MA398 Matrix Analysis and Algorithms: Exercise Sheet 3

1. Write a Python function that takes a matrix A as input and calculates $\rho(A)^k$, $\rho(A^k)$, $\|A^k\|$, and $\|A\|^k$ for some k (you can decide the value of k).

   Then, create a normal matrix and run the function to show that for normal matrices, $\rho(A)^k = \|A\|_2^k$.

2. (Converting to IEEE Standard 754 Format) Given the decimal number 78.625, perform the following steps to convert it to the IEEE Standard 754 single-precision format:

   - Convert the decimal number to binary format.
   - Normalize the binary number to the form 1.m.
   - Add the bias (for single precision format) to the exponent value.
   - Construct the final IEEE Standard 754 single-precision representation.
   - Provide the detailed steps of your calculation and the final answer.

   Answer:

   (a) Convert the decimal number to binary format.
   - 78 in decimal is equivalent to 1001110 in binary.
   - 0.625 in decimal is equivalent to 0.101 in binary (since it equals $2^{-1} + 2^{-3}$).
   - So, 78.625_{10} becomes 1001110.101_2.

   (b) Normalize the binary number to the form 1.m.
   - 1001110.101_2 can be rewritten as 1.001110101×2^6 (we've shifted the binary point 6 places to the left).

   (c) Add the bias (for single precision format) to the exponent value.
   - In the single precision format, the bias is $2^{(8-1)} - 1 = 127$.
   - We add this bias to the actual exponent of the number (which is 6). So, the biased exponent becomes $6 + 127 = 133_{10}$. 133_{10} in binary is 10000101_2.

   (d) Construct the final IEEE Standard 754 single-precision representation.
   - The sign bit is 0 (since the number is positive).
   - The exponent bits are 10000101 (from step 3).
   - The fraction bits are taken from the mantissa we calculated in step 2, i.e., 001110101. But we need 23 bits for the fraction in total. So, we pad the fraction with zeroes to the right, i.e., 00111010100000000000000.

   Therefore, the final IEEE Standard 754 single-precision representation of 78.625_{10} is:
   - 0 10000101 00111010100000000000000.

   To double-check your work, you can use one of the many IEEE 754 calculators available online. They can be a useful tool for ensuring you've followed each step correctly.

3. (Understanding Landau Notation) Consider the following three algorithms for a problem, where n is the size of the input:

   Algorithm A: Requires $3n^2 + 5n + 10$ operations.

   Algorithm B: Requires $7n \log_2(n) + 3n + 2$ operations.

   Algorithm C: Requires $5n^3 + 8n + 12$ operations.

(a) Using the Landau (Big O) notation, provide the time complexity for each algorithm.

(b) Explain in your own words what this means in terms of the algorithms' performance as the size of the input (n) grows.

(c) Assuming that $f(x) = 5x^3 + 8x + 12$, demonstrate the usage of Θ-notation to provide a tight bound for the function. Provide the values for c_1, c_2 and x_0.

Answer:

(a) Time complexity in Landau (Big O) notation:

- Algorithm A: The term with the highest growth rate is $3n^2$, so in Big O notation, the time complexity is $\mathcal{O}(n^2)$.
- Algorithm B: The term with the highest growth rate is $7n\log_2(n)$, so in Big O notation, the time complexity is $\mathcal{O}(n \log n)$.
- Algorithm C: The term with the highest growth rate is $5n^3$, so in Big O notation, the time complexity is $\mathcal{O}(n^3)$.

(b) Interpretation of the time complexity:

- Algorithm A: The time complexity of $\mathcal{O}(n^2)$ means that as the size of the input (n) grows, the number of operations required by the algorithm grows quadratically. If the input size doubles, the algorithm would approximately take four times the number of operations when the input size was n.
- Algorithm B: The time complexity of $\mathcal{O}(n \log n)$ means that the number of operations grows at a rate somewhere between a linear function and a quadratic function. This is often associated with algorithms that divide the problem into smaller parts, solve them independently, and then combine the results (like in divide-and-conquer strategies).
- Algorithm C: The time complexity of $\mathcal{O}(n^3)$ indicates that the number of operations grows cubically as the size of the input grows. This is typical of algorithms that involve three nested loops over the input data.

(c) Using Θ-notation for the function $f(x) = 5x^3 + 8x + 12$:

- Here, the function is clearly dominated by the $5x^3$ term as x grows larger. Therefore, we can say that $f(x) = \Theta(x^3)$.
- To provide a tight bound using Θ-notation, we need to find constants c_1, c_2 and x_0 such that $0 \leq c_1 x^3 \leq f(x) \leq c_2 x^3 \ \forall \ x \geq x_0$.
- Given the positive coefficients in the function, we can choose $c_1 = 1$, $c_2 = 6$ (a number greater than the leading coefficient in $f(x)$) and $x_0 = 1$.
- Therefore, we can say $f(x) = \Theta(x^3)$ with $c_1 = 1$, $c_2 = 6$ and $x_0 = 1$ which satisfies the inequalities for the Θ-notation.

4. (O-notation) Let (i) $f(n) = n + n^2$, (ii) $f(n) = n^2(1 + \sin(n\pi/2))$. For which $\alpha \in \mathbb{Z}$ is the following true as $n \to \infty$?

(a) $f(n) = \mathcal{O}(n^\alpha)$,

(b) $f(n) = \Omega(n^\alpha)$,

(c) $f(n) = \Theta(n^\alpha)$.

Answer: We are given the function $f(n) = n^2(1 + \sin(n\pi/2))$. To find an asymptotic tight bound using the Θ notation, we need to show that $f(n)$ grows at the same rate as some function $g(n)$, up to a constant factor. That is, we need to find functions $g_1(n)$ and $g_2(n)$ such that $f(n) = \Theta(g_1(n)) = \Theta(g_2(n))$.

First, note that $\sin(n\pi/2)$ is equal to 1 when n is odd, and 0 when n is even. Thus, we can write:

$$f(n) = \begin{cases} n^2(1+1) = 2n^2, & \text{if } n \text{ is odd} \\ n^2(1+0) = n^2, & \text{if } n \text{ is even} \end{cases}$$

For odd n, we have $f(n) = 2n^2$. We can show that $f(n) = \Theta(n^2)$ by choosing $g_1(n) = n^2$ and constants $c_1 = 1/2$ and $c_2 = 2$,

$$\begin{aligned} c_1 g_1(n) &= \frac{1}{2} n^2 \\ &\leq f(n) = 2n^2 \\ &\leq 2n^2 = c_2 g_1(n) \end{aligned}$$

Thus, $f(n) = \Theta(n^2)$ for odd n. For even n, we have $f(n) = n^2$. We can show that $f(n) = \Theta(n^2)$ by choosing $g_2(n) = n^2$ and constants $c_1 = 1$ and $c_2 = 1$,

$$\begin{aligned} c_1 g_2(n) &= n^2 \\ &\leq f(n) = n^2 \\ &\leq n^2 = c_2 g_2(n) \end{aligned}$$

Thus, $f(n) = \Theta(n^2)$ for even n.

Since $f(n) = \Theta(n^2)$ in both cases, we can conclude that $f(n)$ has an asymptotic tight bound of $\Theta(n^2)$.

5. This exercise is to practically investigate the concept of algorithm errors due to floating-point representation and arithmetic. Write a Python script to demonstrate the phenomenon of 'catastrophic cancellation', which can occur when subtracting two nearly equal floating-point numbers.

   - Define two floating point numbers that are extremely close to each other. Subtract one from the other and print the result.
   - Analytically, we know what the result of the subtraction should be. Compare this with the actual result computed by Python and compute the absolute and relative errors.
   - Now, add a small number (e.g., 0.001) to one of the original floating point numbers, subtract as before and calculate the absolute and relative errors. Discuss how the errors change and why.

   In these exercises, you are to observe the phenomena of algorithm errors due to floating point representation and approximation errors inherent in computer calculations. The understanding of these concepts is crucial for error analysis in numerical mathematics.