Series 1 Warmup

ETH zürich

Numerical methods for PDEs
Last edited: March 7, 2017
Due date: 2017-03-14 at 23:59

Template codes are available on the course's webpage at `https://moodle-app2.let.ethz.ch/course/view.php?id=3089`.

This is a warmup problem. You do **not need to hand in** this problem.

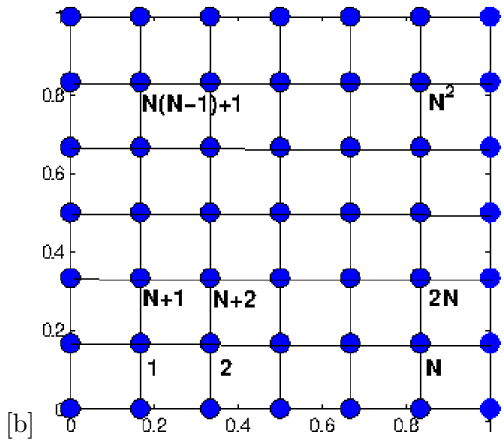
Exercise 1 Finite Differences for Poisson Equation in 2D

In this problem we consider the Finite Differences discretization of the Poisson problem on the unit square:

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega := (0,1)^2, \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{1}$$

for a bounded and continuous function $f \in \mathcal{C}^0(\overline{\Omega})$.

We consider a regular tensor product grid with meshwidth $h := (N+1)^{-1}$ and we assume a lexicographic numbering of the interior vertices of the mesh as depicted in Fig.1.



[b]

Figure 1: Lexicographic numbering of vertices of the equidistant tensor product mesh.

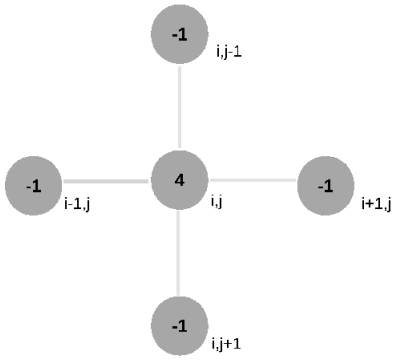


Figure 2: 5-point stencil used in this problem.

We consider the 5-point stencil finite difference scheme for the operator $-\Delta$ described by the 5-points stencil shown in Fig. 2.

1a)

Write the system

$$\mathbf{A}\mathbf{u} = \mathbf{F} \tag{2}$$

corresponding to the discretization of (1) using the stencil in Fig. 2, specifying the matrix $\mathbf{A}$ and the vectors $\mathbf{F}$ and $\mathbf{u}$.

Solution: The equations for the discretized system are:

$$\frac{4u_{i,j} - u_{i,j-1} - u_{i,j+1} - u_{i-1,j} - u_{i+1,j}}{h^2} = f(x_j, y_j), \quad \text{for } (i,j) \in \{1,\dots,N\}^2 \tag{3}$$

If we denote by (x_j, y_j) $(j = 0,\dots,(N+1)^2)$ the coordinates of the node j according to the lexicographic order of Fig. 1, u_j, $j = 0,\dots,(N+1)^2$, denotes the discrete solution, and $f_j := f(x_j, y_j)$ we can rewrite the above expression (taking into account boundary conditions) as:

$$\frac{4u_j - u_{j-1} - u_{j+1} - u_{j-N} - u_{j+N}}{h^2} = f_j, \quad \text{for } (i,j) \in \{1,\dots,N\}^2 \tag{4}$$

Thus, writing the equations as a system, we have a block tridiagonal matrix $\mathbf{A} \in \mathbb{R}^{N^2 \times N^2}$

$$\mathbf{A} = \frac{1}{h^2} \begin{pmatrix} \mathbf{B} & -\mathbf{I} & 0 & \dots & \dots & 0 \\ -\mathbf{I} & \mathbf{B} & -\mathbf{I} & 0 & \dots & 0 \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ 0 & \dots & 0 & -\mathbf{I} & \mathbf{B} & -\mathbf{I} \\ 0 & \dots & \dots & 0 & -\mathbf{I} & \mathbf{B} \end{pmatrix}$$

where $\mathbf{I} \in \mathbb{R}^{N \times N}$ is the identity matrix, and $\mathbf{B} \in \mathbb{R}^{N \times N}$ is the tridiagonal matrix

$$\mathbf{B} = \begin{pmatrix} 4 & -1 & 0 & \dots & \dots & 0 \\ -1 & 4 & -1 & 0 & \dots & 0 \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ 0 & \dots & 0 & -1 & 4 & -1 \\ 0 & \dots & \dots & 0 & -1 & 4 \end{pmatrix}$$

The vectors $\mathbf{F}$ and $\mathbf{u}$ are given by $(\mathbf{F})_j = f(x_j)$ and $(\mathbf{u})_j = u_j$, $j = 1,\dots,N$.

1b)

In the template file `finite_difference.cpp`, implement the function

```
void createPoissonMatrix2D(SparseMatrix& A, int N),
```

to construct the matrix $\mathbf{A}$ in (2), where N denotes the number of interior grid points along one dimension, with `typedef Eigen::SparseMatrix<double> SparseMatrix`. Assume the matrix `A` to have an uninitialized size at the beginning.

Solution: See listing 1 for the code.

Listing 1: Implementation for `createPoissonMatrix2D`

```
//! Create the Poisson matrix for 2D finite difference.
//! @param[out] A will be the Poisson matrix (as in the exercise)
//! @param[in] N number of elements in the x-direction
void createPoissonMatrix2D(SparseMatrix& A, int N) {
     // Fill the matrix A using setFromTriplets - method (see other exercise
    // for how to use it).
    //// NPDE_START_TEMPLATE
    std::vector<Triplet> triplets;
    A.resize(N*N, N*N);
    triplets.reserve(5*N*N-4*N);
    for (int i = 0; i < N*N; ++i) {
        triplets.push_back(Triplet(i, i, 4));
        if (i % N != 0) {
            triplets.push_back(Triplet(i, i-1, -1));
        }
        if (i % N != N - 1) {
            triplets.push_back(Triplet(i, i+1, -1));
        }
        if (i >= N) {
            triplets.push_back(Triplet(i, i-N, -1));
        }
        if (i < N*N - N ) {
            triplets.push_back(Triplet(i, i+N, -1));
        }
    }

    A.setFromTriplets(triplets.begin(), triplets.end());
    //// NPDE_END_TEMPLATE
}
```

1c)

In the template file `finite_difference.cpp`, implement the function

```
void createRHS(Vector& rhs, FunctionPointer f, int N, double dx),
```

to build the vector $\mathbf{F}$ in (2), with `typedef Eigen::VectorXd Vector` and `typedef double(*FunctionPointer)(double, double)`. The argument `f` is a function pointer to the function f in (1), `N` is the number of interior grid points and `dx` is cell width. Again, assume that the vector `rhs` has uninitialized size when passed in input.

Solution: See listing 2 for the code.

Listing 2: Implementation for `createRHS`

```
//! Create the Right hand side for the 2D finite difference
//! @param[out] rhs will at the end contain the right hand side
//! @param[in] f the right hand side function f
//! @param[in] N the number of points in the x direction
//! @param[in] dx the cell width
void createRHS(Vector& rhs, FunctionPointer f, int N, double dx) {
    rhs.resize(N * N);
    // fill up RHS
    // remember that the index (i,j) corresponds to i*N+j
    //// NPDE_START_TEMPLATE
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            const double x = (i + 1) * dx;
            const double y = (j + 1) * dx;
            rhs[j * N + i] = dx * dx * f(x, y);
        }
    }
    //// NPDE_END_TEMPLATE
}
```

1d)

In the template file `finite_difference.cpp`, implement the function

```
void poissonSolve(Vector& u, FunctionPointer f, int N),
```

to solve the system (2), with `u` the vector containing the values of the approximate solution at all the grid points, *including those on the boundary*, and the other arguments as in the previous subproblems.

Solution: See listing 3 for the code.

Listing 3: Implementation for `poissonSolve`

```
//! Solve the Poisson equation in 2D
//! @param[out] u will contain the solution u
//! @param[in] f the function pointer to f
//! @param[in] N the number of points to use (in x direction)
void poissonSolve(Vector& u, FunctionPointer f, int N) {
    // Solve Poisson 2D here
    //// NPDE_START_TEMPLATE
    double dx = 1.0 / (N + 1);
```

```
    SparseMatrix A;
    createPoissonMatrix2D(A, N);

    Vector rhs;
    createRHS(rhs, f, N, dx);

    Eigen::SparseLU<SparseMatrix> solver;
    solver.compute(A);

    if ( solver.info() != Eigen::Success) {
        throw std::runtime_error("Could not decompose the matrix");
    }
    u.resize((N + 2) * (N + 2));
    u.setZero();

    Vector innerU = solver.solve(rhs);

    // Copy vector to inner u.
    for (int i = 1; i < N + 1; ++i) {
        for (int j = 1; j < N + 1; ++j) {
            u[i * (N + 2) + j] = innerU[(i - 1) * N + j - 1];
        }
    }

    //// NPDE_END_TEMPLATE
}
```

1e)

Plot the discrete solution that you get from subproblem **1d)** for $f(x, y) = 8\pi^2 \sin(2\pi x) \sin(2\pi y)$ and $N = 50$, and compare it to the exact solution $u(x, y) = \sin(2\pi x) \sin(2\pi y)$.

Solution: See Fig. 3 for the discrete solution.

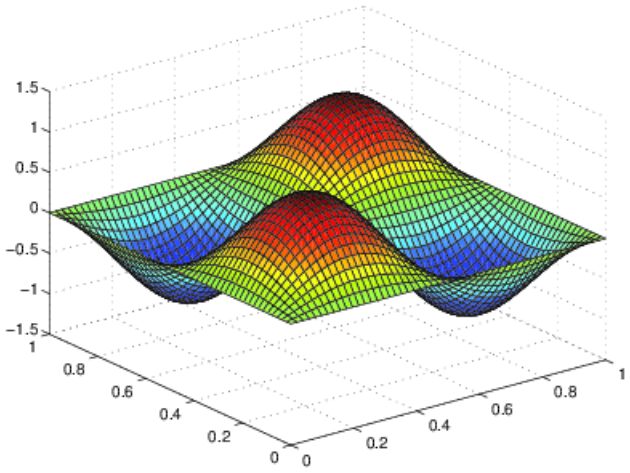


Figure 3: Plot for subproblem **1e)**.