Finite Difference Solver for 2D Elliptic Problems on a Rectangle with Mixed Neuman-Dirchlet Boundary Condtions

Jay Roberts

August 22, 2017

Let $\mathcal{R} = [0, d] \times [-d, d]$. The equation we are considering is

$$\begin{cases} L \cdot D^2 u + D^2 u^T M D^2 u &= F \\ u_x(x, 0) &= 0 \\ u(x, -d) = u(x, d) = u(d, y) &= 0 \end{cases} \tag{1}$$

Where

$$L : \mathbb{R}^2 \to \mathbb{R}^5 \quad M : \mathbb{R}^2 \to \mathbb{R}^{5\times 5} \quad F : \mathbb{R}^2 \to \mathbb{R}$$

and

$$D^2 u^T = (u_x, u_y, u_{xx}, u_{yy}, u_{xy})$$

We note that the question of well-posedness for such problems is non-trivial and not uniersially guarenteed; however, we postpone this question and assume that the L, M, and F, lead to a well posed problem for the remainder of the paper.

1 Discretization:

We use a central difference method on the discretized grid

$$x_i = (i + \frac{1}{2})h \quad y_j = (j - N)h \qquad \begin{matrix} i \in \{0, 1, \ldots, I\} \\ j \in \{0, 1, \ldots, 2I\} \end{matrix}$$

setting $u_{i,j} = u(x_i, y_j)$ the differential approximations at (x_i, y_j)

$$\begin{aligned} \partial_x u &\approx \tfrac{1}{2h}(u_{i+1,j} - u_{i-1,j}) & \partial_{xy} u &\approx \tfrac{1}{4h^2}(u_{i+1,j+1} - u_{i-1,j+1} - u_{i+1,j-1} + u_{i-1,j-1}) \\ \partial_{xx} u &\approx \tfrac{1}{h^2}(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}) & \partial_{yy} u &\approx \tfrac{1}{h^2}(u_{i,j+1} - 2u_{i,j} + u_{i,j-1}) \end{aligned}$$

The Neumann boundary condtion is imposed by requiring the artificial boundary points satisfy

$$u_{0j} = u_{-1j} \tag{2}$$

and the Dirchlet by

$$u_{Ij} = u_{(I+1)j} = u_{i0} = u_{i-1} = u_{i2I} = u_{i,2I+1} = 0. \tag{3}$$

Let U be the matrix of approximate values on the grid, that is

$$U_{ij} = u_{ij}.$$

Further we let $N(i,j)$ be the nine neighbors of the approximation value at (x_i, y_j) ordered lexicographically

$$\begin{pmatrix} u_{i+1,j-1} & u_{i+1,j} & u_{i+1,j+1} \\ u_{i,j-1} & u_{i,j} & u_{i,j+1} \\ u_{i-1,j-1} & u_{i-1,j} & u_{i-1,j+1} \end{pmatrix} \quad \rightarrow \quad \begin{pmatrix} 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{pmatrix}.$$

For points on the **interior** of the grid the discretizatio of D^2u at the point (x_i, x_j) becomes

$$D^2u(x_i, y_j) \approx \begin{pmatrix} 0 & -\frac{1}{2h} & 0 & 0 & 0 & 0 & 0 & \frac{1}{2h} & 0 \\ 0 & 0 & 0 & -\frac{1}{2h} & 0 & \frac{1}{2h} & 0 & 0 & 0 \\ 0 & \frac{1}{h^2} & 0 & 0 & \frac{-2}{h^2} & 0 & 0 & \frac{1}{h^2} & 0 \\ 0 & 0 & 0 & \frac{1}{h^2} & \frac{-2}{h^2} & \frac{1}{h^2} & 0 & 0 & 0 \\ \frac{1}{4h^2} & 0 & -\frac{1}{4h^2} & 0 & 0 & 0 & -\frac{1}{4h^2} & 0 & \frac{1}{4h^2} \end{pmatrix} N(i,j) = PN(i,j) \tag{4}$$

We will abreviate $N(i,j)$ by N when there is no danger of confusion. The boundary points are handled seperately for the linear and quadratic portions of the approximation, but in the end our discretization produces a system of quadratic equations of the form

$$AU = b + f(U) \tag{5}$$

The solution will be solved iteratively using SOR for the non-linear part. Specifically if $U^{(n-1)}$ is the $n-1$ stage approximation we define $\tilde{b}^{(n-1)} = b + F(U^{(n-1)})$ and set $U_*^{(n-1)}$ to be the intermediate solution, gotten by solving

$$AU_*^{(n-1)} = \tilde{b}^{(n-1)}$$

The scheme is then updated through SOR by

$$U^n = \omega U_*^{(n-1)} + (1-\omega)U^{(n-1)}$$

1.1 Linear Part

At a point (x_i, y_j) in the **interior** of $\mathcal{R}$ the linear equation obatined from the descritization is

$$LPN(i,j) = b_{ij}$$

The way we think about this is LP is a 9×1 maxtrix that we then convert to a 3×3 matrix which we imbed into a large matrix of zeros, a matrix the size of our descritized grid, and then unravel that lexicographically to get a row. This adds the extra step of broadcasting the row as a matrix in the middle, but it makes implementation easier. For concreteness let $LP = [a_1, a_2, \dots, a_9]$, the process is then

$$[a_1, a_2, \dots, a_9] \;\; \overrightarrow{fold} \;\; \underbrace{\begin{pmatrix} a_7, a_8, a_9 \\ a_4, a_5, a_6 \\ a_1, a_2, a_3 \end{pmatrix}}_{\mathcal{A}} \;\; \overrightarrow{embed} \;\; \begin{pmatrix} 0 & \dots & j & \dots & 0 \\ \vdots & \ddots & & & \\ i & & \mathcal{A} & & \vdots \\ \vdots & & & \ddots & \\ 0 & & \dots & & 0 \end{pmatrix} \;\; \overrightarrow{unwrap} \quad row_{ij}$$

In this way we get a row for each point on the interior of the grid. For the boundary we wrote out the LPN and then imposed the conditions from (2) and (3) and applied a similar construction as above.

1.2 Quadratic Part

Our quadratic descritization uses the current state of our approximation $U^{(n)}$. To handle the boundary we pad the grid with zeros on the $i = N+1$, $j = \pm(N+1)$, boundary and we copy the U_{0j} row for the $i = -1$ boundary giving a new grid

$$\tilde{U}^{(n)} = \begin{pmatrix} 0 & U^{(n)}_{0j} & 0 \\ 0 & U^{(n)} & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

From this we approximate the derivatives at (x_i, y_j) with the central differnece scheme

$$D^2 u \approx PN(i,j)$$

where again $N(i,j)$ is the list of neighbors of the i, j^{th} grid-point taken in lexicographic order.

$$Q_{ij} = N^T P^T M_{ij} PN$$

2 Results

The method was tested on the PDE

$$LD^2u = D^2uMD^2u + F \tag{6}$$

With

$$L = [1.0, 1.0, 2 - x, 2 - y, 1]$$

$$M = \begin{pmatrix} 1 + x^2 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & x + y & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

The forcing term is choosen so that the following function is a solution

$$u(x, y) = (x^2 - 1)(y^2 - 1) + \cos(\pi x/2)\sin(\pi y)$$

Remark: This function is not compactly supported, or more precisely compactly supported in a small neighborhood of the rectangle, which was assumed in the descritization of the quadratic term. Because of this running the solver as is results in errors on the boundary which propogates through the domain. To remedy this we added a "test" argument which when true pads the grid with the correct values for the fucntion used in testing and thus eliminating the boundary error.

The solver was tested on a grid size of $(N + 1) \times (2N + 1)$ for $N \in \{10, 15, 20, \ldots, 80\}$ with the L^∞ norm as our error function. The linear solver was tested as well by constructing the appropriate forcing function for the same test function.

From the table in Figure (2) we see that our error was still on the order of 10^2 even with the most refined grid tested. Sicne our data was on order 1 this is less than ideal. Further, below we have log plots of error and run time with the slope of their best fit lines.

We see that in Figure (2) both the linear solver and full solver have a quadratic rate of convergence which is in line with the theorey. The run time seems to be increasing quadratically but then its rate increases for a more refined mesh. More data must be collected to better understand the time requirements for the solver.

Nodes	Linear Error	Solver Error	Run Time
10	5.3492101784	6.3206347837	0.4862530231
30	0.5626120219	0.5748473484	4.5167329311
50	0.1999783197	0.2015669124	15.2254259586
70	0.1014555545	0.1018693084	44.2371621132
80	0.0775382182	0.0777807885	69.9071290493

Figure 1: Table of Errors and Run Times

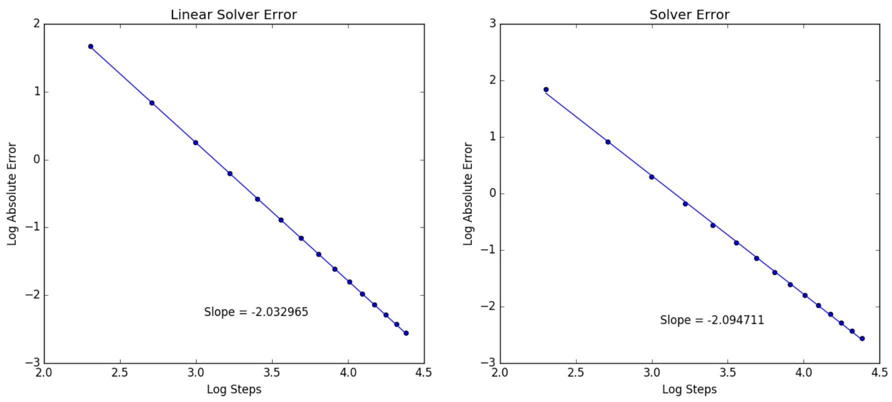

Figure 2: Log-Absolute error for Linear and Full Solvers

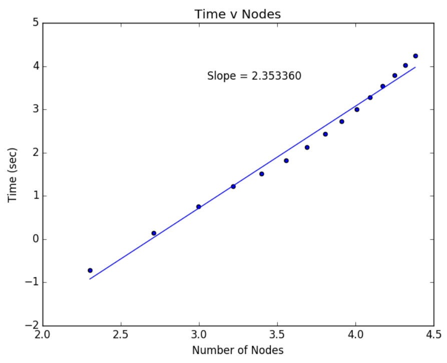

Figure 3: Log-Time for runs

3 Conclusion

A second order solver based on a central difference scheme and SOR approximation for a class of elliptic problems with quadratic nonlinearities was developed and tested. The second order convergence of the algorithm was verified numerically to agree with theoretical predictions. The run time seemed to increase close to quadratically with the mesh size but more analysis is needed.

Specific improvements to the algorithm:

Modify Domain: It would be useful to allow for non-centered and less symmetric rectangular domains. This would be easy to implement in the current form of the code.

Improve Linear Solver: Currently the linear solver uses the Scipy Sparse Linear Algebra routine splu to compute the LU Decomposition of the linear piece and then applies its solve routine. While this greatly increases speed over a basic solver, the linear matrix is not in the correct sparse form and so the solve routine is not as efficient as it could be. The next approach is to use the Scipy Sparse Incomplete LU decomposition and use this as a preconditioner for a conjugate gradient solver. This still would have the Sparse Efficency problem, but it would only be at the decomposition step not the solving step. This too could be implemented easiily in the current form of the code.

Multi-Grid: A slightly more ambitious task would be to modify the code to allow for mutigrid methods. Specifically in the SOR portion a multi-grid approach could dramatically decrease computation time and may prove necessary for high resolution computations. This would likely take signifcant modification of the current code.

Other imporvements that would be beneficial such as a richer geometry of domain or higher convergence rate would require substantially different numerical techniques, such as FEM or collocation methods, and so are beyond the scope of this paper.