Worksheet 1, February 7, 2025

1 Floating point representation and arithmetic

Computers generally use floating point representations to store numbers, following the IEEE standard, where the decimal place can "float" anywhere among the significant digits of the number. Floating point representation is based on *exponential/scientific* notation, stored typically in base 2 but at times base 10. A nonzero number x is written in the form

$$x = \pm \left(1.b_1 b_2 b_3 ... b_{p-2} b_{p-1}\right)_2 \times 2^E \tag{1}$$

where the exponent $E \in [-126, 127]$ serves to float the decimal place to its position, and p stands for the precision. In 32-bit, 1 bit stores the sign, 8 bits for the exponent, and 23 bits for the fraction (b_1 through $b_2 3$) corresponding to precision $p = 24$. Given an x in floating point representation, you can calculate the number by

$$x = \left(2^0 + b_1 2^{-1} + b_2 2^{-2} + b_3 2^{-3} + ...\right) \times 2^E \tag{2}$$

For example, $11/2$ is expressed as

$$\frac{11}{2} = (1.011)_2 \times 2^2$$

Notice that not all numbers x have finite binary expansions – for example, the number 0.1 has an infinitely repeating binary representation

$$1/10 = (0.000110011001100110011...)_2$$

which must be truncated to 23 bits, obtaining

$$1/10 \approx (0.00011001100110011001100)_2 = (1.1001100110011001100)_2 \times 2^{-4}$$

The smallest x that is greater than 1 in floating point representation is

$$(1.00...01)_2 = 1 + 2^{-(p-1)}$$

This second number is what we call *machine epsilon*. The relative rounding error of floating point numbers is bounded by machine epsilon, regardless of rounding mode.

Floating point addition and subtraction require aligning the significands, such that if $x = S \times 2^E$ and $y = T \times 2^F$, T must be shifted $E - F$ positions so both numbers have the same exponent. Then,

$$x \oplus y = (S + T^*) \times 2^E$$

followed by normalization and rounding. Floating point multiplication and division instead involves multiplying the significands, adding the exponents, and then normalizing and rounding the result:

$$x \otimes y = (S \times T) \times 2^{E+F}$$

1.1 Converting to/from floating point representation

Q1 Convert the number 71.125 to floating point representation.

Q2 Convert $(1.0101)_2 \times 2^3$ to decimal representation.

1.2 Floating point arithmetic

Q1 In exact arithmetic, the addition operation is commutative, i.e.

$$x + y = y + x$$

for any two numbers x, y, and also associative, i.e.,

$$x + (y + z) = (x + y) + z$$

for any x, y, and z. Is the floating point addition operator $\oplus$ commutative? Is it associative?

Q2 Consider floating point representation with 32-bit precision. Does there exist some x such that $x \oplus 1 = x$?

Q3 Consider floating point representation with 32-bit precision. What is the largest floating point number x for which $1 \oplus x$ is exactly 1, assuming nearest rounding mode is in effect? (Write x in floating point representation)

2 Fixed point methods

We want to find a positive root x^* of $f(x) := x^2 - x - 2 = 0$ using the fixed point method.

2.1 Convergence of fixed point iteration

Q1 Verify that x^* is a fixed point of $g(x) = x^2 - 2$.

Q2 Can you use the fixed point algorithm to find the root x^* using $g(x)$? Why or why not?

Q3 What other function $h(x)$ can you perform a fixed point iteration on to find x^*?

Q4 How fast do you expect the convergence to be (linear/superlinear/quadratic/etc.)?

2.2 Implementing fixed point iteration

Q1 Implement fixed point iteration for both $g(x)$ and $h(x)$, with $x_0 = 5$. Choose a termination criteria and explain your choice.

Q2 Verify convergence behavior numerically by plotting error $|x_k - x^*|$ against iteration k. Consider carefully how you plot this. (Should you use `plot`, `semilogy`, or `loglog`?)

Q3 Diagnose the convergence rate numerically. Does it match the theoretical expectation?

Q4 What if we did not know x^*? What should you plot instead? Can you relate these quantities to the actual error $|x_k - x^*|$?

2.3 Steffensen's Method

The goal of this task is to analyse Steffensen's Method for finding the root of a function $f : \mathbb{R} \to \mathbb{R}$, defined by the iteration

$$x_{k+1} = x_k - \frac{[f(x_k)]^2}{f(x_k + f(x_k)) - f(x_k)}, \qquad k = 1, 2, \dots. \tag{3}$$

Q1 Explain how this iteration relates to Newton's method and the Secant method.

Q2 Show that the sequence $\{x_k\}_{k=0}^{\infty}$ converges quadratically if x_0 is sufficiently close to the solution.

(a) Step 1: Rewrite (3) by expanding $f(x_k + f(x_k))$ around x_k (to order $f(x_k)^2$).

(b) Step 2: Consider the Taylor series of $f(\xi)$ around x_k:

$$0 = f(\xi) = f(x_k) + f'(x_k)(\xi - x_k) + \frac{1}{2}f''(z_k)(\xi - x_k)^2$$

where z_k is some value between x_k and ξ. Use this in combination with part (a) to write an expression for $\xi - x_{k+1}$ in terms of $\xi - x_k$.

(c) Compute the limit

$$\lim_{k\to\infty} \frac{|\xi - x_{k+1}|}{|\xi - x_k|}$$

Q3 Compare the computational cost of Steffensen's method to that of Newton's method and the secant method. When would you use which of the methods?

Q4 If there is time, try implementing Steffensen's method and verify the convergence is as you expecto