Computational Mechanics II - Project 1

Philip Moseley

April 27, 2010

1 Introduction

The advection-diffusion equation models situations where a substance is both transported and diffused through a medium. The one dimensional advection-diffusion equation for a particle concentration ϕ can be written

$$u\frac{d\phi}{dx} = v\frac{d^2\phi}{dx^2} \tag{1.1}$$

Where u is the transportation velocity and v is the diffusion coefficient, both of which are assumed to be constant throughout the domain.

Solving the advection-diffusion equation using the standard Galerkin method can lead to oscillatory solutions caused by spatial numerical instabilities. The Petrov-Galerkin method is designed to alleviate these problems by essentially adding a viscous damping force. The implementation here is the Streamlined Upwind Petrov-Galerkin (SUPG) method in one dimension.

A specific model is implemented to show the differences between the Galerkin and Petrov-Galerkin methods. The simulation is a one meter segment of a long tube filled with solution. There is a steady state concentration of 5% at $x = 0$ and 20% at $x = 1$. The velocity $u = 2.0$ m/s and the diffusion coefficient $v = 0.025\ m^2/s$.

2 Analytical Solution

The advection-diffusion equation (1.1) can be solved as follows

$$\int_\Omega \frac{d^2\phi}{dx^2}d\Omega = \int_\Omega \frac{u}{v}\frac{d\phi}{dx}d\Omega \tag{2.1a}$$

$$\frac{d\phi}{dx} = \frac{u}{v}(\phi + C_1) \tag{2.1b}$$

This equation is solved for the homogeneous and inhomogeneous parts separately. The homogeneous part and it's general solution are

$$\frac{d\phi_h}{dx} = \frac{u}{v}\phi_h \tag{2.2a}$$

$$\phi_h = C_2 e^{\frac{ux}{v}} \tag{2.2b}$$

The inhomogeneous part and it's solution are

$$\frac{d\phi_i}{dx} = \frac{u}{v}(\phi_i + C_1) \tag{2.3a}$$

$$\phi_i = -C_1 \tag{2.3b}$$

The final solution is therefore

$$\phi = \phi_h + \phi_i = C_1 + C_2 e^{\frac{ux}{v}} \tag{2.4}$$

The constants C_1 and C_2 are solved from the boundary conditions, and for this problem are determined to be $C_1 = 0.05$ and $C_2 = 2.707 \times 10^{-36}$.

3 Finite Element Solutions

The Galerkin finite element discretization follows from the general form of the advection-diffusion equation (1.1). Using the test function $w(x)$ and integrating over the domain,

$$\int_\Omega w\left(u\frac{d\phi}{dx} - v\frac{d^2\phi}{dx^2}\right)d\Omega = 0 \tag{3.1}$$

From this, the weak form is

$$\int_\Omega wu\frac{d\phi}{dx}d\Omega + \int_\Omega v\frac{dw}{dx}\frac{d\phi}{dx}d\Omega = 0 \tag{3.2}$$

For a uniform finite element mesh, the discrete equations become

$$\boldsymbol{\phi}\int_{\Omega^e} u\mathbf{N}^T\frac{d\mathbf{N}}{d\mathbf{x}}d\Omega^e + \boldsymbol{\phi}\int_{\Omega^e} u\left(\frac{d\mathbf{N}}{d\mathbf{x}}\right)^T\frac{d\mathbf{N}}{d\mathbf{x}}d\Omega^e = 0 \tag{3.3}$$

The Petrov-Galerkin formulation modifies the Galerkin test function by adding an additional term

$$\tilde{w} = w + \gamma\frac{dw}{dx} \tag{3.4}$$

where $\gamma = \alpha\frac{\Delta x}{2}$. In one dimension it is possible to select α such that the solution at the nodes is exact,

$$\alpha = \coth(P_e) - \frac{1}{P_e} \tag{3.5}$$

which is based on the Peclet number,

$$P_e = \frac{u\Delta x}{2v} \tag{3.6}$$

which will be shown to have a large effect on the stability of the solution.

The modified test function leads to the same terms as the regular Galerkin formulation in (3.2), plus additional Petrov-Galerkin stabilization terms,

$$\int_\Omega wu\frac{d\phi}{dx}d\Omega + \int_\Omega v\frac{dw}{dx}\frac{d\phi}{dx}d\Omega + \int_\Omega u\gamma\frac{dw}{dx}\frac{d\phi}{dx}d\Omega - \int_\Omega v\gamma\frac{dw}{dx}\frac{d^2\phi}{dx^2}d\Omega = 0 \tag{3.7}$$

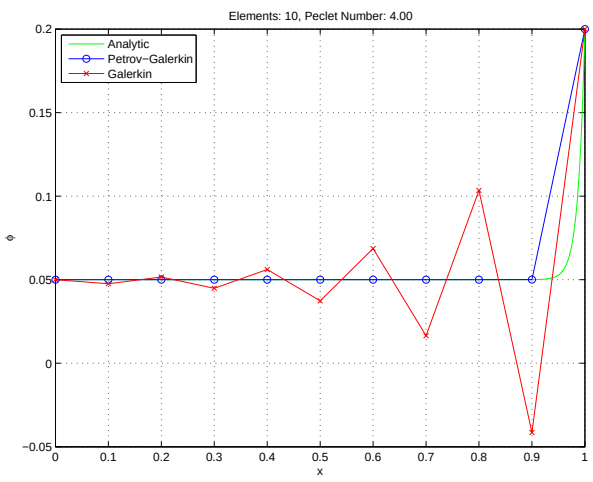
The fourth term depends on the second derivative of ϕ, but for linear shape functions this term drops to zero. So for linear shape functions in a uniform grid, the discrete form of the Petrov-Galerkin is

$$\boldsymbol{\phi}\int_{\Omega^e} u\mathbf{N}^T\frac{d\mathbf{N}}{d\mathbf{x}}d\Omega^e + \boldsymbol{\phi}\int_{\Omega^e} v\left(\frac{d\mathbf{N}}{d\mathbf{x}}\right)^T\frac{d\mathbf{N}}{d\mathbf{x}}d\Omega^e + \boldsymbol{\phi}\int_{\Omega^e} u\gamma\left(\frac{d\mathbf{N}}{d\mathbf{x}}\right)^T\frac{d\mathbf{N}}{d\mathbf{x}}d\Omega^e = 0 \tag{3.8}$$

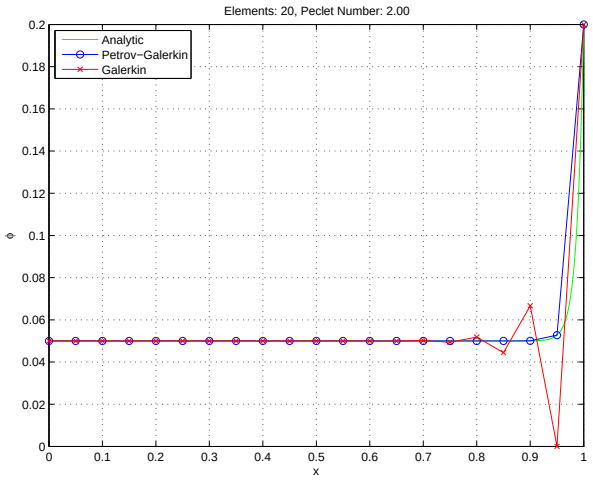
If non-linear shape functions are used, the discarded fourth term must additionally be implemented.

It can be shown that the stability of the Galerkin method depends on the Peclet number, (3.6). If the Peclet number is less than one, then the Galerkin discrete solution approaches the exact solution. For Peclet numbers greater than one, the Galerkin discrete solution becomes oscillatory and the Petrov-Galerkin formulation is required. To illustrate this, the example problem introduced in Section 1 is run with several different Peclet numbers. To achieve the different Peclet numbers the resolution of the mesh is varied, using 10, 20, 50, and 100 elements.

The 10 and 20 element meshes result in Peclet numbers of 4.00 and 2.00, respectively. As shown in Figure 1, the Galerkin method is unstable for these meshes and oscillates badly as x increases. Due to the proper choice of α from (3.5), the Petrov-Galerkin method achieves the exact solution at the nodes, although the choice of linear shape functions limits the accuracy between nodes.



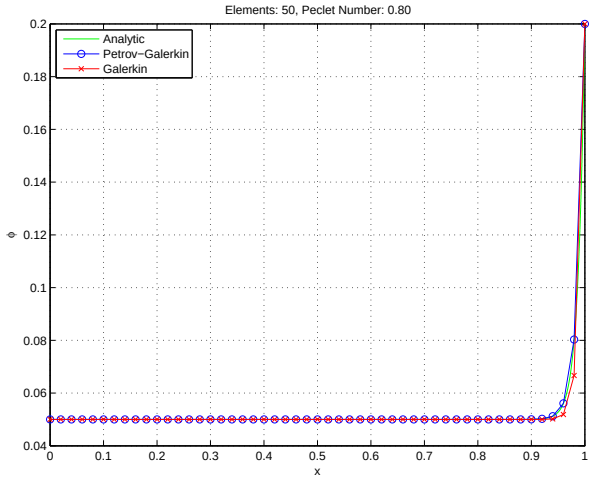
(a) $P_e = 4.00$



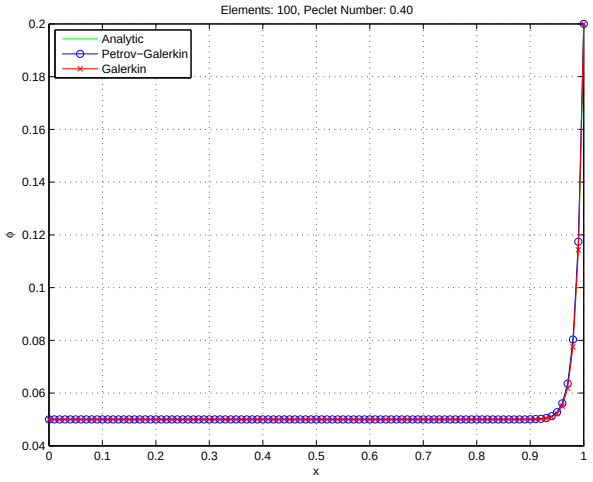
(b) $P_e = 2.00$

Figure 1: Stability at Peclet numbers greater than 1.0

For the finer resolution meshes (50 and 100 elements), the Peclet numbers are 0.80 and 0.40, respectively. The Galerkin method is stable for these values, and compares favorably to the Petrov-Galerkin and analytic solutions, see Figure 2.



(a) $P_e = 0.80$



(b) $P_e = 0.40$

Figure 2: Stability at Peclet numbers less than 1.0

For all cases the Petrov-Galerkin calculates exact solutions at the nodes, due to the choice of α in one dimension. However, the finer meshes give a better resolution of the solution. For the Galerkin method, increasing mesh resolution results in more accurate values at the nodes.

4 SUPG Parameters

The choice of γ in the Petrov-Galerkin formulation plays a major role on the stability and accuracy of the solution. The γ term essentially adds an artificial viscosity to the solution to dampen oscillations. Incorrect choices of γ can lead to the solution being over-damped and inaccurate, or under-damped and oscillatory. In one dimension, γ can be calculated such that the solution is exact at the nodes, (3.5).

In Figure 3 the solution with several values of γ has been plotted. With low damping values the solution remains unstable, and with high values the solution is damped so much that the solution is no longer accurate.

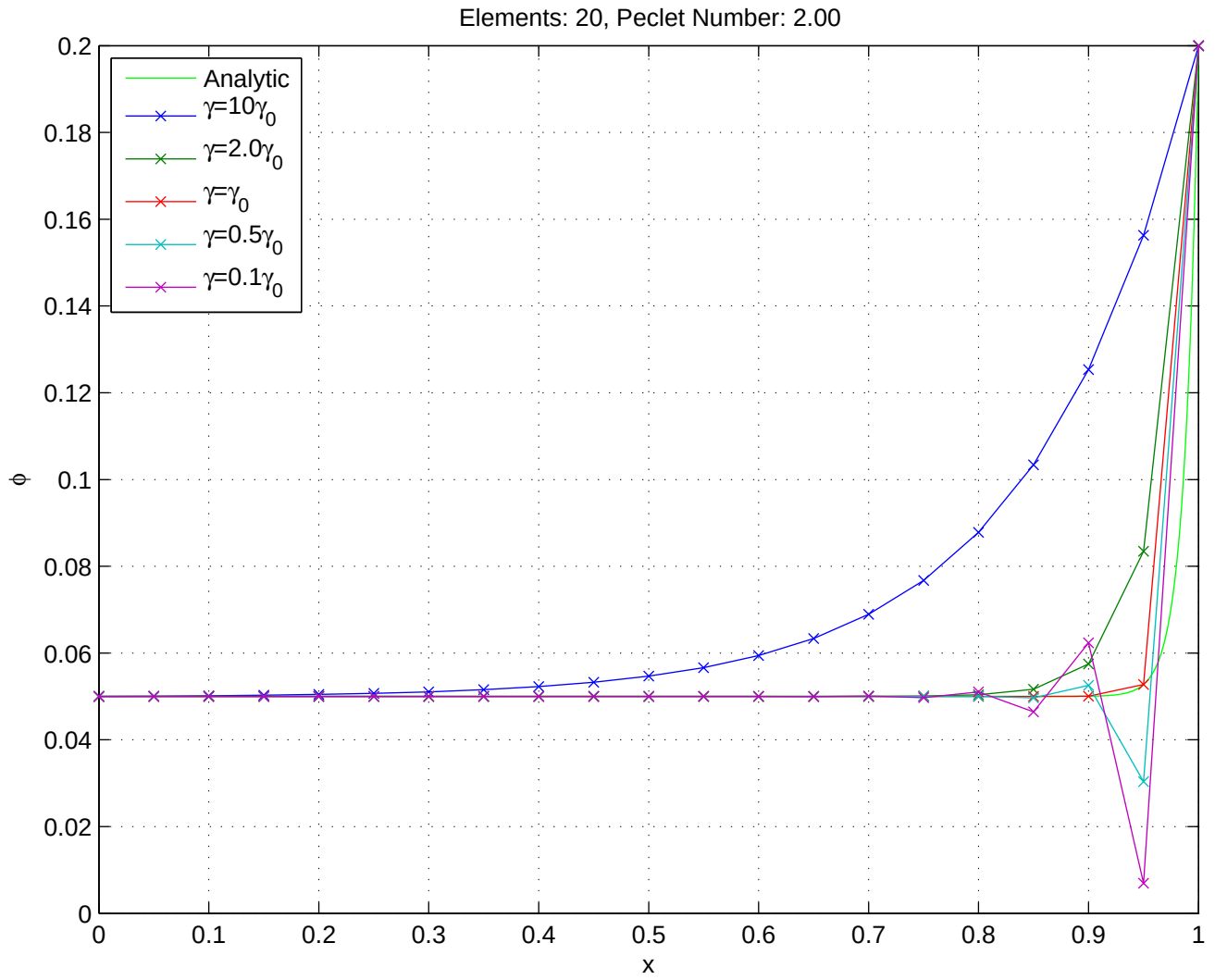
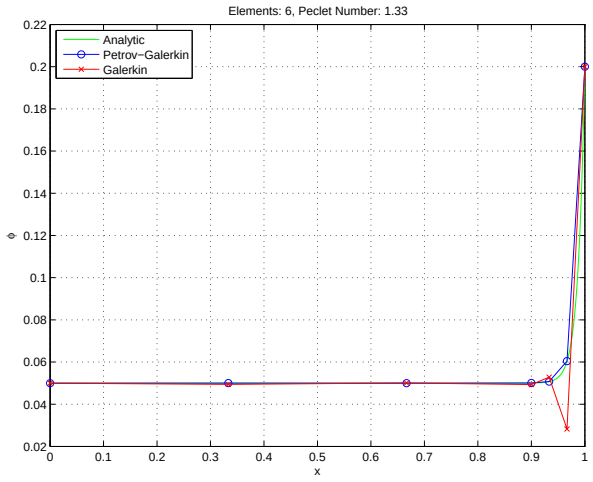


Figure 3: Different γ values, based off the γ_0 exact solution.

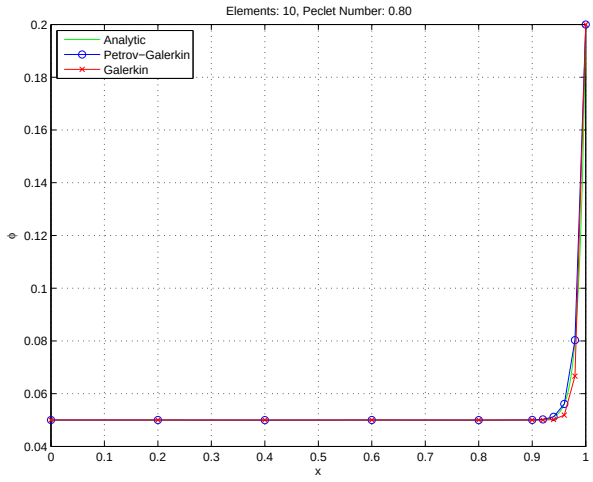
5 Non-uniform Mesh

The stability problems of the Galerkin formulation are spatial instabilities, caused by problems with the mesh. As shown in Section 3, refining the uniform mesh eliminates the oscillations, at the cost of increased computational effort. Selectively refining the mesh around key areas allows the Galerkin solution to remain stable without unnecessary increased computation.

In the example problem, the mesh is refined near the spike in concentration. The mesh on $x = [0.9, 1.0]$ is ten times finer than the mesh from $x = [0.0, 0.9]$. The Peclet number is calculated based on the element length in the refined area. For this simple problem and simple non-uniform mesh, this Peclet number still results in exact solutions for the Petrov-Galerkin formulation.



(a) $P_e = 1.33$ with 6 elements



(b) $P_e = 0.80$ with 10 elements

Figure 4: Solutions for a simple non-uniform mesh.

For a more adaptively refined mesh, the calculation of the γ becomes more difficult, and it may not be possible to calculate a γ resulting in the exact solutions.

6 Matlab Code

```
% Philip Moseley
%  Computational Mechanics II — Project 1
%  Solve the 1d advection diffusion equation using both the
%  Galerkin and Petrov—Galerkin for comparison purposes.
function advection_diffusion
clc; close('all');

for nel=[10,20,50,100] % Number of elements
%======================= Preprocessing ======================%
nne = 2;             % Number of nodes in an element.
ngp = 1;             % Number of Gauss points.
totlength = 1.0;     % Total length of the domain.
vel = 2.0;           % Particle velocity u.
diff_coef = 0.025;   % Diffusion coefficient v.
area = 1.0;          % Cross—sectional area.
alpha = 10^8;        % Penalty factor for enforcing esential bcs.

% Determine the number of nodes and element connectivity matrix.
conn = zeros(nel,nne); nn = 1;
for i=1:nel
    nn = nn—1;
    for j=1:nne, nn = nn+1; conn(i,j) = nn; end
end
le = totlength/nel;              % Element length.
nds = 0:le/(nne—1):totlength;    % Nodal coordinates.
phi = zeros(nn,1);               % Global concentration vector.
[gp,gw] = lgwt(ngp,—1,1);        % Gauss quadrature points and weights.
[N,B,J] = shape_functions(nne); % Shape functions.

% Boundary Conditions.
fixed_nodes = [1,nn];    % List of nodes with essential BCs.
phi_E = [0.05, 0.20];    % Prescribed values at nodes in fixed_nodes.
forced_nodes = [];       % List of nodes with natural BCs.
phi_N = [];              % Prescribed values at nodes in forced_nodes.

% Exact Solution, for error analysis. Need to modify C1,C2 based on BCs.
C2 = 0.15/(exp(vel/diff_coef)—1);
C1 = 0.05—C2;
phi_exact = @(x) C1 + C2*exp((vel/diff_coef)*x);

nds
%========================== Solution ============================%
% Calculate Peclet number.
Pe = vel*le/(2.0*diff_coef)
gamma = le/2.0 * (coth(Pe)—1.0/Pe)

% Calculate with Galerkin and Petrov—Galerkin methods.
for pg=[0,1]      % Run the Galerkin, then Petrov—Galerkin.
    L = sparse(nn,nn);      % Global convective matrix.
    K = sparse(nn,nn);      % Global diffusion matrix.
    F = sparse(nn,1);       % Global external concentration vector.
    % Develop element stiffness and force matrices.
    for e=1:nel
        xe = nds(conn(e,:));
        Le = zeros(nne);
        Ke = zeros(nne);

        % Gauss Quadrature.
        for q=1:length(gp)
            Je = J(gp(q),abs(xe(2)—xe(1)));
            Be = inv(Je)*B(gp(q));
            Le = Le + area*det(Je)*vel*N(gp(q))'*Be*gw(q);
            Le = Le + pg*area*vel*gamma*det(Je)*Be'*Be*gw(q);
            %TODO: need another term here if using non—linear shape fns.
            Ke = Ke + area*det(Je)*diff_coef*Be'*Be*gw(q);
        end

```

```
68          % Assembly.
69          I = conn(e,:);
70          L(I,I) = L(I,I) + Le;
71          K(I,I) = K(I,I) + Ke;
72      end
73
74      % Essential Boundary Conditions (penalty method).
75      BC = sparse(nn,nn);
76      for i=1:length(fixed_nodes)
77          I = fixed_nodes(i);
78          BC(I,I) = BC(I,I) + alpha;
79          F(I) = F(I) + alpha*phi_E(i);
80      end
81      % Natural Boundary Conditions.
82      for i=1:length(forced_nodes)
83          I = forced_nodes(i);
84          F(I) = F(I)+area*phi_N(i);
85      end
86      %%% SOLUTION %%%
87      phi(:,pg+1) = (K+L+BC)\F;
88  end
89
90  % Calculate exact solution for comparison purposes.
91  nds_ex = 0:le/(nne-1)/1000:totlength;
92  phi_ex = phi_exact(nds_ex);
93
94  %======================== Post processing ========================%
95  full(phi)
96  figure; plot(nds_ex,phi_ex,'-g',nds,phi(:,2),'-bo',nds,phi(:,1),'-rx');
97  legend('Analytic','Petrov-Galerkin','Galerkin','Location','NW');
98  grid on; xlabel('x'); ylabel('\phi');
99  title(sprintf('Elements: %d, Peclet Number: %.2f',nel,Pe));
100 print('-depsc', sprintf('mesh_%d-pe_%.2f.eps',nel,Pe));
101 end
102
103 %======================== Helper Functions ========================%
104 function [N,B,J] = shape_functions(nne)
105 switch nne
106     case 2
107         N = @(x) [(1-x)/2.0     (x+1)/2.0];
108         B = @(x) 0.5*[-1,1];
109         J = @(x,le) le/2;
110     case 3
111         N = @(x) [x*(x-1)/2     -(x+1)*(x-1)    x*(x+1)/2];
112         B = @(x) [(2*x-1)/2     -2*x            (2*x+1)/2];
113         J = @(x,le) 0.0;
114     otherwise
115         N = []; B = []; J = [];
116         nne
117 end
```