Màster en Métodes Numèrics **Aitor Bazán Escoda**

Universitat Politècnica de Catalunya

Finite Elements in Fluids

Homework 2: Steady-transport examples

Problem statement

1D convection-diffusion equation with constant coefficients and Dirichlet boundary conditions:

$$a u_x - \nu u_{xx} = s \quad x \,\epsilon\, [0,1]$$

$$u(0) = u_0; u(1) = u_1$$

There are 3 examples to go through:

- $s = 0, u_0 = 0, u_1 = 1$
- $s = 1, u_0 = 0, u_1 = 0$
- $s = \sin(\pi x), u_0 = 0, u_1 = 1$

From the strong form to the weak form, these steps have been followed:

First, the differential function is multiplied by the test function $v, v = 0 \; on \; \Gamma_D$.

$$-\int_\Omega v\nu u'' \, d\Omega + \int_\Omega a u' d\Omega = \int_\Omega v s \, d\Omega \rightarrow -\int_0^1 v(x) \frac{\nu \partial^2 u(x)}{dx^2} dx + \int_0^1 a v(x) \frac{\partial u(x)}{dx} dx = \int_0^1 v(x) s(x) \, dx$$

Second, by integrating by parts:

$$-\int_0^1 v(x) \nu \frac{\partial^2 u(x)}{dx^2} dx = \int_0^1 \nu \frac{\partial v(x)}{dx} \frac{\partial u(x)}{dx} dx - \nu \frac{\partial u(x) v(x)}{dx}\bigg|_0^1$$

Rearranging terms

$$\int_0^1 \nu \frac{\partial v(x)}{dx} \frac{\partial u(x)}{dx} dx = -\int_0^1 a v(x) \frac{\partial u(x)}{dx} dx + \int_0^1 v(x) s(x) dx + \nu \frac{\partial u(x) v(x)}{dx}\bigg|_0^1$$

Imposing the approximation function and substituting into the equation

$$u(x) \cong u(x)^h = \sum_{i=1}^{n} N_i(x) u_i$$

$$\int_0^1 \nu \frac{\partial v(x)}{dx} \frac{\partial}{dx} \sum_{i=1}^{n} N_i(x) u_i \, dx = -\int_0^1 a v(x) \frac{\partial \sum_{i=1}^{n} N_i(x) u_i}{dx} dx + \int_0^1 v(x) s(x) dx + \nu \frac{\partial \sum_{i=1}^{n} N_i(x) u_i \, v(x)}{dx}\bigg|_0^1$$

$$\sum_{i=1}^{n}\int_0^1 \nu \frac{\partial v_j(x)}{dx}\frac{\partial N_i(x)u_i}{dx}$$
$$= -\int_0^1 a v_j(x)\frac{\partial \sum_{i=1}^{n} N_i(x)u_i}{dx}dx + \int_0^1 v_j(x)s(x)dx + \sum_{i=1}^{n} \nu \frac{\partial N_i(x)v_j(x)u_i}{dx}\Bigg|_0^1$$

By taking, for convenience, the assumption that

$$v_j(x) = N_j(x)$$

Then, it all yelds the weak form of the given function in its strong form, where the test function v vanishes in Dirichlet boundary.

$$\textstyle\sum_{i=1}^{n}\int_0^1 \nu \frac{\partial N_j(x)}{dx}\frac{\partial N_i(x)u_i}{dx} = -\int_0^1 aN_j(x)\frac{\partial \sum_{i=1}^{n} N_i(x)u_i}{dx}dx + \int_0^1 N_j(x)s(x)dx + \sum_{i=1}^{n} \nu \frac{\partial N_i(x)N_j(x)u_i}{dx}\Big|_0^1$$

Task statements, resolution and conclusions

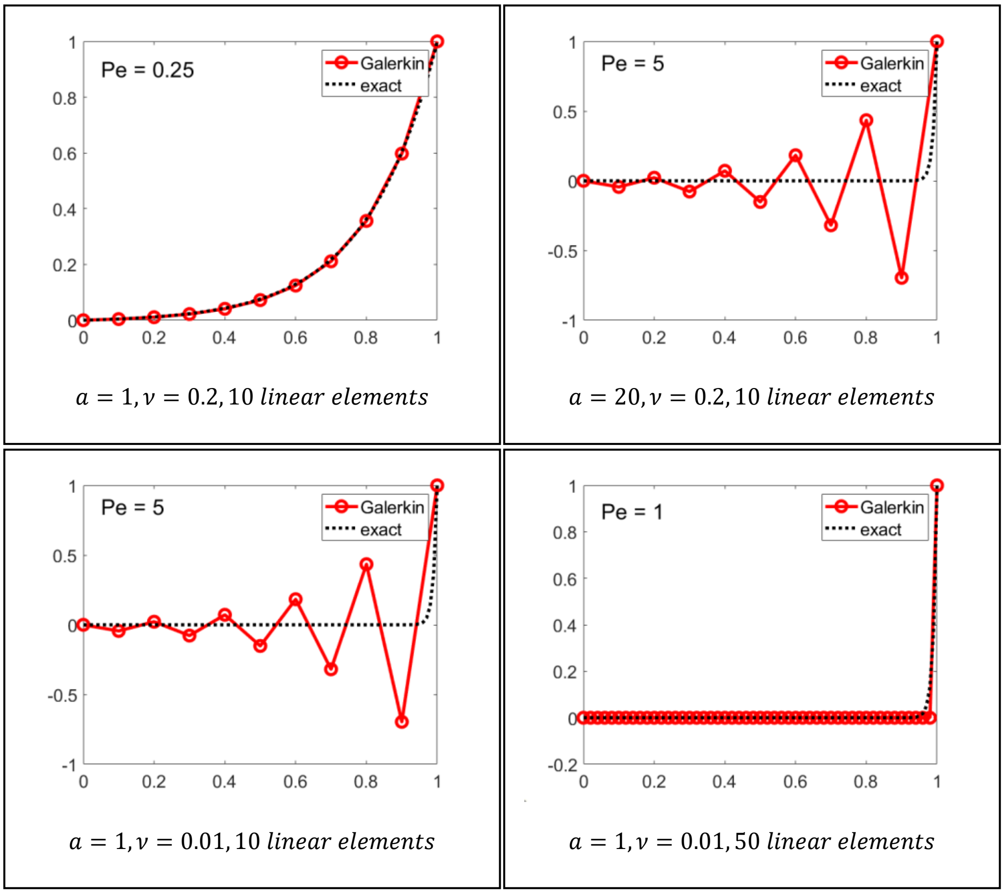
1. By using Galerkin's method:

$$a = 1, \nu = 0.2, 10\ linear\ elements$$

$$a = 20, \nu = 0.2, 10\ linear\ elements$$

$$a = 1, \nu = 0.01, 10\ linear\ elements$$

$$a = 1, \nu = 0.01, 50\ linear\ elements$$



As it is shown above, Galerkin method is not suitable to solve convection-dominated problems. Here, the dominancy is predicted by the Péclet number which expresses the ratio of convective to diffusive transport.

The problem statement consists of solving a 1D boundary value problem with a constant source term, $s = 0, u_0 = 0, u_1 = 1$, in a dimensionless domain L=1. With that uniform source, are avoided the truncation errors due to spatial discretization of the term. This way, truncation error that just arises from Galerkin discretization of the model problem, may be attributed to the representation of the convection and diffusion operators.

The exact solution to this problem is:

$$y(x) = \frac{1 - e^{\frac{ax}{nu}}}{1 - e^{\frac{a}{nu}}}$$

The numerical approximation to the exact solution is computed with several values of a, nu, and number of elements as described before.

As the results are displayed in the figures presented above, one notes that the Galerkin solution loses accuracy by non-physical oscillations when the Péclet number is larger than 1.

$$Pe = \frac{ah}{2\nu}$$

The meaning underlies in the fact that the method lacks its best approximation property when non-symmetric convection operator dominates over the diffusion one in the transport equation. Therefore, spurious node-to-node oscillations appear.

The solution for not having high Péclet numbers lies in the fact that one can use finner meshes by reducing the element size when the convection parameter is high and diffusion is low.

2. Case 3:

$$a = 1, \nu = 0.01, 10 \; linear \; elements$$

Is solved for the three examples of 1st exercise by using:

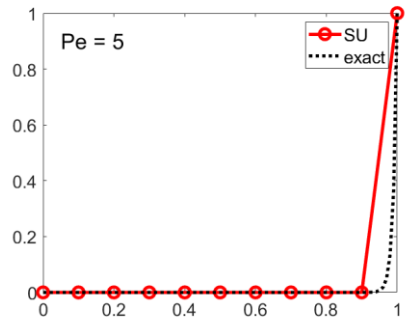
- Streamline upwind
- SUPG
- GLS

With the optimal stabilisation parameter.

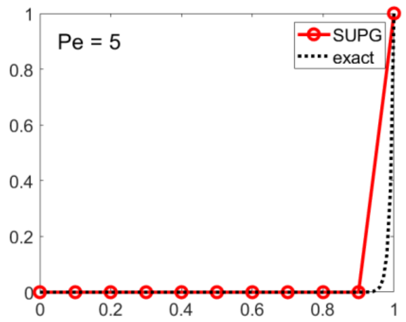
All cases have linear elements

It is now compared the first example with all solvers:

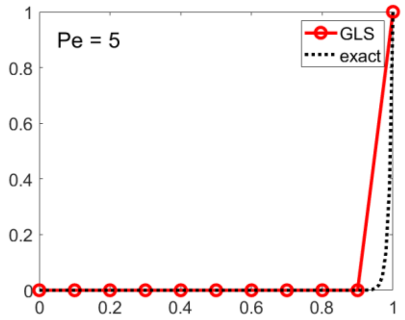
$$s = 0, u_0 = 0, u_1 = 1$$



Stream upwind method

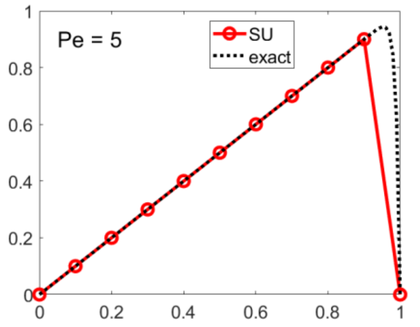


Stream upwind Petrov-Galerkin

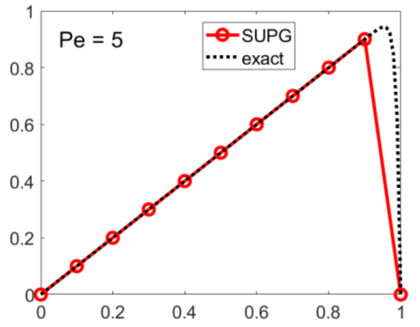


Galerkin Least-Squares

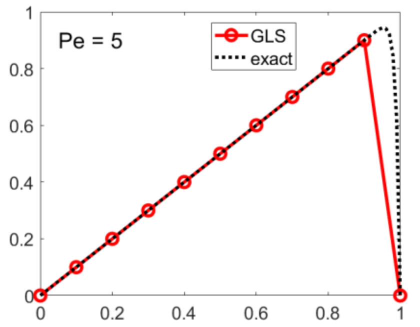
$$s = 1, u_0 = 0, u_1 = 0$$



Stream upwind method

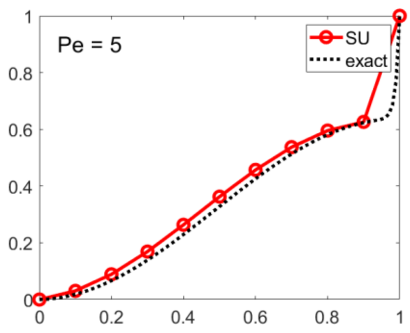


Stream upwind Petrov-Galerkin

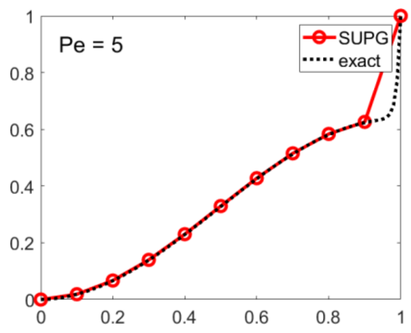


Galerkin Least-Squares

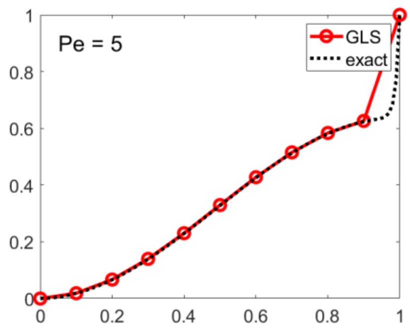
$$s = \sin(\pi x), u_0 = 0, u_1 = 1$$



Stream upwind method



Stream upwind Petrov-Galerkin



Galerkin Least-Squares

$$s = 10e^{-5x} - 4e^{-x}, u_0 = 0, u_1 = 1$$

To use the exact solution to compare with the solution of the different methods, the analytical solution of the partial differential equation has been obtained.

As it is a second order PDE with non-homogeneous solution, the steps carried out are:

The PDE has particular and homogeneous solution (y_p, y_h)

The particular solution is calculated first:

Let a = a and b = nu;

$$y_{p_1} = A10e^{-5x}$$

$$y_{p_1}{}' = -50Ae^{-5x}$$

$$y_{p_1}{}'' = 250Ae^{-5x}$$

$$-250bAe^{-5x} - 50aAe^{-5x} = 10e^{-5x}$$

$$A = \frac{10}{-250b - 50a}$$

$$y_{p_1} = \frac{10}{-250b - 50a} 10e^{-5x} = \frac{e^{-5x}}{-2.5b - 0.5a}$$

The second particular solution:

$$y_{p_2} = -A4e^{-x}$$

$$y_{p_2}{}' = 4Ae^{-x}$$

$$y_{p_2}{}'' = -4Ae^{-x}$$

$$4bAe^{-x} + 4aAe^{-x} = -4Ae^{-x}$$

$$A = -\frac{1}{b+a}$$

$$y_{p_2} = -\frac{1}{b+a}4e^{-x}$$

The homogeneous solution calculated with the proper characteristic equation:

$$y_h = c_1 + e^{\frac{ax}{b}}c_2$$

$$y_L = y_h + y_{p_1} + y_{p_2} = c_1 + e^{\frac{ax}{b}}c_2 + \frac{e^{-5x}}{-2.5b - 0.5a} - \frac{1}{b+a}4e^{-x}$$

If the solution is solved for the following boundary conditions,

$$u_0 = 0, u_1 = 1$$

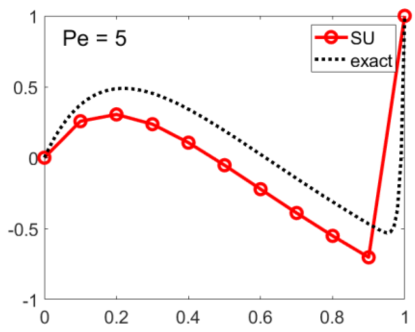
$$c_1 = -\frac{1}{e^{\frac{a}{b}} - 1}\left[1 + 4\frac{(1-e^{-1})}{b+a} + \frac{1-e^{-5}}{-2.5b - 0.5a}\right] - \frac{4}{b+a} - \frac{1}{-2.5b - 0.5a}$$

$$c_2 = -\frac{1}{e^{\frac{a}{b}} - 1}\left[1 + 4\frac{(1-e^{-1})}{b+a} + \frac{1-e^{-5}}{-2.5b - 0.5a}\right]$$

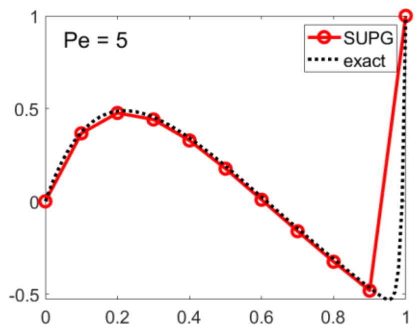
So, the exact analytical solution is used to check method’s accuracy.

Therefore, if the new source is applied:

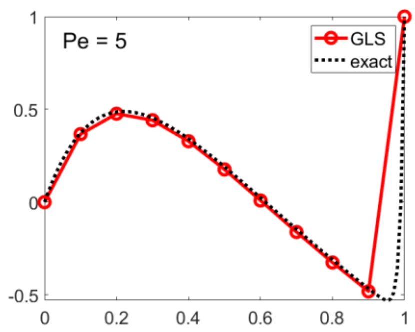
$$s = 10e^{-5x} - 4e^{-x}, u_0 = 0, u_1 = 1$$



Stream upwind method



Stream upwind Petrov-Galerkin



Galerkin Least-Squares

The SU method produces smooth solutions for high Péclet numbers. When the mesh is uniform for linear elements and constant coefficients, SU delivers exact solution.

In the cases where there is a spatially variable source term (cases 3 and 4), the solution is not fitted for this method.

On another hand, SUPG method performs better than the SU method when the source term is a spatially variable term. The definition of the stabilization parameter, tau, and the fact that this parameter is not symmetric also introduces difficulties in determining the stability of such method.

For the cases shown above in which the linear elements are used, no difference can be depicted when comparing SUPG and GLS methods since second-order derivatives are zero in the element interiors.

3. Quadratic elements are implemented.

To do so, it has implemented the way the user inputs the reference to choose linear or quadratic elements.

```
% Reference element: numerical quadrature and shape functions
p = cinput('Input: 1: linear element, 2: quadratic element= ', 1);
% p = 2;
referenceElement = SetRefereceElement(p);
```

Moreover, the number of points for quadratic elements now are 2*nElem+1 and Connectivity matrix has nElem*nen(numb of nodes in an element).

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

The second derivative of the shape functions is also implemented:

```
N2x_ig = N2xi(ig,:)*2/(h^2);
```

In the case of SUPG method, the K element has been calculated as follows:

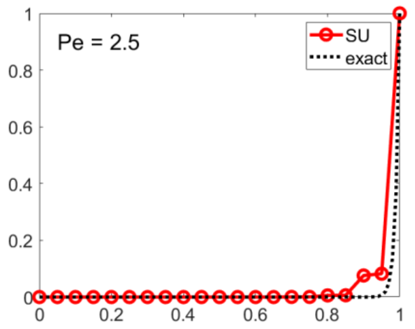
```
Ke = Ke + w_ig*(N_ig'*(a*Nx_ig) + Nx_ig'*(nu*Nx_ig) +
(tau*a*Nx_ig)'*(a*Nx_ig-nu*N2x_ig));
```

While for GLS method:

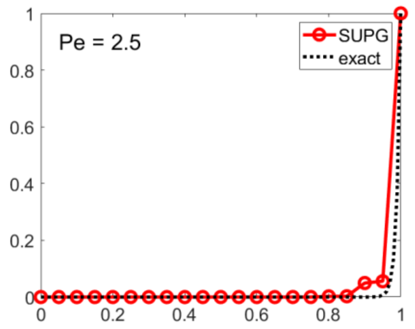
```
Ke = Ke + w_ig*(N_ig'*(a*Nx_ig) + Nx_ig'*(nu*Nx_ig) +
(a*Nx_ig'-N2x_ig')*tau*(a*Nx_ig)-nu*N2x_ig);
```

The results for quadratic elements are presented below:

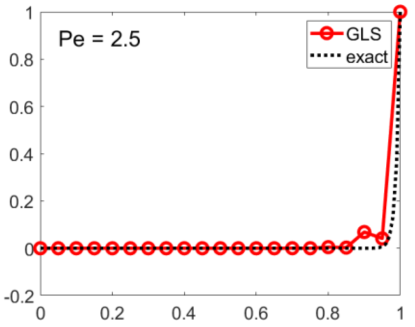
$$s = 0, u_0 = 0, u_1 = 1$$



Stream upwind method

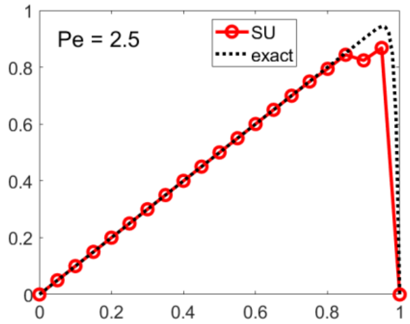


Stream upwind Petrov-Galerkin

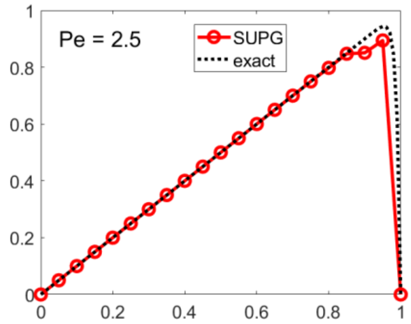


Galerkin Least-Squares

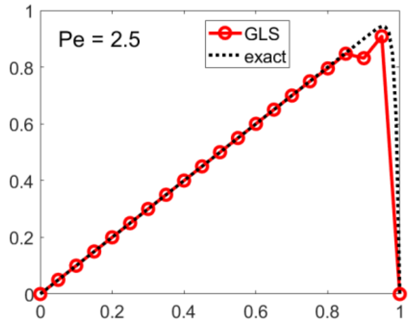
$$s = 1, u_0 = 0, u_1 = 0$$



Stream upwind method

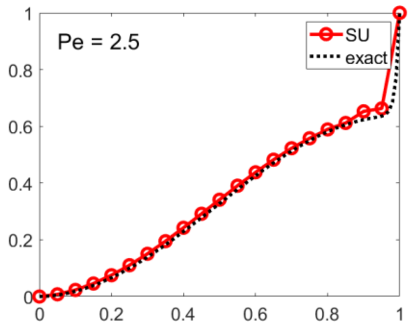


Stream upwind Petrov-Galerkin

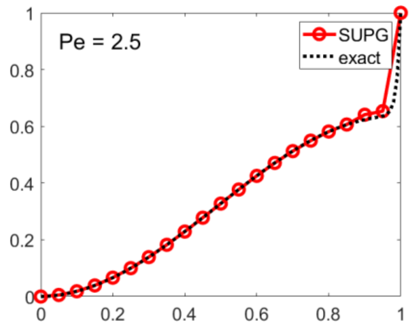


Galerkin Least-Squares

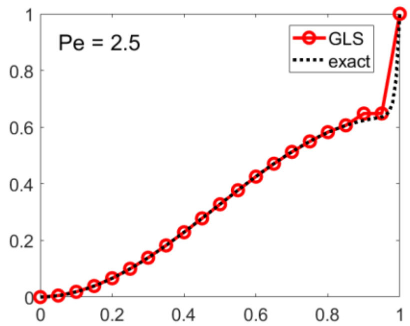
$$s = \sin(\pi x), u_0 = 0, u_1 = 1$$



Stream upwind method

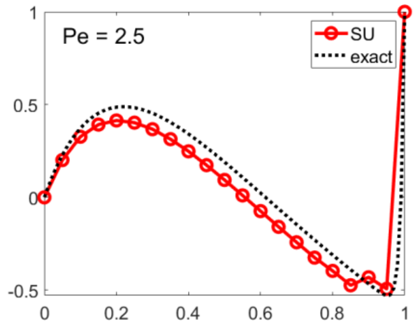


Stream upwind Petrov-Galerkin

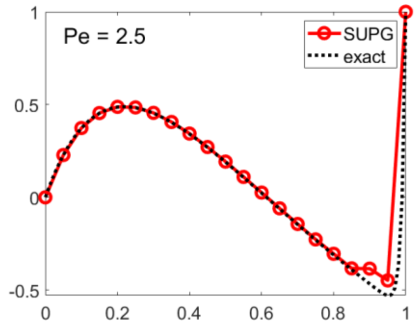


Galerkin Least-Squares

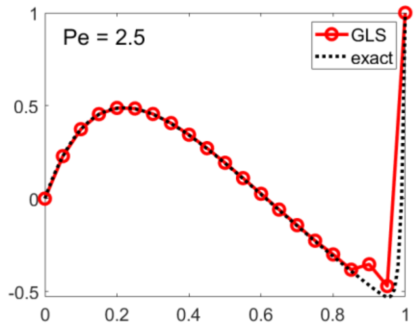
$$s = 10e^{-5x} - 4e^{-x}, u_0 = 0, u_1 = 1$$



Stream upwind method



Stream upwind Petrov-Galerkin

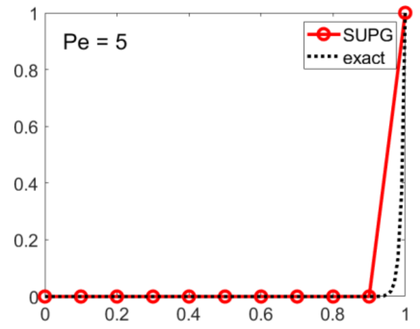


Galerkin Least-Squares

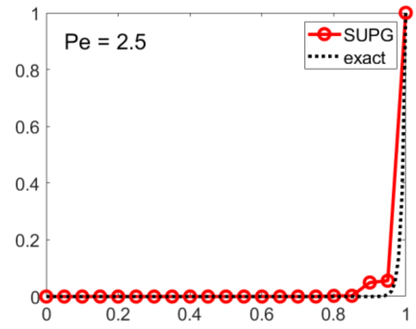
SU with quadratic elements fits better the exact solution but it is still far from it.

There is no minor difference between SUPG and GLS except for the addition of second derivatives and diffusion term in the formulation.

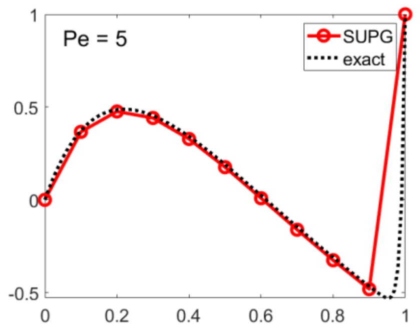
If referring to comparison between linear and quadratic elements it can be seen how with linear elements, when source term is constant or homogeneous, the method behaves better than with quadratic elements. On the contrary, when the source term is spatially distributed (cases 3 and 4), the quadratic elements fit better the exact solution.



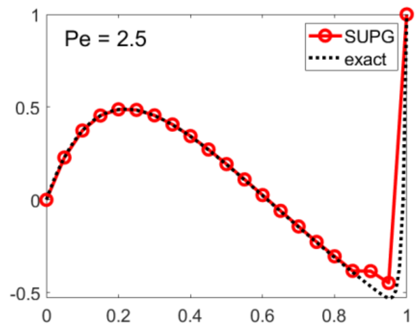
Stream upwind Petrov-Galerkin linear elements



Stream upwind Petrov-Galerkin quadratic elements



Stream upwind Petrov-Galerkin linear elements



Stream upwind Petrov-Galerkin quadratic elements