Computational Physics
Homework 7

Kieran Finn

November 6, 2014

1 Problem a

write a program to solve the 2D Poisson equation in a rectangular region with Dirichlet BCs by the Gauss-Seidel method.

1.1 Description of program

The main programs used for this were *relaxation_algorithms.py* and *main.py*. *relaxation_algorithms.py* contains the method *SOR*, which carries out one iteration of the SOR algorithm given the current phi, a grid containing the RHS and the lattice spacing l, which is a 2-vector. Since Gauss-Seidel is simply a a special case of SOR I use the method *GS* for this question, which just calls SOR with $\omega = 1$.

I use a unique ordering system whereby I start with the outermost layer and spiral inwards until I reach the centre. In this way I hope to transport the information about the boundaries to the centre in the quickest amount of time.

2 Problem b

Test the program on a known function.

2.1 description of the program

I choose to use the test function

$$\phi(x, y) = 3x + 2y + 4\sin(3x)\cos(y) \tag{1}$$

so that the equation I am solving is

$$\nabla^2\phi(x,y) = -40\sin(3x)\cos(y) \tag{2}$$

complete with appropriate boundary conditions.

I initialise my grid phi to be all zero and then set the boundary using the exact function. I then iterate the GS algorithm using the function described in the previous section. At each iteration I calculate the norm of the error, which I do by taking the average of $|\nabla_N^2\phi - f|$ over the entire grid (excluding boundaries), where ∇_N^2 is the discretised Laplace operator and f is the RHS of the Poisson equation.

I also create a 3d plot of the current phi, as well as the theoretical value and string these together into a video in the end.

2.2 experimental results

Figure 1 shows the development of the errors with iteration. We see that for the 11×11 grid, we very quickly enter the asymptotic region where the error follows an exponential. However, the 101×101 grid features a lot more transience and, indeed, if we look at the corresponding video, we see it is a lot slower to converge.

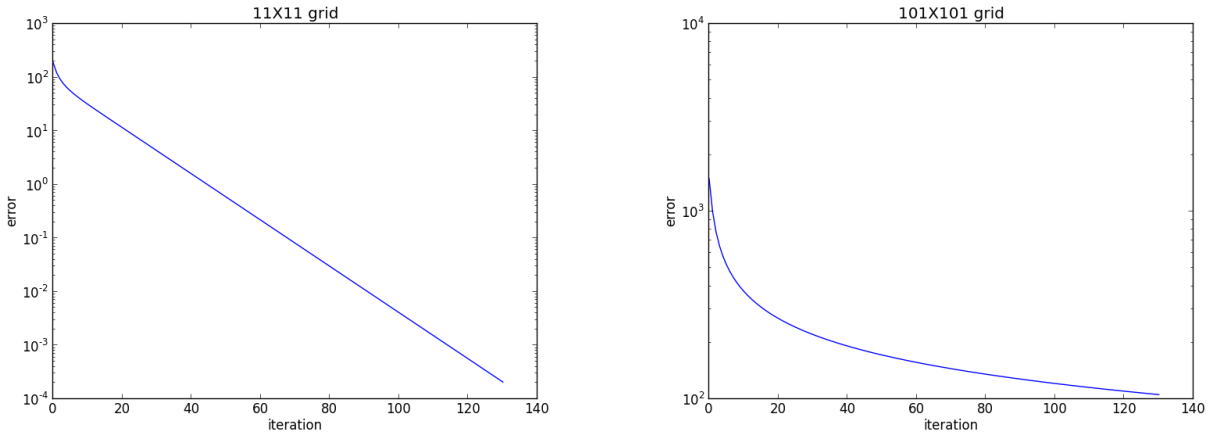

Figure 1: Evolution of errors

3 Problem c

generalise to SOR algorithm

3.1 description of the program

See section 1.

4 Problem d

Measure convergence rate as a function of lattice size (I use N instead of L) and ω.

4.1 Description of the program

For this problem I used *convergence.py* to generate the data and *plotting.py* to plot it.

convergence.py is based heavily on *main.py* including the same test function, the same norm and the same initialisation (although this time we iterate over N and ω. For each value of N and ω I iterate a given (N-dependent) number of times to remove any transience. I then iterate a further 20 times, recording the ratio of each successive pair of errors (calculated using the norm function described above). The average of these is then used as my best estimate o the convergence rate.

I also use the test function $\phi(x, y) = 2 + 3x - 2y + xy$ with $\nabla^2\phi = 0$ to compare.

4.2 Experimental results

Figure 2 shows the convergence rate verses ω for various values of the lattice spacing, N. We see that, although it has roughly the same shape as the theoretical curve, it is a little bit higher. I'm not sure why this is, perhaps a bug in my program.

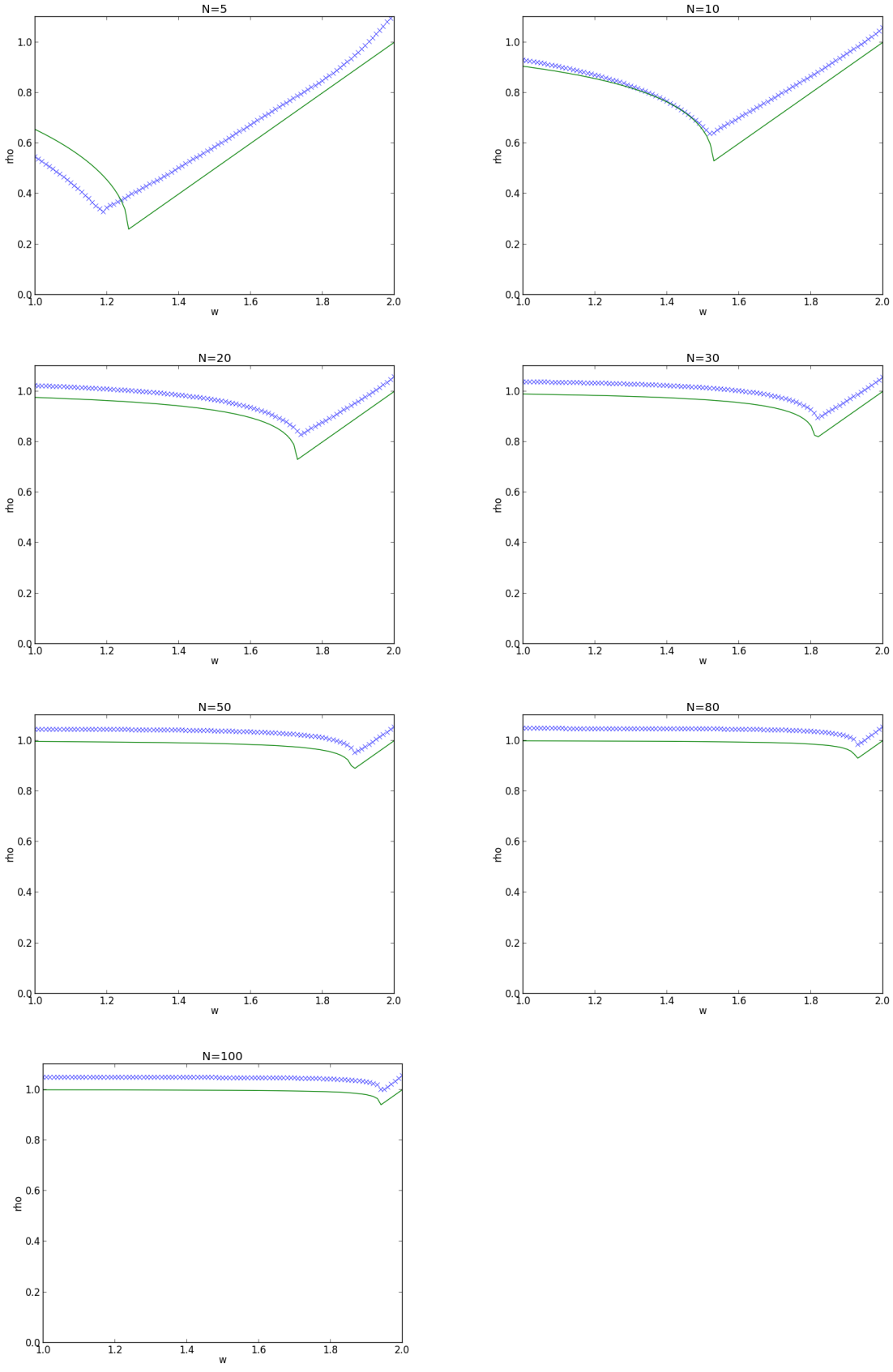

Figure 2: Convergence rates verses ω for different values of the lattice spacing, N.

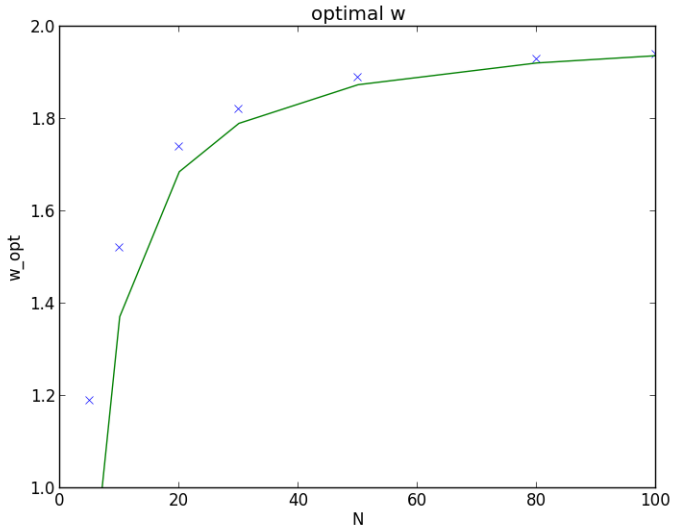

Figure 3: Optimal ω verses lattice spacing

Figure 3 shows how the optimal ω scales with lattice spacing. Again there is a small discrepancy with the theory but I have roughly the right shape.

Figure 4 shows convergence rates verses ω for the function $\phi(x, y) = 2 + 3x - 2y + xy$ and a lattice size of 20, showing that it does not depend on the boundary data.

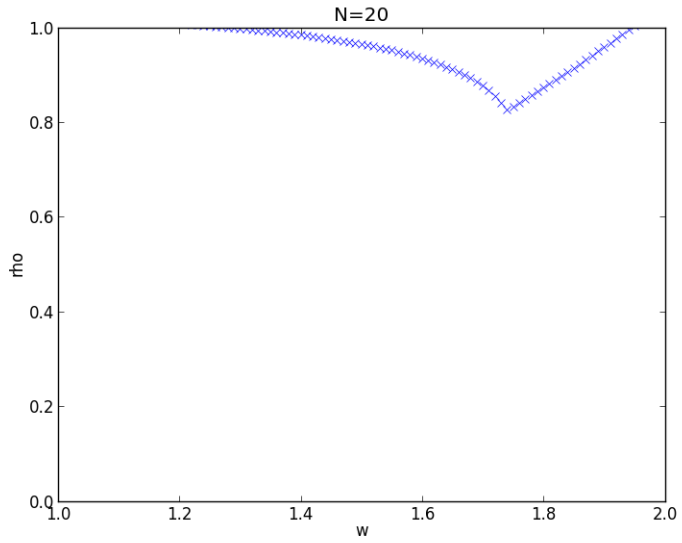

Figure 4: Convergence rate verses ω for the function $\phi(x, y) = 2 + 3x - 2y + xy$.

5 Problem e

Compare to theory

5.1 Theoretical results

5.1.1 Jacobi eigenvalues in 2D

We start by finding the eigenvalues for the 2D Jacobi algorithm. We will solve this based on the knowledge of the 1D eigenvalues, namely

$$\mu_i^{(1)} = \cos\left(\frac{i\pi}{l}\right). \tag{3}$$

let the eigenvector associated with eigenvalue $\mu_i^{(1)}$ be v_i, so that $D^{(1)}v_i = \mu_i^{(1)}v_i$, where $D^{(1)}$ is the discrete laplace operator in 1D. The 2D laplace operator is just 2 copies of the 1D, $D^{(2)} = D_x^{(1)} + D_y^{(1)}$, we therefore consider the ansatz $\phi_{i,j}(x, y) = v_i(x)v_j(y)$. Since $= D_x^{(1)}$ will not be affected by anything that only varies in the y direction and vice versa, we immediately see that

$$D^{(2)}\phi_{i,j} = (\mu_i^{(1)} + \mu_j^{(1)})\phi_{i,j} \tag{4}$$

so that the 2D eigenvalues are the sums of any two of the 1D eigenvalues, i.e. $\mu^{(2)} = 2\cos\left(\frac{i\pi}{N}\right) + 2\cos\left(\frac{j\pi}{N}\right) - 4$. However, what we really need are the eigenvalues of the Jacobi

iteration matrix which is given by

$$J^{(2)} = \frac{1}{4}(D^{(2)} + 4I) \tag{5}$$

This shows us that the eigenvalues are

$$\mu = \frac{1}{2}\left[\cos\left(\frac{i\pi}{N}\right) + \cos\left(\frac{j\pi}{N}\right)\right] \approx 1 - \left(\frac{\pi}{2N}\right)^2 (i^2 + j^2) \tag{6}$$

for any $i, j \in \mathbb{Z}$.

5.1.2 Eigenvalues for SOR

Using the results of Varga et al we know the eigenvalues of SOR satisfy

$$(\lambda + \omega - 1)^2 = \mu^2\omega^2\lambda. \tag{7}$$

The LHS of this equation can be expanded to give

$$\lambda^2 + \left(2\omega - 2 - \mu^2\omega^2\right)\lambda + (\omega - 1)^2 = 0. \tag{8}$$

The optimal value of lambda occurs when the discriminant of this quadratic exactly equals 0, since this is the point where the eigenvalues leave the real axis and start moving along the circle in the complex plane of radius $(\omega - 1)$. This can be seen because, when the discriminant is negative

$$\lambda = \frac{-b}{2} \pm i\sqrt{ac - \left(\frac{b}{2}\right)^2} \tag{9}$$

$$\Rightarrow \quad \Im(\lambda)^2 + \Re(\lambda)^2 = ac = r^2 \tag{10}$$

Therefore the optimal ω satisfies

$$\left(\mu^2\omega^2 - 2(\omega - 1)\right)^2 - 4(\omega - 1)^2 = 0 \tag{11}$$

$$\Rightarrow \quad \mu^2\omega^2 - 2(\omega - 1) = 2(\omega - 1) \tag{12}$$

$$\Rightarrow \quad \omega^2 - \mu^2\omega^2 = 4 - 4\omega + \omega^2 \tag{13}$$

$$\Rightarrow \quad \omega\sqrt{1 - \mu^2} = 2 - \omega \tag{14}$$

$$\Rightarrow \quad \omega = \frac{2}{1 + \sqrt{1 - \mu^2}} \tag{15}$$

we take $\mu = \cos\left(\frac{\pi}{N}\right)$, which is the largest eigenvalue and we find

$$\omega = \frac{2}{1+\sin\left(\frac{\pi}{N}\right)} \tag{16}$$

$$\approx 2 - \frac{2\pi}{N} \tag{17}$$

Therefore the optimal convergence rate is

$$\lambda_{opt} = \omega_{opt} - 1 = 1 - \frac{2\pi}{N} \tag{18}$$

comparing this to Gauss-Seidel, which can be found by plugging $\omega = 1$ into equation 8

$$\lambda_{GS} = \mu^2 \approx 1 - \frac{\pi^2}{N^2} \tag{19}$$

we see that we have reduced the critical slowing down from $\frac{1}{N^2}$ to $\frac{1}{N}$, i.e. we now only need order N iterations instead of order N^2.