Solution of the 2D steady heat equation with the finite difference method

September 30^{th}, 2024

Ben Wilfong

Files

```
HW2_root
├── CMakeLists.txt
├── homework.pdf
├── main.inp
├── make.sh
├── src
│   ├── m_global_parameters.f90
│   ├── m_helpers.f90
│   ├── m_iterative_methods.f90
│   └── p_main.f90
└── plots
    ├── *.csv
    └── plots.m
```

Compiling

This code can be compiled in a terminal by running `chmod u+x ./make.sh` followed by `./make.sh` in the `HW2_root/` directory. This will compile and link the source code in the `src/` directory and create an executable called `main` in the `HW2_root/` directory.

Running

The problem parameters are defined in the namelist file `main.inp` in `HW2_root/`. The input parameters are:

- N: number of cells in each direction
- solver: which iterative method to use. 0 = Jacobi, 1 = Gauss-Seidel, 2 = SOR
- multigrid: logical to enable a 2 level algebraic multigrid method
- omega: omega factor used for the SOR iteration
- bench: logical enable multiple runs of the code for benchmarking runtime
- max_iter: maximum number of iterations allowed

Once the input parameters are set, the code is ran with `./main`. After execution, the solution is written to `HW2_root/T.csv` with the first row holding the x coordinates, the second rows holding the y coordinates, and the following rows holding the solution T. The residuals are written to `HW2_root/residuals.csv` where each column corresponds to an iteration.

1. Part A: Discretization as a System of Linear Equations

Suppose that the two-dimensional steady state heat equation

$$\lambda\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}\right)+Q=0 \tag{1}$$

is discretized onto $\{x_0, x_1, \ldots, x_N\}$ and $\{y_0, y_1, \ldots, y_N\}$ and that T_{ij} is the temperature at coordinate (x_i, y_j). The second order derivatives in the x- and y-directions can be approximated with the second order central discretizations:

$$\frac{\partial T_{ij}}{\partial x}\approx\frac{T_{i-1,j}-2T_{i,j}+T_{i+1,j}}{(\Delta x)^2} \qquad \frac{\partial T_{ij}}{\partial y}\approx\frac{T_{i,j-1}-2T_{ij}+T_{i,j+1}}{(\Delta x)^2}.$$

These approximations can be substituted into eq. (1) yielding:

$$\lambda\left(\frac{T_{i-1,j}+T_{i+1,j}-4T_{i,j}+T_{i,j-1}+T_{i,j+1}}{h^2}\right)=-Q_{i,j},$$

assuming that $h=\Delta x=\Delta y$. The Dirichelet boundary conditions on the left and right of the domain are given by $T_{0,j}=0$ and $T_{N,j}=2y_j^3-3y_j^2+1$. The Neumann boundary conditions at the top and bottom of the boundry are implemented via the ghost cells

$$T_{i,-1}=T_{i,1} \quad \text{and} \quad T_{i,N+1}=T_{i,N-1}.$$

This discretization can be cast as a block tridiagonal system of linear equations. The solution vector T is given by:

$$T=\begin{pmatrix}T_0\\T_1\\\vdots\\T_N\end{pmatrix} \text{ where } T_{j\in 0:N}=\begin{pmatrix}T_{0,j}\\T_{1,j}\\\vdots\\T_{N,j}\end{pmatrix}.$$

The right hand side b is given by:

$$b=\begin{pmatrix}b_0\\b_1\\\vdots\\b_{N-1}\\b_N\end{pmatrix} \text{ where } b_{j=0}=\begin{pmatrix}0\\0\\\vdots\\0\\0\end{pmatrix}, b_{j\in 1:N-1}=\begin{pmatrix}2y_j^2-3y_j^2+1\\-Q_{1,j}/\lambda\\\vdots\\-Q_{N-1,j}/\lambda\\0\end{pmatrix}, b_{j=N}=\begin{pmatrix}0\\0\\\vdots\\0\\0\end{pmatrix}.$$

The matrix A is given by:

$$A=\begin{bmatrix}B_0 & C_0 & & & \\ A_1 & B_1 & C_1 & & \\ & \ddots & \ddots & \ddots & \\ & & A_{N-1} & B_{N-1} & C_{N-1}\\ & & & A_N & B_N\end{bmatrix}. \tag{2}$$

The submatrices $A_j,\ B_j,\ C_j,\ X$, and Y are given by:

$$A_{j=1:N-1} = \begin{bmatrix} 1/2h^2 & & & & \\ & 1/2h^2 & & & \\ & & \ddots & & \\ & & & 1/2h^2 & \\ & & & & 1/2h^2 \end{bmatrix}, \quad A_{j=N} = \begin{bmatrix} -1 & & & & \\ & -1 & & & \\ & & \ddots & & \\ & & & -1 & \\ & & & & -1 \end{bmatrix},$$

$$B_{j=0} = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}, \quad B_{j=1:N-1} = \begin{bmatrix} 1 & & & & \\ 1/2h^2 & -4/h^2 & 1/2h^2 & & \\ & \ddots & \ddots & \ddots & \\ & & 1/2h^2 & -4/h^2 & 1/2h^2 \\ & & & & 1 \end{bmatrix},$$

$$B_{j=N} = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}, \quad C_{j=0} = \begin{bmatrix} -1 & & & & \\ & -1 & & & \\ & & \ddots & & \\ & & & -1 & \\ & & & & -1 \end{bmatrix},$$

$$C_{j=1:N-1} = \begin{bmatrix} 1/2h^2 & & & & \\ & 1/2h^2 & & & \\ & & \ddots & & \\ & & & 1/2h^2 & \\ & & & & 1/2h^2 \end{bmatrix},$$

Writing the system in this way is much more compact than attempting to write the matrix A explicitly. The A_i matrices account for the $T_{i,j-1}$ contributions. The C_i matrices account for the $T_{i,j+1}$ contributions. The B_i matrices account for the local left and right neighbor contributions. X and Y account for the $T_{i,j+2}$ and $T_{i,j-2}$ contributions to the one-sided second-order boundary condition discretizations. This discretization assumes that at the corners of the domain the direchlet boundary condition is enforced because at these locations, the neuman boundary connditions are incompatible with the dirichlet boundary conditions. Each submatrix in A is of size $N+1 \times N+1$. This means that the matrix A has a size of $(N+1)^2 \times (N+1)^2$ assuming a square grid with N^2 nodes. The total number of nonzero elements is only $11(N+1)^2 - 2(N+1)$, making this matrix quite sparse. For the case $N = 100$, only 0.11% of the elements of A are nonzero. As N increases, this percentage continues to decrease.

2. Part B: Iterative Method Forumlation

Jacobi Iteration

$$T_{i,j}^{k+1} = \frac{2h^2 Q_{i,j}}{4\lambda} + \frac{1}{4}\left(T_{i-1,j}^{k} + T_{i+1,j}^{k} + T_{i,j-1}^{k} + T_{i,j+1}^{k}\right)$$

Gauss-Seidel Iteration

$$T_{i,j}^{k+1} = \frac{2h^2 Q_{i,j}}{4\lambda} + \frac{1}{4}\left(T_{i-1,j}^{k+1} + T_{i+1,j}^{k} + T_{i,j-1}^{k+1} + T_{i,j+1}^{k}\right)$$

Sucessive Over-relaxtion Iteration

$$T_{i,j}^{k+1} = (1-\omega)T_{i,j}^{k} + \frac{2\omega h^2 Q_{i,j}}{4\lambda} + \frac{\omega}{4}\left(T_{i-1,j}^{k+1} + T_{i+1,j}^{k} + T_{i,j-1}^{k+1} + T_{i,j+1}^{k}\right)$$

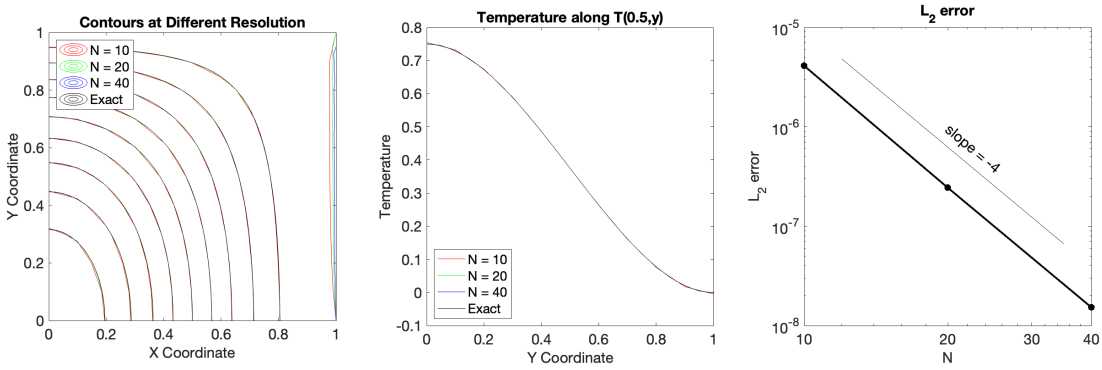

FIGURE 1. Left: Contours of solution at different N. Center: Comparision of solution along T(y,0.5). right: Error as a function of N.

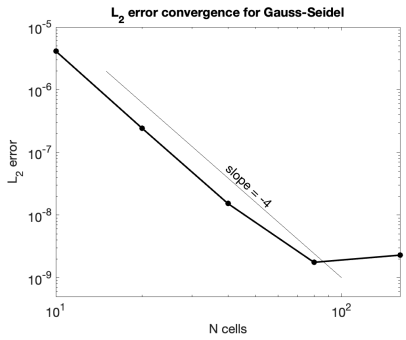

FIGURE 2. Error vs. N for the Gauss-Seidel iteration.

3. Part C: Jacobi Iteration

Figure 1 shows the solution contours, solution at $T(y, 0.5)$ and error convergence at three different grid resolutions. The iteration is terminated when the residual, defined as

$$R_{i,j} = \left| \left(\frac{T_{i-1,j} + T_{i+1,j} - 4T_{i,j} + T_{i,j-1} + T_{i,j+1}}{h^2} \right) + Q_{i,j} \right|,$$

is smaller than a specified tolerance for all i and j. For all computations in this report a tolerance of 1×10^{-4} is used. The error is computed using the root sum of square

$$E = \left(\sum_{\forall i,j} |S_{i,j} - T_{i,j}|^2 \right)^{1/2}$$

where $S_{i,j}$ is the exact solution at (x_i, y_j).

4. Part D: Spatial Convergence of Gauss-Seidel

Figure 2 shows the spatial convergence using Gauss-Seidel iteration. At small resolutions, the method converges with fourth order accuracy. At higher resolutions, the convergence flatlines due to the first order approximations used for the no flux boundary conditions.

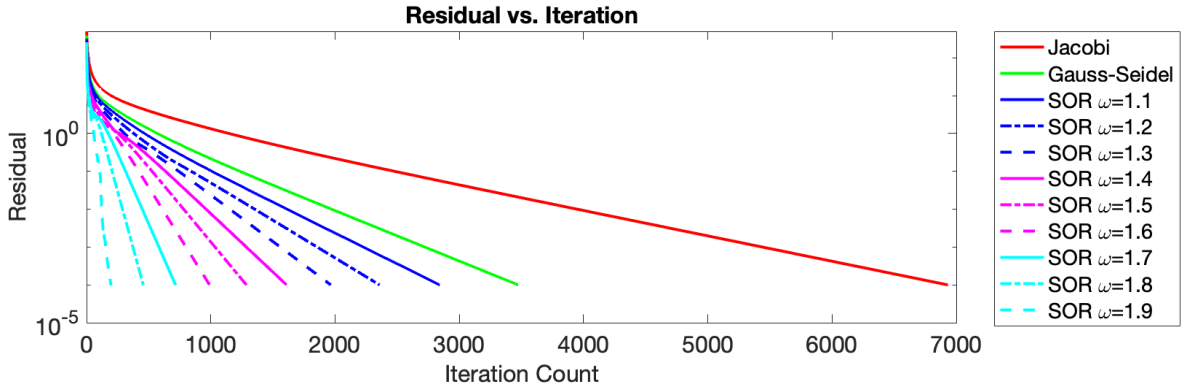

FIGURE 3. Residual vs. iteration count for the implemented iterative schemes.

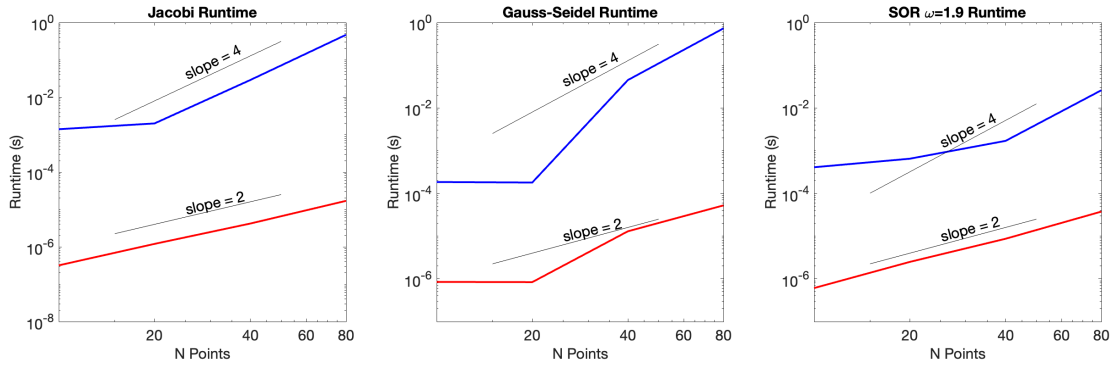

FIGURE 4. Runtime as a function of N

5. Part E: Residuals as a function of iteration

Figure 3 shows the residual as a function of iteration count for Jacobi Iteration, Gauss-Seidel iteration, and Successive overrelaxtion at a variety of ω's. The convergence of SOR monotonically improves at ω ranges from 1.1 to 1.9. Above $\omega = 1.9$ however more iterations are required and the method eventually diverges. Jacobi iteration takes the longest to converge, followed by Gauss-Seidel while SOR converges the quickest as expected.

6. Part F: Runtime

Figure 4 shows the runtime as a function of N for Jacobi iteration, Gauss-Seidel Iteration, and SOR. Red lines show the average time per iteration and blue lines show the time to convergence. The average time per iteration appears to have $\mathcal{O}(N^2)$ time complexity for all methods which is expected because the number of unknowns grows with N^2. Total run time appears to have $)(N^4)$ time compleity, suggesting that the number of iterations required to reach convergence grows with N^2. This trend seems reasonable given that the number of unknowns grow with N^2 and information can only travel from each grid point and its neighbors at each iteration.

7. Part G: 2-Level Multigrid

Figure 5 shows the time to convergence for all four methods. The algebraic multigrid approach is two orders of magnitude faster than the simple single mesh iterative approaches. The implemente

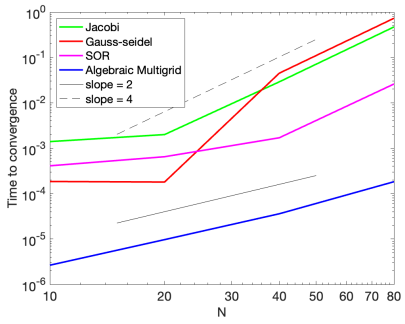

FIGURE 5. Time to convergence comparision of all methods

two level multigrid method performs five smoothing iterations on the fine mesh before solving for the error on the coarse mesh and applying the correction to the fine mesh. While Jacobi, Gauss-Seidel, and SOR all require hundreds to thousands of iterations with N =4, the algebraic multigrid converges in just six iterations comprising of 30 gauss-seidel iterations on the fine mesh.

Your feedback says "results?" with no indication of what results your're looking for. The question states "Solve the problem using a 2-level geometric multigrid method. Compare the computational time with other methods.". It appears to me that the expected results are a comparision of the computational time with the other methods, which I clearly present. I'm not sure why points were deducted for this. I gave exactly what was asked for.