CS31400 Homework 4

John Stanwick

October 9th 2023

1 Section 1

1.1 3.25

To represent 3.25_{10} in an 8-bit floating-point format with 3 bits for the mantissa and 4 bits for the excess-7 exponent, we first convert it to binary as 11.01_2. Normalizing, we get $1.101_2 * 2^1$. The exponent is 1, which in excess-7 format is $1 + 7 = 8_{10}$ or 1000_2. The mantissa is the three bits after the binary point, 101. Combining the sign bit (0 for positive), the exponent 1000, and the mantissa 101, we get the 8-bit representation 01000101.

1.2 -81.5

To represent -81.5_{10} in an 8-bit floating-point format with 3 bits for the mantissa and 4 bits for the excess-7 exponent, we start by converting it to binary, yielding -1010001.1_2. Normalizing, we express this as $-1.0100011_2 * 2^6$. The exponent is 6, which when adjusted for the excess-7 format becomes 13_{10} or 1101_2 in binary. From the normalized form, the three-bit mantissa is 010. Combining the sign bit (1 for negative), the four-bit exponent 1101, and the three-bit mantissa 010, the 8-bit representation of -81.5_{10} is 11101010.

1.3 $\frac{1}{32}$

To represent $1/32_{10}$ in an 8-bit floating-point format with 3 bits for the mantissa and 4 bits for the excess-7 exponent, we begin by converting it to binary, resulting in 0.00001_2. Normalizing, we achieve the form $1.0_2 * 2^{-5}$. The exponent is -5, which, when adjusted using the excess-7 format, becomes 2_{10} or 0010_2 in binary. From the normalized value, the three-bit mantissa is 000. Taking the sign bit (0 for positive), the four-bit exponent 0010, and the three-bit mantissa 000, the 8-bit representation for $1/32_{10}$ is 00010000.

2 Section 2

2.1 Part A

The 8-bit floating point representation for -0.625 in a format with 3 bits for the mantissa and 4 bits for an excess-7 exponent starts with the sign bit, which is 1 for a negative number. The number $-5/8$ is normalized in binary form to -1.01×2^{-1}. Ignoring the implied leading '1', the mantissa becomes 010. With the exponent of -1, when adjusted for the excess-7 bias, it becomes 6 or 0110 in binary. Thus, the complete 8-bit pattern for -0.625 is 10110010.

2.2 Part B

Given the 8-bit pattern 01001001, in a format with 3 bits for the mantissa and 4 bits for the excess-7 exponent, the sign bit is 0 indicating a positive number.

The exponent part is 1001, which simplifies to 2^1 after the excess-7 exponent. The mantissa is then equal to 001, which gives a final result of 1.1×2^1, or 3 for the given equivalent base-10 representation of 01001001.

3 Section 3

3.1 Part A

For the 7-bit floating-point representation with 3 bits for the excess-3 exponent and 3 bits for the mantissa, we start by normalizing 0.275_{10}. The closest representation that is available for a 3 bit mantissa is 0.01, or 0.25_{10}. The exponent should be equal to 2^2 in order to normalize the value. As a result, the equivalent representation should be equal to 0110001.

3.2 Part B

Given the 7-bit pattern 0100110 in a floating-point representation with 3 bits for the excess-3 exponent and 3 bits for the mantissa: the sign bit is 0, indicating a positive number. The exponent part is 100, which is 4 in base-10. Subtracting the excess-3 bias, we get an exponent of $4 - 3 = 1$. The mantissa is 110. Considering the implied leading '1', this is 1.10_2 in binary. Thus, 0100110 represents the base-10 integer 6.

3.3 Part C

Given the 7-bit pattern 1001100 in a floating-point representation with 3 bits for the excess-3 exponent and 3 bits for the mantissa: the sign bit is 1, indicating a negative number. The exponent part is 001, which is 1 in base-10. Subtracting the excess-3 bias, we get an exponent of $1 - 3 = -2$. The mantissa is 100. Considering the implied leading '1', this becomes 1.00_2 in binary, which is 1 in decimal. Given the exponent of -2, the value becomes $1 \times 2^{-2} = 0.25$ in decimal. Therefore, 1001100 represents the decimal value -0.25.

4 Section 4

4.1 Chopping

When storing any number through the chopping method, the absolute error will be equal to True Value − Stored Value. In the of chopping the digits $d_t d_{t+1}...$, one can recognize that the absolute error of this value will be equal to the terms chopped off. As such, if one were to chop any real number, β, they would then actualize an error of β in that respective decimal place. The natural extension of this logic is that the error term associated for the chopping method is equal to $\beta^{1-t}\beta^e$, where t represents the digits that are included in the storage of x.

The β^e term represents the base. The $1-t$ term represents the β digits that have been chopped from the approximation.

4.2 Rounding

When storing a number through the rounding method, the absolute error will again be equal to True Value − Stored Value. The rounding method is unique in that the last digit (d_{t-1}) proposed for storage may be rounded up or down dependent upon the d_t term. This truth is reflected in the proposed system of equations for $fl(x)$, where the last term is either 0 or β^{1-t}. When calculating the absolute error, we can simply average these two terms to determine that the error proposed by the rounding method is equal to $\frac{1}{2}\beta^{1-t}$. One must then simply multiply this result by the original base error to determine that the total error of a floating point utilizing the rounding method is equal to $\frac{1}{2}\beta^{1-t}\beta^e$. Thus, we may determine that the rounding method is less prone to error than the chopping method.

5 Section 5

5.1 Part A

$$f_1(x,\delta) = cos(x+\delta) - cos(x)$$

$$cos(\phi) - cos(\psi) = -2sin(\frac{\phi+\psi}{2})sin(\frac{\phi-\psi}{2})$$

$$cos(x+\delta) - cos(x) = -2sin(\frac{x+\delta+x}{2})sin(\frac{x+\delta-x}{2})$$

$$f_2(x,\delta) = -2sin(x+\frac{\delta}{2})sin(\frac{\delta}{2})$$

Assessing $\frac{f_1(x,\delta)}{\delta}$ finds that $\frac{cos(x+\delta)-cos(x)}{\delta}$, which is the derivative of $-sin(x)$. As such, it is an appropriate approximation of $-sin(x)$ when δ is sufficiently small.

Assessing $\frac{f_2(x,\delta)}{\delta}$ is slightly more strenuous. If δ is very small, then $sin(\frac{\delta}{2})$ goes to $\frac{\delta}{2}$. Thus, we obtain $\frac{-2sin(x+\frac{\delta}{2})\frac{\delta}{2}}{\delta} = -sin(x+\frac{\delta}{2})$. As stated earlier, δ is assumed to be sufficiently small, which means that this approximation is equal to $-sin(x)$.

5.2 Part B

Using the product-to-sum identities of cosine and sine:

$$cos(A)cos(B) = \frac{1}{2}[cos(A-B) + cos(A+B)]$$

$$sin(A)sin(B) = \frac{1}{2}[cos(A-B) - cos(A+B)]$$

With $A = \frac{\phi+\psi}{2}$ and $B = \frac{\phi-\psi}{2}$:

$$sin(\frac{\phi+\psi}{2})sin(\frac{\phi-\psi}{2}) = \frac{1}{2}[cos(\psi) - cos(\phi)]$$

Multiple by -2

$$cos(\phi) - cos(\psi) = -2sin(\frac{\phi+\psi}{2})sin(\frac{\phi-\psi}{2})$$

5.3 Part C

$$g_1(x,\delta) = \frac{f_1(x,\delta)}{\delta} + sin(x)$$
$$g_1(3,10^{-11}) = \frac{cos(x+\delta) - cos(x)}{\delta} + sin(x)$$
$$g_1(3,10^{-11}) = \frac{cos(3+10^{-11}) - cos(3)}{10^{-11}} + sin(3)$$
$$g_1(3,10^{-11}) = 4.94996 \times 10^{-12}$$

$$g_2(x,\delta) = \frac{f_2(x,\delta)}{\delta} + sin(x)$$
$$g_2(3,10^{-11}) = \frac{-2sin(3+\frac{10^{-11}}{2})sin(\frac{10^{-11}}{2})}{10^{-11}} + sin(3)$$
$$g_2(3,10^{-11}) = -0.14112$$

$g_1(x,\delta)$ is a dramatically better approximation of sin(x) than $g_2(x,\delta)$. One can recognize this analytically because the $sin(\delta)$ portion of $f_2(x,\delta)$ becomes very imprecise as $\delta \to 0$. The small angle approximation which allows one to derive $-sin(x)$ as the resultant approximation for $\frac{f_2(x,\delta)}{\delta}$ reflects this fact. As $\delta \to 0$, the $sin(\delta)$ term becomes more and more imprecise, eventually causing the massive error that one observes in the above example.

6 Section 6

6.1 Part A

- The number 15.0625 in binary scientific notation is 1.1110001×2^3.
- For the significand, exponent, and sign bits (32 bits total):
  - The sign bit is 0 (because the number is positive).
  - The exponent is 3+127 = 130 in bias-127 format, which is represented as 10000010 in binary.
  - The significand is 11100010000000000000000
  - Combining these, the 32-bit representation is: 01000001011100010000000000000000.
- In hexadecimal, the above 32 bits are represented as 0x41710000.

6.2 Part B

- The number -35.6875 in binary scientific notation is 1.000111011×2^5.
- For the significand, exponent, and sign bits (32 bits total):
  - The sign bit is 1 (because the number is negative).
  - The exponent is 132 in bias-127 format, which is represented as 10000100 in binary.
  - The significand is 00011101100000000000000
  - Combining these, the 32-bit representation is: 11000010000011101100000000000000
- In hexadecimal, the above 32 bits are represented as 0xc20ec000.

6.3 Part C

- The number 86.5626 in binary scientific notation is 1.0101101001×2^6.
- For the significand, exponent, and sign bits (32 bits total):
  - The sign bit is 0 (because the number is positive).
  - The exponent is 133 in bias-127 format, which is represented as 10000101 in binary.
  - The significand is 01011010010000000000000
  - Combining these, the 32-bit representation is: 01000010101011010010000000000000.
- In hexadecimal, the above 32 bits are represented as 0x42ad2000.

7 Section 7

$$k(x) = |x\frac{f'(x)}{f(x)}|$$

7.1 Part A

$$k(x) = |x\frac{1}{x+c}| = |\frac{x}{x+c}|$$

As x approaches $-c$, the denominator approaches zero, which causes the relative condition number to become large. Therefore, the function $f(x) = x+c$ is ill-conditioned around the point $x = -c$.

7.2 Part B

$$k(x) = |x\frac{c}{cx}| = 1$$

Since k is equal to 1, the function is well conditioned.

7.3 Part C

$$k(x) = |x\frac{cx^{c-1}}{x^c}| = |\frac{xc}{x}| = c$$

Since k is a constant c for this function, the function is well conditioned as long as c itself is a rational and well-conditioned (non infinite) constant.

7.4 Part D

$$k(x) = |x\frac{e^x}{e^x}| = x$$

k(x) is large when x is very large. When x is small, the function is well conditioned.

7.5 Part E

$$k(x) = |x\frac{cos(x)}{sin(x)}|$$

The function could be ill conditioned when $\frac{cos(x)}{sin(x)}$ is not very small and x is very large. Additionally, the function is ill conditioned when $sin(x)$ approaches zero.

7.6 Part F

$$k(x) = |x\frac{sec^2(x)}{tan(x)}|$$

The function is ill conditioned when $\frac{sec^2(x)}{tan(x)}$ is not very small and x is very large. Additionally, the function is ill conditioned when $tan(x)$ approaches zero.

7.7 Part G

$$k(x) = |x\frac{1}{xln(x)}| = |\frac{1}{lnx}|$$

The function is ill conditioned as x approaches 1, as lnx at x = 1 is equal to 0, which would make the k(x) term undefined.

7.8 Part H

$$k(x) = |x\frac{\frac{x}{\sqrt{1+x^2}}}{\sqrt{1+x^2}}| = |\frac{x^2}{1+x^2}|$$

The function is well conditioned, as this term cannot be greater than one, nor can this term become undefined.