## Problem Set 6 Solutions

1. Show that the error in the centered difference approximation of the first derivative is second order. In other words, for $D_0u(x) = \dfrac{u(x+h) - u(x-h)}{2h}$, show that $E_0 = D_0u(x) - u'(x) = \mathcal{O}(h^2)$.

   Taylor series gives us :

   $u(x+h) = u(x) + hu'(x) + \dfrac{h^2}{2}u''(x) + \dfrac{h^3}{3!}u'''(x) + \dfrac{h^4}{4!}u^{(4)}(x) + ...$

   and $u(x-h) = u(x) - hu'(x) + \frac{h^2}{2}u''(x) - \frac{h^3}{3!}u'''(x) + \frac{h^4}{4!}u^{(4)}(x) - ...$

   Therefore,

   $$D_0u(x) = \frac{\left(u(x) + hu'(x) + \frac{h^2}{2}u''(x) + \frac{h^3}{3!}u'''(x) + \frac{h^4}{4!}u^{(4)}(x) + ...\right) - \left(u(x) - hu'(x) + \frac{h^2}{2}u''(x) - \frac{h^3}{3!}u'''(x) +}{2h}$$

   $$= \frac{2hu'(x) + \frac{2h^3}{3!}u'''(x) + \frac{2h^5}{5!}u^{(5)}(x) + ...}{2h}$$

   $$= u'(x) + \frac{h^2}{3!}u'''(x) + \frac{h^4}{5!}u^{(5)}(x) + ...$$

   Therefore, $E_0 = D_0u(x) - u'(x) = \frac{h^2}{3!}u'''(x) + \frac{h^4}{5!}u^{(5)}(x) + ...$

   Then, incorporating $\frac{u'''(x)}{3!}$ into $C$, we see $E_0 = Ch^2 +$ lower order terms.

   Therefore $E_0 = \mathcal{O}(h^2)$.

2. Show that finite difference approximation of the second derivative given by

   $$u''(x) \approx D_2u(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} \tag{1}$$

   is also second-order accurate.
   (Note: This also means that the five-point finite difference approximation of the 2-D Laplacian (that we went over in class) is also second-order accurate. )

   Using the same Taylor Series expansions from problem (1), we get

   $$D_2u(x) = \frac{\left(u(x) + hu'(x) + \frac{h^2}{2}u''(x) + \frac{h^3}{3!}u'''(x) + ...\right) + \left(u(x) - hu'(x) + \frac{h^2}{2}u''(x) - \frac{h^3}{3!}u'''(x) + ...\right) - 2u(x)}{h^2}$$

   $$= \frac{h^2u''(x) + \frac{2h^4}{4!}u^{(4)}(x) + ...}{h^2}$$

   $$= u''(x) + \frac{2h^2}{4!}u^{(4)}(x) + ...$$

   Therefore, $E_2 = D_2u(x) - u''(x) = \frac{2h^2}{4!}u^{(4)}(x) + ...$, so $E_2 = \mathcal{O}(h^2)$.

3. Write a Matlab function to use 5-point 2-D Laplacian to estimate $\Delta u(x, y)$ at *one* point, $(x, y)$. Your function should take as inputs the *function* $u$, as well as $x$,$y$, and $h$. Use it to approximate the value of $\Delta u(1, 2)$ for $u = e^{xy} + x^2$. Use `u=@(x,y) exp(x*y)+x^2` to define this function in your script. Perform a refinement study, using relative errors, for values of $h = 2^{-2}, 2^{-3}, ..., 2^{-12}$ and demonstrate that the finite difference operator is second-order accurate. Explain how you have demonstrated this.

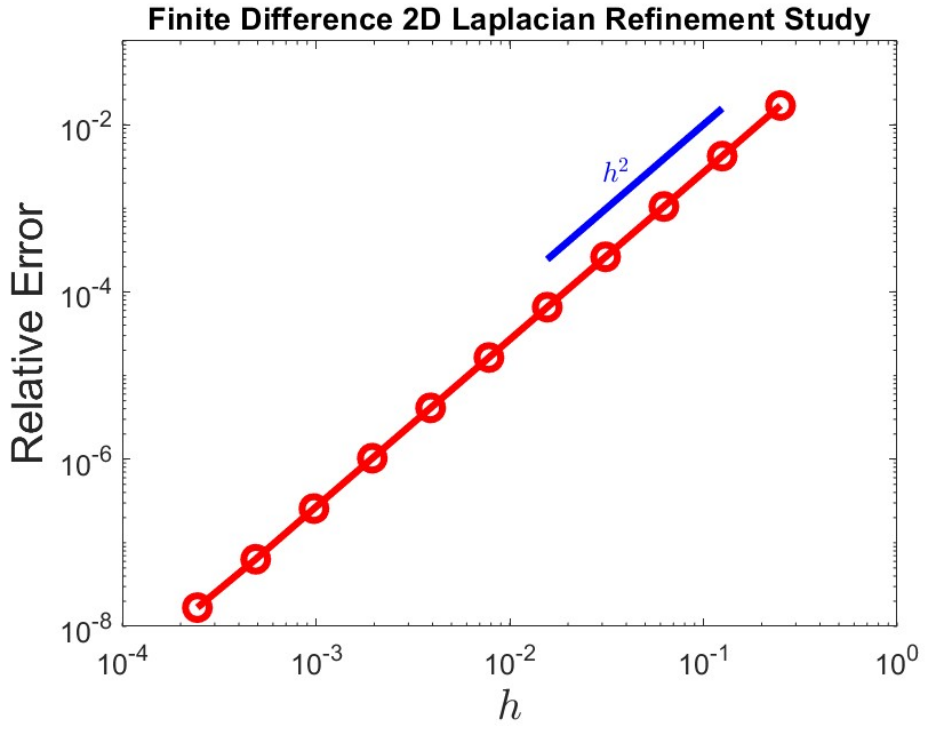
See PS6N3.m for the script and Laplacian2D_onept.m for the function. Below is the refinement study. To get the relative error, I first found the analytical value.

$u_x = ye^{xy} + 2x \implies u_{xx} = y^2 e^{xy} + 2$

$u_y = xe^{xy} \implies u_{yy} = x^2 e^{xy}$

$\implies \Delta u = (x^2 + y^2)e^{xy} + 2.$

The red line gives the refinement study, and the blue line gives a line with a slope of 2, which I obtained by doing a loglog plot of a function of the form $Ch^2$. Since these are parallel, we can confirm that the slope of the red line is 2, and hence our finite difference approximation is second-order.



4. Consider the PDE given by $u''(x) = \sin(\pi x)$ on $[0,1]$ such that $u(0) = 0$ and $u(1) = 1$.

(a) Solve this boundary value problem analytically.

$u''(x) = \sin(\pi x) \implies u'(x) = -\frac{1}{\pi}\cos(\pi x) + C_1 \implies u(x) = -\frac{1}{\pi^2}\sin(\pi x) + C_1 x + C_2$

Then $0 = u(0) = C_2$ and $1 = u(1) = C_1$

Therefore, the solution is $u(x) = -\frac{1}{\pi^2}\sin(\pi x) + x$

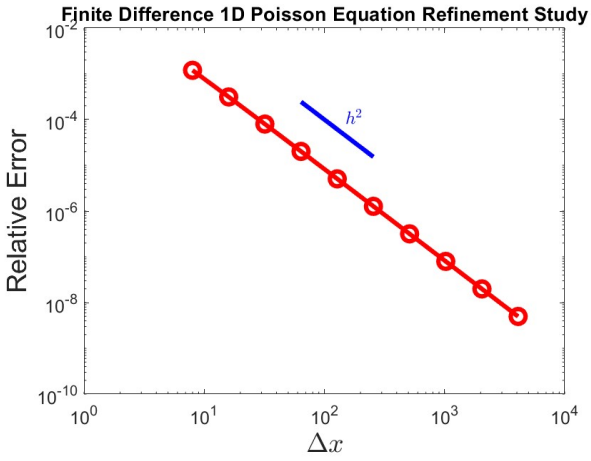
(b) Using the second-order approximation of the second derivative, write the associated linear algebra problem that can be solved to approximate the solution to this PDE for a given value of $h$. For $[a,b]$, use gridpoints located at $x_0 = a, x_1 = a+h, x_2 = a+2h, ...x_N = b-h, x_{N+1} = b$. Therefore, your unknowns will be $u_1, ..., u_N$, and you will have $h = \frac{b-a}{N+1}$.

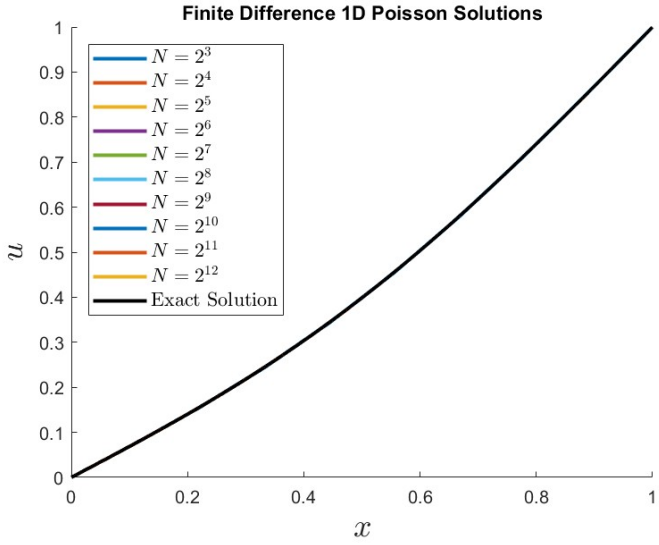
The system is $A\mathbf{u} = \mathbf{b}$ for

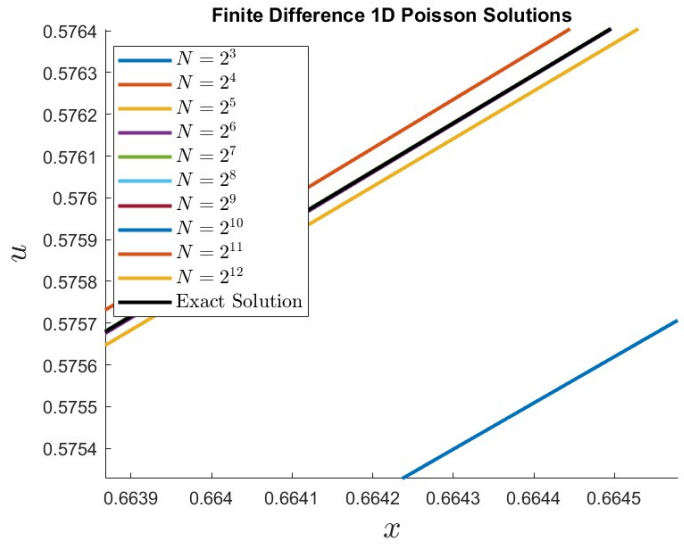
$$A = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & 0 & & & 0 \\ 1 & -2 & 1 & & & \\ 0 & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ 0 & & & & 1 & -2 \end{bmatrix}, \mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{bmatrix}, \text{ and } \mathbf{b} = \begin{bmatrix} \sin(\pi h) \\ \sin(2\pi h) \\ \sin(3\pi h) \\ \vdots \\ \sin(N\pi h) - \frac{1}{h^2} \end{bmatrix}$$

(c) Solve this PDE in Matlab with Matlab's backslash for values of $N = 2^3, 2^4, ..., 2^{12}$. Perform a refinement study, using the relative error of your solution, in the max norm. In your refinement study log log plot, use $N$ for the $x$-axis. (This is more natural than the use of $h$ that we must use for problem (3). This is because convergence shows error going down as you increase $N$, left to right.) What order of convergence do you get? Also plot your approximate solutions and the true solution.

See PS6N4c.m for the script. It was written to be more general than just this problem. As we can see by comparing to a line of slope $-2$, we get second order convergence for our solution using this finite difference discretization. In the solution plots, we really can't distinguish between the different solutions because the error is so low. Therefore, we can zoom in on one area to get a better look, and we see that as we increase $N$, ie as we refine the grid, our solution gets closer to the exact solution.


Finite Difference 1D Poisson Equation Refinement Study
Relative Error
$10^{-2}$
$10^{-4}$
$10^{-6}$
$10^{-8}$
$10^{-10}$
$h^2$
$10^0$
$10^1$
$10^2$
$10^3$
$10^4$
$\Delta x$


Finite Difference 1D Poisson Solutions
$u$
$N = 2^3$
$N = 2^4$
$N = 2^5$
$N = 2^6$
$N = 2^7$
$N = 2^8$
$N = 2^9$
$N = 2^{10}$
$N = 2^{11}$
$N = 2^{12}$
Exact Solution
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0.2
0.4
0.6
0.8
$x$


Finite Difference 1D Poisson Solutions
$u$
$N = 2^3$
$N = 2^4$
$N = 2^5$
$N = 2^6$
$N = 2^7$
$N = 2^8$
$N = 2^9$
$N = 2^{10}$
$N = 2^{11}$
$N = 2^{12}$
Exact Solution
0.5764
0.5763
0.5762
0.5761
0.576
0.5759
0.5758
0.5757
0.5756
0.5755
0.5754
0.6639
0.664
0.6641
0.6642
0.6643
0.6644
0.6645
$x$

5. Consider the PDE given by $\Delta u = -e^{(x-0.25)^2+(y-0.25)^2}$ on $[0,1] \times [0,1]$, with *homogeneous Dirichlet boundary conditions*, which means that $u = 0$ on the border around the square domain. (Note: This simplifies the right-hand-side of the linear algebra problem that results from using finite differences.)

   (a) (10 points)If we use finite differences, as described in 2-D in class, what is the right-hand-side of the resulting linear algebra equation? Then, create this in Matlab for $N = 2^2$.

   Note: For the Matlab part, even though the right-hand-side is a vector, I recommend finding the values in a *matrix* first, using the grid given by :
   `xg=h*(1:N)+xmin; yg=h*(1:N)+ymin; [xg,yg]=ndgrid(xg,yg);`
   Once you have the above grid, you can define a function `f` and then do `f(xg,yg)`. This will give you the values of $f$ on the entire grid, and this is your right-hand-side, in matrix form.
   Then use the `reshape` command to change it to a vector.

   Note: Again, your *interior* gridpoints, for which your solution is unknown should run from $x_1 = a + h, x_2 = a + 2h, ...x_N = b - h$ an similarly for $y$.

The right hand side is
$$\begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ \vdots \\ f_{N1} \\ f_{12} \\ f_{22} \\ f_{32} \\ \vdots \\ f_{N2} \\ \vdots \\ f_{1N} \\ f_{2N} \\ f_{3N} \\ \vdots \\ f_{NN} \end{bmatrix} = \begin{bmatrix} f(\mathbf{x}_{11}) \\ f(\mathbf{x}_{21}) \\ f(\mathbf{x}_{31}) \\ \vdots \\ f(\mathbf{x}_{N1}) \\ f(\mathbf{x}_{12}) \\ f(\mathbf{x}_{22}) \\ f(\mathbf{x}_{32}) \\ \vdots \\ f(\mathbf{x}_{N2}) \\ \vdots \\ f(\mathbf{x}_{1N}) \\ f(\mathbf{x}_{2N}) \\ f(\mathbf{x}_{3N}) \\ \vdots \\ f(\mathbf{x}_{NN}) \end{bmatrix} = \begin{bmatrix} f(h,h) \\ f(2h,h) \\ f(3h,h) \\ \vdots \\ f(Nh,h) \\ f(h,2h) \\ f(2h,2h) \\ f(3h,2h) \\ \vdots \\ f(Nh,2h) \\ \vdots \\ f(h,Nh) \\ f(2h,Nh) \\ f(3h,Nh) \\ \vdots \\ f(Nh,Nh) \end{bmatrix} = \text{for } f(x,y) = -e^{(x-0.25)^2+(y-0.25)^2}$$

The code is in PS6N5.m For $N = 4$, I got the following rhs:

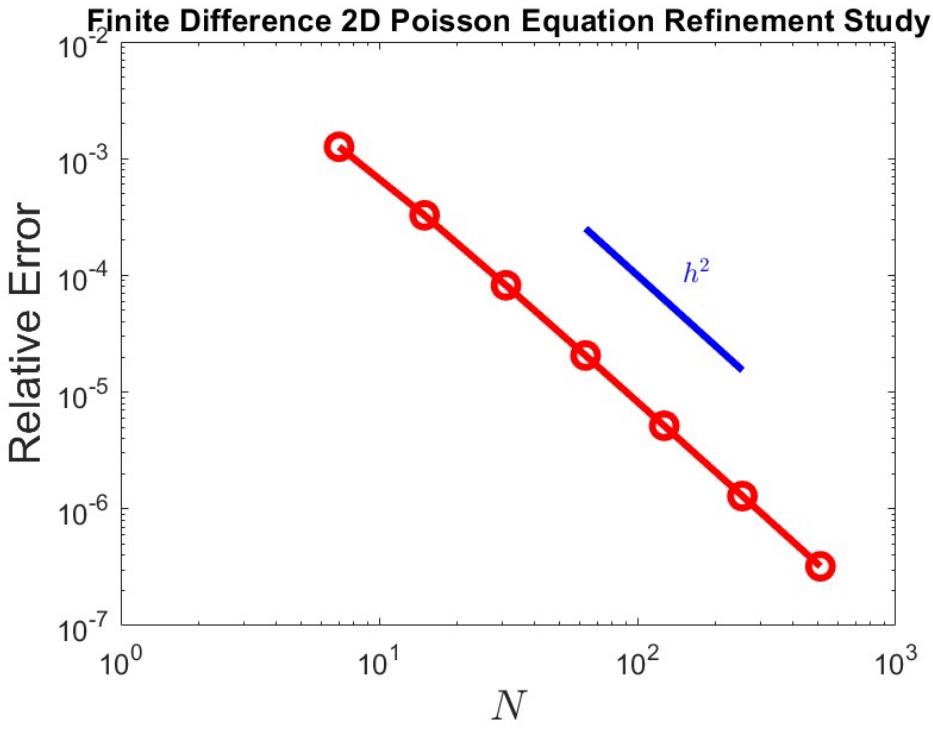
$$\begin{bmatrix} -1.005012520859401 \\ -1.025315120524429 \\ -1.133148453066826 \\ -1.356625003006224 \\ -1.025315120524429 \\ -1.046027859908717 \\ -1.156039570268022 \\ -1.384030645980752 \\ -1.133148453066826 \\ -1.156039570268022 \\ -1.277621313204887 \\ -1.529590419663379 \\ -1.356625003006224 \\ -1.384030645980752 \\ -1.529590419663379 \\ -1.831252208885773 \end{bmatrix}.$$

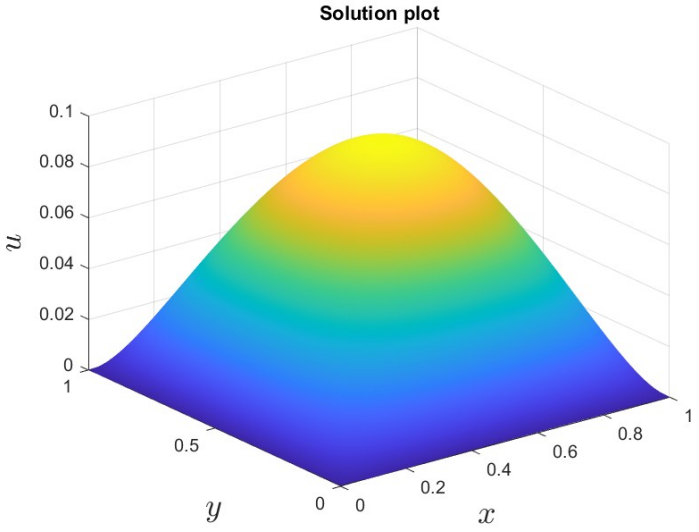
(b) A matlab function, `lap2d.m`, is provided in Catcourses which gives the 2-D discrete Laplacian matrix, without the $1/h^2$ factor. Use this and part (a), and solve the system using `backslash` to solve the system for $N = 2^3 - 1, 2^4 - 1, ...2^{10} - 1$.

Provide a refinement study. Instead of using relative error, use the max norm of the difference of successive solutions. To do this, find the difference between your solutions at $N = 2^3$ and $N = 2^4$ and then the difference of your solutions at $N = 2^4$ and $N = 2^5$, etc. For each of those pairs, evaluate these differences on the coarser mesh. Hints: You will find it easier to restrict the finer solution to the coarser grid if you first reshape it back into a matrix. Then, the "-1" in the values of $N$ will allow you to easily do this restriction.

What order of convergence do you see? Also use the command `mesh` to provide a 3-D plot of the solution on the finest grid.

See PS6N5.m for the code. We again see second-order convergence of the max norm difference in my successive differences.


Finite Difference 2D Poisson Equation Refinement Study
Relative Error
$10^{-2}$
$10^{-3}$
$10^{-4}$
$10^{-5}$
$10^{-6}$
$10^{-7}$
$10^{0}$
$10^{1}$
$10^{2}$
$10^{3}$
$N$
$h^2$


Solution plot
$u$
0.1
0.08
0.06
0.04
0.02
0
1
0.5
0
$y$
0
0.2
0.4
0.6
0.8
1
$x$

6. (a) Write a Matlab function to solve $A\mathbf{x} = \mathbf{b}$ using the Jacobi iterative method. It should take as an input $A$,$\mathbf{b}$, an initial guess $\mathbf{x}_0$, a tolerance, and a maximum number of iterations (optional). It should give as output the approximate solution $\mathbf{x}$, and the number of iterations performed. Use the *relative* stopping criteria of your choice. What did you use? Note: In class, I wrote it in component form, and therefore used a loop over $i$. Try to code it without such a loop in order to increase efficiency.

See Jacobi.m for the function. My method stops when $||\mathbf{r}^k||_2 < tol(||\mathbf{b}||_2)$. You may also use $||\mathbf{r}^k||_2 < tol(||\mathbf{x}^k||_2)$, $||\mathbf{x}^{k+1} - \mathbf{x}^k||_2 < tol(||\mathbf{x}^k||_2)$, or $||\mathbf{x}^{k+1} - \mathbf{x}^k||_2 < tol(||\mathbf{b}||_2)$, and you can also use other norms.

(b) Write a Matlab function to solve $A\mathbf{x} = \mathbf{b}$ using the Gauss-Seidel iterative method. It should take as an input $A$,$\mathbf{b}$, an initial guess $\mathbf{x}_0$, a tolerance, and a maximum number of iterations (optional). It should give as output the approximate solution $\mathbf{x}$, and the number of iterations performed. Use the same stopping criteria as you did for part (a). Make sure to not use any built-in Matlab functions to invert any matrices.

See GaussSeidel.m for the function. Note: You should not be finding inverse matrices when you need to apply $M^{-1}$. Instead, you should use your forward substitution function.

(c) Use your functions to solve $A\mathbf{x} = \mathbf{b}$ for $A$, a $1000 \times 1000$ matrix.

In class, we discussed how/when you would know that these Splitting methods would converge using the eigenvalues, but we did not discuss what matrices you would therefore get convergence for. To ensure that you are using a matrix for which these methods will converge, make a matrix that is *strictly diagonally dominant*, which means that, for every row, the magnitude of the diagonal entry is larger than the sum of the magnitudes of the other entries in the row, ie

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}| \tag{2}$$

Do this by creating a random matrix of numbers from 0 to 10 and then changing the diagonal entries to ensure the above relationship holds.

What tolerance did you use? Time these two methods, as well as using LU decomposition with pivoting and Matlab's backslash. Compare the relative differences, in max norm, between your solutions and Matlab's in order to validate your codes. Also report the number of iterations the methods required. Comment on your results.

See PS6N6.m for the code. Below gives the elapsed time for my problem for each of the methods, as well as the iteration counts and relative differences, with respect to the

max norm for the two iterative methods. I used a tolerance of $10^{-10}$. A few things stand out. Firstly, the slowest method was Jacobi. It also took many more iterations than Gauss-Seidel. This agrees with our discussion in class that $D^{-1}$ is not a very good approximation for $A^{-1}$, and we therefore expect slow convergence. However, by just including the lower triangular part of $A$, we are able to get much faster convergence for Gauss-Seidel, in only 15 iterations. Furthermore, it was able to do this faster than the LU decomposition with pivoting, demonstrating the usefulness of iterative methods when it is not necessary to get even closer to the exact solution. Matlab's direct solve was completed in even less time, so if we want to beat theirs, we would want to explore iterative methods that converge even more quickly.

| Method | Time | Iteration Count | $\|\|x - x_M\|\|_\infty / \|\|x_M\|\|_\infty$ |
|---|---|---|---|
| Jacobi | 5.86 s | 22901 | $3.7(10)^{-11}$ |
| Gauss-Seidel | 0.096 s | 15 | $1.3(10)^{-10}$ |
| LU decomp | 2.44 s | | |
| Matlab | 0.016 s | | |

7. The function `FDjacobi_2D.m` gives the Jacobi method for this finite difference implementation of the Poisson equation, and `FDGaussSeidel_2D.m` gives one version of the Gauss-Seidel method for this finite difference implementation of the Poisson equation. What does one step of the Gauss-Seidel method look like (using the same notation as the end of page 4 of Lecture 17 notes)? What is the difference between it and Jacobi method?

   Use the Gauss-Seidel method to solve the PDE from number (5) for $N = 2^3, 2^4, ...2^6$ with a tolerance of $10^{-8}$.

   The function `FDSOR_2D.m` gives a related method called SOR, or Successive Over-Relaxation. Use this to solve the PDE from number (5) for $N = 2^3, 2^4, ...2^9$ with a tolerance of $10^{-8}$.

   For GS and SOR, how many iterations were required for each solve? Comment on your results. Also comment on why we would solve it this way instead of the way we did it in number (5).

   One step of the Gauss-Seidel method presented here is

$$u_{ij}^{k+1} = \frac{1}{4}(u_{i-1,j}^{k+1} + u_{i,j-1}^{k+1} + u_{i,j+1}^{k} + u_{i+1,j}^{k}) - \frac{h^2}{4} b_{ij} \tag{3}$$

   The difference between this and Jacobi is that the first two terms in the sum are from the *new* step $(k+1)$ instead of the previous step $k$.

   See PS6N7.m for the code. The iterations are given in the table below. The iteration counts from the SOR method are much better than those from Gauss-Seidel, and as the grid is refined, the ratio between them grows, so for fine grids, the improvement of SOR is significant. Another thing to notice is that the iteration counts greatly increase as we refine the grid. This matches what we said in class that we can tell from the eigenvalues that as we decrease $h$, the convergence rate slows. Therefore, if we need to use an iterative method, it would be much better to use one whose convergence rate does not change as we change $h$ and/or with a faster rate.

   We would solve it with an iterative method instead of directly as we did in number (5) due to the memory needed. For this problem, $N = 2^{12}$ begins to run into memory issues for a standard computer with our LU decomposition with partial pivoting and for $N = 2^{13}$ for the Matlab backslash, whereas we are able to handle this using this iterative method (though you were not asked to because the time would still be large). The issue comes about much more quickly in 3-D, where $N = 2^8$ would result in memory issues.

| Iteration Counts | | | | | | | |
|---|---|---|---|---|---|---|---|
| Method | $N = 2^3$ | $N = 2^4$ | $N = 2^5$ | $N = 2^6$ | $N = 2^7$ | $N = 2^8$ | $N = 2^9$ |
| Gauss-Seidel | 133 | 440 | 1513 | 5289 | | | |
| SOR | 69 | 93 | 145 | 246 | 438 | 804 | 1504 |