln2 with an absolute error $|\ln 2 - S_n| < 10^{-3}$

c) Write a computer code that approximates ln2 by partial sums

$$S_n = 1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots+\frac{(-1)^{n-1}}{n}$$

Then find numerically how close the partial sum, S_n, actually approximates ln2 for the estimated in b) number n, i.e. find the absolute approximation error, $|\ln 2 - S_n|$.

d) Running your computer code, find how many terms, n, are actually needed to approximate ln2 with the same absolute error. An elegant way of implementing it in your computer code is using a while-loop.

Exercise 6. Write a computer code that implements Aitken's **iterated** acceleration. Then perform the following steps.

a) Run your computer code using the first 9 partial sums, S_1, S_2, …, S_9 from the previous exercise and report how many correct decimal places for ln2 you have obtained.

b) Evaluated the absolute and relative approximation errors.

c) Estimate how many terms in a partial sum, S_n, you will need in order to achieve the same absolute error.

Exercise 7. Number series are ubiquitous in mathematics. Infinite series are also of paramount importance in physics, computer science, statistics, finance etc. We are mostly interested in convergent or summable series. By definition, a convergent series has a limit

$$\sum_{i=1}^{\infty} a_i \equiv \lim_{n\to\infty}\sum_{i=1}^{n} a_i$$

In practice, we find the sum of an infinite series by a partial sum (truncation)

$$\sum_{i=1}^{\infty} a_i \approx S_n \equiv \sum_{i=1}^{n} a_i$$

In doing so, a question arises how to choose n in order to guarantee

$$\left|\sum_{i=1}^{\infty} a_i - S_n\right| = \left|\sum_{i=n+1}^{\infty} a_i\right| \le \varepsilon$$

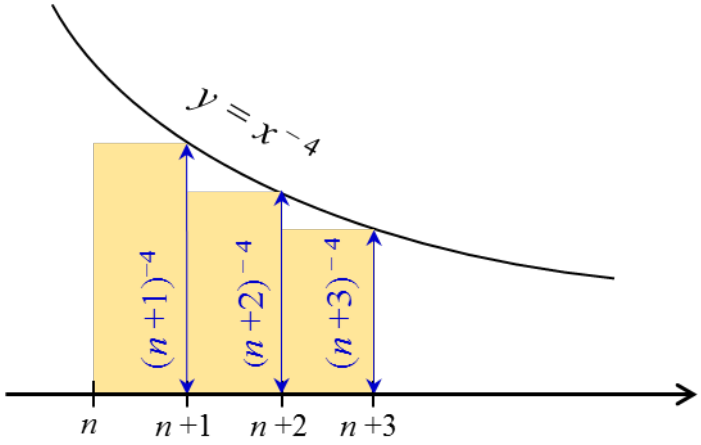
Here, ε is an a-priory-set tolerance ε. In case of alternating series, the answer is elegantly given by the first neglected term as was studied in Exercise 6. For the series of constant signs, the answer becomes problem dependent as is demonstrated bellow. Consider a well-known series.

$$\frac{\pi^4}{90} = 1+\frac{1}{2^4}+\frac{1}{3^4}+\frac{1}{4^4}+\cdots$$

A naïve approach is to take a partial sum, $S_n = \sum_{k=1}^{n} k^{-4}$, where n is chosen so that the next term, $(n+1)^{-4} < \varepsilon$ is less than the tolerance, ε. Strictly speaking, we must guarantee that all the neglected terms add up to less than ε.

$$(n+1)^{-4} + (n+2)^{-4} + \ldots = \sum_{k=n+1}^{\infty} k^{-4} < \varepsilon$$

Often, an infinite series with monotonically decreasing terms may be bounded from above (and bellow) by an infinite definite integral with suitably chosen integrand, $f(x)$, as demonstrated in the next figure where $f(x) = x^{-4}$.



Clearly,

$$\sum_{k=n+1}^{\infty} k^{-4} < \int_{n}^{\infty} x^{-4} dx = \frac{1}{3n^3}$$

Thus, it suffices to choose n such that

$$\sum_{k=n+1}^{\infty} k^{-4} < \int_{n}^{\infty} x^{-4} dx = \frac{1}{3n^3} < \varepsilon$$

Investigate how inaccurate is the naïve approach and how accurate is the suggested bound. To this end, perform the following steps setting the tolerance $\varepsilon = 10^{-6}$.

a) Estimate n in the naïve approach, i.e. find n such that $(n+1)^{-4} < \varepsilon$
b) Write a computer code that calculates the partial sum for the n found in a) and evaluate the absolute and relative approximation errors.
c) Estimate n for the suggested upper bound, calculate an improved approximation, S_n, and find the absolute and relative approximation errors.

Exercise 8. The summation of an explicit sequence (finite series) is a very common and important operation in numerical analysis. If many numbers of dissimilar magnitude are added in a finite precision arithmetic, many significant places in the accuracy of the sum may be lost. J.H. Wilkinson pointed out that the error bounds for floating-point addition are smallest, among all orderings, if the numbers are added in order of increasing magnitude. Verify this statement by summing a finite series

$$S_n = \frac{1}{1\cdot 2} + \frac{1}{2\cdot 3} + \frac{1}{3\cdot 4} + \ldots + \frac{1}{(n-1)n} = 1 - \frac{1}{n}$$

To raise your satisfaction, try to prove this formula but this is not a part of the exercise. Clearly, the "natural" summation is "anti-optimal". Write a computer code that performs this summation in direct (natural) and inverse (optimal) order, i.e.

$$S_n = \frac{1}{(n-1)n} + \frac{1}{(n-2)(n-1)} + \ldots + \frac{1}{2\cdot 3} + \frac{1}{1\cdot 2}$$

Select n to be a power of 2 that is close to 10^6. The following observation might be helpful in converting powers of 2 to powers of 10 and vice versa, $1024 = 2^{10} \approx 10^3$, which suggests, $2^{20} \approx 10^6$. An exact formula, which gives the same result, is given bellow

$$n = 2^m \approx 10^6 \longrightarrow m = \lceil 6\log_2 10 \rceil$$

Then the exact sum, $S_n = 1 - 1/n = 1 - 2^{-m}$, will be exactly represented in both the single and double precision which simplifies the comparison. Finally, do the following.

a) Calculate S_n in the direct order and evaluate the absolute error, which is almost exactly equal to the relative one.
b) Calculate again S_n in the inverse order and evaluate the absolute error.
c) Calculate once again S_n in the direct order but this time using the so called Kahan summation algorithm also known as the compensated summation. This is done by keeping a separate variable to accumulate small errors.

Direct summation	Inverse summation

`sn = 0;`
`for k = 1:n-1`
`  ak = 1.0/(k*(k+1));`
`  sn = sn + ak;`
`end`
	`sn = 0;`
	`for k = n-1:-1:1`
	`  ak = 1.0/(k*(k+1));`
	`  sn = sn + ak;`
	`end`

The Kahan summation algorithm may be written as

```
sn = 0;
c  = 0;
for k = 1:n-1
   ak = 1.0/(k*(k+1));
   y  = ak - c;
   t  = sn + y;
   c  = (t - sn) - y;
   sn = t;
end
```