Numerical Calculation

Beniamin Bogosel

March 10, 2026

Beniamin BOGOSEL

Faculty of Exact Sciences,
Aurel Vlaicu University of Arad,
2 Elena Drăgoi Street,
Arad, Romania
beniamin.bogosel@uav.ro
https://beniamin-bogosel.github.io/

1 Introduction

Numerical computations involve using mathematical software for evaluating complex mathematical expressions. These computations are useful from an application point of view. In this course we will explore basic elements regarding numerical computations and numerical analysis in general. We will work with existing software that can perform such computations and we will learn about the numerical methods involved and possible issues that need to be handled correctly (error analysis, roundoff errors, etc).

Motivation. The usage of numerical analysis software is ...

Example 1.1. a) Evaluating mathematical constants or non-linear functions for quantities for which the values are not explicit:

$$\sin 1, \exp(2), \log 2, \cos 3, \pi, \sqrt{3}.$$

Generally, any irrational number has a non-periodic decimal form, therefore any result computed and shown by a numerical software is **an approximation**. An approximation implies an **error** that needs to be understood and estimated, in order to have a **reliable computation**.

b) Computing integrals:

$$\int_a^b e^{x^2} dx, \int_a^b f(x) dx.$$

Unless a primitive is known for the function in the integral, such calculations are not explicit. An application for computing integrals is **computations of areas and volumes**.

c) Solving equations: in general, equations cannot be solved analytically as soon as we have something more complex than first or second degree equations.

- $3x + 4 = 0$ (explicit solution available)

- $x^2 - 3x + 2 = 0$ (formulas available)
- $x^7 + x^3 + 3x - 2 = 0$ (polynomial equation: general formulas not available for degree higher than 5)
- $e^x + \sin x - 3 = 0$ (non-linear equation: when combining two or more elementary functions solutions cannot be found analytically).

d) Systems of equations:

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix}$$

The 2×2 case can be solved by hand, but in applications we can easily reach large linear systems. Physical applications where discretizations of PDE models are involved can easily reach hundreds of thousands or even millions of unknowns. Using software and algorithms for such systems is a must.

The goal of this course is solving math problems and performing mathematical computations using numerical software. Here are some examples:

- **Mathcad**: will be used in this course and in the lab. It provides a "what you see is what you get" environment. Computations and text can be intercalated in order to produce a nicely formatted document. This software requires licensing.
- **Python**+modules like NumPy, SciPy, etc. This software is more "programming oriented". One writes code in order to perform numerical computations. It is possible to use **Jupyter Notebooks** to make results more "readable" intercalating formatted text cells with code cells. Python is freely available.
- **Pari-GP**: a free, programming language. Allows to easily perform calculations, symbolic computing and numerical computations.

Evaluating expressions: The first and most elementary type of numerical computation is the evaluation of expressions involving mathematical constants and elementary functions.

$$e^2 + \pi - \sin\frac{\pi}{9} + \arctan 5 + ... = \text{ some value.}$$

Remember that **every such computation is an APPROXIMATION**.

For example:

$$\pi = 3.141592...$$

gives an approximation of π using 6 decimal digits after the decimal point. Since π is an irrational number, any computation, being finite, is approximate. The goal is to **be as precise as possible** and to **minimize the error**.

Symbolic computations. It is possible to perform symbolic mathematical computations using software, mimicking the usual algebraic rules:

$$(a+b)^2 = a^2 + 2ab + b^2, (a+b)^4 = a^4 + 4ba^3 + 6b^2a^2 + 4b^3a + b^4.$$

It is also possible to do exact computations using **rational numbers**. However, when we do multiple computations the **denominators become very big** as the following computations show:

$$\begin{aligned}
1+\frac{1}{2}+\frac{1}{3}+...+\frac{1}{10} &= \frac{7381}{2520}\\
1+\frac{1}{2}+\frac{1}{3}+...+\frac{1}{20} &= \frac{55835135}{15519504}\\
1+\frac{1}{2}+\frac{1}{3}+...+\frac{1}{100} &= \frac{14466636279520351160221518043104131447711}{2788815009188499086581352357412492142272}
\end{aligned} \tag{1.1}$$

Integer types are usually defined as follows:

- n bits are used to store the integer
- one bit is used for the sign
- the remaining $n-1$ bits are used to express numbers ranging from

$$-2^{n-1} \text{ to } 2^{n-1}-1.$$

We arrive at the following limits for 32 and 64 bits:

nb of bits	lower bound	upper bound
32	-2147483648	2147483647
64	-9223372036854775808	9223372036854775807

Since all variables used need to be **stored in the computer memory** we can quickly arrive at the limits of available memory when using exact arithmetic, especially if operations with vectors or matrices are involved.

Working with a fixed number of digits before and after the decimal point is not a practical solution. One can easily imagine adding or multiplying two such numbers obtaining a bigger number which does not fit into the desired format.

The solution adopted in all main computational standard is the use of floating point precision. Main details about the `float32` and `double` data types found in most programming languages can be found at the links below:

- `float32`: single precision floating point
- `double`: double precision floating point

Brefly the situation is as follows:

- the numbers are stored using N bits (containing 0 or 1)
- one bit is used for fixing the sign
- a number d of the remaining bits is used for the exponent, encoded as an *unsigned integer* defined using the d bits. Subtract $2^{d-1}-1$ to include negative exponents clustered around 0.
- the remaining $N-d-1$ bits are used to construct the **significant digits** of the fractional part of the number consider.

Example: Consider the `float32` number defined by the bits:

$$0\ \ 10110110\ \ 01100011011101011010101.$$

- The sign: $(-1)^{b_{31}}$, that is (-1) to the power in the leading bit, plus here
- The exponent: 10110110. Convert to int and subtract 127: obtaining 55
- The fraction: $1.01100011011101011010101$. After the decimal point we have the bits contained in the fraction. Thus we obtain 1.3885141611099243

	N bits	d bits for exponent	$N-d-1$ bits for fraction
float32	32	8	23
double	64	11	52

Table 1: Structure of floating point numbers for float32 and double formats.

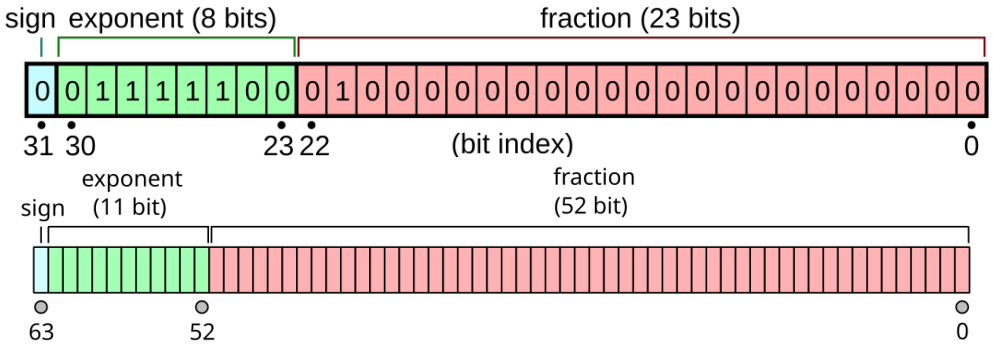

Figure 1: Structure of the `float32` and `double` floating point numbers

- The number stored using the bits above is

$$2^{55} \cdot 1.3885141611099243 \approx 50026494868586495.40.$$

Advantages of working with floating point numbers:

- flexibility: we can represent **small** and **large numbers**
- uniform memory consumption given by the number of bits
- the `double` format is useful for many practical applications.
- multiplication can be done without losing much precision: using the exponent helps

Disadvantages of working with floating point numbers:

- **roundoff errors** may occur when performing various operations.
- simple operations may lead to large **relative errors**. Example: adding numbers having different magnitudes.

Exercises.

1. Following the example involving 32 and 64 bit integer types, what is the required number of bits to store integers in the computation (1.1).

2. Consider a floating point system using 16 bits: one for the sign, 4 for the exponent and the remaining bits for the fractional part. Find the smallest and largest positive real numbers that can be represented in this system.

3. Open the programming language of your choice and perform the computation $1 + 10^{-16}$. Comment on the result.

4. **Machine epsilon**. Find the smallest number $\varepsilon > 0$ such that $(1 + \varepsilon) > 1$. This number is called the machine epsilon. It is about 10^{-7} in `float32` and around $2 \cdot 10^{-16}$ in `double`. Write code that can find this number in your favorite programming language.

5. Can the number $1/10$ be represented exactly using a binary representation?

2 Roundoff errors: should we care?

Patriot missile failure. Check this link for details.

The Patriot Missile is an aerian defense system: it detects incoming missiles and destroys them. In the initial design engineers decided to measure time using tenths of a second. For this the internal clock, measuring the time multiplying $1/10$ (represented in a 24 bit fixed point reister) with an integer number (the time counter). The binary expansion of $1/10$ is

$$0.00011001100110011001100110011001100110...$$

which does not terminate. The 24 bit register stored

$$0.00011001100110011001100$$

instead, introducing an error of

$$0.00000000000000000000000011001100_{(2)} \approx 0.000000095,$$

small you'd say... After 100 hours the number of tenths of a second multiplied with the number aboge produces an error of

$$0.000000095100606010 = 0.34 \text{ seconds}.$$

A Scud missile travels about 1676 meters per second so it travels half a kilometer in this time. The Patriot was therefore unable to track the missle correctly and was destroyed.

A programming inconsistency produced a significant real life problem. Errors in computations should **not be taken lignthly**.

S. Rump experiment. Consider the following function:

$$f(x,y) = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y). \tag{2.1}$$

Evaluate it at the point $(\tilde{x}, \tilde{y}) = (77617, 33096)$ and interpret the results.

Try this experiment with different data types in different software and decide which result is correct.

Answer: In Mathcad the result is $-1.1806e + 21$. In Octave we obtain the same result. Other programming languages where `float32` or `double` formats are used will provide wrong answers. **Why?**

Let's analyze the structure of the function for the given input. We have two terms: $T_1 = 5.5\tilde{y}^8$ and $T_2 = 333.75\tilde{y}^6 + \tilde{x}^2(11\tilde{x}^2\tilde{y}^2 - \tilde{y}^6 - 121\tilde{y}^4 - 2)$. These terms have large modulus and almost cancel:

$$T_1 = +791711134066896136110113470152494284 8$$
$$T_2 = -791711134066896136110113470152494285 0$$

Therefore $T_1 + T_2 = -2$ meaning that

$$f(\tilde{x}, \tilde{y}) = T_1 + T_2 + \tilde{x}/(2\tilde{y}) = -2 + \tilde{x}/(2\tilde{y}) \approx -0.8273960599.$$

When performing the addition $T_1 + T_2$ using floating point formats which do not have enough working precision, information is lost, leading to wrong results.

The resulting **practical problem** is the fact that **no warning is given to the user** concerning the potential error in the result. A user used to trust the output of a calculator would not question the result and therefore will end up using a wrong result.

Such behavior lead to the development of a modern branch of mathematics and computer science related to **validated computing** where every possible source of error, including **roundoff errors**, is taken into account.

Roundoff errors. In numerical computing a roundoff error is the difference beween the result produced by an exact algorithm and an algorithm using finite precision. These errors are due to the fact that real numbers are not represented exactly and mathematical operation made with floating point numbers leads to errors. Therefore, one of the goals in numerical analysis is to estimate and quantify errors made in the computations.

Example 2.1. Try the following computations in the software of your choice:

- $10^{20} - (10^{20} - 1)$
- $10^{20} - (10^{20} - 10^{10})$
- $10^{20} - (10^{20} - 10^{k})$, $k \in \{0, 1, ..., 16\}$.

What are the exact results? Compute the errors made in these computations.

Absolute errors, relative errors. When performing arithmetic operations we have small errors, which can be classified in at least two ways. Let $\overline{x}$ be the result of a computation of an expression x.

- **Absolute error**: $|\overline{x} - x|$, the euclidean distance between the computed number and the actual number.
- **Relative error**: $|\overline{x} - x|/|x|$, the absolute error divided by the magnitude of the number to be computed.

In practice, the most relevant measure of error is the relative error since it measures the precision attained in comparison with the magnitude of the number. This reduces biases of the form: *I work with small numbers, therefore I obtain small errors*.

Exercises.

1. Write a program which computes $n! = 1 \cdot 2 \cdot ... \cdot n$ for a given positive integer n. Use your program to compute the first 40 factorials. Do you notice anything strange? Try to explain what is happening.

2. Write a number which finds the smallest representable number for a given type and the machine epsilon. What are the values you get?

3. (a) Find a double precision floating point number $x \in (1, 2)$ such that $x \otimes \frac{1}{x} \neq 1$.

   (b) Find the smallest such number.

4. Define the function
$$f(x, y) = 9x^4 - y^4 - 2y^2.$$
   a) Compute $f(40545, 70226)$. Write a program which computes the desired value. Decide if the result computed is correct or not. What is the exact value? Explain the different values obtained using the different types used.

   b) Find an equivalent re-writing of the expression $f(x, y)$ to diminish the errors.

5. Compute the absolute and relative errors for the numerical evaluation of the computations in Example 2.1, comparing the numerical result with the analytical one.

6. Consider the sequence $u_0 = \frac{1}{10}$, $u_{n+1} = 11u_n - 1$. Give an analytical formula for u_n for any n. Evaluate the first 50 values of this sequence in `float32` and `float64`. Explain the observed phenomenon. What is the precision lost at every iteration?

7. Compute the expressions proposed at the indicated value. Propose a rewriting that dminishes the error.

   (a) $\sqrt{x^4 + 4} - 2$ around $x = 0$.

   (b) $e^x - e$ around $x = 1$

   (c) $\ln x - 1$ around $x = e$

   (d) The roots of the polynomial $x^2 + 10^{15}x + 1$

3 Evaluating expressions, symbolic computations

The usage of numerical software facilitates computation of complex mathematical expressions, not feasible by hand. In the following we will perform various computations in the software of your choice.

Exercise 1. Evaluate the expressions:

$$1.(4) + 2.(43) + 3.0(54) + 4.00(38) + 8.24(353) + 9.0283(3324)$$

$$\cfrac{1}{2+\cfrac{3}{4+\cfrac{5}{6+\cfrac{7}{8+\frac{9}{10}}}}}$$

$$\sqrt{1+\sqrt{2+\sqrt{3}}}+\sqrt[3]{1+\sqrt[3]{2}+\sqrt[3]{3}}+\sqrt[4]{1+\sqrt[4]{2}+\sqrt[4]{3}}$$

$$\sqrt{1+\sqrt[3]{2+\sqrt[4]{3+\sqrt[5]{4+\sqrt[6]{5+\sqrt[7]{6}}}}}}$$

$$\ln(\ln(11)) + \log_7(\log_5(13)) + \ln(\log_{11}(17)) + \log_{13}(\ln(19))$$

$$1101010101.101011_{(2)} + 34527.352_{(8)} + acef_{(16)}$$

$$e^{\pi} + \pi^e + \sqrt{2}^{\sqrt[3]{3}} + \sqrt[3]{3}^{\sqrt{2}} + \ln(2)^{\frac{1+\sqrt{3}}{2}}.$$

$$\frac{1}{3!} + \frac{1+\sqrt{5}}{4!} + \frac{\log_9(7!)}{6!} + \frac{\pi^{\sqrt{5}}}{\cos(8!)} + \cos(e^7).$$

Exercise 2. Consider the sequence $S_n = 1 + \frac{1}{2} + ... + \frac{1}{n}$ and define $\gamma_n = S_n - \ln(n)$. Find numerically the approximation of γ_n for $n = 10^6$. Considering γ to be the numerical approximation of γ_n computed previously, and $\phi = (1+\sqrt{5})/2$, the golden ratio, find which one of the number is largest:

$$\left(\frac{11}{3} + \sqrt{\frac{11}{3}}\right)^{\sqrt[65]{53^{22}}}, \pi^{e^2}, (\gamma + \phi + 1/\phi)^{33\sqrt{3}/7}.$$

In previous examples the computer performs floating point operations to obtain approximations of the needed results. In the following let us use symbolic computations to evaluate expressions.

Exercise 3. In the software of your choice (Mathcad, Pari-GP) perform the following symbolic computations:

- evaluate operations with rational numbers (addition, multiplication)
- factorize integers
- expand or factor algebraic expressions

Note: In Mathcad we can use the keyword `Ctrl + .` to perform symbolic evaluation. On the symbolic evaluation arrow we write the keyword needed to indicate the operation performed (factor, expand). In Pari GP computations are made in symbolic mode whenever rational or symbolic variables are used.

Remark 3.1. Numerical software can do impressive symbolic computations. Nevertheless, there are cases where symbolic solutions are not computable in closed form. In these cases approximations in floating point precision are used.

4 Solving equations

Exercise 4. a) Linear equations: Solve $ax + b = 0$ using symbolic computations.
b) Quadratic equations: Solve $ax^2 + bx + c = 0$ using symbolic computations.
c) Cubic equations: Solve $ax^3 + bx^2 + cx + d = 0$ using symbolic computations.
d) Solve fourth order equations.
e) What can we say about higher order equations?

Exercise 5. Consider the polynomial $f(x) = x^3 - 3x + 2$. Solve the equation $f(x) = 0$ using:

- the `solve` command
- the `polyroots` command

Search the documentation to see what is the difference between these two commands.

Exercise 6. Consider the nonlinear equation $x = \cos x$.
a) Does this equation have a unique solution in $\mathbb{R}$?
b) Find this solution numerically using `root` in MathCad. Choose various starting points and observe the behavior.