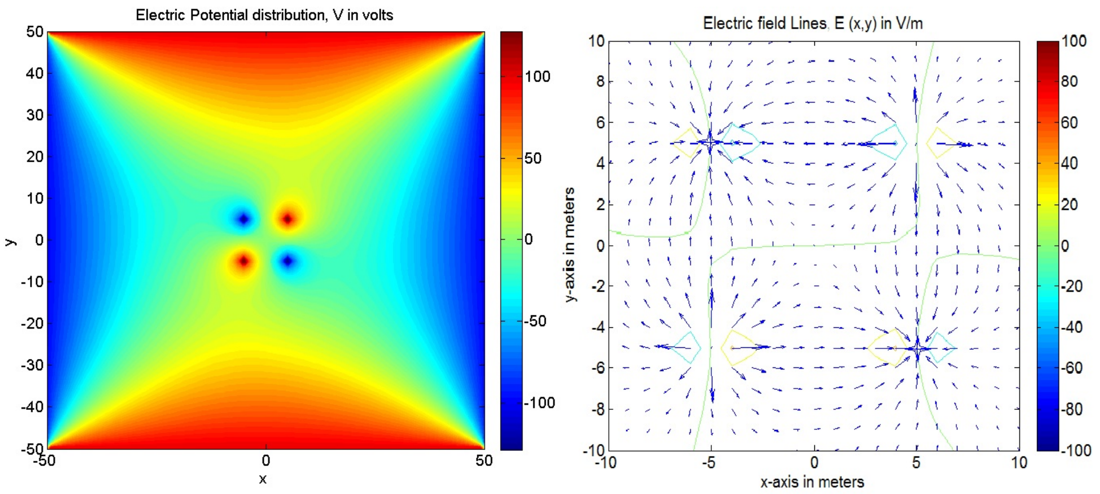
The Poisson Equation in a Rectangle

MATH2601: Control of Partial Differential Equations

Location: http://people.sc.fsu.edu/∼jburkardt/classes/control_2019/poisson_2d/poisson_2d.pdf



Electric field potential is an example of a Poisson problem.

The Poisson Problem

Given a Poisson equation on a 2D rectangular region, use finite differences to create a model of the equation, set up the corresponding linear system, display the approximate solution and estimate its error.

1 The Poisson equation

The Poisson equation can be written for an open finite region $\Omega \subset \mathbb{R}^n$ as:

$$-\Delta u(x) = f(x) \quad \text{for all } x \in \Omega$$

with several choices of boundary conditions to be imposed for $x \in \partial\Omega$:

$$\begin{aligned} \textit{Dirichlet:} \quad & u(x) = g(x) \\ \textit{Neumann:} \quad & \frac{\partial u}{\partial n} = g(x) \\ \textit{Robin:} \quad & \alpha * u(x) + \beta * \frac{\partial u}{\partial n} = g(x) \end{aligned}$$

Here, Δ is the Laplacian operator, and can be written several ways:

$$\Delta u(x) = \nabla \cdot \nabla u(x) = \nabla^2 u(x) = \text{div} \cdot \ \text{grad } u(x) = \sum_{i=1}^{n} \frac{\partial^2 u}{\partial x_i^2}$$

2 A Poisson equation on a 2D rectangle

We take as our domain Ω the interior of the 2D rectangle $(a, b) \times (c, d)$. We will assume that at every point along the boundary, we have imposed Dirichlet boundary conditions, and that the functions $f(x, y)$ and $g(x, y)$ have been given. At this point, mathematically, we would seek a function $u \in C^2(\Omega)$ which satisfies the Poisson equation and the specified boundary conditions. Unfortunately, there is no general procedure for deriving the formula of such a solution function. Since Poisson problems arise in many practical situations, the only feasible approach may be a computational approximation.

We begin by discretizing the geometry. That is, we we will create an $nx \times ny$ grid of points, with spacings $hx = \frac{b-a}{nx-1}$ and $hy = \frac{d-c}{ny-1}$, so that a typical discrete point is:

$$(x_i, y_j) = ((i-1) * hx, (j-1) * hy) \quad \text{for } 1 \le i \le nx, 1 \le j \le ny$$

We will seek an approximate solution only at this discrete set of points, and we may use the notation $u_{i,j}$ to indicate the value of our discrete solution at (x_i, y_j).

Points corresponding to $i = 1$, $i = nx$, $j = 1$ or $j = ny$ are *boundary points* where the Dirichlet boundary conditions will be imposed:

$$u_{i,j} = g(x_i, y_j)$$

The remaining points are *interior points*, at which a discretized version of the Poisson equation will be imposed:

$$\frac{-u_{i-1,j} + 2u_{i,j} - u_{i+1,j}}{hx^2} + \frac{-u_{i,j-1} + 2u_{i,j} - u_{i,j+1}}{hy^2} = f(x_i, y_j)$$

Thus, for each discrete point, we now have an explicit formula (at boundary points) or an implicit linear equation (at interior points). We can think of this combinary of conditions on boundary and interior points as a linear system of the form:

$$A * u = rhs$$

It now remains to organize the process of defining the matrix A and right hand side rhs. Once this is done, the solution process is a relatively straightforward linear algebra task.

3 Assembling the linear system

We have $nx \times ny$ equations to construct, and we need a way to order them. Unfortunately, there are several "natural" ways to number such an array. We will choose to start at the lower left corner, and proceed to the right. Once a row is done, we move up to the next row, and so on. Thus, the point associated with the coordinates (x_i, y_j) will be counted on step $k = i + (j-1) * nx$.

Counting in this way, the first row of nx points, for which $j = 1$ will each generate a boundary condition equation, of the form

$$u_{i,1} = g(x_i, y_1) \quad 1 \le i \le nx$$

Rows $j = 2$ through $ny - 1$ will each begin and end with a boundary condition equation, sandwiching $nx - 2$ discretized Poisson equations:

$$\begin{aligned} u_{1,j} &= g(x_1, y_j) \\ \frac{-u_{i-1,j} + 2u_{i,j} - u_{i+1,j}}{hx^2} + \frac{-u_{i,j-1} + 2u_{i,j} - u_{i,j+1}}{hy^2} &= f(x_i, y_j) \\ u_{nx,j} &= g(x_{nx}, y_j) \end{aligned}$$

The last row of nx points will have the same form as the first row.

It is helpful to see the pattern that this system of equations forms when written in matrix form. If we suppose $nx = ny = 5$, then the first block of equations, which are entirely boundary conditions, has the "interesting part":

```
   1  2  3  4  5                                     RHS

1  1  0  0  0  0                                     g(x1,y1)
2  0  1  0  0  0                                     g(x2,y1)
3  0  0  1  0  0                                     g(x3,y1)
4  0  0  0  1  0                                     g(x4,y1)
5  0  0  0  0  1                                     g(x5,y1)
```

The second block of equations begins and ends with a boundary condition, and in between, specifies 3 discrete Poisson relations. Here, to make the illustration legible, we pretend that the coefficients of the Poisson relation are simply +4 for the central point, and -1 for the four neighbors.

```
    1  2  3  4  5   6  7  8  9 10   11 12 13 14 15   RHS

 6  0  0  0  0  0   1  0  0  0  0    0  0  0  0  0   g(x1,y2)
 7  0 -1  0  0  0   0  4  0  0  0    0 -1  0  0  0   f(x2,y2)
 8  0  0 -1  0  0   0  0  4  0  0    0  0 -1  0  0   f(x3,y2)
 9  0  0  0 -1  0   0  0  0  4  0    0  0  0 -1  0   f(x4,y2)
10  0  0  0  0  0   0  0  0  0  1    0  0  0  0  0   g(x5,y2)
```

The linear system has two more blocks of equations like this, followed by a final block that is once again entirely boundary conditions.

4 A MATLAB code to solve a Poisson equation

The following program accepts input from the user which defines the discretization of a rectangle, and the right hand side functions $f(x, y)$ and $g(x, y)$, and returns matrices U, X, and Y containing the computed solution and coordinate values at the grid points:

```matlab
function [ U, X, Y ] = poisson ( nx, ny, xmin, xmax, ymin, ymax, f, g )

  hx = ( xmax - xmin ) / ( nx - 1 );
  hy = ( ymax - ymin ) / ( ny - 1 );

  x = linspace ( xmin, xmax, nx );
  y = linspace ( ymin, ymax, ny );

  A = zeros ( nx * ny, nx * ny );
  rhs = zeros ( nx * ny, 1 );

  keq = 0;
  for j = 1 : ny
    for i = 1 : nx
      keq = keq + 1;
      if ( i == 1 || i == nx || j == 1 || j == ny )
        A(keq,keq) = 1.0;
        rhs(keq) = g ( x(i), y(j) );
      else
        A(keq,keq) = 2.0 / hx / hx + 2.0 / hy / hy;
        A(keq,keq-1)  = - 1.0 / hx / hx;
        A(keq,keq+1)  = - 1.0 / hx / hx;
        A(keq,keq-nx) = - 1.0 / hy / hy;
        A(keq,keq+nx) = - 1.0 / hy / hy;
        rhs(keq) = f ( x(i), y(j) );
      end
    end
  end

  u = A \ rhs;

  U = ( reshape ( u, nx, ny ) )';
  [ X, Y ] = meshgrid ( x, y );

  return
end
```

Listing 1: poisson.m

5 Solving a particular problem

To solve a specific problem, the user needs to specify the discretization of the rectangle, and supply the names of MATLAB objects that evaluate $f(x,y)$ and $g(x,y)$. Here is an aexample:

```
nx = 21;
ny = 41;

xmin = 0.0;
xmax = 1.0;
ymin = 1.0;
ymax = 3.0;

[ U, X, Y ] = poisson ( nx, ny, xmin, xmax, ymin, ymax, @f, @g );

function value = f ( x, y )
  value = pi * pi * ( x .* x + y .* y ) .* sin ( pi * x .* y );
  return
end
function value = g ( x, y )
  value = sin ( pi * x .* y );
  return
end
```

Listing 2: poisson_test.m

6 Postprocessing

The RMS error can be a useful statistic. Assuming that the user-supplied function $g(x,y)$ is actually a formula for the exact solution everywhere, we can easily compute as follows:

```
err = 0.0;
for i = 1 : ny
  for j = 1 : nx
    err = err + ( U(i,j) - g(X(i,j),Y(i,j)) )^2;
  end
end
err = sqrt ( err / nx / ny );
```

Listing 3: Computing the RMS error

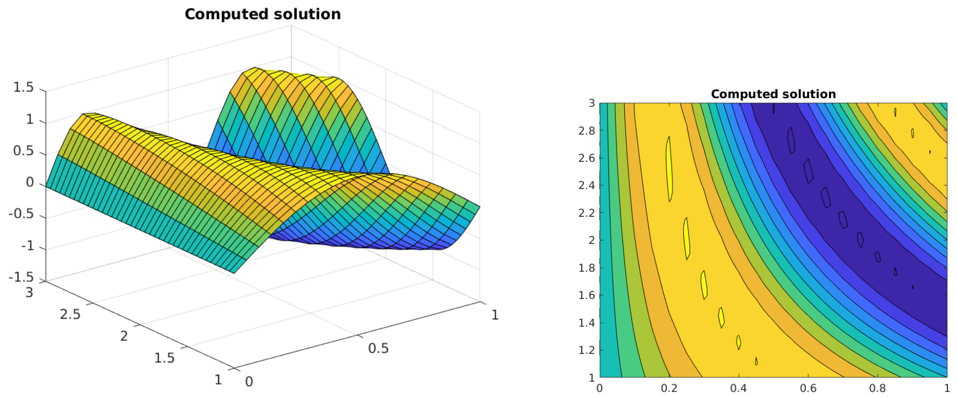
A convergence study can be done by repeatedly doubling the fineness of the mesh, and plotting the corresponding sequence of RMS errors.

Surface or contour plots of the computed solution can also be made:

```
figure ( 1 )
surf ( X, Y, U );
title ( 'Computed solution' );
print ( '-dpng', 'poisson_surface.png' );

figure ( 2 )
contourf ( X, Y, U )
title ( 'Computed solution' );
print ( '-dpng', 'poisson_contours.png' );
```

Listing 4: Surface and contour plots



Surface and contour plots of computed solution.