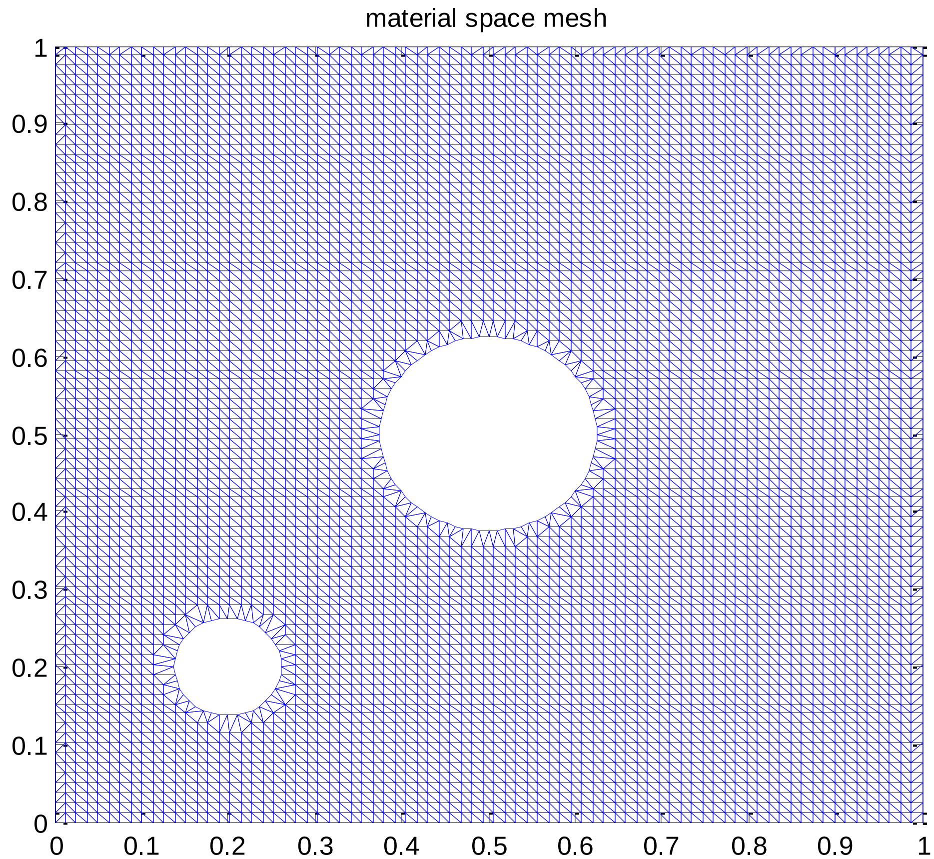
Problem 2

SOLUTION PLOT:

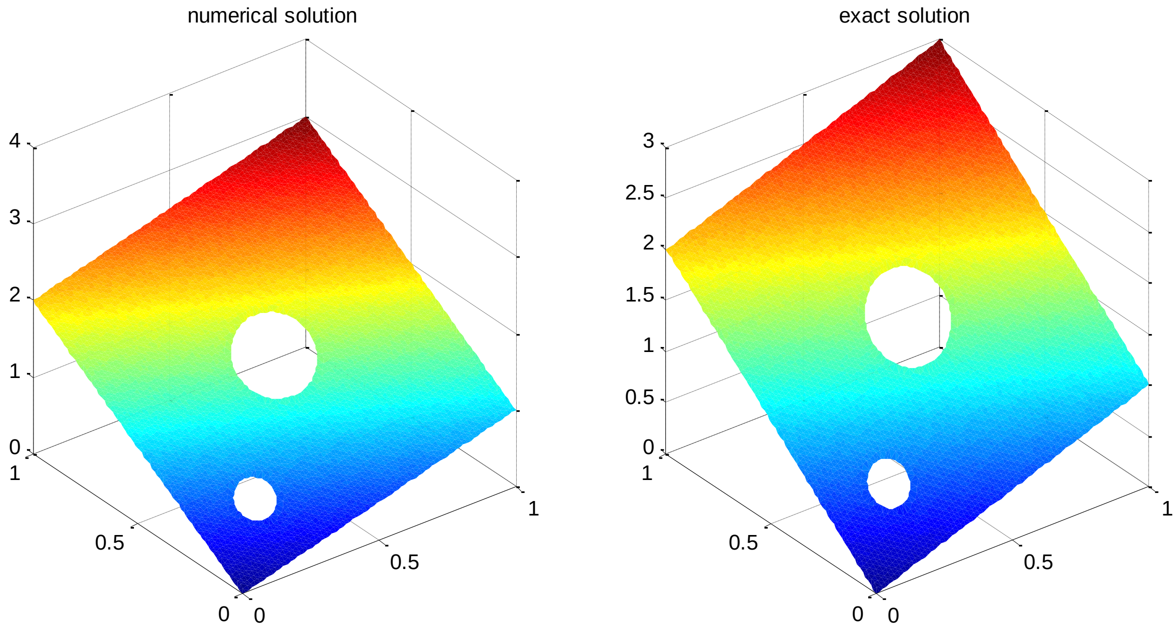


Test 1 log:

Poisson 2D FEM Solver with full Neumann boundary condition
The exact solution is:
u_exact = @(x,y)x+2*y
Time cost for loading mesh:0.11406
Time cost for computing boundary segment mesh:0.88534
Time cost for computing boundary segment normals:0.028482
Time cost for computing Neumann boundary condition:0.058502
Time cost for building linear system:7.356
minres converged at iteration 478 to a solution with relative residual 9.3e-011.
Time cost for solving with minres:0.51201
Time cost for plotting:4.9084
Error: max_abs(u - u_exact) = 0.000000
END

Test 1 plot:

u_exact =   @(x,y)x+2*y



Test 2 log:

Poisson 2D FEM Solver with full Neumann boundary condition
The exact solution is:

u_exact =

    @(x,y)x^2+y^2

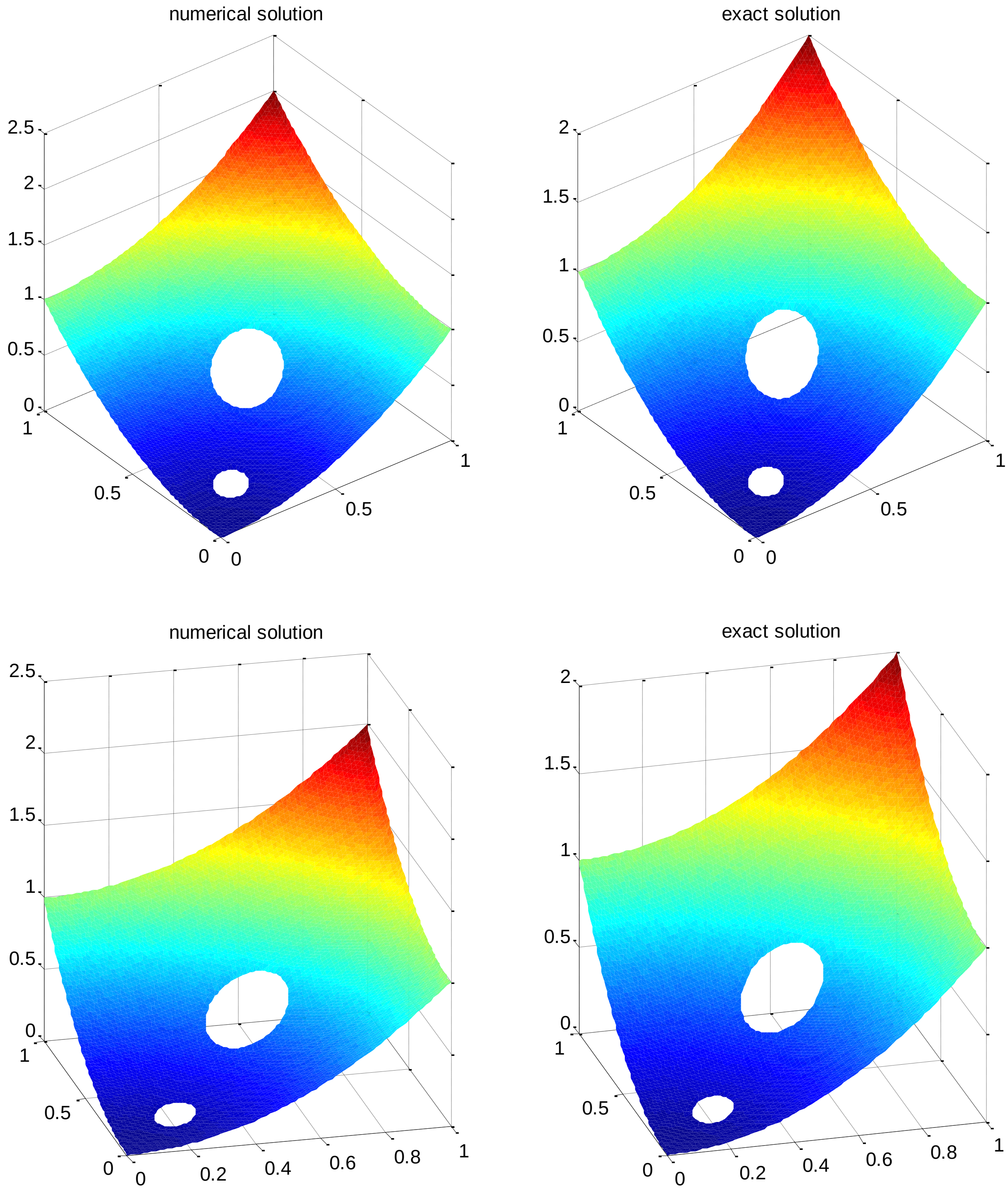
Time cost for loading mesh:0.094044
Time cost for computing boundary segment mesh:0.89073
Time cost for computing boundary segment normals:0.025945
Time cost for computing Neumann boundary condition:0.060791
Time cost for building linear system:7.6786
minres converged at iteration 359 to a solution with relative residual 8.8e-011.
Time cost for solving with minres:0.34801
Time cost for plotting:3.8702

Error: max_abs(u - u_exact) = 0.000230

END

Test 2 plot:

u_exact =   @(x,y)x^2+y^2



Test 3 log:

Poisson 2D FEM Solver with full Neumann boundary condition
The exact solution is:

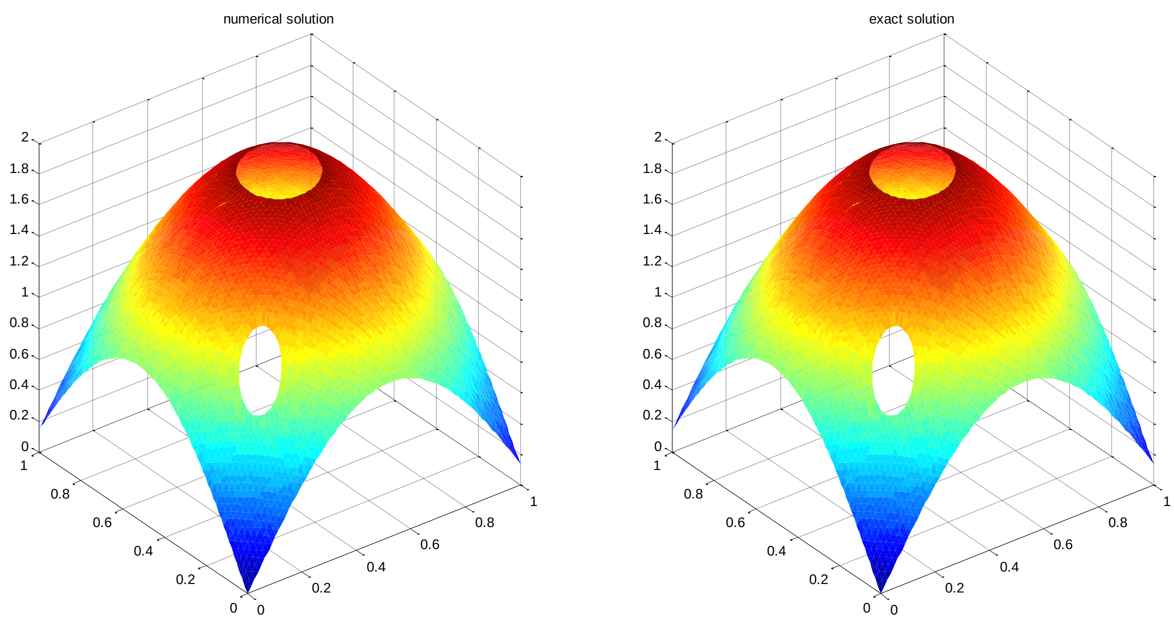
u_exact =

    @(x,y)sin(3*x)+sin(3*y)

Time cost for loading mesh:0.093307
Time cost for computing boundary segment mesh:0.88665
Time cost for computing boundary segment normals:0.02303
Time cost for computing Neumann boundary condition:0.061168
Time cost for building linear system:7.4813
minres stopped at iteration 889 without converging to the desired tolerance 1e-010
because the method stagnated.
The iterate returned (number 889) has relative residual 0.069.
Time cost for solving with minres:0.78513
Time cost for plotting:4.1036
Error: max_abs(u - u_exact) = 0.000808
END

Test 3 plot:

u_exact = @(x,y)sin(3*x)+sin(3*y)



MATLAB CODE:

```
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Poisson 2d FEM solver.m
%
% Solves -div(grad(u)) = f
%       n dot grad(u) = g (full Neumann boundary)
% over a triangle mesh
%
% Equivalent to solving Au=G+F, where Aij = int_Omega{Grad_Ni dot Grad_Nj}
%                                    Gi  = int_PartialOmega{g Ni}
%                                    Fi  = int_Omega{f Ni}
%
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
clear all; close all; clc
fprintf('###  Poisson 2D FEM Solver with full Neumann boundary condition  ###\n')

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% input the problem
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

test_number = 3;

if test_number == 1
   u_exact = @(x,y) x+2*y;                  % exact solution
   grad_u_exact = @(x,y) [1, 2];      % exact grad(u)
   f = @(x,y) 0;                       % exact forcing term

elseif test_number == 2
   u_exact = @(x,y) x^2+y^2;                  % exact solution
   grad_u_exact = @(x,y) [2*x, 2*y];      % exact grad(u)
   f = @(x,y) -4;                      % exact forcing term

elseif test_number == 3
   u_exact = @(x,y) x^3+y^3+x^2+y^2;                  % exact solution
   grad_u_exact = @(x,y) [3*x^2+2*x, 3*y^2+2*y];      % exact grad(u)
   f = @(x,y) -(6*x+2+6*y+2);                      % exact forcing term
end

fprintf('The exact solution is: \n')
display(u_exact)

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% some pre-processing for the mesh
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

tic;

elements = load('mesh_with_holes.dat');                              % element data
N_elements = size(elements,1);
nodes = load('nodes.dat');                                          % node data
```

```
N_nodes = size(nodes,1);
time_cost = toc; display(strcat('Time cost for loading mesh: ',num2str(time_cost))); tic;

boundary_segments = generate_boundary_segments_from_mesh(elements,nodes);    % boundary segment data
boundary_nodes = boundary_segments(:,1);                                      % boundary node data
time_cost = toc; display(strcat('Time cost for computing boundary segment mesh: ',num2str(time_cost))); tic;

[nodal_normals boundary_segment_normals] = compute_normals(elements,nodes,boundary_segments); %
outward normals on nodes (zero for non-boundary nodes) and on boundary segs
time_cost = toc; display(strcat('Time cost for computing boundary segment normals: ',num2str(time_cost))); tic;

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% find the proper input Neumann boundary condition
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

[g_nodal g_boundary] =
compute_flux(elements,nodes,boundary_segments,nodal_normals,boundary_segment_normals,grad_u_exact);
  % g is defined on all nodes, but we only care about those on boundary
time_cost = toc; display(strcat('Time cost for computing Neumann boundary condition: ',num2str(time_cost)));
tic;

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% build and solve FEM system Au=G+F
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

[A G F] = build_system(elements,nodes,boundary_segments,f,g_boundary);
rhs = G+F;
time_cost = toc; display(strcat('Time cost for building linear system: ',num2str(time_cost))); tic;

u = minres(A,rhs,1e-10,1000);
time_cost = toc; display(strcat('Time cost for solving with minres: ',num2str(time_cost))); tic;

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% plot the final solution u
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

% compute the 3d plot coordinates
for i = 1:size(nodes,1)
    x(i) = nodes(i,1);
    y(i) = nodes(i,2);
    z(i) = u(i);
    z_exact(i) = u_exact(nodes(i,1),nodes(i,2));
end

% shift u and u_exact by their minimum: because if u satisfy the equation, then u+C also does
min_z = min(z);
min_z_exact = min(z_exact);
for i = 1:size(nodes,1)
    z(i) = z(i)-min_z;
    z_exact(i) = z_exact(i)-min_z_exact;
end
```

```
% plot the material space
figure
triplot(elements,x,y);
title('material space mesh')

% plot numerical solution
figure
subplot(1,2,1)
trisurf(elements,x,y,z,'EdgeColor','none');
title('numerical solution')

% plot exact solution
subplot(1,2,2)
trisurf(elements,x,y,z_exact,'EdgeColor','none');
title('exact solution')

time_cost = toc; display(strcat('Time cost for plotting: ',num2str(time_cost))); tic;

fprintf('Error: max_abs(u - u_exact) = %f\n', max(abs(z - z_exact)))
fprintf('END\n')
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% generate_boundary_segments_from_mesh.m
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

function boundary_segments = generate_boundary_segments_from_mesh(elements,nodes)

boundary_segments = [];

N_nodes = size(nodes,1);
N_elements = size(elements,1);

edge_table = sparse(N_nodes,N_nodes); % edge_table(i,j) = 1 means edge i-j exists

for t = 1:N_elements
    edge1 = [elements(t,1) elements(t,2)];
    edge2 = [elements(t,2) elements(t,3)];
    edge3 = [elements(t,3) elements(t,1)];

    edge_table(edge1(1),edge1(2)) = 1;
    edge_table(edge2(1),edge2(2)) = 1;
    edge_table(edge3(1),edge3(2)) = 1;
end

[is,js,vs] = find(edge_table);

for e = 1:nnz(edge_table) % loop over non-zero entries
    i = is(e);
    j = js(e);
    v = vs(e);
    if v~=1
        display('Fatal error!')
    end
```

```
    if edge_table(j,i) == 0
        boundary_segments = [boundary_segments; i,j];
    end
end

end

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% compute_normals.m
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

function [nodal_normals boundary_segment_normals] = compute_normals(elements,nodes,boundary_segments)

N_nodes = size(nodes,1);
N_elements = size(elements,1);
N_boundary_segments = size(boundary_segments,1);

nodal_normals = zeros(N_nodes,2);
boundary_segment_normals = zeros(N_boundary_segments,2);

for i = 1:N_boundary_segments
    node_a_index = boundary_segments(i,1);
    node_b_index = boundary_segments(i,2);

    segment_vec = nodes(node_b_index,:)-nodes(node_a_index,:);
    segment_normal = left_handed_perpendicular_vec_normalized(segment_vec);

    boundary_segment_normals(i,:) = segment_normal;

    nodal_normals(node_a_index,:) = nodal_normals(node_a_index,:)+0.5*segment_normal;
    nodal_normals(node_b_index,:) = nodal_normals(node_b_index,:)+0.5*segment_normal;
end

for i = 1:N_nodes
    if nodal_normals(i,:)~=[0 0]
        nodal_normals(i,:) = nodal_normals(i,:)/norm(nodal_normals(i,:));
    end
end

end


function result = left_handed_perpendicular_vec_normalized(v)

    result = [v(2), -v(1)];
    result = result/norm(result);

end
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% compute_grad_nis.m
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
function grad_nis = compute_grad_nis(node_pos1,node_pos2,node_pos3,element_area)

    grad_nis = zeros(3,2);
    twice_area = 2*element_area;

    grad_nis(1,1) = (node_pos2(2)-node_pos3(2)) / twice_area;
    grad_nis(1,2) = (node_pos3(1)-node_pos2(1)) / twice_area;
    grad_nis(2,1) = (node_pos3(2)-node_pos1(2)) / twice_area;
    grad_nis(2,2) = (node_pos1(1)-node_pos3(1)) / twice_area;
    grad_nis(3,1) = (node_pos1(2)-node_pos2(2)) / twice_area;
    grad_nis(3,2) = (node_pos2(1)-node_pos1(1)) / twice_area;

end
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

% compute_flux.m

```
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

function [g_nodal g_boundary] =
compute_flux(elements,nodes,boundary_segments,nodal_normals,boundary_segment_normals,grad_u_exact)

N_nodes = size(nodes,1);
N_elements = size(elements,1);
N_boundary_segments = size(boundary_segments,1);

g_nodal = zeros(N_nodes,1);
g_boundary = zeros(N_boundary_segments,1);

for b = 1:N_boundary_segments
    node_index = boundary_segments(b,1);
    node_position = nodes(node_index,:);
    x = node_position(1);
    y = node_position(2);
    g_nodal(node_index) = dot(grad_u_exact(x,y), nodal_normals(node_index,:));

    boundary_mid_point = 0.5 * (nodes(boundary_segments(b,1),:) + nodes(boundary_segments(b,2),:));
    x = boundary_mid_point(1);
    y = boundary_mid_point(2);
    g_boundary(b) = dot(grad_u_exact(x,y), boundary_segment_normals(b,:));
end

end
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

% compute_element_area.m

```
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

function area = compute_element_area(elements,nodes,e)

    index_a = elements(e,1);
    index_b = elements(e,2);
    index_c = elements(e,3);
```

```
    A = nodes(index_a,:);
    B = nodes(index_b,:);
    C = nodes(index_c,:);

    vec1 = B-A;
    vec2 = C-A;

    area = 0.5 * abs(vec1(1)*vec2(2)-vec1(2)*vec2(1));

end

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% build_system.m
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

function [A G F] = build_system(elements,nodes,boundary_segments,f,g_boundary)

A = build_A(elements,nodes);
G = build_G(elements,nodes,boundary_segments,g_boundary);
F = build_F(elements,nodes,f);

end

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% build_A
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
function A = build_A(elements,nodes)

N_elements = size(elements,1);
N_nodes = size(nodes,1);

A = sparse(N_nodes,N_nodes);

for e = 1:N_elements
    first_node = nodes(elements(e,1),:);
    second_node = nodes(elements(e,2),:);
    third_node = nodes(elements(e,3),:);

    element_area = compute_element_area(elements,nodes,e);

    grad_nis = compute_grad_nis(first_node,second_node,third_node,element_area);

    for a = 1:3
        for b = 1:3
            A(elements(e,a),elements(e,b)) = A(elements(e,a),elements(e,b)) +
element_area*dot(grad_nis(a,:),grad_nis(b,:));
        end
    end
end

end
```

```
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% build_G
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
function G = build_G(elements,nodes,boundary_segments,g_boundary)

N_elements = size(elements,1);
N_nodes = size(nodes,1);
N_boundary_segments = size(boundary_segments,1);

G = zeros(N_nodes,1);

for b = 1:N_boundary_segments
    first_node_index = boundary_segments(b,1);
    second_node_index = boundary_segments(b,2);
    first_node = nodes(first_node_index,:);
    second_node = nodes(second_node_index,:);

    segment_length = norm(second_node-first_node);

    G(first_node_index) = G(first_node_index) + 0.5*segment_length*g_boundary(b);
    G(second_node_index) = G(second_node_index) + 0.5*segment_length*g_boundary(b);
end

end

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% build_F
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
function F = build_F(elements,nodes,f)

N_elements = size(elements,1);
N_nodes = size(nodes,1);

F = zeros(N_nodes,1);

for e = 1:N_elements
    first_node = nodes(elements(e,1),:);
    second_node = nodes(elements(e,2),:);
    third_node = nodes(elements(e,3),:);

    triangle_center_position = (first_node+second_node+third_node)/3;

    element_area = compute_element_area(elements,nodes,e);

    for a = 1:3
        F(elements(e,a)) = F(elements(e,a)) +
(1/3)*element_area*f(triangle_center_position(1),triangle_center_position(2));
    end
end

end
```