ID5130 ASSIGNMENT-1 REPORT

Name: Keerthiharan A Roll no: CH22B026

March 5, 2024

Question 1

. The calculations are handwritten and attached at the end.

Question 2

Given function is

$$f(x) = sin(5x) \quad for \quad 0 \leq x \leq 3. \tag{1}$$

The fourth-order accurate Padé scheme for the interior points is given by:

$$f'_{j+1} + 4f'_j + f'_{j-1} = \frac{3}{h}(fj + 1 - fj - 1) \tag{2}$$

The third-order accurate Padé scheme for the exterior points is given by:

$$f'_0 + 2f'_1 = \frac{1}{h}(-\frac{5}{2}f_0 + 2f_1 + \frac{1}{2}f_2) \tag{3}$$

$$f'_n + 2f'_{n-1} = \frac{1}{h}(\frac{5}{2}f_n - 2f_{n-1} - \frac{1}{2}f_{n-2}) \tag{4}$$

where h is the grid spacing and n is the number of grid points in the x direction. Exact solution is given by :

$$f(x) = 5cos(5x) \quad for \quad 0 \leq x \leq 3. \tag{5}$$

Upon Numerically solving this by,

(a) Serial Program for Derivative Computation using tridiagonal LU Decomposition

Taking number of grid points as 25, h = 3/25 The result obtained is plotted below

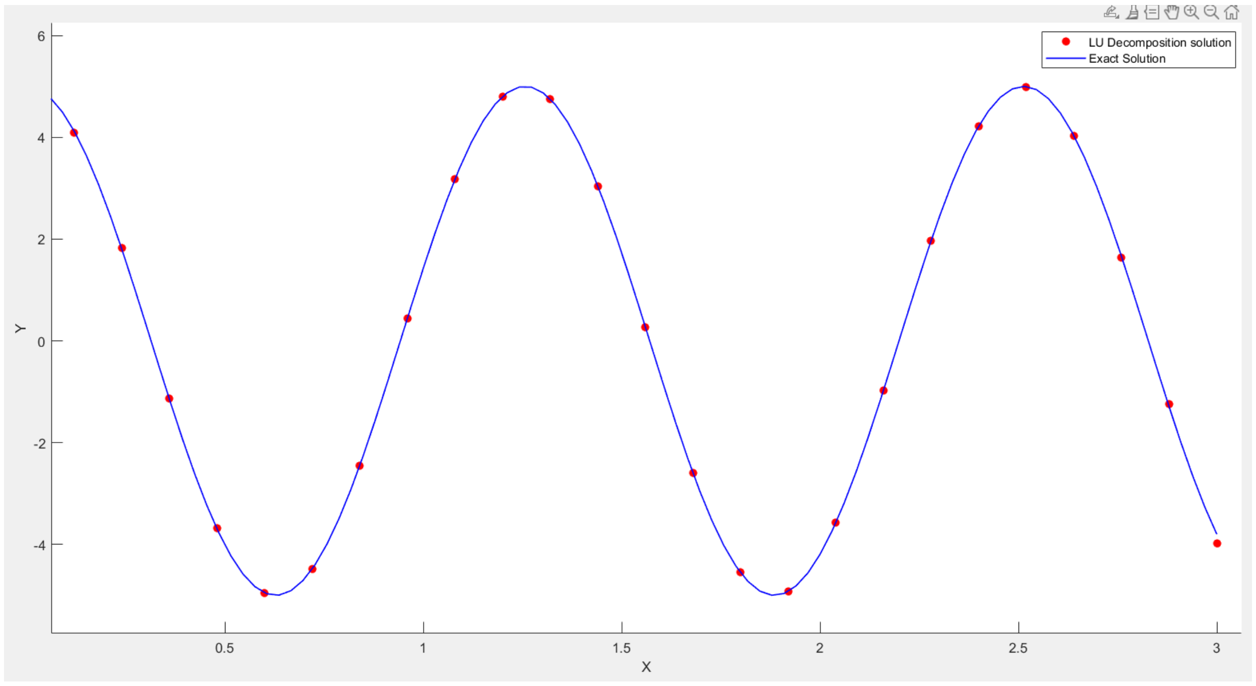

Figure 1: Analytical and Numerical Solutions for $n = 25$

(b) Using Recursive Doubling for Derivative Computation

Using OPENMP the Parallel version of Recursive Doubling method was implemented using 2 threads and solution obtained is plotted below,

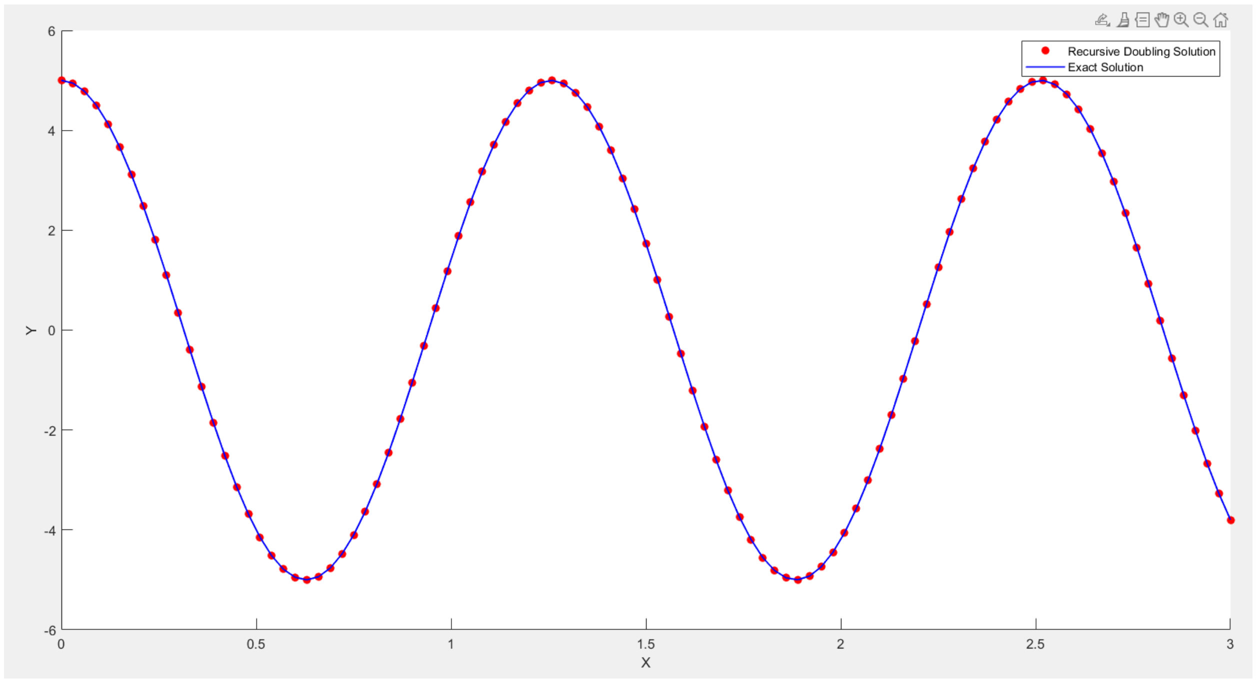

Figure 2: Analytical and Numerical Solutions for $n = 100$ with $p = 2$

The graph shows both recursive doubling gives answers very close to the exact solution as the n is large and this is 4th order accurate scheme.

Comparing the time it took for executing Recursive doubling using 2,4,8 threads

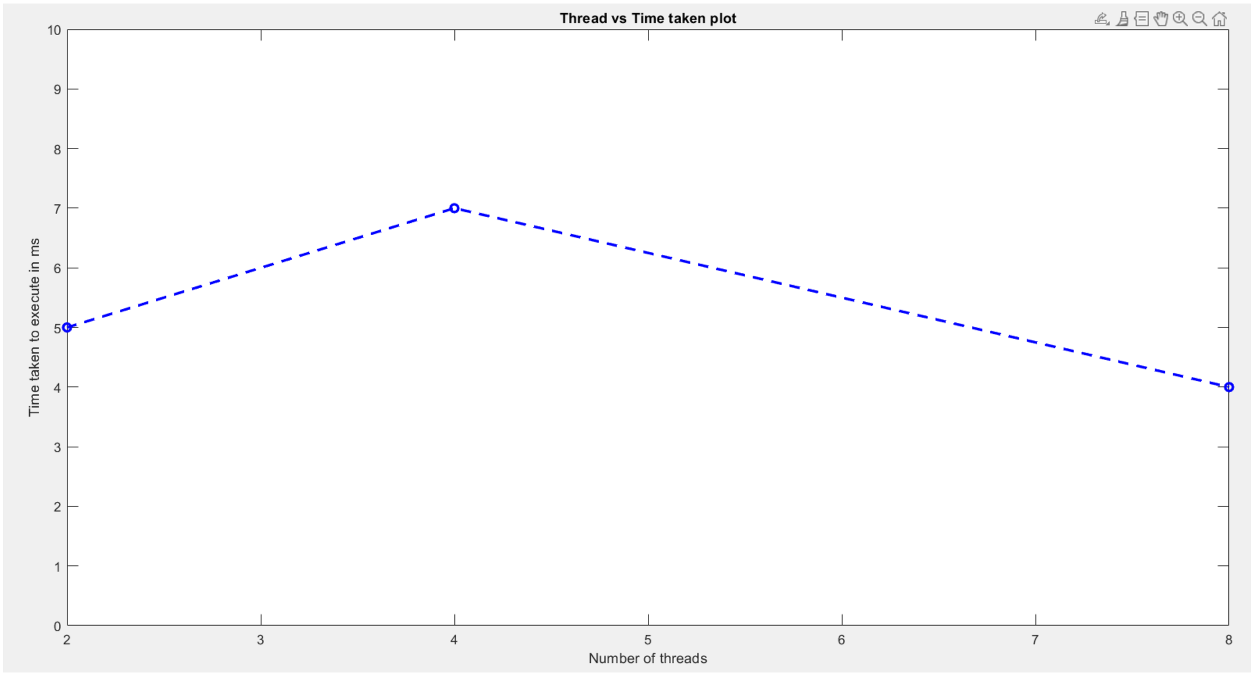

Figure 3: Time taken in Millie seconds vs number of threads for $n = 1000$

The graph shows 8 threads took less time than 2 and 4, which is expected but 4 threads took more time than 2 which is quite unexpected, this can be due to the fact that for 4 threads the performance is not much improved as it takes more time to fork and join that the time compensated by running in parallel.

Question 3

Consider the Poisson equation given by:

$$\nabla^2\phi = -q; \quad q = 2(2 - x^2 - y^2); \quad \phi(\pm 1, y) = 0; \quad \phi(x, \pm 1) = 0;$$

$$\phi_{i,j}^{(k+1)} = \frac{1}{4h}(\phi_{i+1,j}^{(k)} + \phi_{i-1,j}^{(k+1)} + \phi_{i,j+1}^{(k)} + \phi_{i,j-1}^{(k+1)}) + \frac{\Delta^2}{4}q_{i,j}$$

where, $\Delta = \Delta x = \Delta y$.

Initial guess: $\phi(x, y) = 0$ everywhere.

The exact solution is given by: $\phi = (x^2 - 1)(y^2 - 1)$. Solving this elliptic equation numerically,

(a) Serial Gauss-Seidel Method

Gauss Seidel is employed to solve the system to within 1% of the exact solution and the result is plotted. It took **191 iterations** and 0.005s to get solution with maximum 1% error of exact solution.

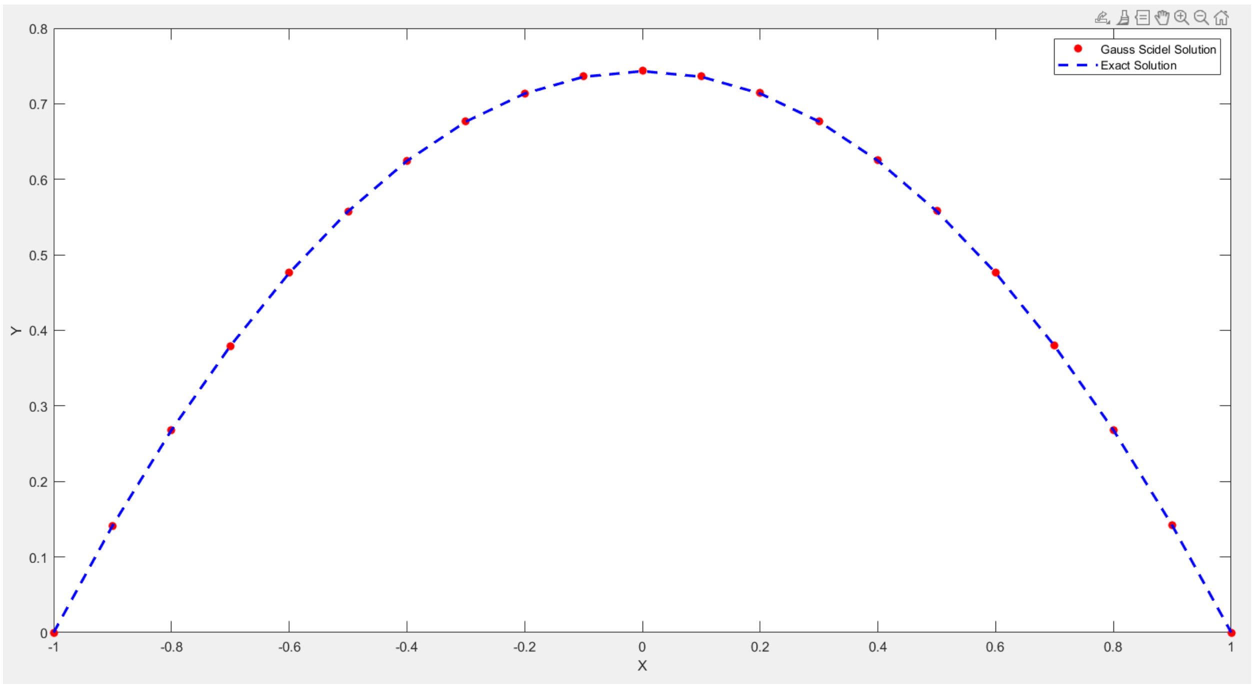

Figure 4: Gauss Seidel Solution of ϕ vs x for $y = 0.5$

(c) Verifying Parallel Solutions and Performance Analysis

The OpenMP program developed in part (b) for Gauss-Seidel method using the diagonal approach and red-black coloring approach is plotted at y = 0.5, both the results closely matches with serial Gauss Seidel method and exact solution.

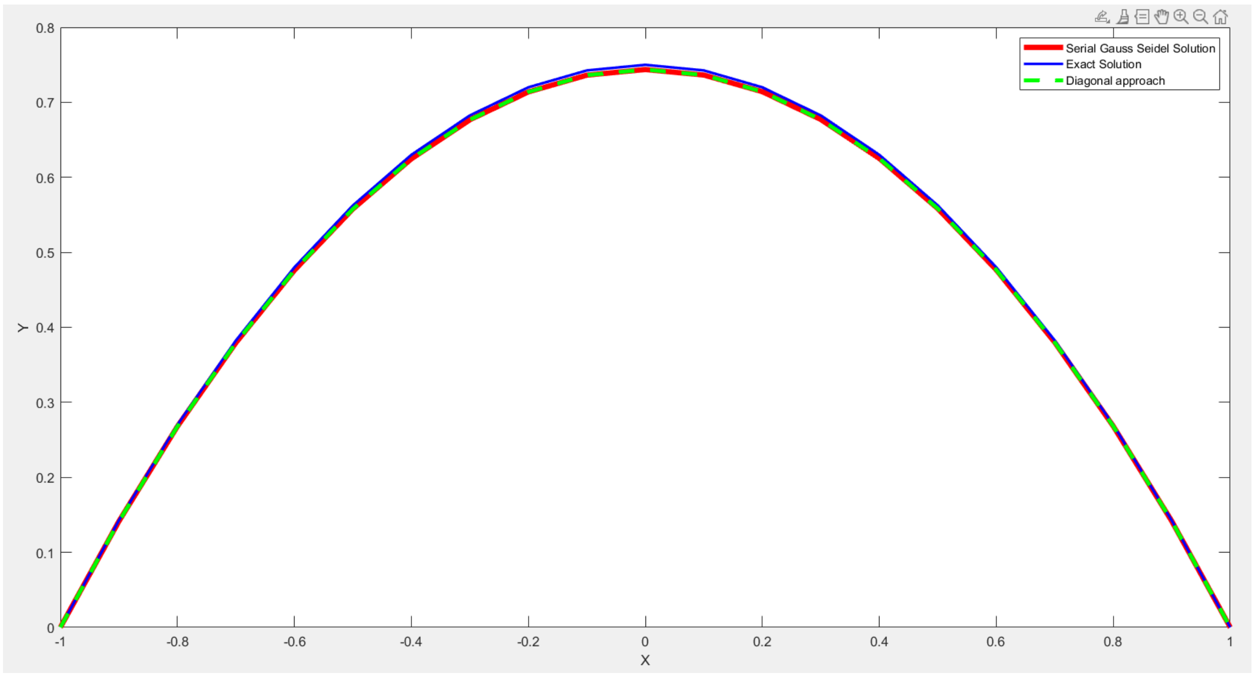

Figure 5: Guass Seidel, Red-Black and Diagonal approach Solutions of ϕ vs x for $y = 0.5$

Execution time for all three methods is plotted here the parallel programs were run on 8 threads.

n = [21 201 401];

delta = [0.1 0.01 0.005];

Time in seconds

red-black = [0.041 16.863 423.849];

serial = [0.005 38.892 780.086];

diag-approach = [0.38 448.157 3628.17];

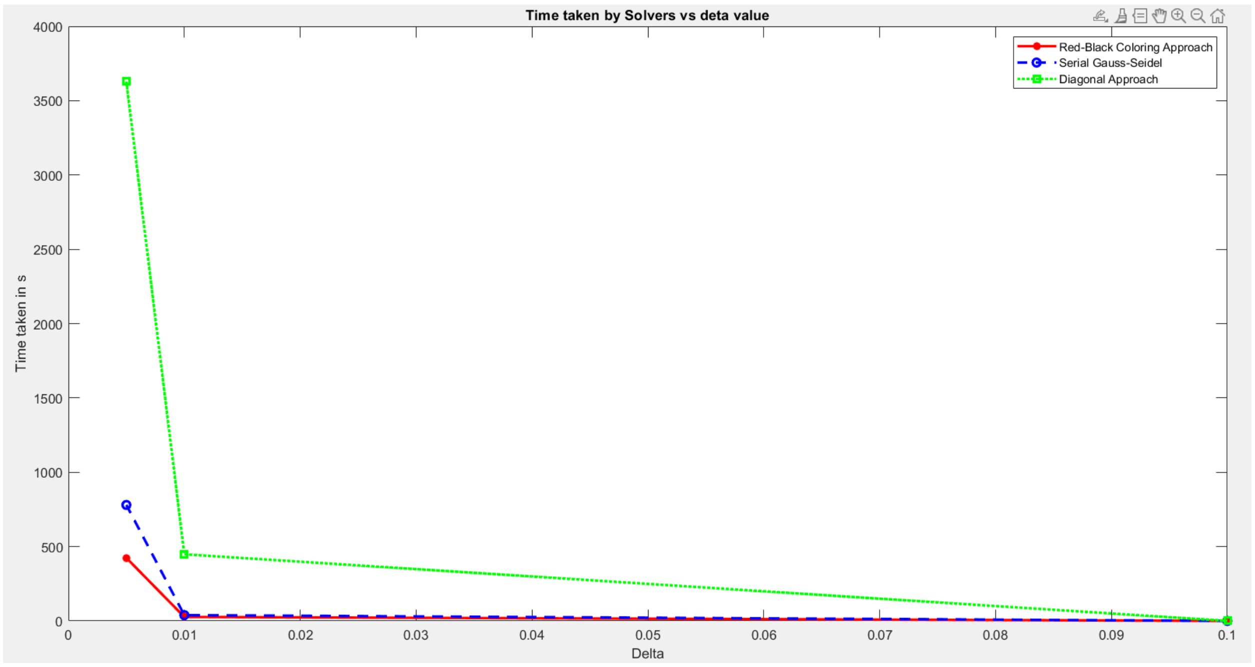

Figure 6: Time taken vs step size (delta)

The results shows Red-Black coloring is the best parallel method, even though all three methods have similar run time for large delta and small n, for large n Red-Black outperforms other methods showing better performance and Diagonal method shows the worst performance. Even though all the method converges in similar number of iterations.

(c) Comparing Parallel methods for $\Delta = 0.005$

The following plot gives execution time as a function of number of threads. As the number of threads should be proportional to the number of cores. Performance will drop off sharply as the number of threads exceeds the number of available cores. My System has only 4 processors hence increasing the threads brings down it's performance.

Number of threads = [2 4 8 16];

Red-black = [447.128 415.961 423.849 432.303];

Diagonal solver = [2335.63 3155.19 3684.42 4084.42];

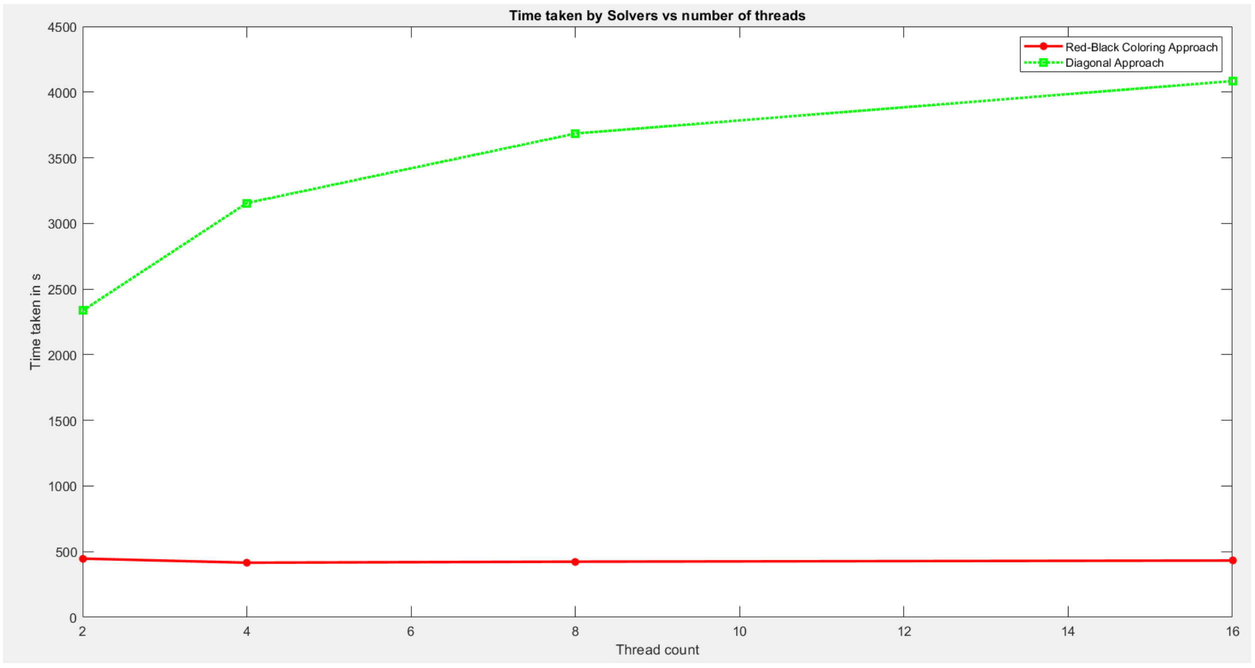

Figure 7: Time taken in seconds vs thread count (delta)

It can be inferred that **Red-Black coloring** is superior in performance to Diagonal approach