Finite Elements for Fluids

Juan Pedro Roldán Blasco

April 2018

1 Steady transport 2D

The assignment included:

- Quadratic element formulation
- GLS method
- Zero Dirichlet BC on the outlet boundary
- Comparison between convection-reaction and reaction dominant cases

1.1 Quadratic element formulation

In order to include quadratic elements, the shape-functions matrices were modified to include the second derivatives terms (which are zero for linear elements). This will appear in every coded method as the N2x and N2y terms.

```
N2xi = [(eta.*(eta - 1))/2,...
    (eta.*(eta - 1))/2,...
    (eta.*(eta + 1))/2,...
    (eta.*(eta + 1))/2,...
    -eta.*(eta - 1),...
    1 - eta.^2,...
    -eta.*(eta + 1),...
    1 - eta.^2,...
    2.*eta.^2 - 2];
N2eta = [(xi.*(xi - 1))/2, (xi.*(xi + 1))/2,    (xi.*(xi + 1))/2,...
     (xi.*(xi - 1))/2, 1 - xi.^2,...
    -xi.*(xi + 1),...
      1 - xi.^2,...
    -xi.*(xi - 1),...
      2.*xi.^2 - 2];
  N2xieta = [(eta.*xi)/4 + (eta.*(xi - 1))/4 + (xi.*(eta - 1))/4 + ((eta - 1).*(xi - 1))/4,...
(eta.*xi)/4 + (eta.*(xi + 1))/4 + (xi.*(eta - 1))/4 + ((eta - 1).*(xi + 1))/4,...
(eta.*xi)/4 + (eta.*(xi + 1))/4 + (xi.*(eta + 1))/4 + ((eta + 1).*(xi + 1))/4,...
(eta.*xi)/4 + (eta.*(xi - 1))/4 + (xi.*(eta + 1))/4 + ((eta + 1).*(xi - 1))/4,...
                                        - eta.*xi - xi.*(eta - 1),...
                                        - eta.*xi - eta.*(xi + 1),...
                                        - eta.*xi - xi.*(eta + 1),...
                                        - eta.*xi - eta.*(xi - 1),...
                                                    4.*eta.*xi];
```

Figure 1: Second derivative matrix of the shapefunctions N(ξ,η)

```
xxi = Nxi_ig*(Xe(:,1));
xeta = Neta_ig*(Xe(:,1));
yxi = Nxi_ig*(Xe(:,2));
yeta = Neta_ig*(Xe(:,2));


Jacob2 =  [xxi.^2, 2*xxi.*yxi, yxi.^2
Nxi_ig.*Neta_ig*(Xe(:,1)), xxi.*yeta+xeta.*yxi, Neta_ig.*Nxi_ig*(Xe(:,2))
xeta.^2, 2*xeta.*yeta, yeta.^2];
dvol2 = wgp(ig)*det(Jacob2)/det(Jacob);


res2 = Jacob2\[N2xi_ig;N2xieta_ig;N2eta_ig];
N2x = res2(1,:);
N2xy = res2(2,:);
N2y = res2(3,:);
```

Figure 2: Conversion from natural to general coordinates

1.2 GLS method

Galerkin Least Squares method was implemented by adding the stabilization term $\int(\mathbf{a}\cdot\nabla\omega-\nabla\cdot(\nu\nabla\omega)+\sigma\omega)\tau(\mathbf{a}\cdot\nabla u-\nabla\cdot(\nu\nabla u)+\sigma u-s)$. We can see that we are including second derivatives in the stabilization term. Figure 3 shows a piece of the FEM_system.m funtion of the 2D_SteadyTransport were GLS implementation was written. N2x and N2y account for the second derivatives of the shape-functions matrices.

```
elseif method == 3
    % GLS
    reac = N_ig'*sigma*N_ig;
    Ke = Ke +(nu*(Nx'*Nx+Ny'*Ny) + N_ig'*(ax*Nx+ay*Ny) + reac +...
        tau*(ax*Nx+ay*Ny-nu*(N2x+N2y)+reac)'*((ax*Nx+ay*Ny-nu*(N2x+N2y)*dvol2 + reac)))*dvolu;
    aux = N_ig*Xe;
    f_ig = SourceTerm(aux); % s term
    fe = fe + (N_ig+tau*(ax*Nx+ay*Ny-nu*(N2x+N2y)*dvol2+ sigma*N_ig))'*(f_ig*dvolu);
end
```

Figure 3: GLS method implementation for steady transport problem

1.3 Zero Dirichlet Boundary Conditions

In order to impose zero Dirichlet boundary conditions on the outlet boundary we had to slightly modify the boundary conditions routine in the main.m script. The difference between only Dirichlet and Dirichlet + Neumann BC can be observed in figures 5 and 6. When zero dirichlet (u = 0) are imposed, we may find sharp gradients of the solution in the vicinity of the outlet boundary.

```
    % nodes on which solution is u=1
    nodesDir1 = nodes_x0( X(nodes_x0,2) > 0.2 );
    % nodes on which solution is u=0
    nodesDir0 = [nodes_x0( X(nodes_x0,2) <= 0.2 ); nodes_y0; nodes_x1; nodes_y1];
     C = [nodesDir1, ones(length(nodesDir1),1);
          nodesDir0, zeros(length(nodesDir0),1)];
    nDir = size(C,1);
    neq  = size(f,1);
    A = zeros(nDir,neq);
    A(:,C(:,1)) = eye(nDir);

    b = C(:,2);
 Ktot = [K A';A zeros(nDir,nDir)];
ftot = [f;b];
end
```

Figure 4: Boundary conditions when only Dirichlet BC are considered. The vector of u = 0 has been expanded with nodes_x1 and nodes_y1. 20x20 mesh.

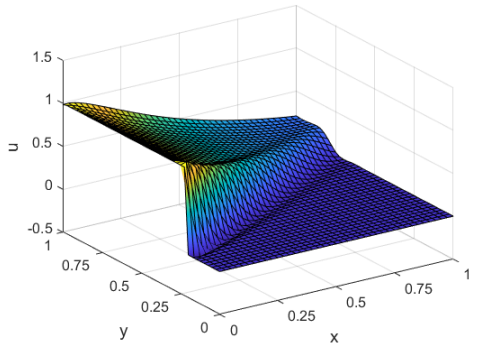

Figure 5: Solution for $a = 0.5$, $\nu = 10^{-4}$ and $\sigma = 1$, Dirichlet and Neumann boundary conditions. 20x20 mesh.

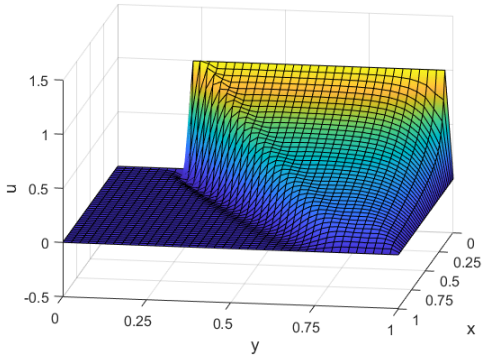

Figure 6: Solution for $a = 0.5$, $\nu = 10^{-4}$ and $\sigma = 1$, only Dirichlet boundary conditions. Notice the gradient at the right side of the graph (for y = 1). 20x20 mesh.

1.4 Convection-reaction vs Reaction cases

The convection-reaction case can be seen previously in figure 5. If we now compare the solution with 7 we can see that the differences are clear. For the reaction dominant case the solution inmediatly tends to zero. This is due to the fact that it reacts (an thus, disappears) faster than it is transported. When convection velocity is predominant, it is travelling further until it reacts completely (u = 0). Notice the oscillations in the internal boundary layer between non-reacted and reacted.

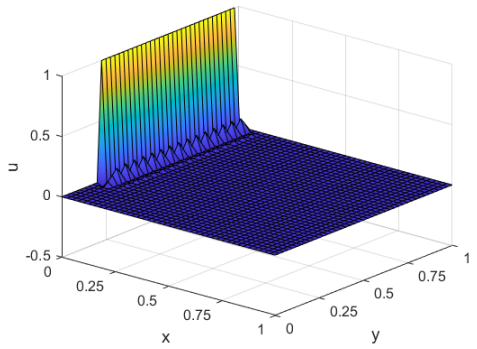

Figure 7: Solution for $a = 10^{-3}$, $\nu = 10^{-4}$ and $\sigma = 1$, only Dirichlet boundary conditions. 20x20 mesh.

2 Unsteady 1D convection

The objectives were to develop the Leap frog, Taylor-Galerkin and two steps Taylor-Galerkin methods. Figure 8 shows their implementation in the provided code. The development is done in the subsections below.

```
case 5 % Third order Taylor-Galerkin + Galerkin
    A = M + a^2*dt^2*K/6;
    B = -a*C*dt - 0.5*dt^2*a^2*K;
    methodName = '3rd Taylor-Galerkin';
    A2 = 0;
    B2 = 0;
    C2 = 0;
case 6 % Leap frog + Galerkin
    A = M;
    B = -2*a*C*dt;
    methodName = 'Leap frog';
    A2 = 0;
    B2 = 0;
    C2 = 0;
case 7 % Two steps Galerkin
    A = M;
    B = M - (dt*a/3) * C - ((dt*a)^2/9) * K;
    A2 = M;
    B2 = -dt*a*C;
    C2 = - 0.5*((dt*a)^2)*K;
    methodName = 'Two steps Galerkin';
```

Figure 8: Implementation of Taylor-Galerkin, Leap Frog and Two step Taylor Galerkin methods

2.1 Leap frog method

The formulation of the method is

$$\frac{u^{n+1} - u^{n-1}}{2\Delta t} = u_t^n \tag{1}$$

Substituting into the equation for $u_t = -a\nabla u$ for the expression of the problem and developing the Galerkin spatial discretization we obtain

$$(\omega, u^{n+1}) = (\omega, u^{n-1}) - 2a\Delta t(w, \nabla u^n) \tag{2}$$

Which can be converted using the matricial expression into $\mathbf{M}u^{n+1} = \mathbf{M}u^{n-1} - 2a\Delta t\mathbf{C}u^n$. This method is not self-starting, and so, to compute the first time-step we will use Third order Galerkin.

2.2 Third order Taylor-Galerkin method

The time discretization formulation of this method is

$$\frac{\Delta u}{\Delta t} = \frac{u^{n+1} - u^n}{\Delta t} = u_t + \frac{1}{2}\Delta t u_{tt} + \frac{1}{6}\Delta t^2 u_{ttt} \tag{3}$$

With time derivatives

$$\begin{aligned} u_t &= -a\nabla u \\ u_{tt} &= (a\nabla)^2 u \\ u_{ttt} &= (a\nabla)^2 u_t = (a\nabla)^2 \frac{u^{n+1} - u^n}{\Delta t} \end{aligned} \tag{4}$$

The Galerkin spatial discretization yields the following scheme

$$(\omega, (1 - \frac{\Delta t^2}{6})(a\nabla)^2 \frac{\Delta u}{\Delta t}) = (\omega, -a\nabla u^n + \frac{\Delta t}{2}(a\nabla)^2 u^n) \tag{5}$$

Integrating by parts the second derivative terms and using the matricial expressions.

$$\begin{aligned} (w, \frac{\Delta u}{\Delta t}) + \frac{\Delta t}{6}a^2(\nabla\omega, \nabla\Delta u) &= -(\omega, a\nabla u^n) - \frac{\Delta t}{2}a^2(\nabla\omega, \nabla u) \\ (\mathbf{M} + \frac{\Delta t}{6}a^2\mathbf{K})\Delta u &= (-a\mathbf{C} - a^2\frac{\Delta t}{2}\mathbf{K})u \end{aligned} \tag{6}$$

2.3 Two step third order Taylor Galerkin

The expression of the two step third order Taylor Galerkin

$$\begin{aligned} \hat{u} &= u^n + \frac{1}{3}\Delta t u_t^n + \frac{1}{9}\Delta t^2 u_{tt}^n \\ u^{n+1} &= u^n + \Delta t u_t^n + \frac{1}{2}\Delta t^2 \hat{u}_{tt}^n \\ u_t &= -a\nabla u \\ u_{tt} &= (a\nabla)^2 u \end{aligned} \tag{7}$$

After the spatial (Galerkin) discretization, the expression becomes

$$\begin{aligned}(\omega, \hat{u}^n) &= (\omega, u^n) - \frac{1}{3}\Delta t a(\omega, \nabla u) + \frac{1}{9}\Delta t^2(\omega, (a\nabla)^2 u^n) \\ (\omega, u^{n+1}) &= (\omega, u^n) - a\Delta t(\omega, \nabla u) + \frac{1}{2}\Delta t^2(\omega, (a\nabla)^2 \hat{u}^n)\end{aligned} \tag{8}$$

Integrating by parts the second derivative terms (neglecting integrals on the boundary)

$$\begin{aligned}(\omega, \hat{u}^n) &= (\omega, u^n) - \frac{1}{3}\Delta t a(\omega, \nabla u) - \frac{1}{9}\Delta t^2(\omega, (a\nabla)^2 u^n) \\ (\omega, \Delta u) &= -a\Delta t(\omega, \nabla u) + \frac{1}{2}\Delta t^2(\omega, (a\nabla)^2 \hat{u}^n)\end{aligned} \tag{9}$$

Using the matricial expressions:

$$\begin{aligned}\mathbf{M}\hat{u} &= \mathbf{M}u - \frac{1}{3}\Delta t a \mathbf{C}u^n - \frac{1}{9}\Delta t^2 a^2 \mathbf{K}u^n \\ \mathbf{M}\Delta u &= -\Delta t a \mathbf{C}u^n - \frac{1}{2}\Delta t^2 a^2 \mathbf{K}\hat{u}\end{aligned} \tag{10}$$

2.4 Methods behaviour

Third order Taylor Galerkin and two steps Taylor Galerkin include some numerical diffusion, losing accuracy. Leap frog method is able to better capture the transported profile, although some oscillations appear.

3 Unsteady 2D transport

The assignment asked for the implementation of a high order method for unsteady transport and the comparison between methods. The chosen one was the 4^{th} order explicit leap frog method.

3.1 4^{th} order explicit leap frog method

The discrete formulation of the method (time derivatives of source term not included) is:

$$\begin{aligned}\frac{u^{n+1} - u^{n-1}}{2\Delta t} &= u_t^n + \frac{1}{6}\Delta t^2 u_{ttt}^n \\ u_t &= s - \mathbf{a}\cdot\nabla u \\ u_{ttt} &= (\mathbf{a}\cdot\nabla)^2 \frac{u^{n+1} - u^n}{\Delta t}\end{aligned} \tag{11}$$

Developing the formulation and premultiplying for ω (Galerkin discretization) yields

$$\frac{u^{n+1}-u^{n-1}}{2\Delta t} = s^n - \mathbf{a}\cdot\nabla u^n + \frac{1}{6}\Delta t^2(\mathbf{a}\cdot\nabla)^2\frac{u^{n+1}-u^n}{\Delta t};$$
$$u^{n+1}-u^{n-1} = s^n - (2\Delta t)\mathbf{a}\cdot\nabla u^n + \frac{1}{3}\Delta t^2(\mathbf{a}\cdot\nabla)^2(u^{n+1}-u^n); \qquad (12)$$
$$(\omega, u^{n+1}-u^{n-1}) = (\omega, s^n - 2\Delta t\mathbf{a}\nabla u^n + \frac{1}{3}\Delta t^2(\mathbf{a}\cdot\nabla)^2(u^{n+1}-u^n))$$

Expressions (ω, u), $(a\omega, \nabla u)$, $(a\nabla\omega, \nabla u)$ and (ω, s) can be written as $\mathbf{M}$u, $\mathbf{C}$u , $\mathbf{K}$u and $\mathbf{v}_1$ respectively. $\mathbf{C}_o$u ($\mathbf{C}$u evaluated at the boundary) and $\mathbf{K}$ are obtained after integrating by parts the term $(\omega, \frac{1}{3}\Delta t^2(\mathbf{a}\cdot\nabla)^2(u^{n+1}-u^n))$. The matricial expression of the method then becomes:

$$(\mathbf{M} + \frac{1}{3}dt^2(\mathbf{K}-\mathbf{C}_o))u^{n+1} = -2\Delta t\mathbf{C}^t + \frac{1}{3}\mathbf{K}\Delta t^2 - \frac{1}{3}C_o\Delta t^2 + 2\Delta t v_1 \qquad (13)$$

See that it is not self-starting, as it needs two previous timesteps in order to compute the next one. For the first step we used the TG3 method. The 4^{th} leap frog method was tested using C = 0.5. Values close to or greater than 1 yielded results of the order of 10^{10} or greater (remember that the method in 1D is only stable for Courant number lower than 1).

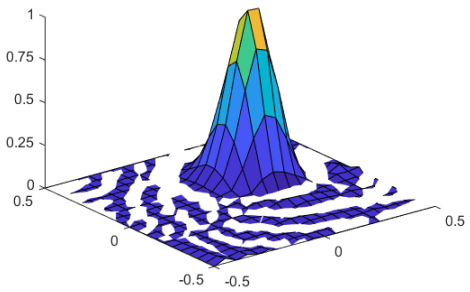

Figure 9: 4th order explicit Leap Frog result. White areas in the flat zone are due to oscillations introduced by the Galerkin spatial discretization

```
elseif meth == 7 % TG3 - 2S
    alpha = 1/9;
    A1 = M;
    B1 = -(dt/3)*C'- alpha*dt^2*(K - Co);
    f1 = (dt/3)*v1 + alpha*dt^2*(v2 - vo);
    A2 = M;
    B2 = -dt*C';
    C2 = - (dt^2/2)*(K-Co);
    f2 = dt*v1 - (dt^2/2)*(v2 - vo);
elseif meth == 8 % 4th order leap frog
    A = 3*M + (dt^2)*(K-Co);
    B = dt*((-6*C') + (K*dt) - (Co*dt));
    f = 6*dt*v1; % Source term
```

Figure 10: 4th order explicit Leap Frog code (matrices)

```
if meth == 7    % 2-step method
    btot = [B1*u(:,n)+ f1; bcod];
    aux =  U1\(L1\btot);
    u_m =  u(:,n) + aux(1:numnp);
    btot = [B2*u(:,n) + C2*u_m + f2; bcod];
    aux  = U2\(L2\btot);
    u(:,n+1) = u(:,n) + aux(1:numnp);
elseif meth == 8
    if n == 1% 4th order Leap Frog, not self-starting
        B1 = dt*(C - (dt/2)*K - Mo + (dt/2)*Co);
        A2 = M +(dt^2/6)*(K - Co);
        Atot2 = [A2 Acod';Acod zeros(nDir,nDir)];
        [L2,U2] = lu(Atot2);
        f1 = dt*((dt/2)*(v2 - vo) +v1);
        btot = [B1*u(:,n)+f1; bcod];
        aux  = U2\(L2\btot);
        u(:,n+1) = u(:,n) + aux(1:numnp);
    else
        btot = [B*u(:,n)+ 3*M*u(:,n-1)+f; bcod];
        aux  = U\(L\btot);
        u(:,n+1) =  aux(1:numnp);
    end
```

Figure 11: 4th order explicit Leap Frog code (loop over time steps)

3.2 Comparison between methods

Lumped mass matrix solutions are less accurate than their generalized counterparts which are computationally more expensive, as their scheme does not use a diagonal mass matrix. High order schemes account for more accurate results as expected. However, the Galerkin spatial discretization introduces numerical oscillations for this pure-convective problem. The least-squares solution avoids this issue, but introduces numerical dissipation, and thus, the maximum and minimum values are lowered.

Method	TG2	LW-FD	TG3	CN	CN-FD	CJ	TG3-2S	LF4
u_{max} (10^{-1})	9.357	6.285	9.309	9.703	6.377	9.276	9.304	9.708
u_{min} (10^{-2})	-2.730	-25.28	-3.136	-5.062	-29.12	-3.333	-3.129	-5.066

Table 1: Solution of the rotating cone problem a = 1, mesh of 20x20 elements and $\Delta t = 2\pi/200$. Courant = 0.62832.

4 Unsteady convection-diffusion

4.1 Propagation of a cosine profile

The following methods had to be tested and compared. Our mesh had 50 elements, so as to able to compare the accuracy when few elements are used.

- Second order Lax-Wendroff
- Third order explicit Taylor Galerkin
- Crank Nicholson finite element
- Fourth order implicit Taylor Galerkin

Regarding the time-step, we have to note that as we increase it, the Courant number decreases. We tried C = 1, C = 0.5 and C = 0.25. Lax-Wendroff method is unstable for C = 1 (as expected for $C^2 > 1/3$), and shows some inaccuracy in the final solution. Figure 12 shows the result for C = 0.5. Third order explicit Taylor Galerkin also shows some innacuracy, but is stable for all the values of the Courant number as expected. Crank Nicholson method shows less accuracy than Taylor-Galerkin (worse phase response and order of the method), with increasing error as we increase the Courant number (consistent with theory). Figure 14 shows the oscillations that arise in the solution for C = 0.5. Fourth order Taylor-Galerkin shows the best response for every value of Courant of the whole set of methods, and is also stable for every one of them. For C = 0.5 figure 15 shows this result.

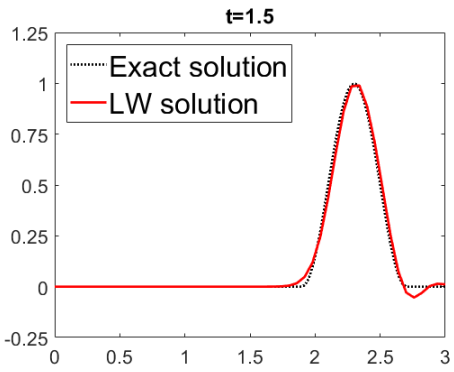

Figure 12: Cosine profile problem with C=0.5

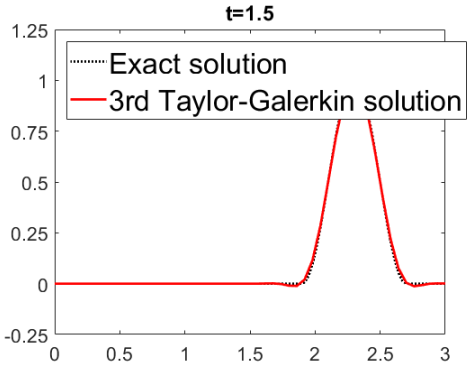

Figure 13: Cosine profile problem with C=0.5

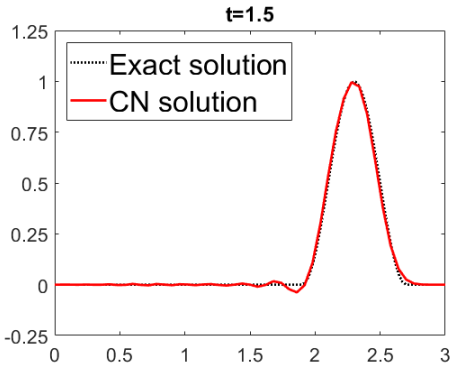

Figure 14: Cosine profile problem with C=0.5

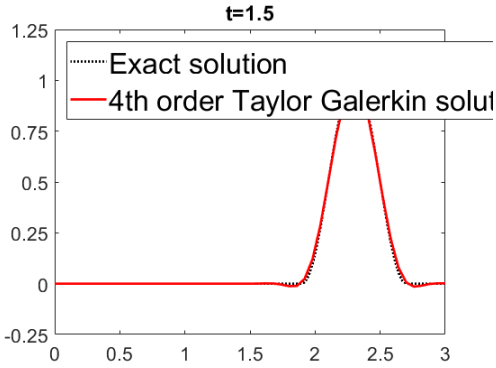

Figure 15: Cosine profile problem with C=0.5

4.2 Propagation of a steep front

The problem had to be solved using:

- Crank Nicholson + Galerkin
- Crank Nicholson + Least Squares
- Second order Lax Wendroff
- Two steps Lax Wendroff

Comparing both Crank Nicholson methods, we can see that the solution using Galerkin (see figure 16) presents oscillations due to the spatial discretization method. If we do this discretization via the Least Squares method (see figure 17) this oscillations dissappear except at the front.
Lax Wendroff scheme (figure 18)shows also numerical innacuracy, as it is a no-monotone method. We could better its behaviour by adding a non-linear diffusive term that would allow the metho to fully represent the steep front solution. If we use a second-step Lax Wendroff with Galerkin discretization (figure 19) with the same Courant number, the solution is completely unstable. For stable Courant numbers (figure 20 for C = 0.24) we also find important oscillations of the solution.

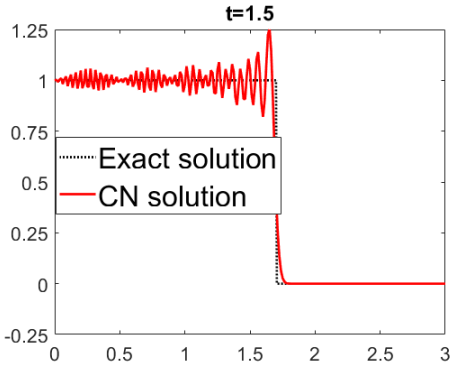

Figure 16: Steep front problem solves with Crank Nicholson with C=0.5

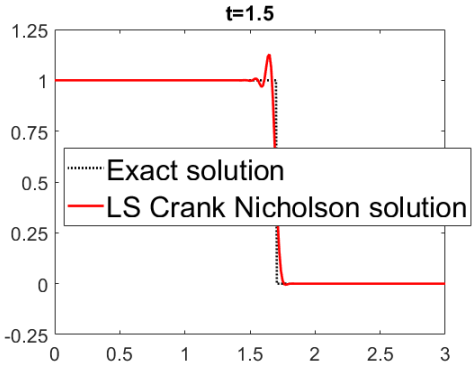

Figure 17: Steep front problem solved with Leas squares Crank Nicholson with C=0.5

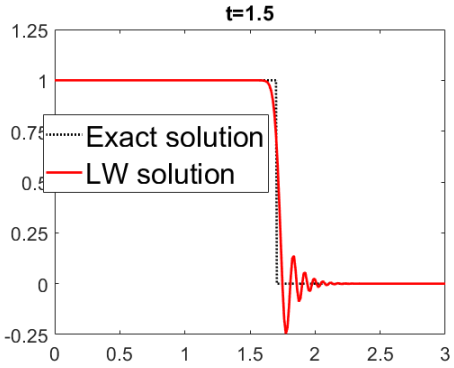

Figure 18: Steep front problem solved with Lax-Wendroff with C=0.5

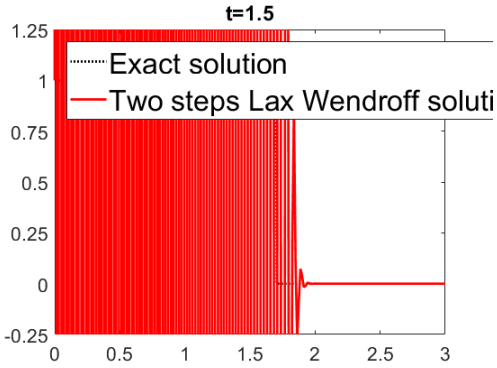

Figure 19: Steep front problem solved with 2 steps Lax-Wendroff with C=0.5

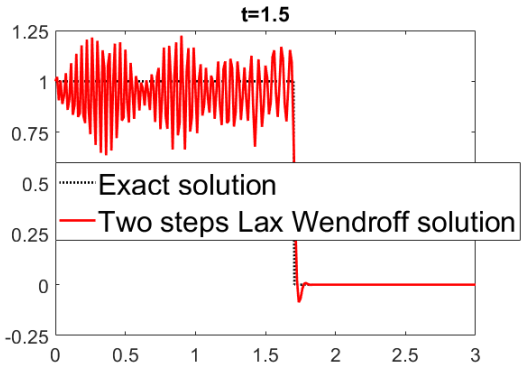

Figure 20: Steep front problem with 2 steps Lax-Wendroff with C=0.24

```
case 5 % Third order Taylor-Galerkin + Galerkin
    A = M + a^2*dt^2*K/6;
    B = -a*C*dt - 0.5*dt^2*a^2*K;
    methodName = '3rd Taylor-Galerkin';
    A2 = 0;
    B2 = 0;
    C2 = 0;
case 6 % Leap frog + Galerkin
    A = M;
    B = -2*a*C*dt;
    methodName = 'Leap frog';
    A2 = 0;
    B2 = 0;
    C2 = 0;
case 7 % Two steps Galerkin
    A = M;
    B = M - (dt*a/3) * C - ((dt*a)^2/9) * K;
    A2 = M;
    B2 = -dt*a*C;
    C2 = - 0.5*((dt*a)^2)*K;
    methodName = 'Two steps Taylor Galerkin';
```

Figure 21: Coded systems for the convection problem

```
case 8 % Least-squares Cranc Nicholson (page 121)
    A = M + 0.5*a*dt*C' + 0.5*a*dt*C + K*(dt*a/2)^2;
    B = -a*dt*C - K*0.5*(a*dt)^2;
    A2 = 0;
    B2 = 0;
    C2 = 0;
    methodName = 'LS Crank Nicholson';
case 9 % Two steps Laxx Wendrof
    A2 = M;
    B2 = -a*dt*C; % Second step
    A = M;
    B = M -a*0.5*dt*C;
    C2 = 0;% First step
    methodName = 'Two steps Lax Wendroff';
case 10 % 4th order TAylor Galerkin
    A = M;
    B = M - (dt*a/3) * C - ((dt*a)^2/12) * K;
    A2 = M;
    B2 = -dt*a*C;
    C2 = - 0.5*((dt*a)^2)*K;
    methodName = '4th order Taylor Galerkin';
```

Figure 22: Coded systems for the convection problem

```
if method == 6
    % Third order Galerkin for first step
    A1 = M + a^2*dt^2*K/6;
    B1 = -a*C*dt - 0.5*dt^2*a^2*K;
    A1 = A1(ind_unk,ind_unk);
    B1 = B1(ind_unk,ind_unk);
    Duold = A1\(B1*u(ind_unk,1) + f);
    u(ind_unk,2) = u(ind_unk,1) + Duold; Du;
    for n = 2:nStep
        u(ind_unk,n+1) = A\(B*(u(ind_unk,n)) + f +A*u(ind_unk,n-1));
    end
elseif method == 7 || method == 10
    for n = 1:nStep
        ut = A\(B*u(ind_unk,n));
        Du = A2\(B2*u(ind_unk,n) + C2*ut + f) ;
        u(ind_unk,n+1) = u(ind_unk,n) + Du;
    end
elseif method == 9
    for n = 1:nStep
        uh = A\(B*u(ind_unk,n));
        Du = A2\(B2*uh + 0.5*f );
        u(ind_unk,n+1) = u(ind_unk,n) + Du;
    end
```

Figure 23: Time integration loop for the different methods coded

4.3 Gaussian hill

Changes in diffusion coefficient had to be tested for the Gausian hill problem. It can be seen that as ν decreases, the problem becomes a pure convection case, and the initial condition only gets transproted with the convective velocity. The discretization were done using R33 and GLS.

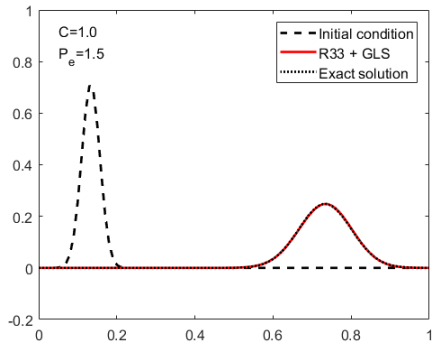

Figure 24: Gaussian hill with $\nu = 3.310^{-3}$

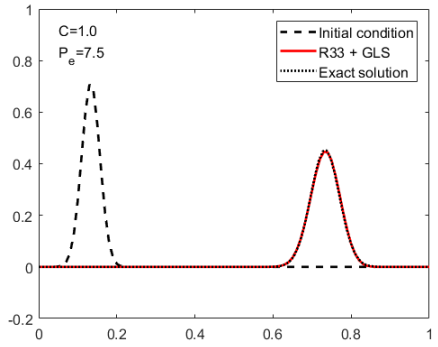

Figure 25: Gaussian hill with $\nu = 6.710^{-4}$

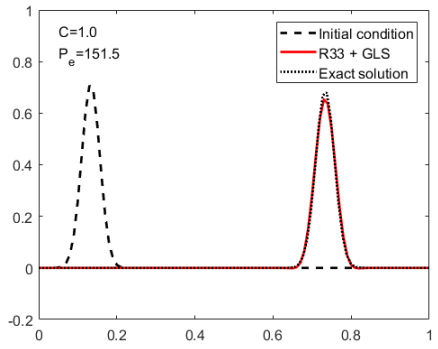

Figure 26: Gaussian hill with $\nu = 3.310^{-5}$