Project 1: Relaxation Methods
Due date: 17 February 2012 12:00hrs

For full credit, hand in a tidy and efficiently short presentation of your results and how they come about, in a manner that can be understood and reproduced by your peers. All problems and solutions are for your personal use only. Please do not pass solutions or problems on to incoming or other students who have not taken the course (yet). Noncompliance with these rules is a breach of academic integrity.

Solutions must be sent by email as .pdf file of documentation, and results plus source code plus executable as one .tar archive to Drs. MacLachlan (maclach@gwu.edu) and Griesshammer (hgrie@gwu.edu).

You may form study groups to discuss and attack projects as team. However, the solution you hand in must be your own alone. Do not share code.

I reserve the right to award zero points for any illegible, chaotic or irreproducible section of your homework.

Project Description
Solving an electro-static problem numerically by the Relaxation Method.

Relaxation Method for the Poisson Equation
The electric potential $\Phi$ in the presence of charged matter is determined from the Poisson equation,

$$ \nabla^2 \Phi(\vec{r}) = -4\pi \rho(\vec{r}) , \quad (1) $$

with $\rho$ the charge density; or in Cartesian coordinates in 3 spatial dimensions:

$$ \frac{\partial^2}{\partial x^2} \Phi(x,y,z) + \frac{\partial^2}{\partial y^2} \Phi(x,y,z) + \frac{\partial^2}{\partial z^2} \Phi(x,y,z) = -4\pi \rho(x,y,z) . \quad (2) $$

One can solve it e.g. by the well-known separation of variables.

Another technique to solve eq. (1) and other second order differential equations is the Relaxation Method. Though not the most efficient numerical technique, it is relatively easy to implement. For the sake of simplicity, consider the one-dimensional case with the understanding that generalisation to higher dimensions is straightforward. The Laplace equation (Poisson for $\rho \equiv 0$) reads then

$$ \frac{d^2 f}{dx^2} = 0 . \quad (3) $$

The Taylor expansion for forward and backward steps of size $h$, up to second order, is written compactly as

$$ f(x \pm h) = f(x) \pm hf'(x) + \frac{h^2}{2} f''(x) + \mathcal{O}(h^3) . \quad (4) $$

The difference of these expressions leads to the forward-backward second derivative as $h$ approaches zero.

$$ f(x+h) - f(x-h) = 2hf'(x) \rightarrow f'(x) = \frac{f(x+h) - f(x-h)}{2h} \quad (5) $$

Conversely, taking the sum of these two expressions gives the interesting relation,

$$ f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) . \quad (6) $$

The key to the relaxation method lies in eq. (6): The second derivate of $f$ is related to $f$ evaluated at $x$ and $x \pm h$. Solving for $f(x)$ gives the simple-to-use relation

$$ f(x) = \frac{1}{2} \left[ f(x+h) + f(x-h) - h^2 f''(x) \right] . \quad (7) $$

Recall from eq. (3) that $f'' = 0$ in the absence of charged matter, so that the resulting equation has a familiar interpretation:

$$ f(x) = \frac{1}{2} \left[ f(x+h) + f(x-h) \right] . \quad (8) $$
In other words, the scalar field solution to the Laplace equation is, at every point, equal to the average of the values of the nearest neighbours! Generalisation to two dimensions is straightforward:

\[
f(x, y) = \frac{1}{4} \left[ f(x + h, y) + f(x - h, y) + f(x, y + h) + f(x, y - h) \right].
\] (9)

You can easily derive that the analogous Poisson equation (1) takes the form

\[
f(x, y) = \frac{1}{4} \left[ f(x + h, y) + f(x - h, y) + f(x, y + h) + f(x, y - h) + (4\pi)h^2\rho(x, y) \right].
\] (10)

The relaxation method is also used to solve other problems involving second-order partial differential equations, e.g. the Schrödinger equation (see e.g. Presilla and Tambini, Phys. Rev. E52 (1995), 4495-4498; Tang, Nill and Shillinglaw, http://arxiv.org/abs/quant-ph/0011116).

**Discretisation**

The discrete version of eq. (10), assuming equal spacing, takes on the form

\[
f_{i,j} = \frac{1}{4} \left[ f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1} + (4\pi)h^2\rho(i, j) \right].
\] (11)

One now uses this equation to iteratively set the value of \( f_{i,j} \) as the average of its nearest neighbours, see Fig. 1. It is possible to proceed through the iteration in a number of ways. The most prominent ones (and the ones we will explore) are:

1. **In the Jacobi method**, one stores the original values in an array and place the new values in another array. After sweeping over the mesh, the new array is then copied into the old array and the process repeats a desired number of times. In terms of an equation:

   \[
f_{i,j}^{\text{new}} = \frac{1}{4} \left[ f_{i+1,j}^{\text{old}} + f_{i-1,j}^{\text{old}} + f_{i,j+1}^{\text{old}} + f_{i,j-1}^{\text{old}} + (4\pi)h^2\rho(i, j) \right].
\] (12)

   This method unfortunately wastes compute cycles.

2. A more efficient approach is the **Gauß-Seidel method** or continuous updating, where one replaces the value of \( f_{i,j} \) immediately before proceeding to the next mesh point. This is a result of double-computing the values which occur along the interface between zero and non-zero values. So you get to put your own name on a method by just a tiny change in the updating procedure. For this method, the equation analogous to (12) is

   \[
f_{i,j}^{\text{new}} = \frac{1}{4} \left[ f_{i+1,j}^{\text{old}} + f_{i-1,j}^{\text{new}} + f_{i,j+1}^{\text{old}} + f_{i,j-1}^{\text{new}} + (4\pi)h^2\rho(i, j) \right].
\] (13)

   when starting the algorithm at the point \((1, 1)\) and proceeds through the mesh by increasing \(i\) and \(j\).

You should try both methods by hand for a small grid and demonstrate it to yourself. The point is illustrated in Tables 1 and 2. Notice the faster convergence for the Jacobi method, i.e. when values for \( f_{i,j} \) are replaced as they are computed.
Figure 1: Mesh for discretisation problem. The blue dot represents \( f_{i,j} \), and red dots represent the nearest neighbours.

Table 1. Convergence of \( f_{i,j} \) after two iterations with values replaced after each iteration (Jacobi method).

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1.000000</td>
<td></td>
</tr>
<tr>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.062500</td>
<td>0.312500</td>
<td>1.000000</td>
</tr>
<tr>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.062500</td>
<td>0.375000</td>
<td>1.000000</td>
</tr>
<tr>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.062500</td>
<td>0.375000</td>
<td>1.000000</td>
</tr>
<tr>
<td>0.000000</td>
<td>0.312500</td>
<td>0.375000</td>
<td>0.375000</td>
<td>0.437500</td>
<td>1.000000</td>
<td></td>
</tr>
<tr>
<td>1.000000</td>
<td>1.000000</td>
<td>1.000000</td>
<td>1.000000</td>
<td>1.000000</td>
<td>1.000000</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Increased convergence by replacing values of \( f_{i,j} \) as they are computed (Gauss-Seidel method): values after two iterations.

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.062500</td>
<td>0.343750</td>
<td>1.000000</td>
</tr>
<tr>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.093750</td>
<td>0.441406</td>
<td>1.000000</td>
</tr>
<tr>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.105469</td>
<td>0.469727</td>
<td>1.000000</td>
</tr>
<tr>
<td>0.000000</td>
<td>0.343750</td>
<td>0.441406</td>
<td>0.469727</td>
<td>0.588623</td>
<td>0.794312</td>
<td>1.000000</td>
</tr>
<tr>
<td>1.000000</td>
<td>1.000000</td>
<td>1.000000</td>
<td>1.000000</td>
<td>1.000000</td>
<td>1.000000</td>
<td>1.000000</td>
</tr>
</tbody>
</table>

Over-Relaxation

It is possible to gain a further advantage in efficiency by introducing a relaxation parameter \( \omega \) and replacing eq. (13) by

\[
f_{i,j}^{new} = (1 - \omega) f_{i,j}^{old} + \frac{\omega}{4} \left[ f_{i+1,j}^{old} + f_{i-1,j}^{new} + f_{i,j+1}^{old} + f_{i,j-1}^{new} + (4\pi h^2 \rho(i,j)) \right]
\]  (14)

One must determine the value of \( \omega \) which minimises the number of iterations required to reach convergence, but in general \( 1 \leq \omega \leq 2 \). For \( \omega = 1 \), (14) reduces to (13).

For a more thorough discussion, see the accompanying hand-out from Garcia: *Numerical Methods for Physics*, and consult a textbook on Computational Physics (see last semester’s bibliography).
Assignment

Write a fortran95 or C implementation of the RELAXATION METHOD and solve the electrostatic problem described below. The code must be modular and well-documented, both as technical report and inside the code. A makefile must be included. A revision controller (rcs) must be used.

Your assignment is to find the two-dimensional scalar potential, $\Phi(x, y)$, in a rectangular vacuum bounded by four equipotentials using the relaxation method. The aspect ratio of the rectangle is 2, with 40 by 20 grid spacings. The value of the upper and left potentials is $\Phi = 1$, and the value of the lower and right potentials is $\Phi = 0$, see Fig. 2. Complement this initial problem with one in which a charge is placed at the centre of the box so that $h^2 \rho = -1/2$ at that point.

However, the code must be written such that other boundary value problems and charges for the same rectangle can easily be implemented.

![Figure 2: Boundary values for the rectangle.](image)

After some number of iterations, the solution should settle, i.e. more iterations will produce no noticeable effect. At this time your program should know to exit. **Clearly state the convergence criterion.** For this part of the assignment, you may use either eq. (12) or eq. (13). Plot your solution for $\Phi$ as both a surface plot and a contour plot. You may combine these two if you wish.

In part two, determine the optimal value of $\omega$ in eq. (14) both with and without continuous updating. Plot the number of iterations until convergence is reached as function of $\omega$. As before, plot your solution of $\Phi$ as both a surface plot and a contour plot. You may combine these two if you wish.

Appendix – Not Part of the Assignment

As an interesting aside, consider the wave equation in 1$d$,

$$\nabla^2 \Phi = -\Phi(x), \quad (15)$$

which is obviously solved by $\Phi(x) = A \sin x + B \cos x$. Now, eq. (10) becomes

$$\Phi(x) = \frac{1}{2} \left[ \Phi(x + h) + \Phi(x - h) + h^2 \Phi(x) \right]. \quad (16)$$

You can solve for $\Phi(x)$:

$$\Phi(x) = \frac{1}{2} \frac{[\Phi(x + h) + \Phi(x - h)]}{1 - h^2/2} \quad (17)$$

Plotting this reiterative relation yields indeed a sinusoidal function, as long as you do not choose pathological boundary conditions. Try this for other second order differential equations, its fun.
Time-line

week 1 **Due by 26 January, 12:00** Preliminary programme without continuous updating (without charge). First plots of $\Phi$.

Prepare *specific points for discussion in class*: Which boundary conditions exactly do you have to impose to start the first sweep? Think about good convergence criteria (several!).

week 2 **Due by 2 February, 12:00** Programme with continuous updating and with the central charge. Compare the speed of convergence with and without continuous updating. Implement over-relaxation.

week 3 **Due by 9 February, 12:00** Compare the speed of convergence with and without continuous updating and for several over-relaxation values. GNUplot plotting interface polished; make a “movie” of the field as the sweep progresses.

week 4 **Final due date 17 February** Completed, well-documented source code with all sub-programmes; working executable with polished user-interface; complete technical report and user manual with discussion of sample input and output as .pdf file; makefile; sample input and output.

As usual in research, the time-line and project goals may have to be re-assessed based on progress and our own curiosity.

Figures of Merit

a) **Code (40% of points)**: a stand-alone package which contains the complete modular code in fortran95 or C and all object-files/sub-routines/depending programmes; makefile. Programme compiles under a common, non-commercial compiler without warnings and errors. Programme implements both Gauß-Seidel and Jacobi algorithms, and both over-relaxation and common relaxation. A movie of the sweep-sequence is generated “on the fly”.

b) **Assessment (40% of points)**: Assessment of usefulness and limitations of your convergence criterion; discussion of continuous updating and of over-relaxation results. Plots (contour and surface) of the quantities described above, with clearly labelled axes, comparing also Gauß-Seidel vs. Jacobi; dependence of convergence speed on over-relaxation parameter.

c) **Technical Report/Documentation (20% of points)**: Description of the workings of the code, including which implementation was chosen and why, and its limitations.