MMC laboratory01

Terman Emil FAF161

September 24, 2017

1 1. Hyperbolic Tangent

After implementing this exercise, I noticed that for big numbers, the other 2 given formulas, fail, even though the result is almost 1. That's because, exp(n) or sinh(n) or cosh(n), for $>$ 700 (about so), are huge numbers - so huge, they don't fit in their given memory.
To resolve this problem, I use the lambert formula:
$tanh(x) = \frac{x}{1+\frac{x^2}{3+\frac{x^2}{5+...}}}$
So that I don't have to depend on huge numbers. To make the results more precises, I store the numbers as fraction. In this way, the results are very exact, but the it takes considerably more time to compute.
To emphasize the exactness of fractions, I also used floating points. As a result, I recive a tiny, but yet visible error.
So, if I'd build a program for some rocket science, I would be aware that some numbers, like 0.1, even though it seems a small number, it's actually an periodic sequence in binary representation, and since we can't store infinite sequences - we have to chop the value, making that little error that led to "The Explosion of the Ariane 5"

2 2. Some Code

In computer, not all real numbers can be stored as floating point numbers. That's because, there are some numbers that their binary representation may be too big to fit in the given memory for floats, in which case - the number is either choped or rounded, producing an error. An example of such a thing is 0.1 whith its binary representation: 0.0001100110011... Well, ofcourse the error is not big, unless it's multiplied to a big number, let's say 1000 or 10000. To resolve such cases, we could use Fractions or Fixed point numbers.

> Fractions: instead of storing 0.1, we can store it as 1/10, in which case we avoid the choping or rounding of the number.

> Fixed point numbers: are less prcise, but much more consistent, because the numbers are stored inside an integer.

Conclusion: never use floats for additive operations (float += float) on big scales

3 3. Integrals

Deomnstration:

1) $K=0$

$$\int_0^1 e^{-x}x^0\,dx = -\int_0^1 e^{-x}\,d(-x) = -e^{-x}\Big|_0^1 =$$

$$= -e^{-1} + 1 = 1 - e^{-1}$$

2) Let's check for $K=2$.

$$\int_0^1 e^{-x}x^2\,dx = -\int_0^1 x^2\,de^{-x} = -x^2e^{-x}\Big|_0^1 + 2\int_0^1 e^{-x}x\,dx =$$

$$= -e^{-1} - 2\int_0^1 x\,de^{-x} = -e^{-1} - 2xe^{-x}\Big|_0^1 + 2\int_0^1 e^{-x}\,dx =$$

$$= -e^{-1} - 2e^{-1} + 2 - 2e^{-1} = 2 - 5e^{-1} \checkmark$$

To avoid any error depending on additions or multiplications, I calculated the result in integers, then I substitued e with its value.

My program's results:

k) $< builtinintegral >:< recursive(float) > (< recursive(explicit) >)$

- 0) 0.632121 : 0.632121 (1 - 1/e)
- 1) 0.264241 : 0.264241 (1 - 2/e)
- 2) 0.160603 : 0.160603 (2 - 5/e)
  ...
- 11) 0.033195 : 0.033195 (39916800 - 108505112/e)
- 12) 0.030463 : 0.030463 (479001600 - 1302061345/e)
- 13) 0.028145 : 0.028145 (6227020800 - 16926797486/e)
- 14) 0.026154 : 0.026154 (87178291200 - 236975164805/e)
- 15) 0.024424 : 0.024414 (1307674368000 - 3554627472076/e)

- 16) 0.022909 : 0.023438 (20922789888000 - 56874039553217/e)
- 17) 0.021570 : 0.000000 (355687428096000 - 966858672404690/e)
- 18) 0.020378 : 0.000000 (6402373705728000 - 17403456103284421/e)
- 19) 0.019311 : 0.000000 (121645100408832000 - 330665665962404000/e)
- 20) 0.018350 : 0.000000 (2432902008176640000 - 6613313319248080001/e)
- 21) 0.017480 : 0.000000 (51090942171709440000 - 138879579704209680022/e)
- 22) 0.016689 : 0.000000 (1124000727777607680000 - 3055350753492612960485/e)
- 23) 0.015966 : 0.000000 (25852016738884976640000 - 70273067330330098091156/e)
- 24) 0.015303 : 0.000000 (620448401733239439360000 - 1686553615927922354187745/e)
- 25) 0.014693 : -2147483648.000000 (15511210043330985984000000 - 42163840398198058854693626/e)

Let's say for a given k, the result is of the form: a - b/e

- As k grows, so does a and b. When k reaches 25, b becomes a big number, Taking in consideration that e is irrational, meaning that its value is choped, then it means that it generates a certain error. Multiplied that with 42163840398198058854693626, the error becomes much bigger, therefore, creating erroneous values.

- When two numbers are nearly equal and we subtract them, then we suffer a Loss of Significance error in the calculation. It usually happens when one gets too few significant digits in substraction of two numbers very close to each other.

Let's say we have the number:

1. 1.2345678

and another number, very close to the first one:

2. 1.2344444

When the difference is calculated, we recive:

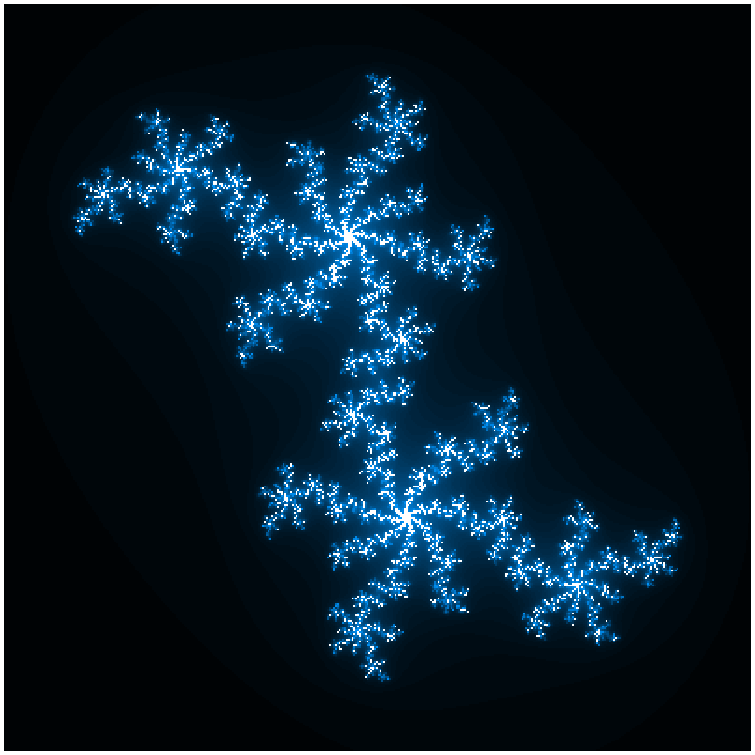
3. 0.0001234

The first 2 numbers have 8 significant digits and the substraction of these numbers lead to a number with 4 significant digits, making a **Loss of Significance**. The first 2 numbers have a realtive round off error of about 10^{-8}, but the result has a round off error of 10^{-4}, which is a much bigger error.

In the case with the integral, we experience a clear loss of significance starting from k = 17. The difference between a and b/e, apparently has too few significant digits exceeding the limit of representable numbers, so the computer rounds the result to 0.

For k = 25, the result would've been 0 aswell, if not for the error of e.

4 4. Fractal



Using the Julia Sets, we can create a very beautiful, infinite and chaotic image.

This formula can describe the chaos: because it's infinite, we could say that everything has a fractal form.

The branching of tracheal tubes, the leaves in trees, the veins in a hand, tiny oxygene molecule, or the DNA molecule - all these can be described as fractals.

Some people say that the spreading of the universe is fractal. Others try to predict the stock market using fractals.

In conclusion: Fractals are not just some pretty pictures, but a whole new way of thinking about the universe

Check out the gifs