


Question 1. (10p)

Convert the following numbers into IEEE 754 single precision numbers:

(a) -45.0


- 1) The positive version of the number: $|-45| = 45$, giving sign(s) = 1
- 2) Divide the number repeatedly by 2, until we get a quotient that is equal to zero:

Number	Division	Quotient	Remainder
45	$\div 2$	22	1
22	$\div 2$	11	0
11	$\div 2$	5	1
5	$\div 2$	2	1
2	$\div 2$	1	0
1	$\div 2$	0	1



- 3) Construct the base 2 representation: $(45)_{10} = (101101)_2$
(Taking remainders from bottom to top as shown in the above table)
- 4) Shifting the decimal: $(101101)_2 = (101101)_2 \times (2)^0 = (1.01101)_2 \times (2)^5$
- 5) Exponent conversion: Take the power of two i.e. 5
 $((5) + (2^8 - 1))_{10} = (132)_{10}$

Number	Division	Quotient	Remainder
132	$\div 2$	66	0
66	$\div 2$	33	0
33	$\div 2$	16	1
16	$\div 2$	8	0
8	$\div 2$	4	0
4	$\div 2$	2	0
2	$\div 2$	1	0
1	$\div 2$	0	1



Construct the base 2 representation: $(132)_{10} = (1000\ 0100)_2$
(Taking remainders from bottom to top as shown in the above table)
 $e = 1000\ 0100$

- 6) Mantissa normalization: 23-bit conversion of 1.01101
 $1.01101 = 011\ 0100\ 0000\ 0000\ 0000\ 0000 = m$
- 7) $s = 1$
 $e = 1000\ 0100$
 $m = 011\ 0100\ 0000\ 0000\ 0000\ 0000$
The conversion of $(-45)_{10}$

S									E									M																
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
1	1	0	0	0	0	1	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			

(b) 0.085

- 1) Since, it's a positive number, sign(s) = 0

- 2) Break the decimal number into two parts:

Integer = 0

Fraction = 0.085

- 3) Conversion of integer part of decimal to binary:

Number	Division	Quotient	Remainder
0	$\div 2$	0	0

$(0)_{10} = (0)_2$

- 4) Conversion of fractional part: Multiply the number repeatedly by 2, until we get a fractional part that is equal to zero:

S. No.	Number	Multiplication	Result	Integer Result	Fraction Result
1	0.085	$\times 2$	0.17	0	0.17
2	0.17	$\times 2$	0.34	0	0.34
3	0.34	$\times 2$	0.68	0	0.68
4	0.68	$\times 2$	1.36	1	0.36
5	0.36	$\times 2$	0.72	0	0.72
6	0.72	$\times 2$	1.44	1	0.44
7	0.44	$\times 2$	0.88	0	0.88
8	0.88	$\times 2$	1.76	1	0.76
9	0.76	$\times 2$	1.52	1	0.52
10	0.52	$\times 2$	1.04	1	0.04
11	0.04	$\times 2$	0.08	0	0.08
12	0.08	$\times 2$	0.16	0	0.16
13	0.16	$\times 2$	0.32	0	0.32
14	0.32	$\times 2$	0.64	0	0.64
15	0.64	$\times 2$	1.28	1	0.28
16	0.28	$\times 2$	0.56	0	0.56
17	0.56	$\times 2$	1.12	1	0.12
18	0.12	$\times 2$	0.24	0	0.24
19	0.24	$\times 2$	0.48	0	0.48
20	0.48	$\times 2$	0.96	0	0.96
21	0.96	$\times 2$	1.92	1	0.92
22	0.92	$\times 2$	1.84	1	0.84
23	0.84	$\times 2$	1.68	1	0.68
24	0.68	$\times 2$	1.36	1	0.36
25	0.36	$\times 2$	0.72	0	0.72
26	0.72	$\times 2$	1.44	1	0.44
27	0.44	$\times 2$	0.88	0	0.88
28	0.88	$\times 2$	1.76	1	0.76
29	0.76	$\times 2$	1.52	1	0.52
30	0.52	$\times 2$	1.04	1	0.04
Mantissa limit crossed (the fraction part is not zero => Result not precise)					
.

Construct the base 2 representation: $(0.085)_{10} = (0.0001\ 0101\ 1100\ 0010\ 1000\ 1111\ 010)_2$

(Taking Integer results from top to bottom as shown in the above table)

5) Shifting the decimal:

$$(0.0001\ 0101\ 1100\ 0010\ 1000\ 1111\ 010)_2 \times (2)^0 = (1.0101\ 1100\ 0010\ 1000\ 1111\ 010)_2 \times (2)^{-4}$$

(Getting 23-digits in the fraction)

6) Exponent conversion: Take the power of two i.e. -4

$$((-4) + (2^8 - 1))_{10} = (123)_{10}$$

Number	Division	Quotient	Remainder
123	÷ 2	61	1
61	÷ 2	30	1
30	÷ 2	15	0
15	÷ 2	7	1
7	÷ 2	3	1
3	÷ 2	1	1
1	÷ 2	0	1

Construct the base 2 representation: $(123)_{10} = (0111\ 1011)_2$

(Taking remainders from bottom to top as shown in the above table)

e = 0111 1011

7) Mantissa normalization: 23-bit conversion of 1.0101 1100 0010 1000 1111 010

$$1.0101\ 1100\ 0010\ 1000\ 1111\ 010 = 0101\ 1100\ 0010\ 1000\ 1111\ 010 = m$$

8) s = 0

e = 0111 1011

m = 0101 1100 0010 1000 1111 010

The conversion of $(0.085)_{10}$

S	E								M																						
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	1	0	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

and then use their binary representation to write these numbers in hexadecimal representation. The conversion table between decimal, binary and hexadecimal representation is given in Figure 1.

Binary	Hex	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Figure 1: Conversion table between decimal, binary and hexadecimal representation.

(a) -45.0

Binary representation:

S									E									M															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
1	1	0	0	0	0	1	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
C				2				3				4				0				0				0				0					

Hexadecimal conversion: 0xC2340000

(a) 0.085

Binary representation:

S									E									M															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0	0	1	1	1	1	0	1	1	0	1	0	1	1	1	0	0	0	0	1	0	1	0	0	0	1	1	1	1	1	0	1	0	
3				D				A				E				1				4				7				A					

Hexadecimal conversion: 0x3DAE147A

Question 2. (20p)

Write a R script that uses the bisection method to find the root of a given function on a given interval,

The required R script has been attached in the folder as file: "BisectionMethod.R"

and apply this program to find the roots of the functions below on the indicated intervals. Be careful to avoid unnecessary function calls in a single iteration. Use a relationship derived in class to determine a priori the number of steps necessary for the root to be accurate within 10⁻⁶

(a) $x^3 - 2$, $[a, b] = [0, 2]$

The solution is in the file: "RootBisectionA.R"

(b) $e^x - 2$, $[a, b] = [0, 1]$

The solution is in the file: "RootBisectionB.R"

Apply Newton's method to find the root of the second function, $f(x) = e^x - 2$, for which the exact solution is $\alpha = \ln 2$.

The R script for Newton's method is: "NewtonMethod.R"

The root can be calculated by running the file: "RootNewtonB.R"

Perform the following experiments:

(c) Compute the ratio:

$$R_n = \frac{\alpha - x_{n+1}}{(\alpha - x_n)^2}$$

and observe whether or not it converges to the correct value as $n \rightarrow \infty$.

Here, $f(x) = e^x - 2$

$f'(x) = e^x \mid f'(\alpha) = 2$ (Since, $\alpha = \ln 2$)

$f''(x) = e^x \mid f''(\alpha) = 2$ (Since, $\alpha = \ln 2$)

$g(x) = x - (f(x)/f'(x))$

$g'(x) = f(x)f''(x)/(f'(x))^2 \mid g'(\alpha) = 0$

Since, $f'(\alpha) \neq 0$, and $g'(\alpha) = 0 \Rightarrow$ Quadratic convergence

By "Quadratic Convergence of Newton's Method Theorem", convergence to alpha at $n \rightarrow \infty$, occurs if for large value of n, $R_n < f''(\alpha)/(2*f'(\alpha))$ ($=R^*=2/2*2 = 1/2 = 0.5$)

Figure: 2, displays the ratio at $p=2$, $R < 0.5$. Thus, R_n converges to the correct value as $n \rightarrow \infty$

The function follows Quadratic convergence,

i.e. if error $e_n = x_n - r$,

then $|e_{n+1}| = R |e_n|^2$

(d) Compute the modified ratio

$$R_n = \frac{\alpha - x_{n+1}}{(\alpha - x_n)^p},$$

for various $p \neq 2$, but near 2

The Ratio at various values of n is as below:

```
[1] "Printing the table with the iterations"
x1 x0 x0.693147180559945 x.0.159174538954862
1 1 0.0000000 6.931472e-01 -0.1591745
2 2 1.0000000 -3.068528e-01 NaN
3 3 0.7357589 -4.261170e-02 NaN
4 4 0.6940423 -8.951214e-04 NaN
5 5 0.6931476 -4.005452e-07 NaN
6 6 0.6931472 -1.005862e-13 NaN
[1] "Printing Ratio at p=2 and Modified Ratio at p = 2.3, 2.5, 3, 4, 5, 6, 7"
x.0.638673837226921 x.0.712904716021179 x.0.767125457798884 x.0.921411577712804 x.1.32931591378394 x.1.91779747659087 x.2.76679690890701 x.3.99164417962706
1 -0.6386738 -0.7129047 -0.7671255 -9.214116e-01 -1.329316e+00 -1.917797e+00 -2.766797e+00 -3.991644e+00
2 -0.4525523 NaN NaN 1.474819e+00 -4.806276e+00 1.566313e+01 -5.104446e+01 1.663484e+02
3 -0.4929742 NaN NaN 1.156899e+01 -2.714979e+02 6.371439e+03 -1.495232e+05 3.508971e+06
4 -0.4999052 NaN NaN 5.584776e+02 -6.239127e+05 6.970146e+08 -7.786817e+11 8.699175e+14
5 -0.6269535 NaN NaN 1.565250e+06 -3.907799e+12 9.756200e+18 -2.435730e+25 6.081037e+31
6 0.0000000 NaN NaN 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
[1] "The found root on the interval [a,b] is:"
[1] 0.6931472
```

Figure: 2

This can be fetched by running the file: "[RootNewtonB.R](#)"

What happens? Comment, in the light of the definition of order of convergence.

Assuming, the function follows Quadratic convergence, and f'' is continuous near α

Error $e_n = \alpha - x_n$, (i)

Using Taylor approximation about x_n

$$f(\alpha) = f(x_n + e_n) = f(x_n) + e_n f'(x_n) + e_n^2 f''(x_n)/2 + O(e_n^3)$$

$$\begin{aligned} \Rightarrow f(x_n) + e_n f'(x_n) + e_n^2 f''(x_n)/2 + O(e_n^3) &= 0 \text{ (Since, } f(\alpha) = 0) \\ \Rightarrow -f(x_n)/f'(x_n) &= e_n + e_n^2 f''(x_n)/2f'(x_n) + O(e_n^3) \dots\dots\dots (ii) \end{aligned}$$

Also,

$$\begin{aligned} e_{n+1} = \alpha - x_{n+1} &= \alpha - (x_n - f(x_n)/f'(x_n)) \\ &= (\alpha - x_n) - f(x_n)/f'(x_n) \dots\dots\dots \text{From (i)} \\ &= e_n - (e_n + e_n^2 f''(x_n)/2f'(x_n) + O(e_n^3)) \dots\dots\dots \text{From (ii)} \\ &= - (e_n^2 f''(x_n)/2f'(x_n) + O(e_n^3)) \end{aligned}$$

By the Fixed-Point Theorem,

$$g(x) = x - (f(x)/f'(x))$$

$$g'(x) = f(x)f''(x)/(f'(x))^2$$

the sequence of iterations converges, if

$$|g'(x)| \leq K < 1 \quad \forall x \text{ in } [a, b]$$

where x_0 in $[a, b]$, and the rate of convergence is $O(K^n)$

$$\text{Similarly, } (|x_{n+2} - \alpha| = R^* |x_n - \alpha|^2) \dots\dots\dots (iii)$$

$$\text{Also, we have } |x_{n+1} - \alpha|^p = R^p |x_n - \alpha|^{p^p}$$

Comparing it with (iii), we get

$$(|x_{n+2} - \alpha| = R |x_{n+1} - \alpha|^p = R^{p+1} |x_n - \alpha|^{p^{*p}})$$

$$\Rightarrow p^2 = 2, \text{ and } R^{1+p} = R^* (=f''(\alpha)/(2*f'(\alpha)))$$

$$\Rightarrow \text{Superlinear convergence of order } p = \sqrt{2}$$

Apply the secant method to the same function, using x_0, x_1 equal to the end points of the given interval. Stop the iteration when the error as estimated by $|x_n - x_{n-1}|$ is less than 10^{-6} .

The Secant Method file can be found as: "[SecantMethod.R](#)"

The roots can be fetched running the file: "[RootSecantB.R](#)"

Compare to your results for Newton and bisection method.

	Bisection Method	Newton Method	Secant Method
Iterations	More number of iterations	Less iteration	Less iterations => faster conversion
# functions	One function	2 functions (one extra for derivative)	1 function required
Accuracy	The result is accurate to five decimal digits	The result is accurate to seven decimal digits	The result is accurate to seven decimal digits
Lesser decimal digit accuracy	Three place decimal accuracy could be seen in 14 th iteration	Lesser decimal digit accuracy can be seen in just 5 th step	Lesser decimal digit accuracy can be seen in just 5 th step
Method	Simple subtraction of values and taking the mid-point by dividing with 2. Simple to implement	Requires derivation, for which the difference between $(x+1)$ & (x) is very small. Or package installation is required	Same concept, but instead of tangent/derivate, requires secant
Assumption	No condition on initial approximation, it always converges	If initial assumption is very close to the solution, derivative is flat, and thus no convergence	Initial assumption should be close to the solution
Solution Assumption	Two initial guesses required	One solution guess is required	Two initial guesses required
Stability	Always converges	Might get into loop	Unstable as is same in Newton

Have your codes print out the entire sequence of iterates to the following files:

bisection1.txt (for function a)), bisection2.txt, newton.txt, and secant.txt (for function b)).

The printed files have been attached in the solution folder.

The solution code for the same has been implemented in the files: "BisectionMethod.R", "NewtonMethod.R", "SecantMethod.R"

A snapshot of the iteration results is as follows:

For Bisection Method for (a):	For Bisection Method for (b) :
<pre>[1] "Printing the table appending the current iteration in the table" x2 x0.690745633944098 x2.1 x2.2 x0 1 2 0.6907456 2.000000e+00 2.000000 0.000000 2 3 1.0000000 1.000000e+00 2.000000 1.000000 3 4 1.5000000 5.000000e-01 1.500000 1.000000 4 5 1.2500000 2.500000e-01 1.500000 1.250000 5 6 1.3750000 1.250000e-01 1.375000 1.250000 6 7 1.3125000 6.250000e-02 1.312500 1.250000 7 8 1.2812500 3.125000e-02 1.281250 1.250000 8 9 1.2656250 1.562500e-02 1.265625 1.250000 9 10 1.2578125 7.812500e-03 1.265625 1.257812 10 11 1.2617188 3.906250e-03 1.261719 1.257812 11 12 1.2597656 1.953125e-03 1.261719 1.259766 12 13 1.2607422 9.765625e-04 1.260742 1.259766 13 14 1.2602539 4.882812e-04 1.260254 1.259766 14 15 1.2600098 2.441406e-04 1.260010 1.259766 15 16 1.2598877 1.220703e-04 1.260010 1.259888 16 17 1.2599487 6.103516e-05 1.259949 1.259888 17 18 1.2599182 3.051758e-05 1.259949 1.259918 18 19 1.2599335 1.525879e-05 1.259933 1.259918 19 20 1.2599258 7.629395e-06 1.259926 1.259918 20 21 1.2599220 3.814697e-06 1.259922 1.259918 21 22 1.2599201 1.907349e-06 1.259922 1.259920 [1] "The found root on the interval [a,b] is:" [1] 1.259921</pre>	<pre>[1] "Printing the table appending the current iteration in the table" x2 x0.690745633944098 x1 x1.1 x0 1 2 0.6907456 1.000000e+00 1.000000 0.000000 2 3 0.5000000 5.000000e-01 1.000000 0.500000 3 4 0.7500000 2.500000e-01 0.750000 0.500000 4 5 0.6250000 1.250000e-01 0.750000 0.625000 5 6 0.6875000 6.250000e-02 0.750000 0.687500 6 7 0.7187500 3.125000e-02 0.718750 0.687500 7 8 0.7031250 1.562500e-02 0.703125 0.687500 8 9 0.6953125 7.812500e-03 0.6953125 0.687500 9 10 0.6914062 3.906250e-03 0.6953125 0.6914062 10 11 0.6933594 1.953125e-03 0.6933594 0.6914062 11 12 0.6923828 9.765625e-04 0.6933594 0.6923828 12 13 0.6928711 4.882812e-04 0.6933594 0.6928711 13 14 0.6931152 2.441406e-04 0.6933594 0.6931152 14 15 0.6932373 1.220703e-04 0.6932373 0.6931152 15 16 0.6931763 6.103516e-05 0.6931763 0.6931152 16 17 0.6931458 3.051758e-05 0.6931763 0.6931458 17 18 0.6931610 1.525879e-05 0.6931610 0.6931458 18 19 0.6931534 7.629395e-06 0.6931534 0.6931458 19 20 0.6931496 3.814697e-06 0.6931496 0.6931458 20 21 0.6931477 1.907349e-06 0.6931477 0.6931458 [1] "The found root on the interval [a,b] is:" [1] 0.6931467</pre>
For Newton Method:	For Secant method:
<pre>[1] "Printing the table with the iterations" x1 x0 x0.693147180559945 x.0.159174538954862 1 1 0.0000000 6.931472e-01 -0.1591745 2 2 1.0000000 -3.068528e-01 NaN 3 3 0.7357589 -4.261170e-02 NaN 4 4 0.6940423 -8.951214e-04 NaN 5 5 0.6931476 -4.005452e-07 NaN 6 6 0.6931472 -1.005862e-13 NaN [1] "The found root on the interval [a,b] is:" [1] 0.6931472</pre>	<pre>[1] "Printing the table with the iterations" x1 x0 x0.693147180559945 1 1 0.0000000 6.931472e-01 2 2 1.0000000 -3.068528e-01 3 3 0.5819767 1.111705e-01 4 4 0.6766927 1.645448e-02 5 5 0.6940814 -9.342191e-04 6 6 0.6931395 7.705915e-06 7 7 0.6931472 3.598951e-09 [1] "The found root on the interval [a,b] is:" [1] 0.6931472</pre>

Question 3. (20p)

Write an R code that does LU factorization with partial pivoting for an arbitrary $n \times n$ matrix ($n < 100$).
Use it to solve the systems of equations $Ax = b$:

The solution R code is attached in file: "[LUFactorization.R](#)"

$$(a) \quad A = \begin{bmatrix} 9 & 3 & 2 & 0 & 7 \\ 7 & 6 & 9 & 6 & 4 \\ 2 & 7 & 7 & 8 & 2 \\ 0 & 9 & 7 & 2 & 2 \\ 7 & 3 & 6 & 4 & 3 \end{bmatrix}, \quad b = [35, 58, 53, 37, 39]^T;$$

The solution is fetched running the file: "[LU_Solution_a.R](#)"

$$(b) \quad A = H_5, \quad b = [5.0, 3.550, 2.81428571428571, 2.34642857142857, 2.01746031746032]^T;$$

The solution is fetched running the file: "[LU_Solution_b.R](#)"

(c) $A = A_5, \quad b = [-4, -7, -6, -5, 16]^T,$

The solution is fetched running the file: "LU_Solution.c.R"

where H_n and A_n denote the families of matrices parameterised by their dimension

in the following way:

$$H_n = [h_{ij}], \quad h_{ij} = \frac{1}{i+j-1},$$

$$A_n = [a_{ij}], \quad a_{ij} = \begin{cases} 1, & i = j; \\ 4, & i - j = 1; \\ -4, & i - j = -1; \\ 0, & \text{otherwise.} \end{cases}$$

In each case, have your code multiply out the L and U factors to check that the routine is working properly.

The L and U factors are multiplied to check the routine