The Exponential Function

Albert Freud

March 5, 2021

Abstract

A 'quick and dirty' implementation of the exponential function, and a study in LaTeX.

1 The exponential function and its properties

The exponential function is defined as:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \tag{1}$$

This can be seen in figure 1:

One of the most important properties of the exponential function is that it is it's own derivative. This is obvious from explicitly differentiating the series representation from equation 1:

$$\frac{\partial}{\partial x} e^x = \sum_{n=0}^{\infty} \frac{\partial}{\partial x} \frac{x^n}{n!} = \sum_{n=1}^{\infty} n \frac{x^{n-1}}{n!} = \sum_{n=1}^{\infty} \frac{x^{n-1}}{(n-1)!} = \sum_{n=0}^{\infty} \frac{x^n}{n!} = e^x \tag{2}$$

2 Numerical implementation

The series representation is also convenient for numerical implementation, as integer powers are easily computed numerically. The natural choice of implementation would then look something like:

```
double ex(double x){
return 1+x+pow(x,2)/2+pow(x,3)/3+pow(x,4)/4+pow(x,5)/5...
}
```

The dots represent not an infinite number of terms, but only a finite e.g. 10. However, this approach is bad for several reasons:

1. We add terms off very different sizes since the sum is computed from left to right, resulting in large numerical error.

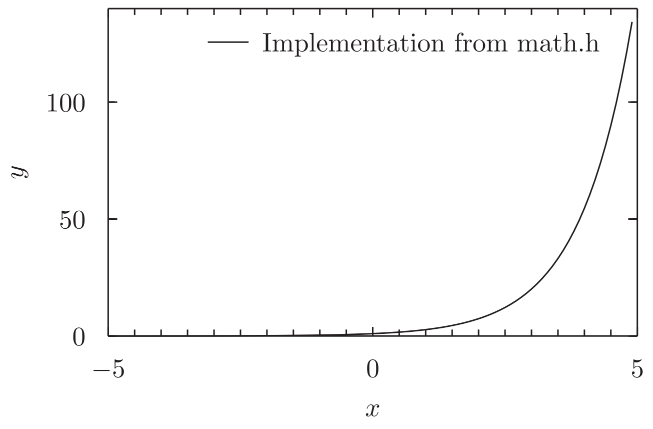

Figure 1: A plot of the exponential function using 'exp' from math.h

2. We perform many operations! For each power of x, we use several multiplations.
3. For large arguments, the convergence of the taylor expansion is not very good resulting in large errors.
4. For negative arguments we add positive and negative terms, which is also very error prone.

Hence, a different approach is recommended:

```
double ex(double x){
if(x<0) return 1/ex(-x);
if(x> 1/.8) return pow(ex(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+...
}
```

This solves all our problems: the sum is calculated from within, meaning that the smallest numbers are added first, the number of operations is reduced, large arguments are handled recursively so that the taylor expansion is only used for small arguments and negative arguments are computed using only strictly positive sums.

3 Test of our implementation

To test our implementation, we plot results against the implementation from the standard C-library. The result can be seen isolated in figure 3 and the comparison is found in figure 3.

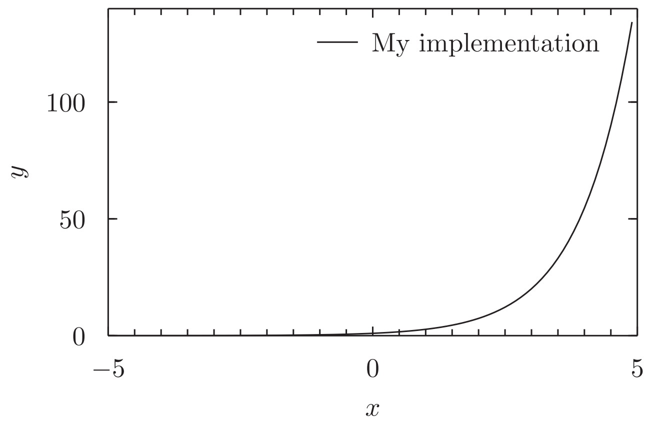

Figure 2: A plot of the exponential function using 'exp' from math.h

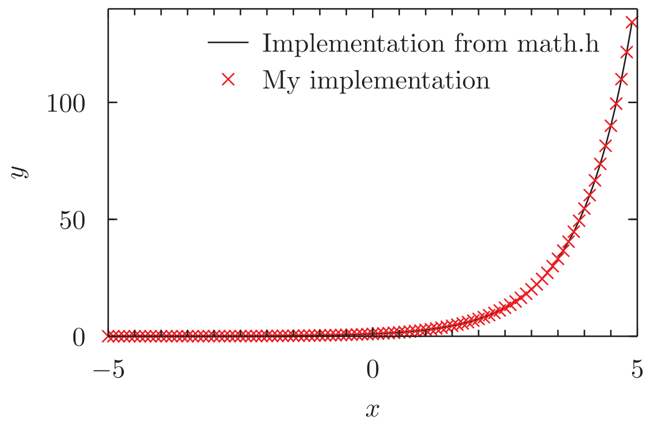

Figure 3: A plot of the exponential function using 'exp' from math.h