Finite Elements in Fluids, Assignment 2

Jose Raul Bravo Martinez, MSc Computational Mechanics

February 18, 2019

Derivation of equations:

Starting from a steady transport equation:

$$\boldsymbol{a} \cdot \nabla u - \nabla \cdot (\nu \nabla u) + \sigma u = s \quad , in \Omega \tag{1}$$

with Dirichlet boundary conditions $u = u_d$ on $\partial\Omega$.

It is necessary to multiply by a "test" or "weight" function and integrate over the domain, to obtain:

$$\int_\Omega w(\boldsymbol{a} \cdot \nabla u) d\Omega - \underbrace{\int_\Omega w \nabla \cdot (\nu \nabla u) d\Omega}_{\text{Apply Divergence Theorem}} + \int_\Omega w \sigma u d\Omega = \int_\Omega s d\Omega \tag{2}$$

Recall the divergence theorem:

$$\int_\Omega f \nabla \cdot G dV = -\int_\Omega \nabla f \cdot G dV + \int_{\partial\Omega} f G \cdot n dS \tag{3}$$

Applying (3) to the second term in (2) to get rid of the second order derivatives, one obtains the weak form of the governing equations:

$$\int_\Omega w(\boldsymbol{a} \cdot \nabla u) d\Omega + \int_\Omega \nabla w \cdot (\nu \nabla u) d\Omega + \int_\Omega w \sigma u d\Omega = \int_\Omega s d\Omega \tag{4}$$

for all w, such that $w = 0$ on $\partial\Omega$.

For simplify the notation, a compact version of the weak from is introduced as:

$$a(w,u) = \int_\Omega \nabla w \cdot (\nu \nabla u) d\Omega, \qquad (w,s) = \int_\Omega w s d\Omega$$

$$c(w,u,a) = \int_\Omega w(a \cdot \nabla u) d\Omega, \qquad (w,h)_{\Gamma_N} = \int_{\Gamma_N} w h d\Gamma$$

This allows to write the weak form 4, in the following compact expression:

$$a(w, u + c(w,u,\boldsymbol{a}) = (w,s) + (w,h)_{\Gamma_N} \tag{5}$$

Implementation of 1D linear elements. Galerkin Approximation.

In order to discretize the weak form by means of the Galerkin method, lets consider the approximation of the exact solution u as:

$$u(\boldsymbol{x}) = u^h(\boldsymbol{x}) = \sum_{A \in \eta \not\eta_D} N_A(\boldsymbol{x}) u_A + \sum_{A \in \eta_D} N_A(\boldsymbol{x}) u_D(\boldsymbol{x}_A) \tag{6}$$

The test functions can be discretized in a similar way, to obtain the set of equations:

$$(\boldsymbol{C} + \boldsymbol{K})\boldsymbol{u} = \boldsymbol{f} \tag{7}$$

Where $\boldsymbol{u}$ is the vector of unknown nodal values, $\boldsymbol{C}$ is the Convection matrix, $\boldsymbol{K}$ is the difussion matrix. Both this matrices are defined as:

$$\boldsymbol{C} = \Pi^e \boldsymbol{C}^e \quad C^e_{ab} = \int_\Omega N_a(\boldsymbol{a} \cdot \nabla N_b) d\Omega \quad \text{convection matrix}$$

$$\boldsymbol{K} = \Pi^e \boldsymbol{K}^e \quad K^e_{ab} = \int_\Omega \nabla N_a \cdot \nu \nabla N_b) d\Omega \quad \text{diffusion matrix}$$

Where Π^e is the assembly operator.
For the case of a 1D linear element, the governing equation is simplified to:

$$\begin{aligned} &au_x - \nu u_{xx} = s(x) \qquad \text{in }]0,\text{L}[\\ &u = u_D \qquad \text{at x=0 and x=L} \end{aligned} \tag{8}$$

The weak form after integration by parts is:

$$\int_0^L (wau_x + w_x \nu u_x) dx = \int_0^L wsdx, \tag{9}$$

or in compact form:

$$a(w,u) + c(w,u,a) = (w,s) \tag{10}$$

Thus, using the definitions given before, the discrete equation for an interior node A, A=2,..., $n_{eq}+1$, where n_{eq} is the number of interior nodes of the spatial discretization:

$$\int_0^L \sum_{B=2}^{n_{eq}+1} (aN_A \frac{B}{\partial x} + \nu \frac{\partial N_A}{\partial x}) \frac{B}{\partial x}) = \int_0^L N_A s dx \tag{11}$$

The shape functions of a linear element are given by:

$$N_1(\xi) = \frac{1}{2}(1-\xi) \qquad N_s \xi \frac{1}{2}(1+\xi), \tag{12}$$

And for a uniform mesh of size h:

$$dx = \frac{1}{2}(x_2 - x_1) d\xi = \frac{h}{2} d\xi, \tag{13}$$

and,

$$\frac{\partial N_b}{\partial x} = \frac{\partial N_b}{\partial \xi} \frac{\partial \xi}{\partial x} = \frac{2}{h} \frac{\partial N_b}{\partial \xi}, \qquad for b = 1, 2. \tag{14}$$

Therefore, the Convection and diffusion matrices are:

$$\boldsymbol{C}^e = a \int_{\Omega^e} \begin{bmatrix} N_1 \frac{\partial N_1}{\partial x} & N_1 \frac{\partial N_2}{\partial x} \\ N_2 \frac{\partial N_1}{\partial x} & N_2 \frac{\partial N_2}{\partial x} \end{bmatrix} dx = \frac{a}{2} \begin{bmatrix} -1 & +1 \\ -1 & +1 \end{bmatrix} \tag{15}$$

$$\boldsymbol{K}^e = \nu \int_{\Omega^e} \begin{bmatrix} \frac{\partial N_1}{\partial x} \frac{\partial N_1}{\partial x} & \frac{\partial N_1}{\partial x} \frac{\partial N_2}{\partial x} \\ \frac{\partial N_2}{\partial x} \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x} \frac{\partial N_2}{\partial x} \end{bmatrix} dx = \frac{\nu}{h} \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} \tag{16}$$

And the forcing vector is:

$$\boldsymbol{f}^e \int_\Omega \{N_1 s1 + N_2 s2, N2(N_1 s1 + N_2 s_2)^T dx\} \tag{17}$$

With these results and assembling the finite element contributions, one obtains the following expression for an interior node j:

$$a\left(\frac{u_{j+1} - u_{j-1}}{2h}\right) - \nu\left(\frac{u_{j+1} - 2u_j + u_{j-1}}{h^2}\right) = \frac{1}{6}(s_{j-1} + 4s_j + s_j + 1) \tag{18}$$

Defining the Peclet number as:

$$\boldsymbol{Pe} = \frac{ah}{2\nu} \tag{19}$$

The importance of the Peclet number can be visualized with a simple example, let us consider equation 8 without the source term:

$$au_x - \nu u_{xx} = 0 \tag{20}$$

Notice that it can be rewritten as:

$$(\boldsymbol{Pe} - 1)u_{i+1} + 2u_i - (\boldsymbol{Pe} + 1)u_{i-1} = 0 \tag{21}$$

and further:

$$(u_{i+1} - u_i) = \frac{\boldsymbol{Pe} + 1}{1 - \boldsymbol{Pe}}(u_i - u_{i-1}) \tag{22}$$

Where it can be seen that values of Peclet number larger than 1 cause oscillations. To illustrate this, lets use the provided MATLAB code of the course. The problem solved by the code is $au_x - nuu_{xx} = f$ on a 1D domain and boundary conditions $u(0) = 0$ and $u(1) = 1$, on a domain [0,1].

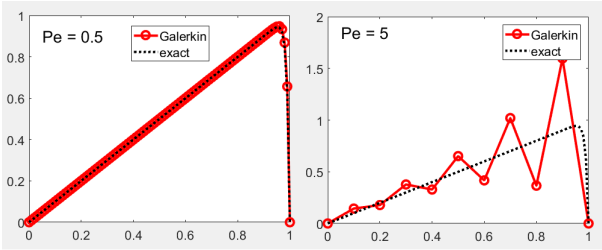

Figure 1: Comparison of Galerkin method using 2 different Peclet number

As seen, in order to obtain stable solutions the Peclet number should be below 1. Sometimes this is difficult to achieve, specially due to the fact that very small elements are to be used in the zones or large gradients.

An alternative to the refinement, are the stabilization methods. The first one to be discussed here is the Streamline Upwind method.

In the Streamline Upwind method (SU), the weak form is:

$$\int_\Omega w(\boldsymbol{a}\cdot\nabla u)d\Omega + \int_\Omega \nabla w\cdot(\nu\nabla u)d\Omega + \int_\Omega w\sigma u d\Omega \underbrace{\int_\Omega \frac{\bar{\nu}}{||\boldsymbol{a}||^2}(\boldsymbol{a}\cdot\nabla w)(\boldsymbol{a}\cdot\nabla u)d\Omega}_{\text{Stabilization Term}} = \int_\Omega s d\Omega \tag{23}$$

Where $\frac{\bar{\nu}}{||\boldsymbol{a}||^2}$ is τ in the code. After running the code using SU method, the following plot is obtained:

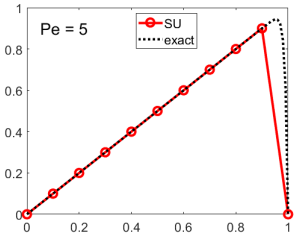

Figure 2: Streamline Upwind method

As can be seen from figure , exact nodal values are obtained. However, this only holds for constant values of the source term s. If one solves a problem where the source term is a function of space the method fails. The following figure shows exactly that.

The source term used in Figure , is $10e^{-5x} - 4e^{-x}$

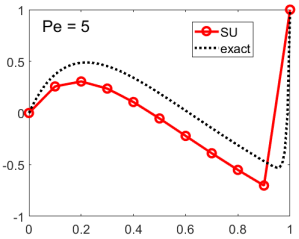

Figure 3: Streamline Upwind method

Notice that the solution not only is not exact at the nodes, but is far from being precise. This is due to the fact that the SU method is not conservative, in the sense that something is being added to the weak form, therefore, another problem (and not the original one) is being solved.

In order to obtain a better solution, the Streamline Upwind Petrov Galerkin (SUPG) method is used. The weak form of SUPG is shown next:

$$\int_\Omega w(\boldsymbol{a}\cdot\nabla u)d\Omega + \int_\Omega \nabla w\cdot(\nu\nabla u)d\Omega + \int_\Omega w\sigma u d\Omega +$$

$$\underbrace{\sum_e \int_{\Omega^e} (\boldsymbol{a}\cdot\nabla w)\tau\left((\boldsymbol{a}\cdot\nabla w) - \nabla\cdot(\nu\nabla u) + \sigma u\right)}_{\text{Stabilization Term}} = \int_\Omega s d\Omega + \underbrace{\sum_e \int_{\Omega^e} (\boldsymbol{a}\cdot\nabla w)\tau s d\Omega}_{\text{Stabilization Term}} \tag{24}$$

The implementation of this is shown next:

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

    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;
        w_ig = wgp(ig)*h/2;
        Nxx_ig = Nxxi(ig,:)*((2/h)^2) ;
        Ke = Ke + w_ig*(N_ig'*a*Nx_ig + Nx_ig'*nu*Nx_ig) ...
            +w_ig*(tau*a*Nx_ig)'*(a*Nx_ig - nu*Nxx_ig + sigma*Nx_ig);
        x = N_ig*Xe; % x-coordinate of the gauss point
        s = SourceTerm(x,example);
        fe = fe + w_ig*(N_ig'*s + (a*Nx_ig)'*tau*s*w_ig;
    end
    % Assmebly
    K(Te,Te) = K(Te,Te) + Ke;
    f(Te) = f(Te) + fe;
end
```

Figure 4: SUPG implementation in 1D

And after running the same case in which SU method failed, the following plot is obtained:
The implementation of this is shown next:

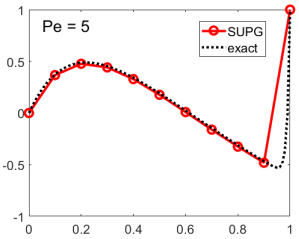

Figure 5: SUPG method for solving the PDE with variable source term

As can be seen, SUPG is nodally exact even when the source term is a function of space. This is because the stabilization is done on a conservative manner, using the residual.

However effective, SUPG presents a slight disadvantage by introducing a non-symmetric stabilization term. Therefore, the Galerkin Least Squares method is also introduced (GLS). The weak form of GLS is shown next:

$$\int_\Omega w(\boldsymbol{a}\cdot\nabla u)d\Omega + \int_\Omega \nabla w\cdot(\nu\nabla u)d\Omega + \int_\Omega w\sigma u d\Omega +$$

$$\underbrace{\sum_e \int_{\Omega^e} \left((\boldsymbol{a}\cdot\nabla w) - \nabla\cdot(\nu\nabla u) + \sigma u\right)\tau\left((\boldsymbol{a}\cdot\nabla w) - \nabla\cdot(\nu\nabla u) + \sigma u\right)}_{\text{Stabilization Term}}$$

$$= \int_\Omega s d\Omega + \underbrace{\sum_e \int_{\Omega^e} \left((\boldsymbol{a}\cdot\nabla w) - \nabla\cdot(\nu\nabla u) + \sigma u\right)\tau s d\Omega}_{\text{Stabilization Term}} \tag{25}$$

This method offers the advantage that its not only conservative, but also symmetric. The implementation of this method is shown next:

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

    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;
        w_ig = wgp(ig)*h/2;
        Nxx_ig = Nxxi(ig,:)*((2/h)^2) ;
        Ke = Ke + w_ig*(N_ig'*a*Nx_ig + Nx_ig'*nu*Nx_ig) ...
            + w_ig*(a*Nx_ig - nu*Nxx_ig + sigma*N_ig)'*tau*(a*Nx_ig - nu*Nxx_ig + sigma*N_ig);
        x = N_ig*Xe; % x-coordinate of the gauss point
        s = SourceTerm(x,example);
        fe = fe + w_ig*(N_ig)'*s + (a*Nx_ig - nu*Nxx_ig + sigma*N_ig)'*tau*s*w_ig;
    end
    % Assembly
    K(Te,Te) = K(Te,Te) + Ke;
    f(Te) = f(Te) + fe;
end
```

Figure 6: GLS implementation in 1D

And the plot after running the code:

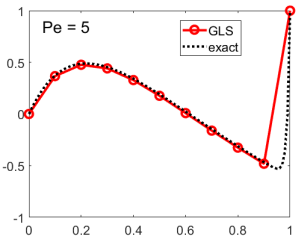

Figure 7: GLS method for solving the PDE with variable source term

Quadratic Elements

For quadratic elements, some remarkable changes are introduced. First, the shape functions used are now the following:

$$N_1(\xi) = \frac{1}{2}\xi(\xi-1) \qquad N_2(\xi) = 1-\xi^2 \qquad N_3(\xi) = \frac{1}{2}\xi(\xi+1) \tag{26}$$

It follows that:

$$\frac{\partial N_b}{\partial x} = \frac{\partial N_b}{\partial \xi}\frac{\partial \xi}{\partial x} = \frac{1}{h}\frac{\partial N_b}{\partial \xi}, \qquad b = 1,2,3. \tag{27}$$

Notice that h is the space from node to node, and not the length of the element.
Then, the Stiffness and Convection matrices are:

$$\boldsymbol{C}^e = a\int_{\Omega^e}\begin{bmatrix} N_1\frac{\partial N_1}{\partial x} & N_1\frac{\partial N_2}{\partial x} & N_1\frac{\partial N_3}{\partial x} \\ N_2\frac{\partial N_1}{\partial x} & N_2\frac{\partial N_2}{\partial x} & N_1\frac{\partial N_3}{\partial x} \\ N_3\frac{\partial N_1}{\partial x} & N_3\frac{\partial N_2}{\partial x} & N_3\frac{\partial N_3}{\partial x}\end{bmatrix} dx = \frac{a}{2}\begin{bmatrix} -1 & 4/3 & -1/3 \\ -4/3 & 0 & 4/3 \\ 1/3 & -4/3 & 1\end{bmatrix} \tag{28}$$

$$\boldsymbol{K}^e = \nu\int_{\Omega^e}\begin{bmatrix} \frac{\partial N_1}{\partial x}\frac{\partial N_1}{\partial x} & \frac{\partial N_1}{\partial x}\frac{\partial N_2}{\partial x} & \frac{\partial N_1}{\partial x}\frac{\partial N_3}{\partial x} \\ \frac{\partial N_2}{\partial x}\frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x}\frac{\partial N_2}{\partial x} & \frac{\partial N_2}{\partial x}\frac{\partial N_3}{\partial x} \\ \frac{\partial N_3}{\partial x}\frac{\partial N_1}{\partial x} & \frac{\partial N_3}{\partial x}\frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial x}\frac{\partial N_3}{\partial x}\end{bmatrix} dx = \frac{\nu}{6h}\begin{bmatrix} 7 & -8 & 1 \\ -8 & 16 & -8 \\ 1 & -8 & 7\end{bmatrix} \tag{29}$$

But even more interesting is that now, not only one stabilization parameter τ is needed, but two of them. One on the side nodes, and one in the center node (the one in the center node is the same one used for both nodes in the linear element case).

In the case of linear SU approximation, the stabilization parameters needed are:

$$\alpha = coth(\boldsymbol{Pe}) - \frac{1}{\boldsymbol{Pe}}; \quad \tau = \alpha\frac{h}{2a} \tag{30}$$

$$\beta = 2((coth(\boldsymbol{Pe}) - 1/\boldsymbol{Pe}) - (cosh(\boldsymbol{Pe}))^2(coth(2\boldsymbol{Pe}) - 1/(2\boldsymbol{Pe})))/(2 - (cosh(\boldsymbol{Pe}))^2);$$

$$\tau_c = \beta\frac{h}{2a} \tag{31}$$

Where τ is the parameter to be used on the inner node, and τ_c is the parameter to be used on the corner nodes. For SUPG and GLS, the parameters are now:

$$\alpha = coth(\boldsymbol{Pe}) - \frac{1}{\boldsymbol{Pe}}; \quad \tau = \alpha\frac{h}{2a} \tag{32}$$

$$\beta = \frac{(2\boldsymbol{Pe}-1)e^{3\boldsymbol{Pe}} + (-6\boldsymbol{Pe}+7))e^{\boldsymbol{Pe}} + (-6\boldsymbol{Pe}-7))e^{-\boldsymbol{Pe}} + (2\boldsymbol{Pe}+1))e^{-3\boldsymbol{Pe}}}{(\boldsymbol{Pe}+3)e^{3\boldsymbol{Pe}} + (-7\boldsymbol{Pe}) - 3)e^{\boldsymbol{Pe}} + (7\boldsymbol{Pe}-3)e^{(-\boldsymbol{Pe}} - (\boldsymbol{Pe}+3)e^{-3\boldsymbol{Pe}}}$$

$$\tau_c = \beta\frac{h}{2a} \tag{33}$$

The implementation of the quadratic elements in the code starts by changing the way the coordinates and connectivities is generated.

```
% Computational domain
dom = [0,1];
example.dom = dom;
% Discretization
disp(' ')
nElem = cinput('Number of elements',10);
nPt = (p*nElem) + 1;
h = (dom(2) - dom(1))/(nElem*p);
X = (dom(1):h:dom(2))';
if p==2
T = [1:p:nPt-1; 2:p:nPt; 3:p:nPt+1]';
else
T= [1:nPt-1; 2:nPt]';
end
```

Figure 8: Quadratic Grid Generation

The implementation of the methods is shown in figure 9.

Notice that the parameter τ is now a matrix with the following shape:

$$\begin{bmatrix} \tau_c & 0 & 0 \\ 0 & \tau & 0 \\ 0 & 0 & \tau_c \end{bmatrix}$$

Where τ_c is the vaue of the stabilization parameter in the corners, and τ is the stabilization parameter in the middle node.

The plots of solving the PDE with each method, using only 5 elements (Same amount of nodes as in the linear case) is shown in figure 10.

It is possible to observe that the Galerkin method one again fluctuates, the SU method again is giving stable results, but far from the analytical solution. SUPG is nodally exact, and GLS is close to the analytical, but exact anymore; this might be due to the τ parameter chosen for SUPG and GLS.

```
% Loop on elements
for ielem=1:nElem
    Te = T(ielem,:);
    Xe = X(Te);
    h = Xe(end) - Xe(1);
    h=h/2;
    Ke = zeros(nen);
    fe = zeros(nen,1);
    % Loop on Gauss points
    for ig = 1:ngaus
        N_ig = N(ig,:);
        Nx_ig = Nxi(ig,:)/h;
        w_ig = wgp(ig)*h;
        Ke = Ke + w_ig*(N_ig'*(a*Nx_ig) + Nx_ig'*(nu*Nx_ig));
        x = N_ig*Xe; % x-coordinate of the gauss point
        s = SourceTerm(x,example);
        fe = fe + w_ig*(N_ig')*s;
    end
    % Assmebly
    K(Te,Te) = K(Te,Te) + Ke;
    f(Te) = f(Te) + fe;
end
```

Galerkin

```
% Loop on elements
for ielem = 1:nElem
    Te = T(ielem,:);
    Xe = X(Te);
    h = Xe(end) - Xe(1);
    h=h/2;
    Ke = zeros(nen);
    fe = zeros(nen,1);
    %Loop on Gauss points
    for ig = 1:ngaus
        N_ig = N(ig,:);
        Nx_ig = Nxi(ig,:)/h;
        w_ig= wgp(ig)*h;
        Nxx_ig = Nxxi(ig,:)/(h^2) ;
        Ke = Ke + w_ig*(N_ig'*a*Nx_ig + Nx_ig'*nu*Nx_ig) ...
            + w_ig*tau*(a*Nx_ig)'*(a*Nx_ig);
        x = N_ig*Xe; % x-coordinate of the gauss point
        s = SourceTerm(x,example);
        fe = fe + w_ig*(N_ig)'*s ;
    end
    %Assmebly
    K(Te,Te) = K(Te,Te) + Ke;
    f(Te) = f(Te) + fe;
end
```

SU

```
% Loop on elements
for ielem = 1:nElem
    Te = T(ielem,:);
    Xe = X(Te);
    h = Xe(end) - Xe(1);
    h=h/2;
    Ke = zeros(nen);
    fe = zeros(nen,1);
    %Loop on Gauss points
    for ig = 1:ngaus
        N_ig = N(ig,:);
        Nx_ig = Nxi(ig,:)/h;
        w_ig= wgp(ig)*h;
        Nxx_ig = Nxxi(ig,:)/(h^2) ;
        Ke = Ke + w_ig*(N_ig'*a*Nx_ig + Nx_ig'*nu*Nx_ig) ...
            + w_ig*tau*(a*Nx_ig - nu*Nxx_ig + sigma*Nx_ig);
        x = N_ig*Xe; % x-coordinate of the gauss point
        s = SourceTerm(x,example);
        fe = fe + w_ig*(N_ig)'*s + tau*(a*Nx_ig)'*s*w_ig;
    end
    %Assmebly
    K(Te,Te) = K(Te,Te) + Ke;
    f(Te) = f(Te) + fe;
end
```

SUPG

```
% Loop on elements
for ielem = 1:nElem
    Te = T(ielem,:);
    Xe = X(Te);
    h = Xe(end) - Xe(1);
    h=h/2;
    Ke = zeros(nen);
    fe = zeros(nen,1);
    % Loop on Gauss points
    for ig = 1:ngaus
        N_ig = N(ig,:);
        Nx_ig = Nxi(ig,:)/h;
        w_ig = wgp(ig)*h;
        Nxx_ig = Nxxi(ig,:)/(h^2) ;
        Ke = Ke + w_ig*(N_ig'*a*Nx_ig + Nx_ig'*nu*Nx_ig) ...
            + w_ig*tau*(a*Nx_ig - nu*Nxx_ig + sigma*Nx_ig)'*(a*Nx_ig - nu*Nxx_ig + sigma*Nx_ig);
        x = N_ig*Xe; % x-coordinate of the gauss point
        s = SourceTerm(x,example);
        fe = fe + w_ig*(N_ig)'*s + tau*(a*Nx_ig - nu*Nxx_ig + sigma*Nx_ig)'*s*w_ig;
    end
    % Assmebly
    K(Te,Te) = K(Te,Te) + Ke;
    f(Te) = f(Te) + fe;
end
```

GLS

Figure 9: Quadratic Implementations

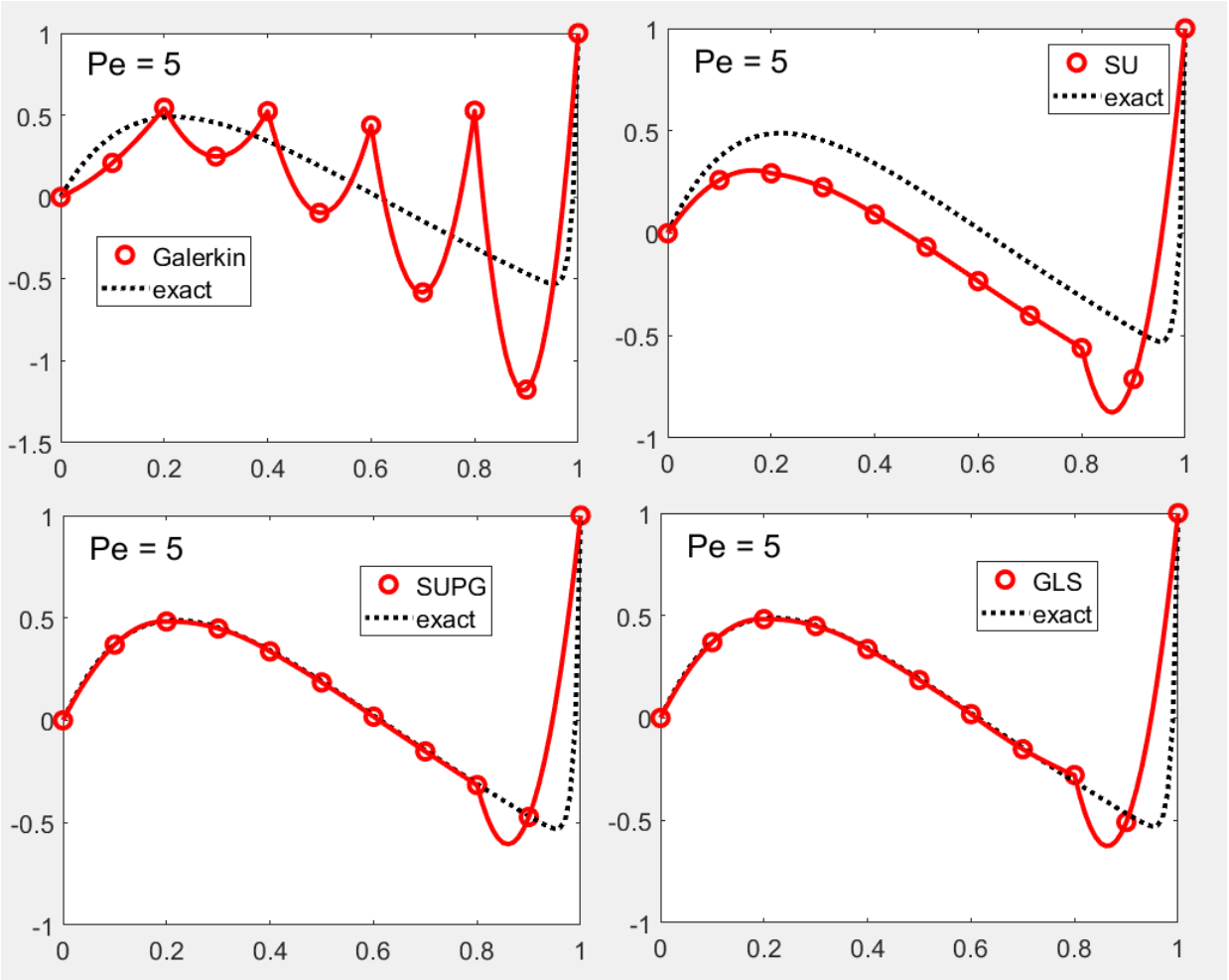

Figure 10: Quadratic Plots