My bs notes

Makus

November 13, 2023

Contents

1 Introduction

just random ass notes, separted by sections.

1.1 forenotes

for the entire part of Computer Science, Chapter 3, the variable naming standard will be as follows:

i is input
o is output

unlesss stated otherwise.

2 math

2.1 general

2.1.1 bases

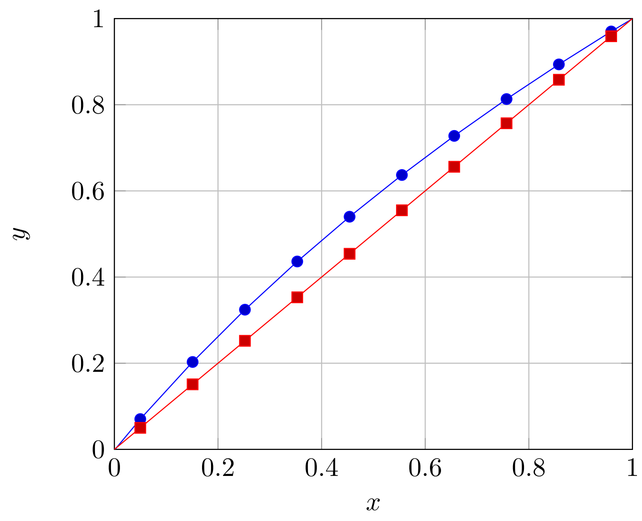
given

$$n = \text{base of number representation}$$
$$x = \text{number that is represented in base } n,$$

Then $\frac{x}{n}$ shiftes everything in x towards the left by one and $x \cdot n$ shiftes everything to the right.

2.1.2 logrithmic base 2

when taking $\log_2(1+x)$ where $0 < x \leq 1$, the output is approx. x.



but by adding μ to this, we can lower the amount of error to the average. to find μ, we can do a mean value theorem stated in calculus portion to find μ, which can be gotten from

$$\text{if} f(x) = \log_2(x+1), \text{ then}$$

$$f'(x) = \frac{1}{(x+1)\ln(2)}$$

$$\text{since the average rate of change is}$$

$$\frac{f(1) - f(0)}{1 - 0} = 1$$

then to get μ, the steps are

$$f'(c) = \frac{1}{(c+1)\ln(2)} = 1$$

$$1 = (c+1)\ln(2)$$

and

2.2 calculus

2.2.1 Newton's iteration

2.2.2 Mean Value Theorem

Mean Value Theorem, or MVT, states that if f(x) is continuous and differentiable within the range $a \leq x \leq b$, then there exists a point c where $f'(c) = \frac{f(b)-f(a)}{b-a}$

3 Computer Science

will be using c++ for every example

3.1 bit shit

3.1.1 bitwise operators to stop forgetting

& is and
| is or
$\wedge$ is xor
$\sim$ for not
>> or << for moving bits left and right respectively

3.1.1.1 bitwise operator quirks?

In C++, every number is represented in binary, or base 2. Thus if anyone were to divide a number by two, then

```
x=x>>1;//shifting towards the right
```

works the same, but is faster. it also rounds up. Conversely, if anyone were to multiply by two, then

```
x=x<<1;//shifting towards the left
```

does the same.

3.1.2 IEEE 754

3.1.2.1 overview

This is how floats in C++ are stored. Floats are stored in 32 bit. the first bit is sign. The second to ninth bit is exponent, and the 10th to 32th bit is the mantissa. the exponent has an offset of 127, because it needs negative exponents. The mantissa, 23 bits, is the representation of the number(scientific notation). The equation to get from float to the final number is given by:

$$(1 + \frac{M}{2^{23}}) \cdot 2^{E-127} \tag{1}$$

3.1.2.2 nitty griddy

The layout of this(the float) is

sign bit:1 bit
exponent:1 byte(8 bits)
mantissa:23 bits

So if we were to follow this and try to extract exponents, we can do

```
o = ((0xff<<23)&i)-127;
```

and extracting the mantissa would be just

```
o = ((~(0xff<<23))&i);
```

assuming the variables i and o are unsigned longs.

3.2 optimizations

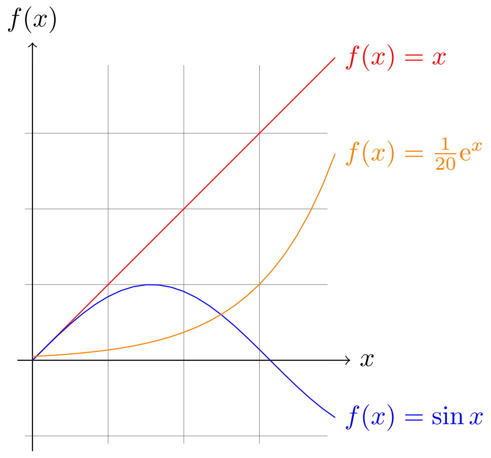
3.2.1 division

Since so much work has been done on addition subtraction and multiplication, the only real operation that needs change is division here, we will focus mostly on floats. A really easy way is to just

```
x=x/y;
```

but in order to optimize this, we will use

4 physics


$f(x)$
x
$f(x) = x$
$f(x) = \frac{1}{20}\mathrm{e}^x$
$f(x) = \sin x$