Exercises for the lecture

Fundamentals of Simulation Methods

Friedrich Röpke & Cornelis Dullemond

Exercise sheet 6

Boundary Value Problems

1. 1-D Heat diffusion problem

To "warm up" to the solving of complicated boundary value problems, we start with a simple 1-D heat diffusion problem. Consider a 1-D block of radioactive material that produces heat at a rate of ϵ Joules per meter. In 3-D this would be in Joules per meter^3, but since we reduce the problem to 1-D, it is just per meter. The heat diffusion coefficient is D. The temperature at the edges at $x = \pm L$ is kept fixed at T_0. The steady state temperature profile obeys

$$-D\frac{\partial^2 T(x)}{\partial x^2} = \epsilon \tag{1}$$

For simplicity let us set $D = 0.5$, $\epsilon = 1$, $T_0 = 1$ and $L = 1$. We use $N = 100$ grid points.

a) Write down the numeric form of this equation on the grid.

b) Write this in a matrix notation.

Now let us think of how to put this into a computer program. The matrix is *sparse*: most of the elements are zero. In fact, it is a special kind of sparse matrix: a tridiagonal matrix.

c) Come up with a way to store the matrix elements of this tridiagonal matrix into only three 1-D arrays of N elements.

d) Design a function/subroutine that multiplies such a matrix with any given vector. Test that it performs the multiplication correctly.

e) Design and test a function/subroutine that uses the forward-elimination backward-substition method to solve a matrix equation.

f) Apply it to the above problem, and plot the solution to the above problem.

g) Verify that the result is correct by multiplying the solution with the matrix and computing the residual, and show that the residual is almost zero.

h) Why is it not exactly zero?

i) Change N to e.g. 1000 and see that the result is the same, but smoother.

As you can see, this is a very efficient method for 1-D boundary value problems. Now let us see how the Jacobi iteration performs.

j) Design a function/subroutine to perform a single Jacobi iteration step for this problem. Take initially $N = 8$ (yes, really that low!). Perform 30 iteration steps.

k) Plot the result of each iteration over each other, and overplot the true solution that we found with the forward elimination, backward substitution.

l) Now do the same, but for $N = 100$. Explain the behavior and the difference to the case for $N = 8$.

2. **1-D Multigrid method**

Although the multigrid method is most useful for 2-D and 3-D problems, let us learn how to use it with a simple 1-D problem.

NOTE: To keep things simple let us no longer use the sparse storage of the matrix (the compact storage without all the zeros), but simply use the full 2-D matrix arrays. This will make it easier for you, and will give more insight, because you can then see the structure of the matrices more easily.

Start with a grid of $2^3 + 1 = 9$ points. The twice courser grid will have $2^2 + 1 = 5$ grid points, and the next $2^1 + 1 = 3$ and finally the last will have $2^0 + 1 = 2$ points.

a) Construct the 2-D matrices $\mathbf{R}^{(3)}$, $\mathbf{R}^{(2)}$, $\mathbf{R}^{(1)}$ for restriction to the next lower level (coarser grid).

b) Construct the 2-D matrices $\mathbf{P}^{(2)}$, $\mathbf{P}^{(1)}$, $\mathbf{P}^{(0)}$ for prolongation to the next higher level (finer grid). Check that the P and R matrices are (apart from a factor) each other's tranpose.

c) Take the 9×9 matrix of the previous section (or the one from the 1-D diffusion problem in the lecture notes), again in full 9×9 shape, and use the $\mathbf{R}^{(i)}$ and $\mathbf{P}^{(i)}$ matrices to create the restricted versions of that matrix. Check that the result makes sense.

d) Construct a recursive subroutine/function that applies the V-shaped multigrid procedure using Jacobi iteration.

e) Compare to the solution you found in the previous exercise.

3. **[BONUS] 3-D Poisson equation solver**

This problem is not obligatory, as it can be quite time-consuming and may require a bit of frustration-resistance. The assignment is to design and test a 3-D Poission equation solver on a cartesian grid of $n \times n \times n$ grid points. Take for a start $n = 10$. At the boundary cells, ribbon cells and corner cells we set $\Phi = 0$. At the center of the grid we make a $2 \times 2 \times 2$ block of cells with a non-zero density, whereas the rest of the grid has zero density.

Use the Biconjugate Gradient (or Biconjugate Gradient Stabilized) method to solve this equation. Advice: make sure that you normalize the equations such that the diagonal elements of the boundary cells are not vastly larger or smaller (in their absolute values) than those of the inner grid cells.

For Python users: you can use `scipy.sparse.linalg.bicgstab()`.

For C or Fortran users, we've uploaded the Numerical Recipes routines you need for calling `linbcg()` in both C and Fortran versions. You can find the documentation online: `http://numerical.recipes`

Make plots of the solution by taking a 1-D or 2-D cut along the center of the box. Does the result look reasonable?