MATH-411 Numerical Analysis

1. Convert the following base 10 numbers to binary. Use overbar notation for nonterminating binary

(a) 10.3

Find the Binray equvlient of the whole Number

$$\left\lfloor \frac{10}{2} \right\rfloor = 5, \quad 10 \bmod 2 = 0$$

$$\left\lfloor \frac{5}{2} \right\rfloor = 2, \quad 5 \bmod 2 = 1$$

$$\left\lfloor \frac{2}{2} \right\rfloor = 1, \quad 2 \bmod 2 = 0$$

$$\left\lfloor \frac{1}{2} \right\rfloor = 0, \quad 1 \bmod 2 = 1$$

Find the Binary equvilenat of the decimal

$$0.3 * 2 = 0.6 \rightarrow \lfloor 0.6 \rfloor = 0$$

$$0.6 * 2 = 1.2 \rightarrow \lfloor 1.2 \rfloor = 1$$

$$0.2 * 2 = 0.4 \rightarrow \lfloor 0.4 \rfloor = 0$$

$$0.4 * 2 = 0.8 \rightarrow \lfloor 0.8 \rfloor = 0$$

$$0.8 * 2 = 1.6 \rightarrow \lfloor 1.6 \rfloor = 1$$

$$0.6 * 2 = 1.2 \rightarrow \lfloor 1.2 \rfloor = 1$$

So the Binary Value is

$$1010.01\overline{0011}_2$$

(b) 1/3

Find whole Binray equvlient of the whole number

$$\lfloor \frac{0}{2} \rfloor = 0, \quad 0 \bmod 2 = 0$$

Find the Binary equvilenat of the decimal

$$\frac{1}{3} * 2 = \frac{1}{3} \rightarrow \left\lfloor \frac{2}{3} \right\rfloor = 0$$

$$\frac{2}{3} * 2 = \frac{4}{3} \rightarrow \left\lfloor \frac{4}{3} \right\rfloor = 1$$

$\frac{1}{3} * 2 = \frac{1}{3} \rightarrow \lfloor\frac{2}{3}\rfloor = 0$

$\frac{2}{3} * 2 = \frac{4}{3} \rightarrow \lfloor\frac{4}{3}\rfloor = 1$

So the Binary Value is

$0.\overline{0101}_2$

(c) 12.8

Find the Binary equvilient of the whole number

$\left\lfloor\frac{12}{2}\right\rfloor = 6, \quad 12 \bmod 2 \quad = 0$

$\left\lfloor\frac{6}{2}\right\rfloor = 3, \quad 6 \bmod 2 = 0$

$\left\lfloor\frac{3}{2}\right\rfloor = 1, \quad 3 \bmod 2 = 1$

$\left\lfloor\frac{1}{2}\right\rfloor = 0, \quad 1 \bmod 2 = 1$

Find the Binary equvilenat of the decimal

$0.8 * 2 = 1.6 \rightarrow \lfloor 1.6 \rfloor = 1$

$0.6 * 2 = 1.2 \rightarrow \lfloor 1.2 \rfloor = 1$

$0.2 * 2 = 0.4 \rightarrow \lfloor 0.4 \rfloor = 0$

$0.4 * 2 = 0.8 \rightarrow \lfloor 0.8 \rfloor = 0$

$0.8 * 2 = 1.6 \rightarrow \lfloor 1.6 \rfloor = 1$

$0.6 * 2 = 1.2 \rightarrow \lfloor 1.2 \rfloor = 1$

So the Binary Value is

$1100.\overline{1100}_2$

2. Find the IEEE double precision representation fl(x), and find the exact difference fl(x)-x for the given real numbers. Check that the relative rounding error is no more than

$\epsilon_{mach} = 2^{-52} = 2.2204 * 10^{-16}$

(a) x = 2.75,

Find the Binray equvlient of the whole Number

$$\left\lfloor \frac{2}{2} \right\rfloor = 1, \quad 2 \bmod 2 = 0$$

$$\left\lfloor \frac{1}{2} \right\rfloor = 0, \quad 1 \bmod 2 = 1$$

Find the Binary equvilenat of the decimal

$$0.75 * 2 = 1.5 \rightarrow \lfloor 1.5 \rfloor = 1$$

$$0.5 * 2 = 1 \rightarrow \lfloor 1 \rfloor = 1$$

The Fixed point Binary is

$$10.1100_2$$

The double persion floating point Binary is

$$1.01100000 \quad 00000000 \quad 00000000 \quad 00000000 \quad 00000000 \quad 00000000 \quad 0000_2 * 2^1$$

Relative Error

$$\frac{|2.75 - fl(2.5)|}{2.75} = 0$$

(b) x = 2.7

Find the Binray equvlient of the whole Number

$$\left\lfloor \frac{2}{2} \right\rfloor = 1, \quad 2 \bmod 2 = 0$$

$$\left\lfloor \frac{1}{2} \right\rfloor = 0, \quad 1 \bmod 2 = 1$$

Find the Binary equvilenat of the decimal

$$0.7 * 2 = 1.4, \rightarrow \lfloor 1.4 \rfloor = 1$$

$$0.4 * 2 = 0.8 \rightarrow \lfloor 0.8 \rfloor = 0$$

$$0.8 * 2 = 1.6 \rightarrow \lfloor 1.6 \rfloor = 1$$

$$0.6 * 2 = 1.2 \rightarrow \lfloor 1.2 \rfloor = 1$$

$$0.2 * 2 = 0.4 \rightarrow \lfloor 0.4 \rfloor = 0$$

$$0.4 * 2 = 0.8 \rightarrow \lfloor 0.8 \rfloor = 0$$

$$0.8 * 2 = 1.6 \rightarrow \lfloor 1.6 \rfloor = 1$$

The Fixed point Binary is

$$10.10\overline{1100}_2$$

The double persion floating point Binary is

$$1.01011001 \quad 10011001 \quad 10011001 \quad 10011001 \quad 10011001 \quad 10011001 \quad 1010_2 * 2^1$$

Relative Error

$$\frac{|2.7 - f\ell(2.7)|}{2.7} = \frac{|10.10\overline{1100}_2 - 1.0101100110011001100110011001100110011001100110011010_2 * 2^1|}{10.10\overline{1100}_2}$$

$$\rightarrow \frac{10_2 * 2^{-51} - 1.\overline{1001}_2 * 2^{-51}}{2.7} = \frac{2 * 2^{51} - 1.6 * 2^{-51}}{2.7} = \frac{0.4}{2.7} * 2^{-51} = 6.5791 * 10^{-17}$$

(c) x = 10/3 or 3 and 1/3

Find the Binray equvlient of the whole Number

$$\left\lfloor \frac{3}{2} \right\rfloor = 1, \quad 3 \bmod 2 = 1$$

$$\left\lfloor \frac{1}{2} \right\rfloor = 0, \quad 1 \bmod 2 = 1$$

Find the Binary equvilenat of the decimal

$$\frac{1}{3} * 2 = \frac{1}{3} \rightarrow \left\lfloor \frac{2}{3} \right\rfloor = 0$$

$$\frac{2}{3} * 2 = \frac{4}{3} \rightarrow \left\lfloor \frac{4}{3} \right\rfloor = 1$$

$$\frac{1}{3} * 2 = \frac{1}{3} \rightarrow \left\lfloor \frac{2}{3} \right\rfloor = 0$$

$$\frac{2}{3} * 2 = \frac{4}{3} \rightarrow \left\lfloor \frac{4}{3} \right\rfloor = 1$$

The Fixed point Binary is

$$11.\overline{0101}_2$$

The double persion floating point Binary is

$$1.10101010 \quad 10101010 \quad 10101010 \quad 10101010 \quad 10101010 \quad 10101010 \quad 1011_2 * 2^1$$

Relative Error

$$\frac{|3.\overline{3} - f\ell(3.\overline{3})|}{3.\overline{3}} = \frac{|11.\overline{1011}_2 - 1.1010101010101010101010101010101010101010101010101011_2 * 2^1|}{11.\overline{1010}_2}$$

$$\rightarrow \frac{1_2 * 2^{-51} - 0.\overline{0101}_2 * 2^{-51}}{3.\overline{3}} = \frac{1 * 2^{-51} - 0.\overline{3} * 2^{-51}}{3.\overline{3}} = \frac{0.\overline{6}}{3.\overline{3}} * 2^{-51} = 8.8818 * 10^{-17}$$

3. A machine stores floating point numbers in 7-bit words. The first bit is stored for the sign of the number, the next three for the biased exponent and the next three for the magnitude of the mantissa. You are asked to represent 33.35 in the above word. The error you will get in this case would be

(A) underflow

(B) overflow

(C) NaN

(D) No error will be registered.

Explain why this is the case

Max value: $1.111_2 * 2^7 \rightarrow 1111000_2 \rightarrow 120_{10}$

D) no error will happen since it is not larger maxium value but due to chopping there will be a high relative error.

4. Consider a binary floating-point number system containing numbers of the form ±0.1d1d2 ×2 e , −4 ≤ e ≤ 6, where d1,d2 ∈ {0,1} are binary bits. Suppose that the system uses a conventional rounding to the nearest policy to convert a real number to its binary floating-point number and to do floating point arithmatic.

(a) What are the smallest and largest positive numbers (in decimal) in this floating point system?

Max: $1.11_2 * 2^6 = 1110000_2 = 112_{10}$

Min: $1.01_2 * 2^{-4} = 0.000101_2 = 0.00015_{610}$

(b) What is εmach in this system?

$\epsilon_{mach} = 0.01_2 * 2^0 = 0.25_{10}$

(c) What is the floating-point representation (in binary and decimal) of the number 9 in this system?

$f(9) = 1.01_2 * 2^{-3}$

(d) Give an example that shows f l(f l(a+b) +c)/= f l(a+ f l(b+c)), where a,b, c are floating-point numbers contained in this system

$f(32 + f(5 + 5)) \; ? = f(5 + f(32 + 5)) \Rightarrow 1.00_2 * 2^5 + (1.01 * 2^2 + 1.01 * 2^2) \; ? = (1.00_2 * 2^5 + 1.01 * 2^2) + 1.01_2 * 2^2$

$\Rightarrow 1.00_2 * 2^5 + 1.01_2 * 2^3 \; ? = 1.00_2 * 2^5 + 1.01_2 * 2^1$

$\Rightarrow 1.01_2 * 2^5 = 1.00_2 * 2^5$

5. As you have likely seen in calculus and analysis, the Maclaurin series for f(x) = e 2x converges for −∞ < x < ∞ and is given by e 2x = ∞ ∑ n=0 (2x) n n! Let An(x) be the nth sum of this series for a given x.

(a) Write a MATLAB program that calculates the terms of the series until the relative error is less than 10−10. To write this program most efficiently, you can take advantage of the fact that the n+1st term is equal to the nth term times 2x n+1 . Also, use the MATLAB exp command to calculate the 'true' values of e 2x . Finally, make sure you a maximum iteration threshold of 120 so that it does not run forever is the estimate is not converging. How many terms are needed to converge to this bound for x = 3, x = −3, and x = −9?

```
format  shortG
inputs = [3,-3,-9];
aprox = zeros(1,3);
actual = exp(2*inputs);

for i=1:size(inputs,2)
    cnt = 0;
    while(cnt<120)
      aprox(i) =  aprox(i)+((2*inputs(i))^(cnt))/factorial(cnt);
      cnt = cnt+1;
      if(abs(1-actual(i)/aprox(i)) < 10^-10)
           break;
      end
    end
    outputs=[cnt,actual(i),aprox(i),abs(1-actual(i)/aprox(i))];
    fprintf("At count "+ outputs(1) + " the aprox value is "+ outputs(2) + " the true value is
           " and the absulte error is " + outputs(4) +".\n")
end
```

```
At count 28 the aprox value is 403.4288 the true value is 403.4288 and the absulte error is 6.2813e-11.
At count 35 the aprox value is 0.0024788 the true value is 0.0024788 and the absulte error is 5.6771e-11.
At count 120 the aprox value is 1.523e-08 the true value is 1.5984e-08 and the absulte error is 0.047147.
```

(b)Now change your stopping criteria in the previous program so that the program stops when An(x) = An+1(x) (when doing this, please comment out the original stopping criteria and put the new on in). Obviously this never happens using infinite precision, but it will on a computer. Why?

```
inputs = [3,-3,-9];
aprox = zeros(1,3);
actual = exp(2*inputs);

for i=1:size(inputs,2)
    cnt = 0;
    while(cnt<120)
      aprox(i) =  aprox(i)+((2*inputs(i))^(cnt))/factorial(cnt);
      cnt = cnt+1;
      if(abs(1-actual(i)/aprox(i)) == 0)
           break;
      end
    end
    outputs=[cnt,actual(i),aprox(i),abs(1-actual(i)/aprox(i))];
    fprintf("At count "+ outputs(1) + " the aprox value is "+ outputs(2) + " the true value is
           " and the absulte error is " + outputs(4) +".\n")
end
```

```
At count 36 the aprox value is 403.4288 the true value is 403.4288 and the absulte error is 0.
At count 120 the aprox value is 0.0024788 the true value is 0.0024788 and the absulte error is 7.2498e-13.
At count 120 the aprox value is 1.523e-08 the true value is 1.5984e-08 and the absulte error is 0.047147.
```

It can converge on the computer because of chooping the number doesn't exsit past there.

(c) For x = 15,6,−6,−15 find the following pieces of information: what is the estimate of e 2x generated by the series, what is the 'true' value, how many iterations does it take, and what are the absolute and relative errors?

```
inputs = [15,-15,6,-6];
aprox = zeros(1,4);
actual = exp(2*inputs);

for i=1:size(inputs,2)
    cnt = 0;
    while(cnt<120)
      aprox(i) =  aprox(i)+((2*inputs(i))^(cnt))/factorial(cnt);
      cnt = cnt+1;
      if(abs(1-actual(i)/aprox(i)) < 10^-10)
           break;
      end
    end
    outputs=[cnt,actual(i),aprox(i),abs(1-actual(i)/aprox(i))];
    fprintf("At count "+ outputs(1) + " the aprox value is "+ outputs(2) + " the true value is
           "\nand the absulte error is " + outputs(4) + " the realtive error  is " +outputs(4)/
end
```

```
At count 72 the aprox value is 10686474581524.46 the true value is 10686474580903.78
and the absulte error is 5.8081e-11 the realtive error  is 3.8721e-12.


At count 120 the aprox value is 9.3576e-14 the true value is -4.8265e-06
and the absulte error is 1 the realtive error  is -0.066667.


At count 41 the aprox value is 162754.7914 the true value is 162754.7914
and the absulte error is 4.5191e-11 the realtive error  is 7.5318e-12.


At count 120 the aprox value is 6.1442e-06 the true value is 6.1442e-06
and the absulte error is 6.1218e-08 the realtive error  is -1.0203e-08.
```

(d) You should notice that the estimate for x = −15 is almost laughably bad. Why do you think the relative error is so much higher for negative values?

Maclaurin series is a polynomial aso the negative portion switching from Postive to negative ever new term.