Floating-Point Computation

(a) Briefly describe the 32-bit IEEE floating-point format, explaining what values (or other mathematical objects) are represented by bit-patterns in this format (you need not give the values corresponding to denormalised numbers).

(b) What value, if any, does the following Java method return, assuming x and old are held as 32-bit IEEE values?

```java
c() { float old=0, x=1;
    while (old != x) { old = x; x = x+1; }
    return x; }
```

Explain your reasoning.

(c) Consider the function computed by the Java method

```java
f(float x) { return x+1; }
```

Discuss how the use of 32-bit IEEE floating-point arithmetic causes it to differ from the mathematical function \( f(x) = x + 1 \).

(d) Given a problem of the form “find \( x \) such that \( f(x) = y \)”, explain informally what it means for it to be ill-conditioned.

(e) The Newton–Raphson iteration for \( \sqrt{a} \) uses \( x_{n+1} = (x_n + a/x_n)/2 \).
Let \( x_n = \sqrt{a} + \epsilon_n \), where the error \( \epsilon_n \) is assumed to be small.

(i) Calculate how the error declines from one iteration to the next.

(ii) Given \( 1 \leq a < 4 \) and \( x_0 = 1.5 \), how many iterations are necessary to achieve approximate 32-bit IEEE accuracy, and 64-bit IEEE accuracy?

(iii) Summarise a possible implementation of square-root on the whole 32-bit IEEE input range rather than just on \([1, 4)\).