Question 1:

(a) -45.0

$45_{10} = 2^5 + 2^3 + 2^2 + 2^0 = 101101_2$
Normalise: 1.01101×2^5
P = 132 (127 + 5)
S = 1
Binary = [S | P | 1.f] = [1 | 10000100 | 01101000000000000000000]
Hex: 0xC2340000

(b) 0.085

Note: 0.085 cannot be represented exactly in 32 bit floating point format so this is an approximation.
To determine fraction part:
0.085 - (1/16) = 0.0225
0.0225 - (1/64) = 0.006875
0.006875 - (1/256) = 0.00296875
And so on. The final result is:
$0.085 = 0.00010101110000101010001111011_2$
Normalise to get $1.01011100001010001111011_2 \times 2^{-4}$
S = 1
P = 123 (127 - 4)
Binary: [S | P | 1.f] = [0 | 01111011 | 01011100001010001111011]
Hex: 0x3DAE147B

Question 2:

See file *root_finding.R*. The *run_all()* function in that file will use the bisection method to find the roots of $f(x) = x^3 - 2$ and save the results to bisection1.txt, then it will use all of the methods to find the roots of $f(x) = e^x - 2$ and will save the results in *bisection2.txt*, *newton.txt* and *secant.txt*.

Use a relationship derived in class to determine *a priori* the number of steps necessary for the root to be accurate within 10^{-6}:

The relationship (with accuracy = $\epsilon = 10^{-6}$) is:

$$n \geq \frac{log(b-a) - log(\varepsilon)}{log(2)}$$

For function a with a = 0, b = 2 this gives n ≥ 20.93.
Function b with a = 0, b = 1 this gives n ≥ 19.93
This matches with my results - function a took 21 iterations and function b took 20 iterations.

(c) & (d):

To calculate the data here I used $x_0=5$ and $f(x) = e^x$.
This document[1] states that when:

- p=2 and $|R_N|$ approaches 1 the convergence rate is at least quadratic.
- p=1 and $|R_N|$ approaches 0,the convergence rate is at least exponential.
- p=3 and$|R_N|$ approaches infinity, the convergence is at least superquadratic.

Newton's method with this choice of *f(x)* meets all three of these conditions.
This means that it converges to the correct value extremely quickly.

N	X_N	X_{N+1}	$\|R_N\|$, p=1	$\|R_N\|$, p=2	$\|R_N\|$, p=3	$\|R_N\|$, p=1.9	$\|R_N\|$, p=2.1
1	4.013476	3.049617	0.709710	0.21374683461	0.064375	0.241000	0.189575
2	3.049617	2.144371	0.615847	0.26134285683	0.110904	0.284732	0.239875
3	2.144371	1.378654	0.472365	0.32549399709	0.224289	0.337844	0.313595
4	1.378654	0.882489	0.276207	0.4029238611	0.587775	0.387993	0.418429
5	0.882489	0.7099926	0.088968	0.4698819225	2.481659	0.397845	0.554963
6	0.7099926	0.6932883	0.008377	0.49730541294	29.521700	0.330576	0.748126
7	0.6932883	0.6931472	0.000138	0.97616626055	6917.305370	0.402237	2.369000

Compare to your results for Newton and bisection method:

For the following I used function b and a tolerance of 10^{-6}

<u>Starting points: 0 and 1 for bisection and secant methods, 1 for Newton's method:</u>
Newton's method converged in only 4 steps.
The secant method converged in 6 steps.
The bisection method converged in 20 steps.
Here Newton's method is obviously the best when we have an initial estimate that is close enough to the actual root. The secant method is not far behind, while the bisection method is notably slower.

<u>Starting points: 0 and 100 for bisection and secant methods, 100 for Newton's method:</u>
The bisection method converged with the correct answer after 24 steps.
The secant method failed to converge. It returned after only two steps with an incorrect answer. This was due to the large difference between the starting points. By changing the starting points to 90 and 100 it converged correctly after 137 steps.

[1] http://cavern.uark.edu/~arnold/4363/OrderConv.pdf

Newton's method converged correctly after 104 steps.
This shows that Newton's method struggles when the starting point is far from the actual root.
It also shows that the secant method struggles when there is a large difference between the starting points, or when one of the starting points is far from the actual root.
Finally it shows that the bisection method is the fastest method when the starting points are far from the actual root.

Question 3:

See file *linear_system_solver.R*. The file contains a function *run_all_parts()*, which sets up and solves all systems as described in the assignment. The output is printed to the screen. The output includes the following: The original matrix, the B matrix, the L, U and LU matrices, the permutation (P) and PA matrices, and finally the resulting X matrix.

I have included a copy of the results in the file *output.txt.*

The results for each part of the question were:
(a): x = [0 1 2 3 4]
(b): x = [1 2 3 4 5]
(c): x = [0 1 2 3 4]