0.1 Rectangular Domain in Two Dimension

Consider a mixed Poisson problem for the rectangular domain $\Omega = \{(x, y) \mid 0 \leq x \leq L_x,\ 0 \leq y \leq L_y\}$. Denote the numbers of mesh points in x- and y-directions by L and M, respectively.

0.1.1 Program Sequence

In order to numerically solve the problem for a rectangle in cartesian coordinates, one needs to perform the following steps.

1. Create a new folder and copy all files of some example that solves the Poisson equation in a rectangle into that new folder.

2. Modify the file `forcing.m`: change the line `f_mat(i,j)=Q` where $Q(x_i, y_j)$ is the desired forcing function.

3. Modify the file `dirichlet_boundary.m`: change the `if`-conditions as needed, and change the lines `diriBC(i,j)=Q` so that $Q = v_D(x_i, y_j)$ specifies the desired Dirichlet boundary condition of the problem.

4. Modify the file `neumann_boundary.m`: change the `if`-conditions as needed, and change the lines `neuBC(i,j)=Q` so that $Q = v_N(x_i, y_j)$ specifies the desired Neumann boundary condition of the problem.

5. Modify the files `x_refine_function.m` and `y_refine_function.m` to introduce necessary mesh refinement. The input parameter `in` is a vector that will contain homogeneous mesh points; the output parameter `out` has to either equal `in` (no mesh refinement) or a function that acts on `in` to concentrate/rarify mesh where needed. Each refinement function must be a monotone increasing function. It is the user's responsibility to satisfy the conditions

   $$\texttt{in(1)=out(1)}, \quad \texttt{in(end)=out(end)} \tag{1}$$

   in order to preserve the square side lengths.

   [Often it is useful to plot `out` as a function of `in` using Matlab `plot(in, out)`.

6. Run the solver routine:

```
[xs ys v relres iter resvec] = rectangle_2d_poisson(
    x_max,y_max,num_xs,num_ys,'x_refine_function','y_refine_function',
    'dirichlet_boundary','neumann_boundary','forcing',useiter)
```

where `x_max` and `y_max` are the domain sizes L_x, L_y in the `x`-and `y`-directions respectively; `num_xs` and `num_ys` are the numbers of mesh points L, M in the corresponding directions; `x_refine_function.m` and `y_refine_function.m` are the filenames of external `Matlab` functions for mesh refinement; `dirichlet_boundary.m` and `neumann_boundary.m` are the filenames of external `Matlab` functions that generate the Dirichlet and Neumann boundary conditions; `forcing.m` is the name of an external `Matlab` function that generates the forcing term at every point of the domain; `useiter` is an optional parameter (default value 0) which, if set to a nonzero value, forces the solver to use an iterative method to solve the sparse linear system.

The normal output of the solver consists of a matrix of approximate solutions `v` of the problem at mesh points stored in matrices `xs` and `ys`. The solution is ready for plotting using the routine

```
surf(xs,ys,v);
```

When the parameter `useiter` is set to a nonzero value, an iterative solver for a linear sparse system is used, and the output values `relres`, `iter` and `resvec` are the final residual, the number of iterations used, and a vector of the history of the residuals, respectively.

The solver applies the following rules to determine which boundary condition to use at each boundary point.

- If at a given boundary point, a Dirichlet boundary condition is specified, then it is used, and the Neumann boundary condition is ignored (if specified at all).
- If at a given boundary point, a Neumann boundary condition needs to be specified, the value provided for `diriBC` for that point in the file `dirichlet_boundary.m` must be set to `NaN`.

The above rules let the user avoid duplicating `if`-conditions when the problem involves a mixture of Dirichlet and Neumann boundary parts.

0.2 Disk Domain in Two Dimensions

In order to numerically solve the problem for a disk of radius R in polar coordinates, the following steps need to be taken.

0.2.1 Program Sequence

1. Create a new folder and copy all files of some example that solves the Poisson equation in a disk into that new folder.

2. Modify the file `forcing.m`: change the line `f_mat(i,j)=Q` where $Q(r_j, \phi_i)$ is the desired forcing function.

3. Modify the file `dirichlet_boundary.m`: change the `if`-conditions as needed, and change the lines `diriBC(i)=Q` so that $Q = v_D(\phi_i)$ specifies the desired Dirichlet boundary condition of the problem.

4. Modify the file `neumann_boundary.m`: change the `if`-conditions as needed, and change the lines `neuBC(i)=Q` so that $Q = v_N(\phi_i)$ specifies the desired Neumann boundary condition of the problem.

5. Modify the files `r_refine_function.m` and `phi_refine_function.m`, similarly to the case of the rectangle. Each refinement function must be a monotone increasing function. It is the user's responsibility to satisfy the conditions to preserve the domain sizes for both variables. Note that for the function `phi_refine_function.m`, one must have

   $$\texttt{in(1)=out(1)=0}, \quad \texttt{in(end)=out(end)=}2\pi.$$

6. Run the solver routine:

   ```
   [xs ys v rs phis relres iter resvec] = polar_2d_poisson(
       R_max,num_rs,num_phis,'r_refine_function','phi_refine_function',
       'dirichlet_boundary','neumann_boundary','forcing',useiter)
   ```

   where `R_max` is the disk radius R, `num_rs` and `num_phis` are the numbers of mesh points L, M in the radial and angular polar directions; `r_refine_function.m` and `phi_refine_function.m` are the filenames of external `Matlab` functions for mesh refinement. For the output values `v` of the numerical solution, the routine outputs both the matrices of corresponding polar coordinates `rs`, `phis` and the corresponding Cartesian coordinates `xs`, `ys`. The rest of input and output parameters is identical to those listed in Section 0.1.1.

0.3 Spherical Domain in Three Dimensions

Finally, in order to numerically solve the problem for a sphere of radius R in spherical coordinates (ρ, θ, ϕ), one needs to perform the following steps.

0.3.1 Program Sequence

1. Create a new folder and copy all files of some example that solves the Poisson equation in a sphere into that new folder.

2. Modify the file `forcing.m`: change the line `f_mat(i,j,k)=Q` where $Q(\rho_k, \theta_j, \phi_i)$ is the desired forcing function.

3. Modify the file `dirichlet_boundary.m`: change the `if`-conditions as needed, and change the lines `diriBC(i,j)=Q` so that $Q = v_D(\theta_j, \phi_i)$ specifies the desired Dirichlet boundary condition of the problem.

4. Modify the file `neumann_boundary.m`: change the `if`-conditions as needed, and change the lines `neuBC(i,j)=Q` so that $Q = v_N(\theta_j, \phi_i)$ specifies the desired Neumann boundary condition of the problem.

5. Modify the files `r_refine_function.m`, `phi_refine_function.m`, and `theta_refine_functi` similarly to the case of the rectangle. It remains the user's responsibility to satisfy the conditions for each refinement function to preserve the domain sizes for all three variables. Note that for the function `theta_refine_function.m`, one must have

   `in(1)=out(1)=0`, `in(end)=out(end)=`π,

   and for the function `phi_refine_function.m`, one must have

   `in(1)=out(1)=0`, `in(end)=out(end)=`2π.

6. Run the spherical solver:

```
[rs thetas phis v xs ys zs relres iter resvec] = sphere_3d_poisson(
    R_max,num_rs,num_thetas,num_phis,
    'r_refine_function','theta_refine_function.m', 'phi_refine_function',
    'dirichlet_boundary','neumann_boundary','forcing',useiter)
```

   where `R_max` is the sphere radius R. The numerical solution `v` is output in the form of a three-dimensional array, together with the corresponding values of spherical coordinates `rs`, `thetas`, `phis`, and Cartesian coordinates `xs`, `ys`, `zs`, and is ready for plotting. The rest of the input and output parameters is analogous to parameters described in Sections 0.1.1 and 0.3.1.

   Unlike the two-dimensional solvers, for the spherical solver, the default value of `useiter=1` is used, i.e., an iterative sparse matrix solver is employed by default.