Note

bcynuaa

January 30, 2024

Contents

1 Problem 1

1.1 Question 1

Suppose we have a v distributed in domain, we have:

$$\int_\Omega (-\Delta u - f)(u - v)\mathrm{d}\Omega = 0 \tag{1}$$

considering an arbitrary w, we use integrate by part:

$$\int_\Omega w\Delta u\mathrm{d}\Omega = \int_{\partial\Omega} w\frac{\partial u}{\partial n}\mathrm{d}\Gamma - \int_\Omega \nabla w \cdot \nabla u\mathrm{d}\Omega \tag{2}$$

and by Cauchy-Schwarz inequality:

$$\int_\Omega \nabla w \cdot \nabla u\mathrm{d}\Omega \leq \frac{1}{2}\left(\int_\Omega \nabla w \cdot \nabla w\mathrm{d}\Omega + \int_\Omega \nabla u \cdot \nabla u\mathrm{d}\Omega\right) \tag{3}$$

apply (2) to (1), we have:

$$\begin{aligned}
-\int_\Omega u\nabla^2 u\mathrm{d}\Omega + \int_\Omega v\nabla^2 u\mathrm{d}\Omega - \int_\Omega fu\mathrm{d}\Omega + \int_\Omega fv\mathrm{d}\Omega = 0 \\
\int_\Omega \nabla u \cdot \nabla u\mathrm{d}\Omega - \int_{\partial\Omega} u\frac{\partial u}{\partial n}\mathrm{d}\Gamma - \int_\Omega fu\mathrm{d}\Omega = \\
\int_\Omega \nabla v \cdot \nabla u\mathrm{d}\Omega - \int_{\partial\Omega} v\frac{\partial u}{\partial n}\mathrm{d}\Gamma - \int_\Omega fv\mathrm{d}\Omega \\
\int_\Omega \nabla u \cdot \nabla u\mathrm{d}\Omega - \int_{\partial\Omega} u\bar{t}\mathrm{d}\Gamma - \int_\Omega fu\mathrm{d}\Omega = \\
\int_\Omega \nabla v \cdot \nabla u\mathrm{d}\Omega - \int_{\partial\Omega} v\bar{t}\mathrm{d}\Gamma - \int_\Omega fv\mathrm{d}\Omega
\end{aligned} \tag{4}$$

apply (3) above, we will have:

$$\begin{aligned}
\int_\Omega \nabla u \cdot \nabla u\mathrm{d}\Omega - \int_{\partial\Omega} u\bar{t}\mathrm{d}\Gamma - \int_\Omega fu\mathrm{d}\Omega \leq \\
\frac{1}{2}\int_\Omega (\nabla u \cdot \nabla u + \nabla v \cdot \nabla v)\Omega - \int_{\partial\Omega} v\bar{t}\mathrm{d}\Gamma - \int_\Omega fv\mathrm{d}\Omega
\end{aligned} \tag{5}$$

denote $U[w]$ as below:

$$U[w] = \int_\Omega \frac{1}{2}\nabla w \cdot \nabla w\mathrm{d}\Omega - \int_{\partial\Omega} w\bar{t}\mathrm{d}\Gamma - \int_\Omega fw\mathrm{d}\Omega \tag{6}$$

we will have:

$$U[u] \leq U[v] \tag{7}$$

finally we reduce the Poisson equation to a minimization problem, and the solution of the Poisson equation is the minimizer of $U[u]$.

1.2 Question 2

From integral by part(2), we have:

$$\int_\Omega \nabla w \cdot \nabla \tilde{u} \mathrm{d}\Omega = \int_{\partial\Omega} w \frac{\partial \tilde{u}}{\partial n} \mathrm{d}\Gamma - \int_\Omega w \Delta \tilde{u} \mathrm{d}\Omega \tag{8}$$

thus we have:

$$\int_\Omega w(-\Delta \tilde{u} - f) \mathrm{d}\Omega + \int_{\partial\Omega} w \left(\frac{\partial \tilde{u}}{\partial n} - \bar{t} \right) \mathrm{d}\Gamma = 0 \tag{9}$$

when $w = 0$ at $\partial\Omega$, we have:

$$\int_\Omega w(-\Delta \tilde{u} - f) \mathrm{d}\Omega = 0 \tag{10}$$

for the arbitrary w, we have:

$$-\Delta \tilde{u} - f = 0 \tag{11}$$

Consider $\psi = \tilde{u} - u$, which satisfies:

$$\nabla^2 \psi = 0 \quad \text{in} \quad \Omega, \quad \psi = 0 \quad \text{on} \quad \partial\Omega \tag{12}$$

thus:

$$0 = \int_\Omega \psi \nabla^2 \psi \mathrm{d}\Omega = \int_{\partial\Omega} \psi \frac{\partial \psi}{\partial n} \mathrm{d}\Gamma - \int_\Omega \nabla \psi \cdot \nabla \psi \mathrm{d}\Omega \tag{13}$$

we will have:

$$\nabla \psi = \vec{0} \quad \text{in} \quad \Omega \tag{14}$$

for $\psi = 0$ at $\partial\Omega$, we have:

$$\psi = 0 \quad \text{in} \quad \Omega \tag{15}$$

which indicates that $\tilde{u} = u$.

1.3 Question 3

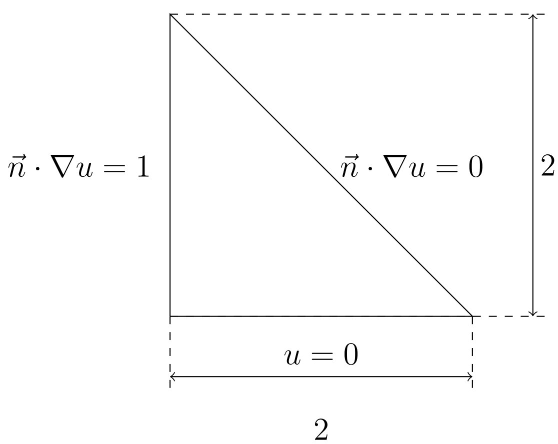


Figure 1: fig: Problem Setting

Considering a shape function ϕ_i in single triangular element, we have:

$$\sum_j a_j \int_{\Omega_e} \nabla\phi_i \cdot \nabla\phi_j \mathrm{d}\Omega = \int_{\partial\Omega_e} \phi_i \frac{\partial u}{\partial n} \mathrm{d}\Gamma \tag{16}$$

Let's consider an element as below:

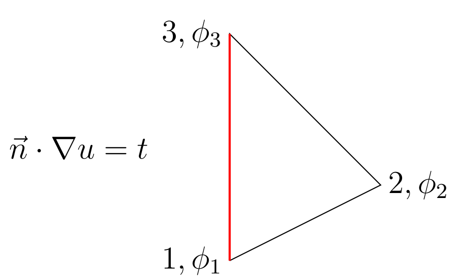


Figure 2: fig: Single Element with Neumann Boundary Condition

we will have the integral as below:

$$\begin{aligned}\int_3^1 \phi_2 t dl &= 0 \\ \int_3^1 \phi_1 t dl &= \frac{t}{2} l_{13} \\ \int_3^1 \phi_3 t dl &= \frac{t}{2} l_{13}\end{aligned} \tag{17}$$

this integral seems to devide part tl_{13} into two parts, which are added separately to node 1 and node 3.

You may read the report last year in folder 'FEM/Autumn2022' for more details of the derivation of the integral. Another option is to watch video bilibili: https://www.bilibili.com/video/BV1qD4y1J7ZU/?spm__id__from=333.999.0.0, a video by me on last year's homework.

Generally, we have:

$$\left[\sum_e \int_{\Omega_e} \nabla\phi_i^e \cdot \nabla\phi_j^e \mathrm{d}\Omega\right](\tilde{u} + u_\Gamma) = \left[\sum_e \int_{\partial\Omega_e} \phi_i^e \frac{\partial u}{\partial n} \mathrm{d}\Gamma\right] \tag{18}$$

1.3.1 MMA Solution

To verify the solution by hand-writing code, we use Mathematica to solve the problem first. The notebook is printed as below:

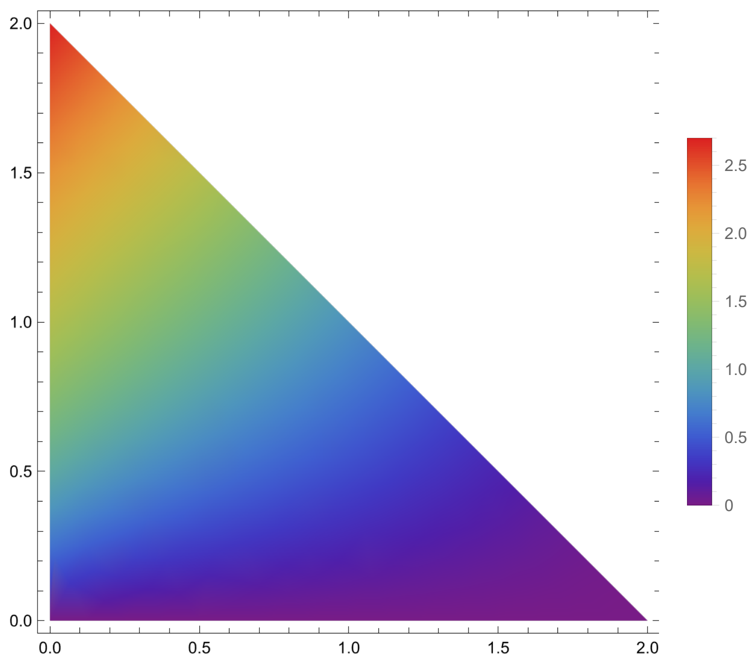
In[∘]:=
```
Clear["Global`*"];
清除
nodelist = {{0, 0}, {2, 0}, {0, 2}};
tri = Polygon[nodelist];
      多边形
tri
```

Out[∘]= Polygon[Number of points: 3 Embedding dimension: 2]

In[∘]:=
```
res = NDSolveValue[{-∇_{x,y}^2 u[x, y] ==
      数值解的值
  NeumannValue[1., x == 0],
  诺伊曼边值
 DirichletCondition[u[x, y] == 0., y == 0]}, u, {x, y} ∈
 狄里克雷条件
 tri]
```

Out[∘]= InterpolatingFunction[Domain: {{0., 2.}, {0., 2.}} Output: scalar]

In[∘]:=
```
figure = DensityPlot[res[x, y], {x, y} ∈ tri,
         密度图
  PlotLegends → Automatic, ColorFunction → "Rainbow"]
  绘图的图例      自动          颜色函数
```

Out[∘]=


In[∘]:=
```
Export["..//Desktop//problem1_mma_solution.pdf", figure];
导出
Export["..//Desktop//problem1_mma_solution.png", figure]
导出
```

Out[∘]= ..//Desktop//problem1_mma_solution.png

1.3.2 Julia Code Solution

First we need include the package we use:

```
using DelaunayTriangulation, CairoMakie;
using SparseArrays, LinearAlgebra;
```

here, `DelaunayTriangulation` is used to generate the mesh, `CairoMakie` is used to plot the mesh and contour. `SparseArrays` and `LinearAlgebra` are used to solve the linear system.

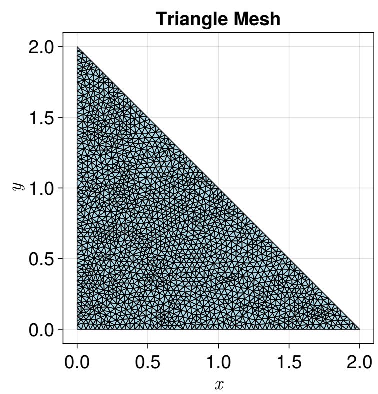


Figure 3: Triangle Mesh

Then we define the domain and generate / draw the triangular mesh, shown in Fig. 3.

```
const triangle_edge_length = 2.;
const max_area = 1e-3;
const min_angle = 31.5;
points = [
    (0., 0.),
    (triangle_edge_length, 0.),
    (0., triangle_edge_length),
    (0., 0.)
];
boundary_nodes, points = convert_boundary_points_to_indices(
    points);
triangle = triangulate(points; boundary_nodes);
refine!(triangle; max_area=max_area, min_angle=min_angle);

```

```
figure = Figure(fontsize=24)
axes = Axis(figure[1, 1], title="Triangle Mesh", titlealign=:
    center, width=400, height=400, xlabel=L"$x$", ylabel=L"$y$")
triplot!(axes, triangle, triangle_color=:lightblue)
save("triangle_mesh.pdf", figure)
save("triangle_mesh.png", figure)
figure
```

In the following code, we generate the stiffness matrix:

```
function generateStiffnessMatrix(triangle::Triangulation)
    n_nodes = length(triangle.points);
    stiffness_matrix = spzeros(n_nodes, n_nodes);
    for triangle_element in each_triangle(triangle)
        id1, id2, id3 = triangle_element;
        x1, y1 = triangle.points[id1];
        x2, y2 = triangle.points[id2];
        x3, y3 = triangle.points[id3];
        double_area = det(
            [1 x1 y1;
            1 x2 y2;
            1 x3 y3]
        );
        k11 = (x2 - x3)^2 + (y2 - y3)^2;
        k12 = (x1 - x3) * (-x2 + x3) + (y1 - y3) * (-y2 + y3);
        k13 = (x1 - x2) * (x2 - x3) + (y1 - y2) * (y2 - y3);
        k22 = (x1 - x3)^2 + (y1 - y3)^2;
        k23 = (x1 - x2) * (-x1 + x3) + (y1 - y2) * (-y1 + y3);
        k33 = (x1 - x2)^2 + (y1 - y2)^2;
        # first row
        stiffness_matrix[id1, id1] += k11 / double_area / 2;
        stiffness_matrix[id1, id2] += k12 / double_area / 2;
        stiffness_matrix[id1, id3] += k13 / double_area / 2;
        # second row
        stiffness_matrix[id2, id1] += k12 / double_area / 2;
        stiffness_matrix[id2, id2] += k22 / double_area / 2;
        stiffness_matrix[id2, id3] += k23 / double_area / 2;
        # third row
        stiffness_matrix[id3, id1] += k13 / double_area / 2;
        stiffness_matrix[id3, id2] += k23 / double_area / 2;
        stiffness_matrix[id3, id3] += k33 / double_area / 2;
    end
    return stiffness_matrix;
end

stiffness_matrix = generateStiffnessMatrix(triangle);
```

```
stiffness_matrix

figure = Figure(fontsize=24)
axes = Axis(figure[1, 1], title="Stiffness Matrix", titlealign=:center, width=400, height=400)
spy!(axes, rotr90(stiffness_matrix), markersize=4, marker=:circle, framecolor=:blue)
hidedecorations!(axes)
save("../images/stiffness_matrix.pdf", figure)
save("../images/stiffness_matrix.png", figure)
figure
```

The stiffness matrix is a sparse matrix, which is shown in Fig. 4.

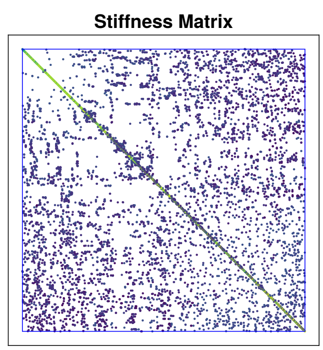


Figure 4: Stiffness Matrix

For Neumann boundary condition, we need to generate the source term vector:

```
function isLeftBoundaryNode(x, y)
    return x == 0.
end

function generateSourceVector(triangle::Triangulation, neumann_boundary_value::Float64)
    n_nodes = length(triangle.points);
    source_vector = zeros(n_nodes);
    for triangle_element in each_triangle(triangle)
        boundary_nodes = [];
        id1, id2, id3 = triangle_element;
```

```julia
        x1, y1 = triangle.points[id1];
        x2, y2 = triangle.points[id2];
        x3, y3 = triangle.points[id3];
        node_dict = Dict(id1 => (x1, y1), id2 => (x2, y2), id3 => (x3, y3));
        for (id, (x, y)) in node_dict
            if isLeftBoundaryNode(x, y)
                push!(boundary_nodes, id);
            end
        end
        if length(boundary_nodes) == 2
            n1_id, n2_id = boundary_nodes;
            n1_x, n1_y = node_dict[n1_id];
            n2_x, n2_y = node_dict[n2_id];
            edge_length = sqrt((n1_x - n2_x)^2 + (n1_y - n2_y)^2);
            source_vector[n1_id] += neumann_boundary_value * edge_length / 2;
            source_vector[n2_id] += neumann_boundary_value * edge_length / 2;
        else
            continue;
        end
    end
    return source_vector;
end
source_vector = generateSourceVector(triangle, 1.);
```

The linear problem has Dirichlet boundary condition, thus we need to modify the stiffness matrix and source vector. Below code is fit for the problem with Dirichlet boundary condition:

```julia
function solve(
    stiffness_matrix::SparseMatrixCSC,
    source_vector::Vector,
    known_nodes_ids::Vector,
    known_nodes_values::Vector
)::Vector
    @assert length(known_nodes_ids) == length(known_nodes_values);
    n_nodes = length(source_vector);
    solution_vector = zeros(n_nodes);
    solution_vector[known_nodes_ids] .= known_nodes_values;
    unknown_nodes_ids = setdiff(1:n_nodes, known_nodes_ids);
    part_stiffness_matrix = stiffness_matrix[unknown_nodes_ids, unknown_nodes_ids];
```

```
    part_source_vector = (source_vector .- stiffness_matrix *
        solution_vector)[unknown_nodes_ids];
    solution_vector[unknown_nodes_ids] .= part_stiffness_matrix
        \ part_source_vector;
    return solution_vector;
end
```

Then we need to get the bottom nodes ID and put them to function above:

```
function isBottomBoundaryNode(x, y)
    return y == 0.;
end

known_nodes_ids = [];
known_nodes_values = [];
for (id, (x, y)) in enumerate(triangle.points)
    if isBottomBoundaryNode(x, y)
        push!(known_nodes_ids, id);
        push!(known_nodes_values, 0.);
    else
        continue;
    end
end
solution_vector = solve(stiffness_matrix, source_vector,
    known_nodes_ids, known_nodes_values);
```

Finally, we plot the solution:

```
figure = Figure(fontsize=24)
axes = Axis(figure[1, 1], title="Solution", titlealign=:center,
    width=400, height=400, xlabel=L"$x$", ylabel=L"$y$")
tr = tricontourf!(axes, triangle, levels=51, solution_vector,
    colormap=:gist_rainbow)
Colorbar(figure[1, 2], tr, label="Colorbar", labelpadding=10,
    width=20, height=400)
save("../images/solution.pdf", figure)
save("../images/solution.png", figure)
figure
```

The solution is shown in Fig. 5, which is consistent with the solution by Mathematica.

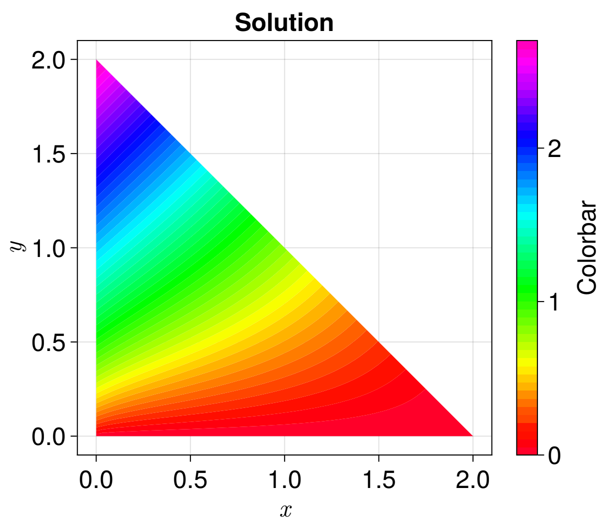


Figure 5: Solution of Problem 1

2 Problem 2