Laboration 4: The Poisson equation and the time-independent Schrödinger equation in 2D

December 8, 2014

Classical solution

The Poisson equation

The Poisson equation with homogenous Dirichlet boundary conditions is

$$-\nabla\left(a(x,y)\nabla u\right) = f, \quad x \in \Omega \tag{1}$$
$$u = 0, \quad x \in \partial\Omega, \tag{2}$$

where f and a are given functions and a is positive on Ω. A classical solution is then a two times continuously differentiable function u, which satisfies (1) and (2).

The Schrödinger equation

The time-independent Schrödinger equation has the following form

$$-\Delta u + a(x,y)u = f, \quad x \in \Omega \tag{3}$$
$$u = 0, \quad x \in \partial\Omega, \tag{4}$$

where f and a are given functions and a is non-negative on Ω. A classical solution is then a two times continuously differentiable function u, which satisfies (3) and (4).

Classical solutions will not always exist (for example in the case when a in (1) is piecewise constant). It is therefore important to also consider weak solutions. Moreover, a weak formulation of the problem is the starting point for the finite element method.

Problem 1. State a weak (variational) formulation of the Poisson equation (1), (2) and of the Schrödinger equation (3), (4).

Finite Element Approximation

The weak formulations derived in Problem 1 can be used as the starting point for the finite element method. However, we will as in the book/lectures use a Robin condition and then approximate the homogenous Dirichlet boundary conditions by a suitable choice of constants. Let

$$c_0 u + c_1 \frac{\partial u}{\partial n} = 0, \tag{5}$$

where c_0, c_1 are given constants and $\partial u/\partial n = \nabla u \cdot n$.

Problem 2. State a weak (variational) formulation of the Poisson equation (1), (5) and of the Schrödinger equation (3), (5). Give the energy norm for these two problems.

Problem 3. State a finite element approximation (piecewise linear approximation) of the solution to the equations in Problem 1 and in Problem 2. That is, replace the spaces in the weak formulations with appropriate spaces of piecewise linear functions.

Triangulations

To construct vector spaces of piecewise polynomials in two dimensions, it is necessary to partition the domain of interest into polygons. Common choices of such polygons are quadrilaterals or triangles. Triangles are used in this laboration.

Let $\Omega = [0, 0] \times [2, 1]$ be the rectangle with corners at origo $(2, 0)$, $(2, 1)$, and $(0, 1)$. Figure 1 shows a triangulation, or mesh, of Ω into three triangles K_i, $i = 1, 2, 3$. The five triangle vertex points $N_i = (x_i, y_i)$, $i = 1, 2, \ldots, 5$ are called the *nodes*.

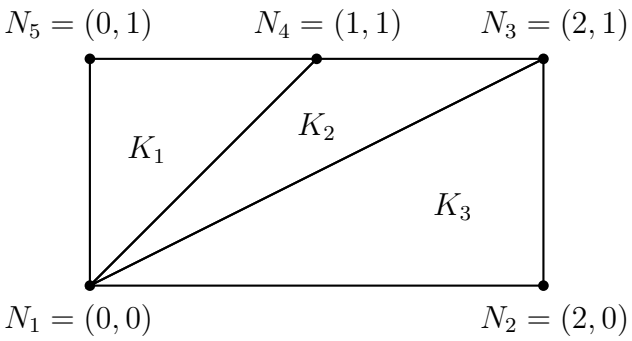

Figure 1: A triangulation of the rectangle $[0, 0] \times [2, 1]$.

The most common way of representing a triangulation (or any mesh for that matter) within a computer is to store a point matrix p containing the node coordinates (x_i, y_i), and a connectivity matrix t, which columns contain information about which nodes belong to which triangle. For example, the point and connectivity matrices of the mesh shown above take the following form

$$p = \begin{bmatrix} 0 & 2 & 2 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}, \qquad t = \begin{bmatrix} 1 & 1 & 2 \\ 4 & 3 & 3 \\ 5 & 4 & 1 \end{bmatrix}. \tag{6}$$

Thus, the coordinates $(x_3, y_3) = (2, 1)$ of node N_3 are given by the matrix entries $p_{1,3}$ and $p_{2,3}$, respectively. In the connectivity matrix t, column i

contains the three node numbers of triangle K_i. The nodes of a triangle are always ordered counter-clockwise.

MATLAB has a non standard set of routines called the PDE toolbox, which provides a flexible environment for the study of finite element methods. In particular, the toolbox includes a mesh generator for creating high quality triangulations of arbitrary two-dimensional geometries.

Let us first define a geometry of, say, the unit square $\Omega = [0,1] \times [0,1]$. Start MATLAB and write at the prompt

```
geom = [2 0 1 0 0 1 0;
        2 1 1 0 1 1 0;
        2 1 0 1 1 1 0;
        2 0 0 1 0 1 0]';
```

Each column of the matrix `geom` describes one of the four sides of the unit square. Read the help for the command `decsg` for a thorough explanation of the geometry format. It is also possible to define geometries via the GUI `pdetool`.

Now, create the mesh by using the command

```
[p,e,t] = initmesh(geom,'hmax',0.5)
```

The outputs `p` and `t` are the mesh data matrices described previously and `e` contains the edge matrix. The extra argument `...'hmax',0.5)` means that no triangle are allowed to have a side length exceeding 0.5. Type doc initmesh to get more information.

Computer Implementation

Consider a single triangle K with nodes at its three corners $N_1 = (x_1^{(1)}, x_2^{(1)})$, $N_2 = (x_1^{(2)}, x_2^{(2)})$, and $N_3 = (x_1^{(3)}, x_2^{(3)})$. To each node N_i, $i = 1, 2, 3$, there is

a hat function φ_i associated, which takes the value one at node N_i and zero at the other nodes. Each hat function is a linear function (i.e., a plane) on K. Hence

$$\varphi_i = a_i + b_i x_1 + c_i x_2, \quad i = 1, 2, 3, \tag{7}$$

where the coefficients a_i, b_i, and c_i, are determined from the requirement

$$\varphi_i(N_j) = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases} \tag{8}$$

Thus, for example, a_1, b_1 and c_1 are given by the equations $\varphi_1(x_1^{(1)}, x_2^{(1)}) = 1$, $\varphi_1(x_1^{(2)}, x_2^{(2)}) = 0$, and $\varphi_1(x_1^{(3)}, x_2^{(3)}) = 0$. In matrix form this (3×3) system of equations take the form

$$\begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ 1 & x_1^{(3)} & x_2^{(3)} \end{pmatrix} \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}. \tag{9}$$

The inverse of a general invertible 3×3 matrix

$$B = \begin{pmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{pmatrix} \tag{10}$$

can be written

$$B^{-1} = \frac{1}{\det(B)} \begin{pmatrix} B_{33}B_{22} - B_{32}B_{23} & B_{32}B_{13} - B_{33}B_{12} & B_{23}B_{12} - B_{22}B_{13} \\ B_{31}B_{23} - B_{33}B_{21} & B_{33}B_{11} - B_{31}B_{13} & B_{21}B_{13} - B_{23}B_{11} \\ B_{32}B_{21} - B_{31}B_{22} & B_{31}B_{12} - B_{32}B_{11} & B_{22}B_{11} - B_{21}B_{12} \end{pmatrix}. \tag{11}$$

Let V denote the matrix in (9). We proved in class that $\det(B) = 2|K|$ and the inverse of V is then

$$V^{-1} = \frac{1}{2|K|} \begin{pmatrix} x_2^{(3)}x_1^{(2)} - x_1^{(3)}x_2^{(2)} & x_1^{(3)}x_2^{(1)} - x_2^{(3)}x_1^{(1)} & x_2^{(2)}x_1^{(1)} - x_1^{(2)}x_2^{(1)} \\ x_2^{(2)} - x_2^{(3)} & x_2^{(3)} - x_2^{(1)} & x_2^{(1)} - x_2^{(2)} \\ x_1^{(3)} - x_1^{(2)} & x_1^{(1)} - x_1^{(3)} & x_1^{(2)} - x_1^{(1)} \end{pmatrix}. \tag{12}$$

Hence, the solution of the linear system (9) is given by

$$\begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} = V^{-1} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \tag{13}$$

$$= \frac{1}{2|K|} \begin{pmatrix} x_2^{(3)} x_1^{(2)} - x_1^{(3)} x_2^{(2)} \\ x_2^{(2)} - x_2^{(3)} \\ x_1^{(3)} - x_1^{(2)} \end{pmatrix}. \tag{14}$$

The basis functions φ_2 and φ_3 are determined from (9) with the right hand side $(0\,1\,0)^T$ and $(0\,0\,1)^T$, respectively.

Note that the gradient of φ_i is just the constant vector $\nabla\varphi_i = (b_i, c_i)$.

As usual the global stiffness matrix A is formed by summing local stiffness matrices A^K. Thus, in the case $a = 1$ we get the (3×3) local stiffness matrix

$$A_{ij}^K = \int_K \nabla\varphi_i \cdot \nabla\varphi_j \, dx = (b_i b_j + c_i c_j) \operatorname{area}(K), \quad i, j = 1, 2, 3. \tag{15}$$

The $3 \cdot 3 = 9$ entries of A^K are to be assembled (at the right locations) into A. In Matlab this can be done as shown below:

```
for K = 1:size(t,2); % loop over the triangles
  nodes = t(1:3,K);  % find triangle K's nodes
  AK = ...           % compute the (3 x 3) stiffness matrix AK
  A(nodes,nodes) = A(nodes,nodes)+AK; % add AK(i,j), i,j=1,2,3,
                                      % to A(nodes(i),nodes(j))
end
```

The local mass matrix on triangle K is

$$M^K = \frac{|K|}{12} \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}. \tag{16}$$

Moreover, since a Robin condition is used the following local matrices and

vectors are necessary:

$$R_{ij}^E = \int_E \varphi_j \varphi_i \, ds = \frac{1}{6}(1 + \delta_{ij})|E|, \quad \text{for edge } E \tag{17}$$

$$F_j^K = \int_K f \varphi_j \, dx \approx \frac{1}{3} f(N_j)|K| \tag{18}$$

The following code is a template routine for assembly the global matrices

```
[p,e,t]=initmesh(geom,'hmax',0.5);
pdemesh(p,e,t) %plot the mesh

n_nodes = size(p,2); % Number of nodes (degrees of freedom).
n_elements = size(t,2); % Number of elements (triangles).

M = sparse(n_nodes,n_nodes); % Allocating matrix.
A = sparse(n_nodes,n_nodes); % Allocating matrix.
R = sparse(n_nodes,n_nodes); % Allocating matrix.
F = sparse(n_nodes,1); % Allocating vector.

for ie = 1:n_elements % Looping over elements.
    dofs = t(1:3,ie); % Node numbers for element ie.
    x = p(1,dofs); y = p(2,dofs); % x and y coords for the three nodes.
    area = polyarea(x,y); % Triarea
    %---------------Matrix M
    Me = (area/12)*[2,1,1;1,2,1;1,1,2]; % Element contribution to M.
    M(dofs,dofs) = M(dofs,dofs) + Me;
    %--------------Matrix A
    b = (y([2,3,1])-y([3,1,2]))/2/area; c = (x([2,3,1]) - x([3,1,2]))/2/area;
    AK = area*(b'*b + c'*c); % Element contribution to A.
    A(dofs,dofs) = A(dofs,dofs) + AK;
    %--------------Vector F
    FK = [f(x(1),y(1));f(x(2),y(2));f(x(3),y(3))]/3*area;
    % element load vector
    F(dofs) = F(dofs) + FK;
 end
c0=?; c1=?;
```

```
for i=1:1:size(e,2)
    E_ind=e(1:2,i);
    x = p(1,E_ind); % node x-coordinates
    y = p(2,E_ind); % node y-
    len = sqrt((x(1)-x(2))^2+(y(1)-y(2))^2); % edge length
    RE=1/6*[2 1;1 2]*len;
    R(E_ind,E_ind)=R(E_ind,E_ind)+RE;
end
U=?; %compute the solution
Energy_error=? %compute the error in the energy norm
figure
pdesurf(p,t,U) %plot solution
```

Problem 4. Implement a finite element solver for the Poisson equation in the unit square with $u = 0$ on the boundary and $a = 1$. Test your code by solving the problem with (in Matlab code)

```
f=@(x,y)  2*pi^2*sin(pi*x)*sin(pi*y);
```

The exact solution is $u = \sin(\pi x)\sin(\pi y)$. Use a mesh with hmax=$0.2, 0.1, 0.05, 0.025$ and compare with the exact solution using the energy norm. The energy norm of the exact solution is $\pi/\sqrt{2}$. Plot this hmax dependent error and the theoretical rate of convergence $\mathcal{O}(h)$ in the same figure using loglog in Matlab. Note that the computed error should be C*hmax for some constant C.

Problem 5. Implement a finite element solver for the Schrödinger equation in the unit square with $u = 0$ on the boundary and $a = 1$. Test your code by solving the problem with (in Matlab code)

```
f=@(x,y)  2*pi^2*sin(pi*x)*sin(pi*y)+sin(pi*x)*sin(pi*y);
```

The exact solution is $u = \sin(\pi x)\sin(\pi y)$. Use a mesh with hmax=$0.2, 0.1, 0.05, 0.025$ and compare with the exact solution using the

energy norm. The energy norm of the exact solution is $\sqrt{\pi^2/2 + 1/4}$. Plot this hmax dependent error and the theoretical rate of convergence $\mathcal{O}(h)$ in the same figure using loglog in Matlab. Note that the computed error should be $Ch_{\max}$ for some constant C.