NUMERICAL ANALYSIS FOR PARTIAL DIFFERENTIAL EQUATIONS
A.A. 2019/2020

LECTURER: Prof. A. Quarteroni   TEACHING ASSISTANT AND TUTOR: Dr. F. Regazzoni

LAB 3
GALERKIN-FINITE ELEMENT METHOD FOR ELLIPTIC PROBLEMS

EXERCISE 1

In the unit square $\Omega \equiv (0,1) \times (0,1)$ we consider the following problem:

$$\begin{cases} -\Delta u = 8\pi^2 \sin(2\pi x)\sin(2\pi y) & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega. \end{cases} \tag{1}$$

1. Give the weak formulation and prove the well posedness of the problem.

   The problem considered features homogeneous Dirichlet conditions. The weak formulation is obtained multiplying the first equation of the system by a generic function $v \in V \equiv H_0^1$ and integrating by parts the Laplacian term. Calling $f = 8\pi^2 \sin(2\pi x)\sin(2\pi y)$ we get

   $$\int_\Omega \nabla u \cdot \nabla v \, d\omega = \int_\Omega f v \, d\omega.$$

   The weak formulation of problem (1) reads: Find $u \in V$ s.t.

   $$a(u,v) = \int_\Omega \nabla u \cdot \nabla v \, d\omega = \int_\Omega f v \, d\omega = F(v) \quad \forall v \in V. \tag{2}$$

   The bilinear form is symmetric, continuous, coercive in V. We have indeed

   $$|a(u,v)| \leq \|u\|_V \|v\|_V \quad \forall u, v \in V.$$

   Moreover for any $u \in V$ we have

   $$a(u,u) = \|\nabla u\|^2_{L^2(\Omega)} \geq \frac{1}{1 + c_\Omega^2} \|u\|_V^2,$$

   where in the last step we have used the Poincaré inequality. Finally we can prove that

   $$|F(v)| \leq \|f\|_{L^2(\Omega)} \|v\|_V \quad \forall v \in V.$$

   From the Lax-Milgram lemma, solution to (2) exists and is unique.

2. Implement in Matlab a linear finite element solver for problem (1), focusing in particular on the assembly of the linear system and of the load vector stemming after the Galerkin finite element approxiamtion.

   The implementation of the finite element solver involves the definition of the data structure `Dati` *within the function* `C_dati.m` *and the modification of the function* `C_matrix2D.m` *For the former we have:*

```matlab
function [Dati]=C_dati(test)

if test=='Test1'
Dati = struct( 'name',              test,...
               ... % Test name
               'domain',            [0,1;0,1],...
               ... % Domain bounds
               'exact_sol',         'sin(2*pi*x).*sin(2*pi*y)',...
               ... % Definition of exact solution
               'force',             '8*pi^2*sin(2*pi*x).*sin(2*pi*y)',...
               ... % Forcing term
               'grad_exact_1',      '2*pi*cos(2*pi*x).*sin(2*pi*y)',...
               ... % Definition of exact gradient (x comp)
               'grad_exact_2',      '2*pi*sin(2*pi*x).*cos(2*pi*y)',...
               ... % Definition of exact gradient (y comp)
               'fem',               'P1',...
               ... % P1-fe
               'nqn_1D',             4,...
               ... % Number of quad. points per edge
               'nqn_2D',             3,...
               ... % Number of quad. points per triangle
               'MeshType',          'TS', ...
               ... % Triangular Structured mesh
               'refinement_vector', [2,3,4,5],...
               ... % Refinement levels for  the error analysis
               'visual_graph',       'N',...
               ... % Visualization of the solution
               'plot_errors',        'Y' ...
               ...% Compute Errors
               );
end
```

The assembly of the linear system and of the load vector is computed in `C_matrix2D.m` *by using this strategy:*

1. *compute local stiffness matrix* `A_loc` *and local load vector* `load`
2. *add the local results to the corresponding global ones through the connectiviy matrix* `connectivity`

```matlab
function [A,f]=C_matrix2D(Dati,femregion)
%% [A,f] = C_matrix2D(Dati,femregion)
%==========================================================================
% Assembly of the stiffness matrix A and rhs f
%==========================================================================
%    called in C_main2D.m
%
%    INPUT:
%          Dati        : (struct)  see C_dati.m
%          femregion   : (struct)  see C_create_femregion.m
%
%    OUTPUT:
%          A           : (sparse(ndof,ndof) real) stiffnes matrix
%          f           : (sparse(ndof,1) real) rhs vector


addpath FESpace
addpath Assembly

```

```matlab
fprintf('============================================================\n')
fprintf('Assembling matrices and right hand side ... \n');
fprintf('============================================================\n')


% connectivity infos
ndof         = femregion.ndof; % degrees of freedom
nln          = femregion.nln;  % local degrees of freedom
ne           = femregion.ne;   % number of elements
connectivity = femregion.connectivity; % connectivity matrix


% shape functions
[basis] = C_shape_basis(Dati);

% quadrature nodes and weights for integrals
[nodes_2D, w_2D] = C_quadrature(Dati);

% evaluation of shape bases
[dphiq,Grad] = C_evalshape(basis,nodes_2D);


% Assembly begin ...
A = sparse(ndof,ndof);  % Global Stiffness matrix
f = sparse(ndof,1);     % Global Load vector

for ie = 1 : ne

    % Local to global map --> To be used in the assembly phase
    iglo = connectivity(1:nln,ie);


    [BJ, pphys_2D] = C_get_Jacobian(femregion.coord(iglo,:), nodes_2D, Dati.MeshType);
    % BJ        = Jacobian of the elemental map
    % pphys_2D = vertex coordinates in the physical domain

    %=============================================================%
    % STIFFNESS MATRIX
    %=============================================================%

    % Local stiffness matrix
    [A_loc] = C_lap_loc(Grad,w_2D,nln,BJ);

    % Assembly phase for stiffness matrix
    A(iglo,iglo) = A(iglo,iglo) + A_loc;

    %==============================================
    % FORCING TERM --RHS
    %==============================================

    % Local load vector
    [load] = C_loc_rhs2D(Dati.force,dphiq,BJ,w_2D,pphys_2D,nln);

    % Assembly phase for the load vector
    f(iglo) = f(iglo) + load;

end
```

This is achieved between the lines 48 and 78 of the presented code. In particular the local contributions are computed through the following functions

```
function [K_loc]=C_lap_loc(Grad,w_2D,nln,BJ)
%% [K_loc]=C_lap_loc(Grad,w_2D,nln,BJ)
%==========================================================================
% Build the local stiffness matrix for the term grad(u)grad(v)
%==========================================================================
%    called in C_matrix2D.m
%
%    INPUT:
%          Grad        : (array real) evaluation of the gradient on
%                        quadrature nodes
%          w_2D        : (array real) quadrature weights
%          nln         : (integer) number of local unknowns
%          BJ          : (array real) Jacobian of the map
%
%    OUTPUT:
%          K_loc       :  (array real) Local stiffness matrix


K_loc=zeros(nln,nln);

for i=1:nln
    for j=1:nln
        for k=1:length(w_2D)
            Binv = inv(BJ(:,:,k));   % inverse
            Jdet = det(BJ(:,:,k));   % determinant
            K_loc(i,j) = K_loc(i,j) + (Jdet.*w_2D(k)) .* ( (Grad(k,:,i) * Binv) * (Grad(k,:,j) * Binv )');
        end
    end
end
```

```
function [f]=C_loc_rhs2D(force,dphiq,BJ,w_2D,pphys_2D,nln)
%% [f]=C_loc_rhs2D(force,dphiq,BJ,w_2D,pphys_2D,nln)
%==========================================================================
% Build the right hand side vector (fv)
%==========================================================================
%    called in C_matrix2D.m
%
%    INPUT:
%          force       : (string) expression of the forcing term
%          dphiq       : (array real) basis functions evaluated at q.p.
%          BJ          : (array real) Jacobian of the map
%          w_2D        : (array real) quadrature weights
%          pphys_2D    : (array real) quadrature nodes in the physical
%                                     space
%          nln         : (integer) number of local unknowns
%
%    OUTPUT:
%          f           : (array real) Local right hand side


f = zeros(nln,1);
% evaluation of the right hand side on the physical nodes
x = pphys_2D(:,1);
y = pphys_2D(:,2);
F = eval(force);

```

```matlab
% Evaluation of the local r.h.s.
for s = 1:nln
    for k = 1:length(w_2D)
        Jdet = det(BJ(:,:,k));  % determinant
        f(s) = f(s) + w_2D(k)*Jdet*F(k)*dphiq(1,k,s);
    end
end
```

3. Compute the $H^1(\Omega)$ and $L^2(\Omega)$ errors as function of the mesh grid h, knowing that the exact solution to problem (1) is $u_{ex} = \sin(2\pi x)\sin(2\pi y)$. Gather these results in a table and provide a comment on them, moving from the theoretical knowledge.

The computation of the $H^1(\Omega)$ and $L^2(\Omega)$ errors is implemented in the provided function `C_compute_errors`*. The error plots can be obtained through the function* `C_convergence_test` *and the results are reported in Figure 1. As one can see the theoretical convergence rates are verified, in particular a quadratic (resp. linear) order of convergence is obtained with respect to the $L^2(\Omega)$-norm (resp. $H^1(\Omega)$-norm). See also Table 1.*

```matlab
function [errors_table,rates]=C_convergence_test(test_name)
%% [errors_table,rates]=C_mesh_test(test_name)
%==========================================================================
% Error analysis varying the mesh size h
%==========================================================================
% Example of usage: [errors_table,rates] = C_mesh_test('Test1')
%
%    INPUT:
%           test_name    : (string)  test case name, see C_dati.m
%
%    OUTPUT:
%           errors_table : (struct) containing the computed errors
%           rates        : (struct) containing the computed rates


warning off;
addpath Assembly
addpath BoundaryConditions
addpath Errors
addpath MeshGeneration
addpath FESpace
addpath Postprocessing


Dati=C_dati(test_name);

refinement_vector=Dati.refinement_vector;
num_test=length(refinement_vector);

for k=1:num_test
    [errors,solutions,femregion,Dati]=C_main2D(test_name,refinement_vector(k));
    Error_L2(k)=errors.Error_L2;
    Error_SEMI_H1(k)=errors.Error_SEMI_H1;
    Error_H1(k)=errors.Error_H1;
    Error_inf(k)=errors.Error_inf;
    ne(k)=femregion.ne;
    h(k)=femregion.h;
    fprintf('==========================================\n');
    fprintf('End test %i\n',k);
    fprintf('==========================================\n');
```

```
end
p=femregion.degree;

%ERROR TABLE
errors_table=struct('ne',ne,...
                    'h',h,...
                    'Error_L2', Error_L2,...
                    'Error_SEMI_H1', Error_SEMI_H1,...
                    'Error_H1', Error_H1,...
                    'Error_inf',Error_inf);



%TABLE
rate_L2=log10(Error_L2(2:num_test)./Error_L2(1:num_test-1))./log10(h(2:num_test)./h(1:num_test-1));
rate_SEMI_H1=log10( Error_SEMI_H1(2:num_test)./ Error_SEMI_H1(1:num_test-1))./log10(h(2:num_test)./h(1:num_test-1));
rate_H1=log10( Error_H1(2:num_test)./ Error_H1(1:num_test-1))./log10(h(2:num_test)./h(1:num_test-1));
rate_inf=log10( Error_inf(2:num_test)./ Error_inf(1:num_test-1))./log10(h(2:num_test)./h(1:num_test-1));

rates=struct('rate_L2',rate_L2,...
             'rate_SEMI_H1',rate_SEMI_H1,...
             'rate_H1',rate_H1,...
             'rate_inf',rate_inf);


% ERROR PLOTS
hs = subplot(2,1,1);
loglog(h,h.^(p+1),'-+b','Linewidth',2);
hold on
loglog(h,Error_L2,'-or','Linewidth',2);
hold on
legend(sprintf('h^%i',p+1),'||.||_{L^2}');
title ('Errors');
ylabel('L^2-error')
xlabel('h');
hs.FontSize = 12;

hs = subplot(2,1,2);
loglog(h,h.^(p),'-+b','Linewidth',2);
hold on
loglog(h,Error_H1,'-or','Linewidth',2);
hold on
legend(sprintf('h^%i',p),'||.||_{H^1}');
ylabel('H^1-error')
xlabel('h');
hold off
hs.FontSize = 12;
```

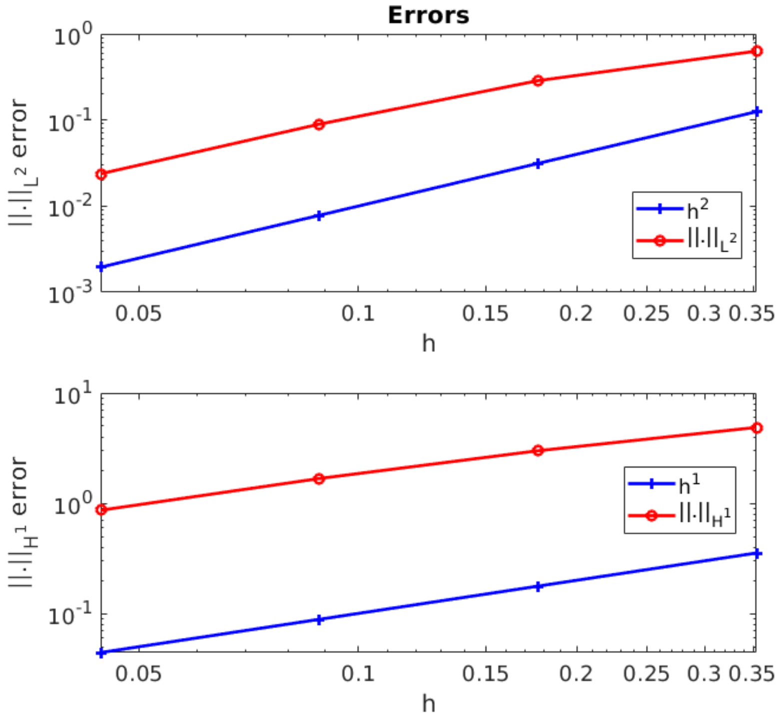

Figure 1: Computed $H^1(\Omega)$ and $L^2(\Omega)$ errors as function of the mesh grid h

h	0.1768	0.0884	0.0442	0.0221
$\|u - u_h\|_{L^2(\Omega)}$	0.2862	0.0895	0.0238	0.0060
$\|u - u_h\|_{H^1(\Omega)}$	2.9914	1.6754	0.8634	0.4351

Table 1: Computed $H^1(\Omega)$ and $L^2(\Omega)$ errors as function of the mesh grid h.

Exercise 2

In the unit square $\Omega \equiv (0,1) \times (0,1)$ we consider the following problem:

$$\begin{cases} -\Delta u + 2u = 0, & \text{in } \Omega, \\ u = g \equiv e^{x+y}, & \text{on } \partial\Omega. \end{cases}$$

1. Give the weak formulation and prove the well posedness of the problem. Find the analytical solution.

   The problem considered features non homogeneous Dirichlet conditions. The weak formulation is obtained multiplying the first equation of the system by a generic function $v \in V \equiv H_0^1$ and integrating by parts the Laplacian term. We get

$$\int_\Omega \nabla u \cdot \nabla v \, d\omega + 2 \int_\Omega uv \, d\omega = 0. \tag{3}$$

   Le us introduce the lifting operator R_g such that $R_g \in H^1(\Omega)$ and $R_g\big|_{\partial\Omega} = g$. We define a new unknown function $\overset{\circ}{u} = u - R_g$ that, by definition, will belong to $H_0^1(\Omega)$. By replacing u in (3) with $\overset{\circ}{u} + R_g$, we get the following problem. Find $\overset{\circ}{u} \in V$ s.t.

$$a(\overset{\circ}{u}, v) = \int_\Omega \nabla \overset{\circ}{u} \cdot \nabla v \, d\omega + 2 \int_\Omega \overset{\circ}{u} \, v \, d\omega = - \int_\Omega \nabla R_g \cdot \nabla v \, d\omega - 2 \int_\Omega R_g v \, d\omega = F(v) \quad \forall v \in V. \tag{4}$$

   The bilinear form is symmetric, continuous, coercive in V. We have indeed

$$|a(u,v)| \leq 3 \|u\|_V \|v\|_V \quad \forall u, v \in V.$$

   Moreover for any $u \in V$ we have

$$a(u,u) \geq \|\nabla u\|^2_{L^2(\Omega)} + 2\|u\|^2_{L^2(\Omega)} \geq \|u\|^2_V.$$

   Finally we can prove that

$$|F(v)| \leq 3 \|R_g\|_V \|v\|_V \quad \forall v \in V.$$

   From the Lax-Milgram lemma, solution to (4) exists and is unique.
   We assume that the solution has the form $u = X(x)Y(y)$. Notice that in particular $g = B_x(x)B_y(y)$, where the boundary functions are $B_x = e^x$ and $B_y = e^y$. With these positions, we write

$$-X''Y - XY'' + 2XY = 0.$$

   Assuming $X \neq 0$ and $Y \neq 0$, we reformulate the equation in the form

$$\frac{X''}{X} = 2 - \frac{Y''}{Y}.$$

   The two sides of the equation above are functions of two different variables, so they are constants. For a real constant K, we have

$$\frac{X''}{X} = K.$$

Assume that $K > 0$, so to obtain

$$X = C_{1,K} e^{\sqrt{K}x} + C_{2,K} e^{-\sqrt{K}x}.$$

By prescribing the boundary function B_x we conclude that

$$C_{1,K} = \begin{cases} 1 & \text{for } K = 1, \\ 0 & \text{for } K \neq 1, \end{cases} \qquad C_{2,K} = 0, \quad \forall K > 0.$$

Proceeding similarly for Y (for $K = 1$) we obtain that $Y = e^y$ and conclude that $u_{ex} = e^{x+y}$. Our theoretical analysis concludes that this is the only solution to the problem.

2. Give the linear finite element approximation using Matlab on uniform grids.

   Let V_h be the subspace of V of linear finite elements. The Galerkin finite element formulation of (4) reads: find $u_h \in V_h$ s.t.

$$a(u_h, v_h) = F(v_h) \quad \forall v_h \in V_h. \tag{5}$$

   By introducing a basis $\{\varphi_j\}_{j=1}^{N_h}$ for the space V_h we can write the generic element of the stiffness matrix A assciated to (5) in the form

$$A_{ij} = \int_\Omega \nabla\varphi_j \cdot \nabla\varphi_i \, d\omega + 2\int_\Omega \varphi_j \varphi_i \, d\omega, \quad i, j = 1, ..., N_h.$$

   To assembly the linear system and the load vector we can proceed as for the previous exercise. Here, in particular, we need to compute the local stiffnes matrix as the sum of two contribution `A_loc` *and* `M_loc`*. The former can be computed by using the* `C_lap_loc.m` *function whereas the latter through the* `C_mass_loc.m` *function defined below.*

```
function [M_loc]=C_mass_loc(dphiq,w_2D,nln,BJ)
%% [M_loc]=C_mass_loc(dphiq,w_2D,nln,BJ)
%==========================================================================
% Build the local mass matrix for the term (uv)
%==========================================================================
%     called in C_matrix2D.m
%
%     INPUT:
%            dphiq        : (array real) evaluation of the basis function on
%                           quadrature nodes
%            w_2D         : (array real) quadrature weights
%            nln          : (integer) number of local unknowns
%            BJ           : (array real) Jacobian of the map
%
%     OUTPUT:
%            M_loc        :  (array real) Local mass matrix

M_loc=zeros(nln,nln);

for i=1:nln
    for j=1:nln
        for k=1:length(w_2D)
            Jdet = det(BJ(:,:,k));       % determinant
            M_loc(i,j) = M_loc(i,j) + (Jdet.*w_2D(k)) .* dphiq(1,k,i).* dphiq
    (1,k,j);
        end
    end
end
```

The results are then summed up together to obtain the global stiffness matrix A.

```
    % Local stiffness matrix
    [A_loc] = C_lap_loc(Grad,w_2D,nln,BJ);
    [M_loc] = C_mass_loc(dphiq,w_2D,nln,BJ);

    % Assembly phase for stiffness matrix
    A(iglo,iglo) = A(iglo,iglo) + Dati.mu*A_loc + Dati.sigma*M_loc;
```

Notice that we have added to the `Dati` *structure the additional fields* `Dati.mu` and `Dati.sigma` *and defined the new test case* `Test2`.

```
function [Dati]=C_dati(test)

if test=='Test2'
Dati = struct( 'name',              test,...
               ... % Test name
               'domain',            [0,1;0,1],...
               ... % Domain bounds
               'mu',                1, ...
               ... % Diffusive term ...
               'sigma',             2, ...
               ... % Reactive term ...
               'exact_sol',         'exp(x+y)',...
               ... % Definition of exact solution
               'force',             '0.*x.*y',...
               ... % Forcing term
               'grad_exact_1',      'exp(x+y)',...
               ... % Definition of exact gradient (x comp)
               'grad_exact_2',      'exp(x+y)',...
               ... % Definition of exact gradient (y comp)
               'fem',               'P1',...
               ... % P1-fe
               'nqn_1D',             4,...
               ... % Number of quad. points per edge
               'nqn_2D',             3,...
               ... % Number of quad. points per triangle
               'MeshType',          'TS', ...
               ... % Triangular Structured mesh
               'refinement_vector', [2,3,4,5],...
               ... % Refinement levels for the error analysis
               'visual_graph',       'N',...
               ... % Visualization of the solution
               'plot_errors',        'Y' ...
               ...% Compute Errors
               );
end
```

3. Compute the $H^1(\Omega)$ and $L^2(\Omega)$ errors as function of the mesh grid h. Gather these results in a table and provide a comment on them, moving from the theoretical knowledge.

   As for the previous exercise, the error plots can be obtained through the function `C_convergence_test` *and the results are reported in Figure 2. As one can see the theoretical convergence rates are verified, in particular a quadratic (resp. linear) order of convergence is obtained with respect to the $L^2(\Omega)$-norm (resp. $H^1(\Omega)$-norm). See also Table 2.*

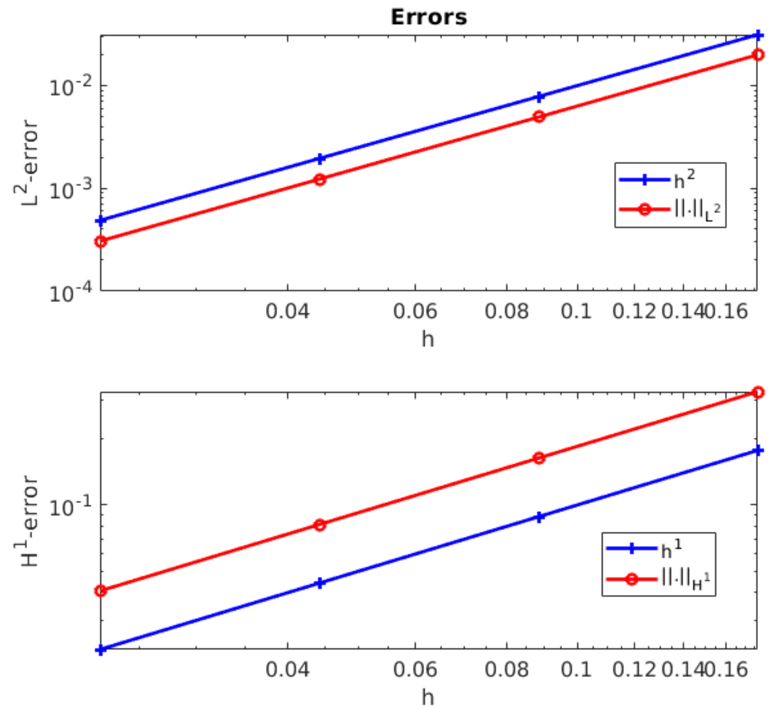

Figure 2: Computed $H^1(\Omega)$ and $L^2(\Omega)$ errors as function of the mesh grid h

h	0.1768	0.0884	0.0442	0.0221
$\|u-u_h\|_{L^2(\Omega)}$	0.0199	0.0049	0.0012	3.0716e-04
$\|u-u_h\|_{H^1(\Omega)}$	0.3263	0.1630	0.0815	0.0408

Table 2: Computed $H^1(\Omega)$ and $L^2(\Omega)$ errors as function of the mesh grid h.

Exercise 3

A simple model used in oceanography is due to Stommel[1]. In this model ocean is assumed to be flat and with uniform depth H, and no vertical movement of the water free boundary it is considered (since it is small compared with the horizontal one). Only Coriolis force, wind action on the surface and friction at the bottom are included. Incompressibility implies that there exists a function ψ, called *stream function*, related to the velocity components by the equations

$$u = -\frac{\partial \psi}{\partial y},\ v = \frac{\partial \psi}{\partial x}.$$

In Stommel model for a rectangular ocean $\Omega = (0, L_x) \times (0, L_y)$, ψ is solution to the following elliptic problem

$$\begin{cases} -\Delta\psi - \alpha\dfrac{\partial \psi}{\partial x} = \gamma \sin(\pi y/L_y), & \text{in } \Omega, \\ \psi = 0, \quad \text{on } \partial\Omega, \end{cases} \tag{6}$$

with $\alpha = \dfrac{H\beta}{R}$, $\gamma = \dfrac{W\pi}{RL_y}$, being R the friction coefficient on the bottom, W a coefficient associated with the surface wind, $\beta = df/dy$ where f is the Coriolis parameter, which is in general function only of y (latitude).

1. Give the weak formulation of (6). Prove the well-posedness of the problem.

   The weak formulation of the problem is obtained with the usual procedure and reads: find $\psi \in V \equiv H_0^1(\Omega)$ *such that*

   $$a(\psi, v) = \int_\Omega \nabla\psi \cdot \nabla v \, d\omega - \alpha \int_\Omega \frac{\partial \psi}{\partial x} v \, d\omega = \gamma \int_\Omega \sin(\pi y/L_y) v \, d\omega \quad \forall v \in V. \tag{7}$$

   Well posedness analysis relies on the Lax-Milgram Lemma. With arguments similar to the ones used in the previous exercise, we can verify that the bilinear form $a(\cdot,\cdot)$ *defined in* (7) *is continuous and coercive. For the coercivity, we just notice that we can write the second term on the left hand side of* (7) *as*

   $$\alpha \int_\Omega \frac{\partial \psi}{\partial x} v \, d\omega = \int_\Omega \nabla \cdot (\mathbf{b}\psi) v,$$

   with $\mathbf{b} = (\alpha, 0)^\top$. *Notice that*

   $$\alpha \int_\Omega \frac{\partial \psi}{\partial x} \psi \, d\omega = \int_\Omega \nabla \cdot (\mathbf{b}\psi)\psi = 0.$$

   From $\nabla \cdot \mathbf{b} = 0$, *we have in fact*

   $$\psi \nabla \cdot (\mathbf{b}\psi) = \psi \mathbf{b} \cdot \nabla \psi = \frac{1}{2} \mathbf{b} \cdot \nabla \psi^2 = \frac{1}{2} \nabla \cdot (\mathbf{b}\psi^2).$$

   Therefore, since $\psi = 0$ *on* $\partial\Omega$,

   $$\int_\Omega \nabla \cdot (\mathbf{b}\psi)\psi \, d\omega = \frac{1}{2} \int_\Omega \nabla \cdot (\mathbf{b}\psi^2) \, d\omega = \frac{1}{2} \int_{\partial\Omega} \psi^2 \mathbf{b} \cdot \boldsymbol{n} \, d\gamma = 0.$$

   Since the functional $F(\cdot)$ *defined on the right hand side of* (7) *is linear and continuous, we conclude that the solution exists and is unique.*

[1] Stommel H. (1948) The westward intensification of wind-driven ocean currents. *Trans. Amer. Geophys. Union* 29(202).

2. Compute the analytical solution of the model (6) for a constant β.

   We can compute this solution following the separation of variables approach. In particular, we assume

$$\psi(x,y) = X(x)Y(y),$$

   with with $X(0) = X(L_x) = 0$ and $Y(0) = Y(L_y) = 0$. Moreover, the forcing term depends only on y in the form $\gamma \sin(\pi y/L_y)$. Observe that the left hand side of the equation in (6) reads now

$$-X''Y - XY'' - \alpha X'Y$$

   and that the second derivative of a sinus function is still a sinus function with the same frequency. This suggests to do the following educated guess,

$$Y(y) = \sin(\pi y/L_y),$$

   which fulfills the boundary conditions. We have then

$$\left(-X'' + \frac{\pi^2}{L_y^2}X - \alpha X'\right)\sin(\pi y/L_y) = \gamma \sin(\pi y/L_y).$$

   The problem is now reduced to the solution of the constant coefficient ordinary differential equation

$$X'' + \alpha X' - \frac{\pi^2}{L_y^2}X = -\gamma.$$

   A particular solution is clearly given by

$$X_P = \frac{L_y^2\gamma}{\pi^2}.$$

   The general solution to the homogeneous equation reads

$$X_H(x) = C_1 e^{\lambda_1 x} + C_2 e^{\lambda_2} x$$

   where $\lambda_{1,2}$ are the roots of the algebraic equation

$$\lambda^2 + \alpha\lambda - \frac{\pi^2}{L_y^2} = 0,$$

   i.e., $\lambda_{1,2} = -\frac{\alpha}{2} \pm \sqrt{\left(\frac{\alpha}{2}\right)^2 + \frac{\pi^2}{L_y^2}}$. For $X = X_P + X_H$, when we prescribe the boundary conditions $X(0) = X(L_x) = 0$ we find C_1 and C_2, and the final solution reads

$$\psi = -\gamma\frac{L_y^2}{\pi^2}(pe^{\lambda_1 x} + (1-p)e^{\lambda_2 x} - 1)\sin(\pi y/L_y)$$

   with $p = (1 - e^{\lambda_2 L_x})/(e^{\lambda_1 L_x} - e^{\lambda_2 L_x})$.

3. Solve the problem numerically with linear finite elements, using the Stommel's parameters: $L_x = 10^5\ m$, $L_y = 2\pi 10^4\ m$, $H = 200\ m$, $W = 0.3\ 10^{-7}m^2s^{-2}$, $R = 0.6\ 10^{-3}ms^{-1}$. Assume at first $\beta = 0$ and then $\beta = 5\ 10^{-10}m^{-1}s^1$. Comment the results.

   Galerkin approximation of (7) has the usual form: find $\psi_h \in V_h$ s.t. $a(\psi_h, v_h) = F(v_h)$ for any $v_h \in V_h$, being V_h the subspace of piece-wise linear functions.

By introducing a basis $\{\varphi_j\}_{j=1}^{N_h}$ for the space V_h we can write the generic element of the stiffness matrix A assciated to the bilinear form $a(\cdot,\cdot)$ in the form

$$A_{ij} = \int_\Omega \nabla\varphi_j \cdot \nabla\varphi_i \, d\omega - \alpha \int_\Omega \frac{\partial \varphi_j}{\partial x} \varphi_i \, d\omega, \quad i,j = 1,...,N_h,$$

while for the load vector we have

$$b_i = \gamma \int_\Omega sin(\pi y/(2\pi))\varphi_i \, d\omega \quad i = 1,...,N_h.$$

To assembly the linear system and the load vector we can proceed as for the previous exercise. Here, in particular, we need to compute the local stiffnes matrix as the sum of two contribution `A_loc` *and* `Adv_loc`*. The former is computed with the* `C_lap_loc.m` *function whereas the latter through the* `C_adv_loc.m` *function defined below.*

```
function [ADV_loc]=C_adv_loc(Grad,dphiq,beta,w_2D,nln,BJ)

ADV_loc=sparse(nln,nln);

for i=1:nln
    for j=1:nln
        for k=1:length(w_2D)
            Binv=inv(BJ(:,:,k));                          % inverse
            Jdet=det(BJ(:,:,k));                          % determinant
            ADV_loc(i,j)=ADV_loc(i,j) + (Jdet.*w_2D(k)) .*  dphiq(1,k,i) *( (
    beta)*(Grad(k,:,j) * Binv )');
        end
    end
end
```

The results are then summed up together to obtain the global stiffness matrix A.

```
    % Local stiffness matrix
    [A_loc] = C_lap_loc(Grad,w_2D,nln,BJ);
    [M_loc] = C_mass_loc(dphiq,w_2D,nln,BJ);
    [Adv_loc]=C_adv_loc(Grad,dphiq, Dati.beta,w_2D,nln,BJ);

    % Assembly phase for stiffness matrix
    A(iglo,iglo) = A(iglo,iglo) + Dati.mu*A_loc + Dati.sigma*M_loc + Adv_loc;
```

Notice that in the `Dati` *structure we added the additional vector field* `Dati.beta` and we defined the new test case `Test3`.

```
function [Dati]=C_dati(test)

if test=='Test3'
Dati = struct( 'name',              test,...
               ... % Test name
               'domain',            [0 1.e5;0 2*pi*1.e4],...
               ... % Domain bounds
               'mu',                1, ...
               ... % Diffusive term ...
               'sigma',             0, ...
               ... % Reactive term ...
               'beta',              [-200*5.e-10/(0.6*1.e-3),0], ...
               ... % Advective term
               'exact_sol',         '0.*x.*y',...
```

```
                ... % Definition of exact solution
                'force',            'pi*0.3*1.e-5/(0.6*1.e-3*2*pi*1.e4)*sin(pi
    *y/(2*pi*1.e4))',...
                ... % Forcing term
                'grad_exact_1',     '0.*x.*y',...
                ... % Definition of exact gradient (x comp)
                'grad_exact_2',     '0.*x.*y',...
                ... % Definition of exact gradient (y comp)
                'fem',              'P1',...
                ... % P1-fe
                'nqn_1D',            4,...
                ... % Number of quad. points per edge
                'nqn_2D',            3,...
                ... % Number of quad. points per triangle
                'MeshType',         'TS', ...
                ... % Triangular Structured mesh
                'refinement_vector', [2,3,4,5],...
                ... % Refinement levels for  the error analysis
                'visual_graph',      'Y',...
                ... % Visualization of the solution
                'plot_errors',       'N' ...
                ...% Compute Errors
                );

end
```

Input data listed above are computed through the script `analytical_solution_Test3.m`*.*

```
Lx = 1.e5;
Ly = 2*pi*1.e4;
H = 200;
W = 0.3*1.e-7;
R = 0.6*1.e-3;
% beta = 0;
beta = 5*1e-10;

alpha = H*beta/R;
gamma = W*pi/(R*Ly);

lambda1 = -alpha*0.5 + sqrt((alpha*0.5)^2+(pi/Ly)^2);
lambda2 = -alpha*0.5 - sqrt((alpha*0.5)^2+(pi/Ly)^2);

c2 = (Ly/pi)^2*gamma*((exp(lambda1*Lx)-1)/(exp(lambda2*Lx)-exp(lambda1*Lx)));
c1 = - c2 - (Ly/pi)^2*gamma;

psi = @(x,y,c1,c2,lambda1,lambda2,Ly,gamma) (c1*exp(lambda1*x) + c2*exp(
    lambda2*x) + (Ly/pi)^2*gamma).*sin(pi/Ly*y);

x = linspace(0,Lx,100);
y = linspace(0,Ly,1000);

[X,Y] = meshgrid(x,y);

surf(X,Y,psi(X,Y,c1,c2,lambda1,lambda2,Ly,gamma),'EdgeColor','none');
```

In Figure 3 are reported the computed solutions ψ_h obtained with $\beta = 0$ and $\beta = 5.10^{-10}$. It is clear the effect of the advective term on the computed solution, cf. Figure 3 (right).

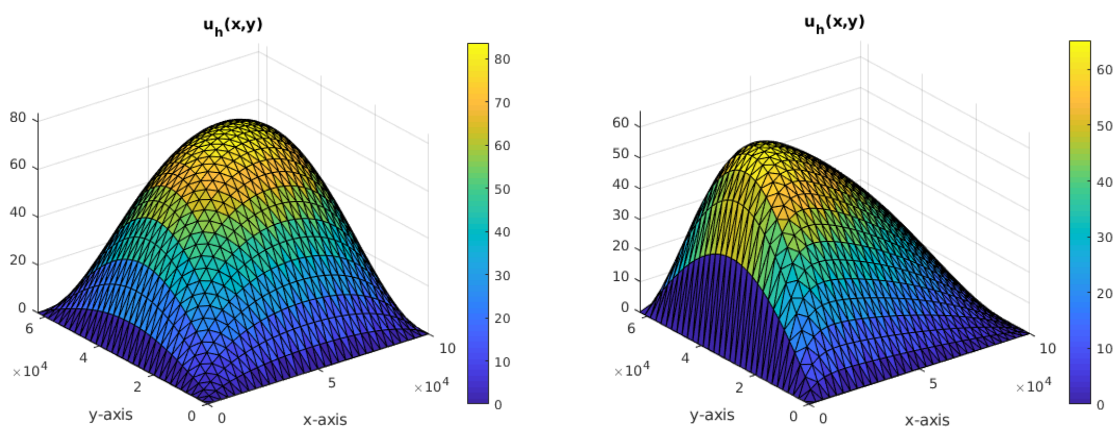

Figure 3: Computed solution ψ_h for problem (6) with $\beta = 0$ (left) and $\beta = 5.10^{-10}$ (right)