A brief introduction to LaTeXdocuments

Laurits Nikolaj Stokholm*

March 11, 2021

Abstract

This short paper was written as part of the course *Practical Programming and Numerical Methods*. Herein, I will introduce the exponential function and show an implementation of it using the programming language of C. The result is in fine agreement with both the exponential function as defined in the standard mathematics package and in the gnuplot tool. Evenmore, we show some basic capabilities of the LaTeXsystem such as the mathematical cross references, sections and including plots generated by the gnuplot software. The project is handled by the make utility tool, and a link to the repository can be found at `https://github.com/LauritsStokholm/ppnm`.

1 A Bird's eye view

Analysis of real-variables

From the beginning, the scalar exponential function $\exp : t \to exp(a \cdot t)$ draws much of its significance from two very peculiar properties it enjoys. For one thing, it satisfies the functional equation

$$f(t+s) = f(t) \cdot f(s). \tag{1}$$

which, in the view of algebraic mappings between groups, gives the group homomorphism from $(\mathbb{R}, +)$ to $(\mathbb{R}_{>0}, \cdot)$, where $\mathbb{R}_{>0} = \{x \in \mathbb{R} \| x > 0\}$. Without the exponential function, this fact would be hidden.

On the other hand, the exponential function satisfies the differential equation

$$\frac{df}{dt} = a \cdot f(t) \tag{2}$$

eq. (1) is the analogue to the slide rule of logarithms as first used by John Napier**john_napier_wiki**. eq. (2) appears to be of a quite different nature. Historically developed to expand on the idea that the growth rate of an amount of money under the influence of continuously calculated compound interest to be directly proportional at any time to the amount attained at that time. Leonhard Euler put eq. (1) and eq. (2) into a coherent context showing the series

$$\sum_{n=0}^{\infty} \frac{t^n}{n!} = \lim_{n\to\infty} \left(1 + \frac{t}{n}\right)^n = \mathrm{e}^t \tag{3}$$

where we differentiate between the function exp and the irrational number e. eq. (3) is what extends the exponential function to almost any other field of mathematics. It is the basis of Eulers identity, which easily extends the exponential function to complex variables; and in fact to operator calculus and functional analysis. Irrefutably vital without exception for the

entire field of mathematics and thus also of the natural sciences.

2 Numerical Implementation

In this section a numerical implemtation of the exponential function will be described. Here, we take ground in the series expansion of eq. (3). A naive method would be to hard code a finite degree polynomial estimation of the series expansion. This would lead to the following downsides and considerations:

- Each term varies in size exponentially and thus as the total sum is computed in sequence from left to right, a numerical error is introduced.
- A large number of functional operations will surely be called, unless an explicit consideration to this is written. For instance, instead of utilising the power function for each term, one can reduce this number by isolating a common factor out.
- Even further, using the quicker multiplication operator instead of the power function, compilation time will be lowered.
- Since the series expansion will be estimated by a finite sum, then for large values of the argument, the series approximation will diverge from the exact series expansion.
- In case of a negative argument, every uneven term will be negative, and and thus another source of error is introduced by adding positive and negative terms.

Luckily, these considerations are not too tedious. There exists a “quick-and-dirty" method to estimate the series expansion eq. (3); giving a numerical implementation of the exponential function. This is given in fig. 1. First of all, the terms are monotonically increasing, so the addition is less error prone. Secondly, there is defined a handler for relatively large values of the argument, such that the series expansion is always for small arguments. Lastly, for negative values the addition is summed using strictly positive terms. eq. (3).

```
double my_exp(double x)
{
if (x<0) return 1/my_exp(-x);
if (x>1./8) return pow(my_exp(x/2),2);
return 1+x*\
(1+x/2*(1+x/3*(1+x/4*(1+x/5*\
(1+x/6*(1+x/7*(1+x/8*(1+x/9*\
(1+x/10)))))))));
}
```

Figure 1: Explicit code for implementation of the exponential function.

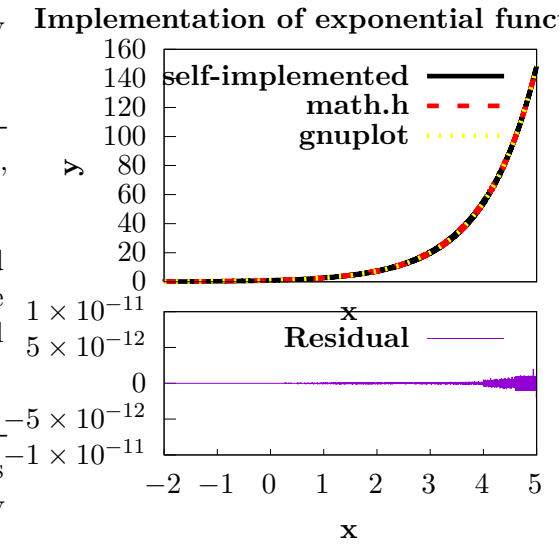

Figure 2: A comparison of the exponential function defined by the header *math.h*, in the plotting tool *gnuplot* and as defined in this paper implementation.

3 Conclusion

The residuals of this quick and dirty implementation compared to the exponential function given in the mathematical header *math.h* is of the order of 10^{-12} which compared to the precision of a double (15 decimals) is very good. The cutoff at the 10th term was arbitrary, and one could of course consider a higher degree polynomial estimation. All in all, we can conclude a successfull implementation of the exponential function.