|epcc|

Numerical Solution of Partial Differential Equations The Two-Dimensional Pollution Model

Introduction

This exercise is split into two parts. First you perform a discrete second derivative operation on $u_{i,j}$: you write a code to calculate $u_{new} = -\nabla^2 u$ on an $M \times M$ grid with spacing $h = 1/(M+1)$. You are supplied with the input function u, and you must implement appropriate boundary conditions (in this case, $u = 0$ on all the edges) and then write code to compute the second derivative. You can then visualise the output. The input function is constructed so that you should recognise the resulting picture.

The second part of the exercise builds on the first, but now you are solving the equation $-\nabla^2 u = 0$. As before you must set the boundary conditions, but now they are a more complicated as they correspond to the pollution from a chimney located in the middle of the right-hand edge. Afterwards, you repeatedly update u using a variety of algorithms (Jacobi, Gauss-Seidel and over-relaxed

Gauss-Seidel), stopping when the residue is sufficiently small. Each update is similar to the code you wrote for the first part of the exercise, so you should be able to re-use much of your program. For example, the definition of the array dimensions and the output routines are identical. Again, you can visualise the result.

1 Computing the Second Derivative

1.1 Template code

You should download the template code (all packaged up into a single tar file `pde.tar`) from the NAHPC course web pages and unpack it:

```
user$ tar xvf pde.tar
```

This will create a directory called `pde` containing two subdirectories, one each for Fortran and C programmers (F and C). You should work in the directory corresponding to your chosen language. The code will run as it stands, although initially it does not do any useful work and simply writes out a visualisation of the input function u. You compile and execute, e.g.:

```
user$ gfortran -o testpde testpde.f90
user$ ./testpde

user$ gcc -o testpde testpde.c
user$ ./testpde
```

You can then visualise the result using the Paraview visualisation package. To do this, launch Paraview and load the csv file, reading the data with the `CSV Reader` option. Click `Apply`, found in the

Properties window that is likely on the lower left of Paraview. Then select `Filters` → `Alphabetical` → `Table To Points` and for the X Column select x, for the Y Column select y and for the Z column select u. Again, click `Apply` and then click on the eye next to `TableToPoints1` in the Pipeline Browser. You may need to move the picture around and click the eye a couple of times before you see a picture. In order to see much of interest you will have to scale the $z-$axis. To do this, go to `Filters` → `Alphabetical` → `Transform`. Uncheck "Show box" and scale the z data (the third column) by 10 (you may need to vary this factor for other data).

The output should look like a smooth lump with slightly ragged edges. Note that we will use the same visualisation procedure for all the following exercises (though the scale factor may have to change to get a clear picture of what is happening). If you overwrite the csv file you can view the new data without going through the above steps by right-clicking on it in the Pipeline Browser and selecting "Reload Files".

1.2 The warm-up exercise

You should add code to set all the boundaries of u to zero. Run your new code and check that the ragged edges have disappeared. Now add code to compute $u_{new} = -\nabla^2 u$. Note that, as is conventional for relaxation methods, we will ignore all the factors of h^2. The equation you must implement is:

$$unew_{i,j} = -u_{i,j-1} - u_{i-1,j} + 4u_{i,j} - u_{i+1,j} - u_{i,j+1}$$

When you visualise the output you should see a familiar picture!

2 Solving the Pollution Equations

We now want to solve:

$$-u_{i,j-1} - u_{i-1,j} + 4u_{i,j} - u_{i+1,j} - u_{i,j+1} = 0$$

with appropriate boundary conditions. When we solve these pollution equations we will set the initial conditions by hand so we do not read anything in from file. You can therefore remove the call to the input routine from your program. You should start with a small value of M, for example $M = 20$, but you should try larger values when you are experimenting with different algorithms.

Then add the following in place of the existing $u_{new} = -\nabla^2 u$ code.

1. zero the entire u array
2. set the values of $u(1.0, y)$ according to the following formula which creates a smooth hump of height 1.0 between y_1 and y_2.

$$\begin{aligned} y <= y_1 : \;& u(1.0, y) = \; 0.0 \\ y_1 < y < y_2 : \;& u(1.0, y) = \; k(y - y_1)^2 (y_2 - y)^2 \\ y >= y_2 : \;& u(1.0, y) = \; 0.0 \end{aligned}$$

   where $k = \left(\frac{2}{y_2 - y_1}\right)^4$. For $y < y_1$ and $y > y_2$ you should set $u(1.0, y) = 0.0$. To implement this you will need to loop over the right-hand edge of the u array (*i.e.* over $u_{M+1,j}$ for $j = 0, 1, 2, \ldots, M, M+1$), compute the y coordinate of each point from the value of j, and set $u_{M+1,j}$ according to the formula above. At this stage you should compile and run your program, and visualise the output. This will give you a good visual check that the source function is correct.

3. sum up the squares of the edge values to compute the norm of the source $|b|$

4. compute the initial residue (appropriately normalised)

5. loop over iterations

    - update u according to equations in Section 2.1
    - compute residue for this iteration (appropriately normalised)

6. until residue is small enough

For this exercise you should set $y_1 = 0.4$, $y_2 = 0.6$ and stop when the residue is less than 10^{-6}.

You can visualise the result as before — do you see what you expect? If your code is not working, a good check is that the initial residue should be exactly 1.0, and should always decrease as the number of iterations increases.

You should implement the Jacobi algorithm at first, then Gauss-Seidel, and finally add over-relaxation, as described in Section 2.1. Be careful to write your program so it is straightforward to switch between alternative algorithms (eg write a separate routine or method for each). This will make it easier to compare their relative performance when solving the same problem.

2.1 Update equations

You should start by implementing the Jacobi algorithm. The 2D update equation is:

$$u_{new} = \frac{1}{4}\left(u_{i,j-1} + u_{i-1,j} + u_{i+1,j} + u_{i,j+1}\right)$$

For Gauss-Seidel, you simply perform the above update in-place, so you do not require the u_{new} array. Implement the Gauss-Seidel algorithm and note how much faster it is in terms of iterations taken to converge.

You should be able to write down the over-relaxed update equations by analogy with the 1D equations given in the lectures. Introduce over-relaxation into your Gauss-Seidel update (a useful check is that for $\omega = 1$ the output should be identical to your previous code which had no over-relaxation) and look for the optimal value of ω, ie the value for which the algorithm converges in the smallest number of iterations. You should search in the range $1.0 \leq \omega \leq 2.0$.

In all cases you should visualise the solution to check that you are getting sensible answers

2.2 Interpolating the Solution

As mentioned in the lectures, the pollution level at the house, $u(0.20, 0.33)$, will probably not correspond to a grid point $u_{i,j}$. We therefore have to perform an interpolation by using an appropriately weighted average of the four surrounding grid points.

The formula you should use is:

$$u(x,y) = u((i+\delta_i)h, (j+\delta_j)h) = u_{i,j} + \delta_i(1-\delta_j)f_i + \delta_j(1-\delta_i)f_j + \delta_i\delta_j f_{i,j}$$

$$f_i \;=\; u_{i+1,j} - u_{i,j}$$

$$
\begin{aligned}
f_j &= u_{i,j+1} - u_{i,j} \\
f_{i,j} &= u_{i+1,j+1} - u_{i,j}
\end{aligned}
$$

Here δ_i and δ_j measure the relative distance of the point in question from the two nearest grid points, and their values are always between 0 and 1. For example, if we had $h = 0.12$ then there would be grid points at $x_5 = 0.60$ and $x_6 = 0.72$. If we wanted to know the value at $x = 0.69$, then δ_x would be equal to 0.75. This is because x is 75% of the distance from the grid point at $i = 5$ to the next point at $i = 6$.

In equations:

$$
i = \frac{x}{h}, \quad j = \frac{y}{h} \qquad\qquad \delta_i = \frac{x - ih}{h}, \quad \delta_j = \frac{y - jh}{h}
$$

3 Introducing a Wind

By looking at the 2D wind equations presented in the lectures, you should be able to write down the associated Jacobi, Gauss-Seidel and over-relaxed update equations when we have a north-easterly wind of strength (a_x, a_y). Solve the equations using a wind of $(2.0, 1.0)$. What effect does it have on the pollution levels at the house? What happens if you increase the strength of the wind?

Note that when calculating the residue you will have to include extra terms for the wind. Furthermore, the off-diagonal elements of the matrix A are no longer equal to -1. This means that there is a scaling factor between the norm of the pollution source and the norm of the associated b vector. To account for this, you must multiply the norm of b by $(1 + a_x h)$ to ensure that the initial residual is exactly equal to 1.0.