1D Steady convection-diffusion equation:

Juan Pedro Roldán Blasco

Objectives of the lab class:

1. Write SUPG_system
2. Write GLS_system
3. Adapt the solution to use quadratic elements

1. SUPG_system*:

The main change in the code was the stabilization term:

```
% Loop on Gauss points
for ig = 1:ngaus
    N_ig = N(ig,:);
    Nx_ig = Nxi(ig,:)*2/h;
    N2x_ig = N2xi(ig,:)*(4/(h^2));
    w_ig = wgp(ig)*h/2;
    x = N_ig*Xe; % x-coordinate of the gauss point
    s = SourceTerm(x,example);
    fe = fe + w_ig*(N_ig)'*s;
    Ke = Ke + w_ig*(N_ig'*a*Nx_ig + Nx_ig'*nu*Nx_ig)  ... %galerkin
        + w_ig*(tau*a*Nx_ig)'*((a*Nx_ig)-nu*N2x_ig-s); %SUPG
end
```

Figure 1: SUPG added term

See for the expression of Ke:
Ke = Ke + w_ig(N_ig'*a'Nx_ig + Nx_ig'*nu*Nx_ig) :* This is the expression of the standard Galerkin solution

w_ig(tau*a*Nx_ig)'((a*Nx_ig) - nu*N2x_ig - s)* : Corresponds to the stabilization term, where the red part is the stabilization term of the SU method, and the blue is the extra term that introduces SUPG method (which completes the expression of the residual of the strong form).

2. GLS system*:

Similarly to the SUPG implementation, we changed the stabilization term for Galerkin Least Squares method:

The expression of Ke is now:
Ke = Ke + w_ig(N_ig'*a'Nx_ig + Nx_ig'*nu*Nx_ig) :* Again, the expression of the standard Galerkin solution

w_ig* (tau*a*Nx_ig-nu*N2x_ig)'*((a*Nx_ig)-nu*N2x_ig-s): The red part, in this case, is the stabilization term that multiplies the residual (in blue). This creates the least squares expression that gives name to the method.

```
% Loop on Gauss points
for ig = 1:ngaus
    N_ig = N(ig,:);
    Nx_ig = Nxi(ig,:)*2/h;
    N2x_ig = N2xi(ig,:)*(4/(h^2));
    w_ig = wgp(ig)*h/2;
    x = N_ig*Xe; % x-coordinate of the gauss point
    s = SourceTerm(x,example);
    fe = fe + w_ig*(N_ig)'*s;
    Ke = Ke + w_ig*(N_ig'*a*Nx_ig + Nx_ig'*nu*Nx_ig)  ... %galerkin
        + w_ig*(tau*a*Nx_ig-nu*N2x_ig)'*((a*Nx_ig)-nu*N2x_ig-s); %GLS
end
```

Figure 2: GLS added term

*Remarks:
Note that in both implementations there is one extra variable that does not appear in the given code. This is N2x_ig. Although we have the expression of the second derivative of the shape functions in isoparametric formulation, there is no expression for general coordinates. In order to do so, we have to take into account that the second derivative matrix must be multiplied by the square of the inverse of the Jacobian, this is, 4/h (being h the element size). This value is obtained after derivation of d^2/dx^2 with respect to isoparametric variable xi.

```
f = zeros(nPt,1);

% Loop on elements
for ielem = 1:nElem
    Te = T(ielem,:);
    Xe = X(Te);
    h = Xe(end) - Xe(1);
    disp(h)

    Ke = zeros(nen);
    fe = zeros(nen,1);
    % Loop on Gauss points
    for ig = 1:ngaus
        N_ig = N(ig,:);
        Nx_ig = Nxi(ig,:)*2/(h);
        N2x_ig = N2xi(ig,:)*(4/((h)^2));
        w_ig = wgp(ig)*h/2;
        x = N_ig*Xe; % x-coordinate of the gauss point
        s = SourceTerm(x,example);
```

Figure 3: Extra term (in blue) added to SUPG and GLS to include the second derivates of shape functions

3. Quadratic elements

To adapt quadratic elements, we added a subroutine that changes the formulation of the elements if we intend to use quadratic elements:

Elements have 3 nodes, with the central node belonging to only 1 element. That is why the nodes in T are chosen with a step of 2 (for the first element, the first node is #1, but for the second is #1+2 = #3).

As the rest of characteristics are defined in SetReferenceElement and the mesh size h (defined as space between nodes in mai) has the correct size in every system solution, we do not have to make more changes.

```
if p ==2
    nPt = 2*nElem+1;
    h = (dom(2) - dom(1))/(2*nElem);
    X = (dom(1):h:dom(2))';
    T = [1:2:nPt-2; 2:2:nPt-1; 3:2:nPt]';
else
    nPt = nElem + 1;
    h = (dom(2) - dom(1))/nElem;
    X = (dom(1):h:dom(2))';
    T = [1:nPt-1; 2:nPt]';
end
```

Figure 4: Sub-routine added in main.m. The original part was between *else* and *end*