CMPS 251 - Assignment 11

Kinan Dak Al Bab

December 7, 2014

1 Range and Precision

1.1 Question

Consider a hypothetical machine representation with 64 bits where 1 bit is used for the sign, 58 bits are used for the mantissa, and the rest are used to store the (biased) exponent in a manner similar to the IEEE representation.

- What are the largest and smallest positive numbers representable?
- What the machine epsilon for this representation.

1.2 Answer

64 bits: 1 bit sign, 58 bits mantiessa, 5 bits.

5 bits means numbers from 0 to 31, so the bias is 15, (15 means 0, 14 means -1, 16 means 1, etc..).

The question says "in a manner similar to the IEEE representation", so I am assuming that the biggest exponent means infinity, and smallest means zero.

The largest number possible has the largest mantissa, i,e, 58 1's, and exponent 30 - 15 = 15. so the smallest number is 1. 58 ones with the point shifted 15 places to the right.
$(111111111111111.11111111111111111111111111111111111111111111)_2$
$(16\ ones.43\ ones)_2$
$around$ 65535.999999999999886
The smallest number possible has the smallest mantissa, i,e, 58 0's, and exponent 0 - 15 = -15. so the smallest number is 0. 14 zeros followed by 59 1's.
$(0.000000000000001111111111111111111111111111111111111111111111111111111111111)_2$
$(0\ .\ 14\ zeros59\ ones)_2$
$around\ 6.103515625000000e-005$

Machine epsilon is the smallest number that can be added to one and have the result representable, we have 58 digits in the mantissa so the smallest 'registered' tick we can add to one is 57's zeros followed by a one, in other words 2^{-58}:
$3.469447\ e-18$

2 Internal Representation

2.1 Question

Which familiar real numbers are approximated by oating point numbers that display the following values with format hex:

- 4059000000000000
- 3f847ae147ae147b

2.2 Answer

We have 16 hex digits, so the representations are in 64 bits.

First Number (in binary): 0100 0000 0101 1001 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000.
The sign bit is positive, the exponent is 100 0000 0101, which is equal to 1029 in decimal. minus the bias (1023), we get that the exponent is 6.
The mantissa has 1. 1001 followed by zeros. we shift the decimal point by six to the right and we add to it 1 on the left, we get 1100100 which is 100 in decimal. So the number is **100**.

Second Number (in binary): 0011 1111 1000 0100 0111 1010 1110 0001 0100 0111 1010 1110 0001 0100 0111 1011.
The sign bit is positive, the exponent is 011 1111 1000, which is equal to 1016 in decimal. minus the bias (1023), we get -7. so the point is shifted 7 places to the left.
The mantissa has 1. 0100 0111 1010 1110 0001 0100 0111 1010 1110 0001 0100 0111 1011 with the point shifted 7 places to the left, so it is 0.000 0001 0100 0111 1010 1110 0001 0100 0111 1010 1110 0001 0100 0111 1011. Since this number is quite horrible I used a matlab script to find it (Script is in the section bellow), and The number appeared to be **0.01**.

Indeed, to double check I changed the format of matlab to 'hex' and inputed the numbers I got, the printed representation were equal to those given to us.

2.3 Helper Script

```
final = 0;
x = [0 0 0 0 0 0 1 0 1 0 0 0 1 1 1 1 0 1 0 1 1 1 0 0 0 0 1 0 1 0 ...
     0 0 1 1 1 1 0 1 0 1 1 1 0 0 0 0 1 0 1 0 0 0 1 1 1 1 0 1 1];
for i = 1:length(x)
   final = final + x(i) * 2^(-i) ;
end
final % Print Result.
```

3 Digit Cancellation

3.1 Question

Consider the following computation $\sqrt{25+(10^{-4})^2}-5$

- Can it be performed reliably in single precision?
- How many correct decimal digits do we expect to get in double precision?
- How can we rewrite it to obtain more accuracy?

3.2 Answer

- According to wolframalpha.com $\sqrt{25+(10^{-4})^2} = \frac{\sqrt{(2500000001)}}{10000}$ Which is nearly $5+9\times10^{-10}$. Now using the formula learned in class, we can calculate how many digits we loose:

$$-log(1-\frac{b}{a}) = -log(1-\frac{5}{5+9\times10^{-10}} \approx 22.33$$

  So we are loosing nearly 23 binary digits, and notice that in single-precision we have only 23 binary digits in the mantesa, therefore it cannot be done reliably in single precision.

- In double precision we have 52 binary digits, and we are loosing 23 digits so we are left with 29 **binary** digits (more or less), roughly each 4 binary digit correspond to a decimal digit so we are left with 7 **decimal** digits we can trust.

  Calculating the values could have been done using first few terms of taylor series around 25 of square root, but this should be more accurate.

- We can re-write in the following way:

$$a-b = \frac{a^2-b^2}{a+b} = \frac{25+10^{-8}-25}{\sqrt{25+10^{-8}}+5}$$

$$= \frac{10^{-8}}{\sqrt{25+10^{-8}}+5}$$

  Indeed When we write the formula in the new form in Matlab we get a better answer (1.000000082740371 e - 009) Which is similar to the one given by wolframalpha, as opposed to a less accurate answer when we write it using the first form.

4 Floating Point Experiments

4.1 Question

In each of the following computations, something goes wrong when the calculations are performed with floating point representations. For every one of them:

1. experiment using Matlab;
2. briefly describe the problem;
3. describe available ways to avoid the problem, if possible.

4.2 First Experiment

a = 4/3
b = a - 1
c = 3 * b
e = 1 - c

Output : e = 2.220446049250313e-016

4/3 is not exactly representable in binary, thus we have a round-off error in a, this error was carried out to b and c and e, causing c to differ from 1 by a very small number (close to machine epsilon), which is why e was not exactly 0.
This can be avoided only by keeping the numbers as fractions (logical datatype) and carrying operations on them in that representation as opposed to machine representation.

4.3 Second Experiment

y = 1e-7;
z = sqrt(1+yˆ2) - 1;

Output: z = 4.884981308350689e-015

Subtracting two numbers that are very close to each other, using 2 terms of taylor series we approximate the value of the square root to around 1.000000000000001 (14 zeros), so using the log rule we find that we cannot trust the last 35 binary digits (last 9 decimal digits approximatly).
We can try to avoid it by writting it on the form $\frac{a^2-b^2}{a+b}$.

4.4 Third Experiment

x = 1.2e-8;
f = (1 - cos(x)) / x^2;
Output: f = 0.770988211545248, expected ouput is close to 0.5.

Subtracting two numbers that are very close to each other (1 - cos(x)), we can use taylor series to estimate the digits lost and all.

Possible way to calculate it better is to use its taylor series, so Let us try to do that with the first few terms of taylor series in Matlab:

x = 1.2e-8;
f = @(x) 1/2 - x^2 / 24 + x^4 / 720; feval(f, x)

ans = 0.500000000000000

4.5 Fourth Experiment

```
1  a = 1;
2  b = 1e9;
3  c = 1;
4  x1 = (-b + sqrt(b^2-4*a*c)) / (2*a);
5  x2 = (-b - sqrt(b^2-4*a*c)) / (2*a);
6
7  \noindent %% Test if it is correct, evaluate equation at x1, x2 ...
       should get 0
8  eq = @(x) a*x^2 + b*x + c;
9  feval(eq, x1)
10 feval(eq, x2)
```

ans = 1
ans = 1

b^2 is a very big number (in order of 10^+18), subtracting 4 from it has no effect (the representable 'tics' on the real line get sparser and sparser as we go towards bigger numbers).
To fix this maybe we can try to use the taylor expansion of the square root **around b^2** to calculate the value of the square root then use it within the formula to find x1 and x2.

4.6 Fifth and Last Experiment

x = 2;
for i = 1:50
x = sqrt(x);
end

for i = 1:50
x = x^2;
end

Output is around 1.6, expected 2 (or close to 2 at least).

The problem seems to be that x is getting smaller and smaller (as we square root it) eventually reaching near machine epsilon when taking is square root is yeileding weird behavior. when we eventually try to power it up back to 2 we would be starting with a wrong value leading us to finish with the far fetched 1.6 .

We have two ways to solve this, we can either try to amplify 2 by some factor before starting, this will make the x reach machine epsilon slower and might succed for 50 iterations (deciding the factor has to be reasoned about but it has to be big enough and at the same time not too big so that we dont lose precision).

The other way is to stop the loop when the difference between the next x and the previous one becomes very small.

```
x = 2 % Loop Break
for i = 1:50
    old_x = x;
    x = sqrt(x);
    if old_x - x < 10^-14
        break
    end
end
for j = i:-1:1
    x = x^2;
end

x1 = x

x = 2 * 100000000000; % Amplifying by a factor chosen by ...
    trial-and-error
for i = 1:50
    x = sqrt(x);
end
for i = 1:50
    x = x^2;
end

x2 = x /  100000000000
```

The new Output Becomes:
x1 = 1.988737457549722 x2 = 1.957296064957755