Implementing and Analyzing a Custom Square Root Function

Talha Ahmad, 400517273

September 26, 2024

1 Overview

This assignment required us to implement a square root function from scratch in C without the `<math.h>` library. In this report, I will be documenting the method I used, how to compile and run the code, the time complexity of the algorithm, and how the program compares to the `sqrt()` function from the `<math.h>` library. At the very end, there is an Appendix that contains the code for the custom square root function.

2 Method

I used the Newton-Raphson method to implement the square root function. The Newton-Raphson method is an iterative method that uses the tangent line of a function to approximate the root of the function. The formula for the Newton-Raphson method is as follows:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

For the square root function, since we know computing the square root is the same as finding the roots of the function $f(x) = x^2 - \text{number}$, we can use the Newton-Raphson method to find the square root of a number where the function is $f(x) = x^2 - \text{number}$. We can differentiate the function to get $f'(x) = 2x$. Then, we can use the formula above to find the square root of a number. This formula, after subbing in the function, becomes:

$$x_{n+1} = x_n - \frac{x_n^2 - \text{number}}{2x_n}$$

This formula can be applied iteratively to get a better and better estimate of the square root, until we reach the desired number of decimal places.

3 Compiling and Running the Code

To compile the code, run the following command in the terminal:

```
gcc -o sqrtUser sqrtUser.c
```

To run the code, run the following command in the terminal:

```
./sqrtUser
```

4 Time Complexity

The time complexity of the Newton-Raphson method is heavily dependent on the initial guess and the number of iterations. However, we can see that the Newton-Raphson method's system for making subsequnt guesses is logarithmic in nature. That is, the number of correct decimal places grows exponentially with each iteration. For example, here is the number of iterations required to get the square root of 2 to 13 decimal places broken down by decimal places:

Decimal Places	Iterations Required
1	1
2	2
3	3
4	4
5	4
6	4
7	5
8	5
9	5
10	5
11	5
12	5
13	6

We can see that as the decimal places increase, the accuracy of each iteration increases exponentially. By the 5th iteration, we already have 12 decimal places of accuracy. This is double the ENTIRE number of decimal places we had in all the previous iterations combined. This is why the time complexity of my implementation is $O(\log n)$.

5 Comparison with `sqrt()` Function

The `sqrt()` function from the `<math.h>` library is implemented using a more sophisticated algorithm than the Newton-Raphson method. This is because the `sqrt()` function is required to be fast and accurate for a wide range of inputs. Moreover, it is also optimized for different architectures and compilers.

For speed, the `sqrt()` function also uses a lookup table for small inputs. By doing so, the function can return the square root of small inputs in constant time. For this reason, it is faster than our implementation which is $O(\log n)$.

6 Appendix

The code for the custom square root function is as follows:

```c
#include <stdio.h>

// Function prototype
double sqrtUser(double number, int n);

int main() {

  // Loop to continue asking for user input and computing the square root
  while (1) {

    // Variables to store user input
    double number;
    int n;

    // Ask for the number the user wants to find the square root of. Also check and make sure it's a number
    // Scanf returns the number of items successfully read. If it's not 1, then the user didn't enter a number and we can exit the program.
    printf("Enter a number: ");
    if (scanf("%lf", &number) != 1) {
      printf("Invalid input\n");
      return 0;
    }

    // Ask for the number of iterations the user wants to use. Also check and make sure it's a number.
    printf("Enter the number of decimal places: ");
    if (scanf("%d", &n) != 1) {
      printf("Invalid input\n");
      return 0;
    }

    // If the number is less than or equal to 0, or the number of decimal places is less than 0, then the input is invalid.
    if (number <= 0 || n < 0) {
      printf("Invalid input\n");
      return 0;
    }

    // Print the result to the user inputted number of decimal places.
    printf("The square root of %lf accurate to %d decimal places is %.*lf\n\n", number, n, n, sqrtUser(number, n));
  }
```

```c
}

// Function to compute the square root of a number using the Newton-Raphson method
double sqrtUser(double number, int n) {

  // We can divide the accuracy by 10 for each decimal place we want to find
  // This gives us the allowed difference between two consecutive estimates
  double accuracy = 1;
  for (int i = 1; i < n; i++) {
    accuracy /= 10;
  }

  // Keep 2 variables for the numbers so we can find the difference between them
  double estimate = number / 2;
  double estimate_next = number;

  // Loop to keep computing the square root until the difference between the two estimates is less than the accuracy
  while (1) {

    // Compute f(x) = x^2 - number
    double f = estimate*estimate - number;

    // Compute f'(x) = 2x
    double f_prime = 2*estimate;

    // Now compute the next estimate using the formula: x_i+1 = x_i - f(x)/f'(x)
    estimate_next = estimate - f/f_prime;

    // If the difference between the two estimates is greater than the accuracy, then update the estimate and continue
    if (estimate-estimate_next > accuracy || estimate_next-estimate > accuracy) estimate = estimate_next;

    // If the difference between the two estimates is less than the accuracy, then we have found the square root
    else break;
  }

  // Return the estimate with the correct number of ACCURATE decimal places
  return estimate_next;
```

```
}
```