Finite Element in Fluids

Homework I

Zahra Rajestari

Contents

Introduction

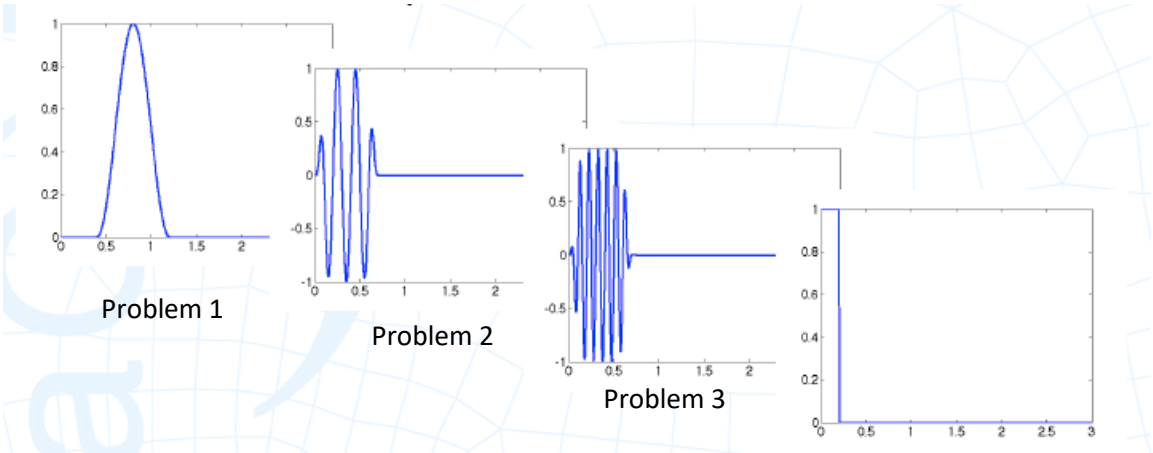
This assignment focuses on studying and implementing different numerical methods for one-dimensional and two-dimensional problems of different types. In the first section we study the behavior of Lax-Wendroff, Lax-Wendroff with lumped mass matrix, Crank-Nicolson and Crank-Nicolson with lumped mass matrix and Leap-frog and two step Taylor Galerkin methods for four different types of one-dimensional problems. In the next section we move to two-dimensional problems and study the behavior of the methods for both steady and unsteady problems. At the end we make some comparisons between different methods implemented to see the pros and cons of each method for each problem type.

1D Convection

Studying the behavior of different methods

The aim is to solve the one dimensional equation $u_t + a\, u_x = 0$ using different methods. The methods used for time discretization are Lax-Wendroff, Lax-Wendroff with lumped mass matrix, Crank-Nicolson and Crank-Nicolson with lumped mass matrix and for all of them the method of space discretization is the Galerkin method. These methods are already implemented and the behavior of each of them is studied in the following.

There are four problems available each of which are evaluated using different methods.



Problem 1

Problem 2

Problem 3

Problem 4

In the following, the Lax_wendroff method has been studied for all problems for two courant numbers of values 0.5 and 0.6.

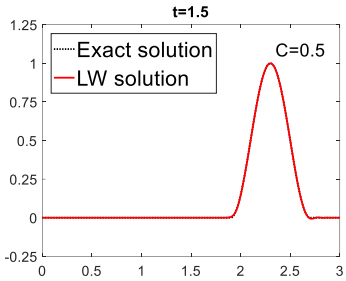


Figure 1 Lax-Wendroff, time-step=240, problem 1

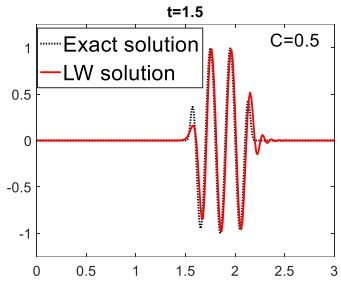


Figure 2 Lax-Wendroff, time-step =240, problem 2

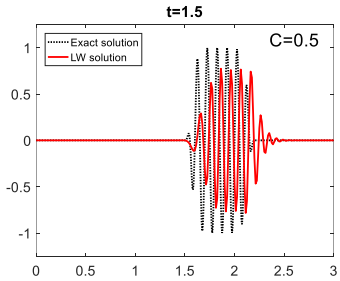


Figure 3 Lax-Wendroff, time-step =240, problem 3

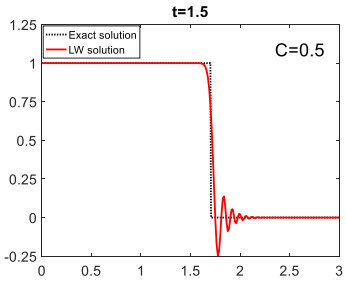


Figure 4 Lax-Wendroff, time-step =240, problem 4

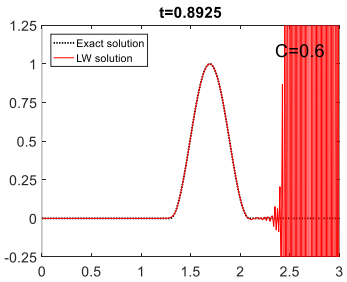


Figure 5 Lax-Wendroff, time-step=200, problem 1

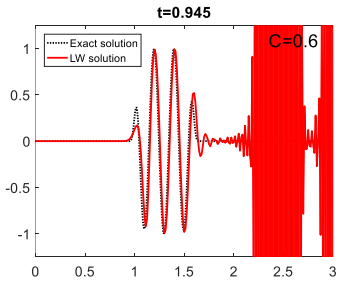


Figure 6 Lax-Wendroff, dt=200, problem 2

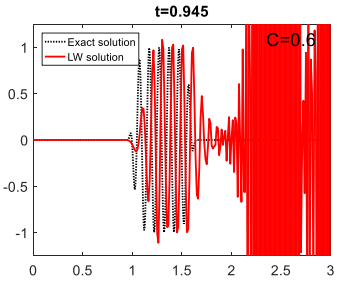


Figure 7 Lax-Wendroff, dt=200, problem 3

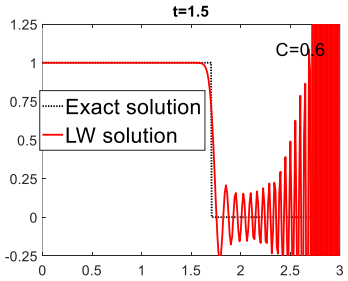


Figure 8 Lax-Wendroff, dt=200, problem 4

For Lax-Wendroff method the stability region lies in $C^2 < \frac{1}{3}$ and as we see in Figure 5, Figure 6, Figure 7 and Figure 8, the method starts to have oscillatory behavior just as this limit is passed (for C=0.6). However, for values lying in the stable region, Figure 1, Figure 2, Figure 3 and Figure 4, it is behaving acceptably.

In the following, we shall study the behavior of Lax-Wendroff with lumped matrix for different problems.

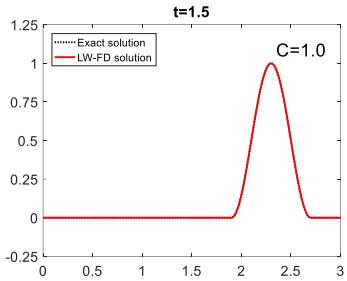


Figure 9 Lax-Wendroff with lumped mass matrix, time-step=120, problem 1

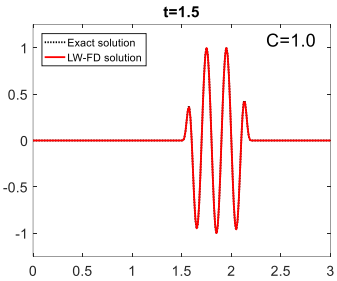


Figure 10 Lax-Wendroff with lumped mass matrix, time-step=120, problem 2

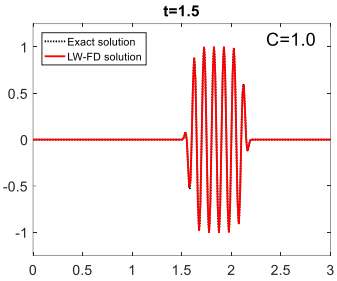


Figure 11 Lax-Wendroff with lumped mass matrix, time-step=120, problem 3

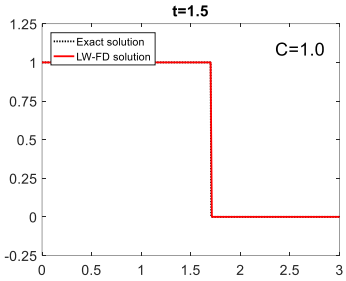


Figure 12 Lax-Wendroff with lumped mass matrix, time-step=120, problem 4

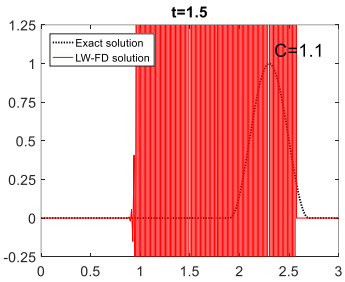


Figure 13 Lax-Wendroff with lumped mass matrix, time-step=110, problem 1

When it comes to using Lax-Wendroff with lumped mass matrix, the stability region changes to C<1, which is in agreement with the results shown in Figure 9 and Figure 13.

The Crank_nicolson method is unconditionally stable and shows good behavior for any Courant:

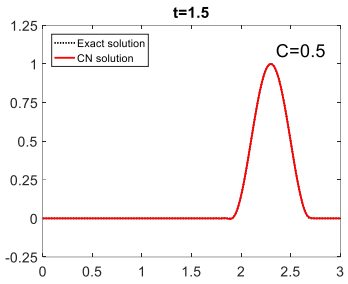


Figure 14 Crank-Nicolson, problem 1

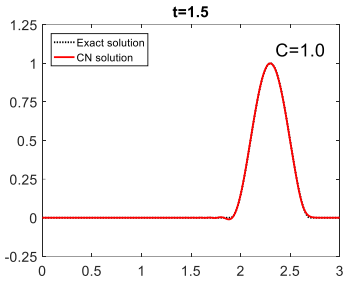


Figure 15 Crank-Nicolson, time-step=120, problem 1

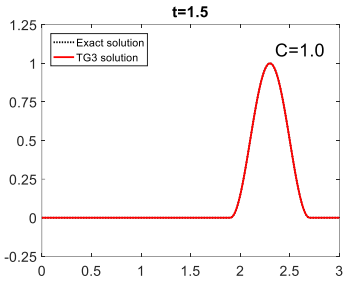


Figure 16 TG3, time-step=120, problem 1

Implementation of Leap-Frog method

After studying the behavior of different methods for different problems, we implement the leap-frog method for time discretization which is based on the following formula:

$$\frac{u^{n+1}-u^{n-1}}{2\triangle t} = u_t^n = s^n - \boldsymbol{a}\cdot\boldsymbol{\nabla}u^n$$

In these examples we have zero source term and Dirichlet boundary condition on the inflow.

Galerkin method is used for space-discretization. Therefore, a weight function is multiplied by the whole leap-frog equation and after integration by parts and substituting the weight and the solution at nodes by the shape functions, the matrices used in Galerkin method can be found in terms of mass, convection and stiffness matrices.

Since A is the matrix multiplying Δu and B is the matrix multiplying the solution of the previous time step, matrices "A" and "B" which are used in the code to solve the system can be written in function *"System.m"* as the following:

```
case 5 % Leap Frog
    A = M;
    B = -2*a*dt*C;
    methodName = 'LF';
```

where M and C are the mass and convection matrices defined as:

$$M_{ab} = \int_\Omega N_a N_b d\Omega \qquad C_{ab} = \int_\Omega N_a(\boldsymbol{a}\cdot\boldsymbol{\nabla}N_b)d\Omega$$

The corresponding changes in MATLAB code "main.m" should also be made as this new method is added. It should be mentioned that the leap-frog method is a non-self-starting method so another method should be used for the first step:

```
%1st iteration with LAx_Wendroff
method=1;
[A0,B0,methodName0] = System(method,M,K,C,a,dt);
A0 = A0(ind_unk,ind_unk);
B0 = B0(ind_unk,ind_unk);
Du_n0 = A0\(B0*u(ind_unk,1) + f);
u(ind_unk,2) = u(ind_unk,1) + Du_n0;
method=5;
for n = 2:nStep
    Du_n = -Du_n0 + (A\(B*u(ind_unk,n) + f));
    u(ind_unk,n+1) = u(ind_unk,n) + Du_n;
    Du_n0 = Du_n;
end
```

The results can then be shown as the following for each problem:

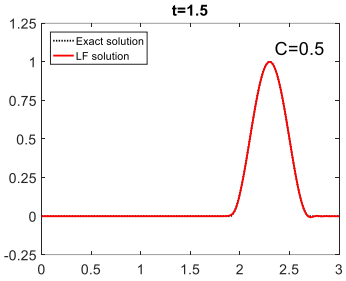


Figure 17 Leap-Frog, time-step=240, problem 1

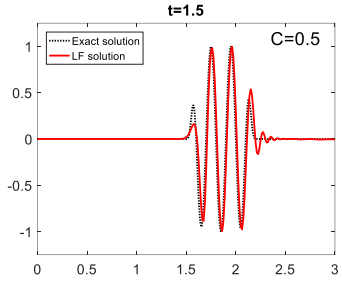


Figure 18 Leap-Frog, time-step=240, problem 2

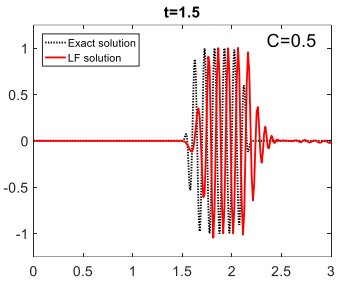


Figure 19 Leap-Frog, time-step=240, problem 3

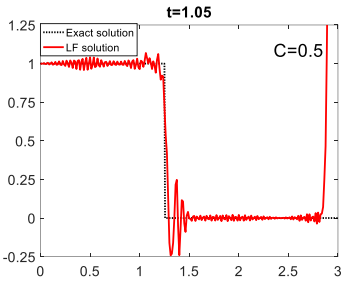


Figure 20 Leap-Frog, time-step=240, problem 4

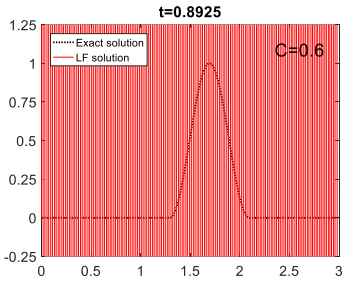


Figure 21 Leap-Frog, time-step=200, problem 1

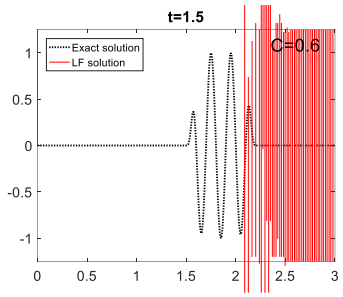


Figure 22 Leap-Frog, time-step=200, problem 2

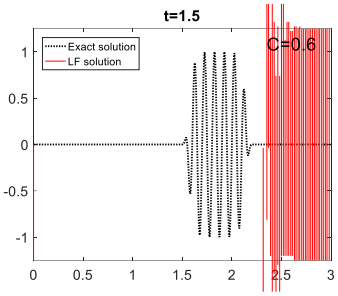


Figure 23 Leap-Frog, time-step=200, problem 3

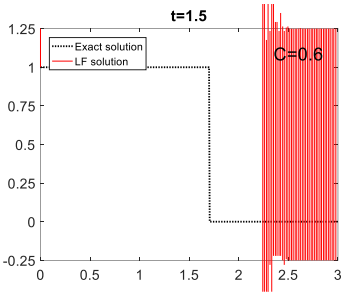


Figure 24 Leap-Frog, time-step=200, problem 4

The stability region for Leap-frog method is $C^2 < \frac{1}{3}$ and as it can be seen in Figure 17, Figure 18, Figure 19 and Figure 20 for all the first three problems this method is showing acceptable results. It should be mentioned that for the last problem, problem 4, the results of leap frog method do not seem to be stable.

In Figure 21, Figure 22, Figure 23 and Figure 24, it can be seen that as we pass the stability limit the results start having oscillatory behavior.

Implementation of the TG3-2S

As the last part of this assignment, the two-step Taylor-Galerkin method is implemented using α=1/9 based on the following formulation:

$$\tilde{u}^n = u^n + \frac{1}{3}\triangle t\, u_t^n + \alpha\, \triangle t^2\, u_{tt}^n,$$

$$u^{n+1} = u^n + \triangle t\, u_t^n + \frac{1}{2}\triangle t^2\, \tilde{u}_{tt}^n,$$

Before implementing the method in MATLAB, we shall first substitute the terms with time derivative according to the main equation and also do the integration by parts to end up with the formulation including mass, convection and stiffness matrices. At the end, for each step, matrices A and B can be defined in the corresponding codes as the following:

```
alpha = 1/9;
K = K(ind_unk,ind_unk);
C = C(ind_unk,ind_unk);
for n = 1:nStep
    Du_hat = A\((-(1/3)*a*dt*C - alpha*a^2*dt^2*K)*u(ind_unk,n) + f);
    u_hat(ind_unk,n) = u(ind_unk,n) + Du_hat;
    Du = A\((-a*dt*C)*u(ind_unk,n) + (-.5*a^2*dt^2*K)*u_hat(ind_unk,n) + f);
    u(ind_unk,n+1) = u(ind_unk,n) + Du;
end
```

where *u_hat* corresponds to $\tilde{u}$ which is the solution of the first step and A and B correspond to the mass and convection matrices respectively.

The results for each problem can be found below:

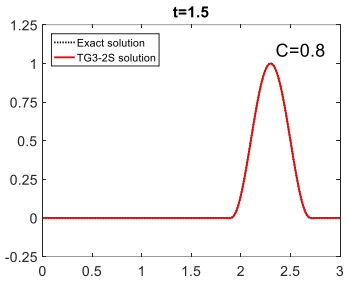


Figure 25 TG3-2S, time-step=150, problem 1

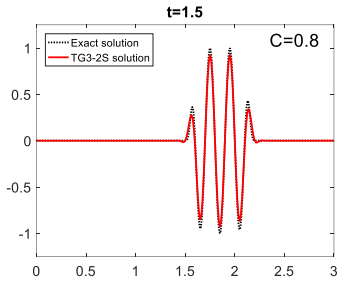


Figure 26 TG3-2S, time-step=150, problem 2

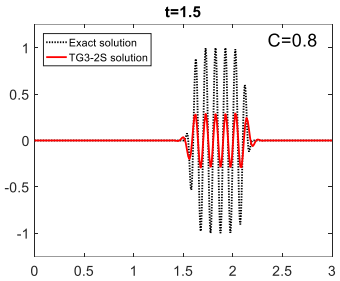


Figure 27 TG3-2S, time-step=150, problem 3

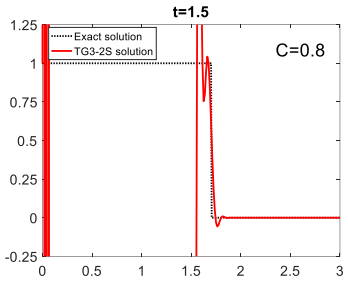


Figure 28 TG3-2S, time-step=150, problem 4

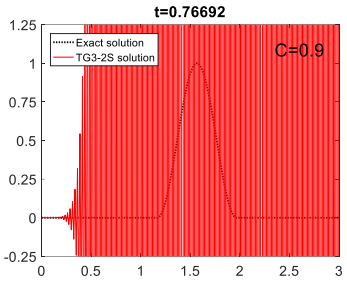


Figure 29 TG3-2S, time-step=133, problem 1

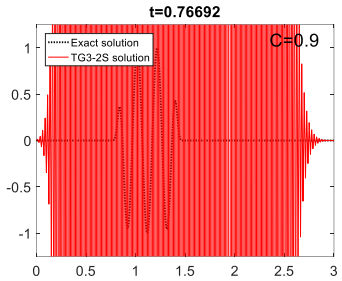


Figure 30 TG3-2S, time-step=133, problem 2

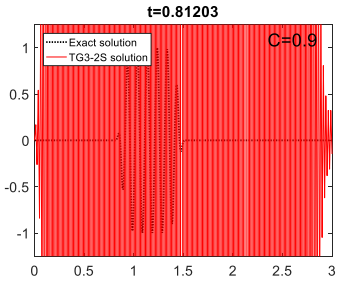


Figure 31 TG3-2S, time-step=133, problem 3

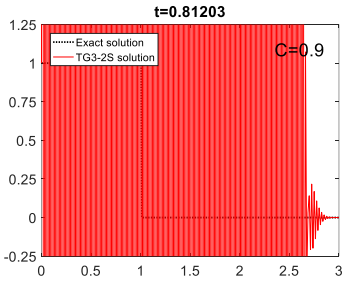


Figure 32 TG3-2S, time-step=133, problem 4

The stability region for two-step Taylor-Galerkin method is $C^2 < \frac{3}{4}$. Figure 25 and Figure 26 show almost good results however Figure 27 and Figure 28 which correspond to problem 3 and 4 do not show great accuracy although the Courant number lies in the stability region. Then, as the stability region is crossed which means for Courant numbers of 0.86 and higher, the solution starts having oscillation as shown in Figure 29, Figure 30, Figure 31 and Figure 32.

Two Dimensional Steady Problems

Implementation of GLS method

The goal is to implement the GLS method for two-dimensional convection-diffusion problem. To do so, we have to define the stiffness matrix and the fource vector based on the formula that we have for this method in the MATLAB function "FEM_system.m" as the following:

```
% GLS
aux = N_ig*Xe;
g_ig = ReactionTerm(aux);
Ke = Ke + (nu*(Nx'*Nx+Ny'*Ny) + N_ig'*(ax*Nx+ay*Ny)+N_ig'*g_ig*N_ig + ...
    tau*((((ax*Nx+ay*Ny)-nu*(Nxx+Nyy)+g_ig*N_ig)'*((ax*Nx+ay*Ny)-nu*(Nxx+Nyy)+g_ig*N_ig)))...
    *dvolu;

f_ig = SourceTerm(aux);
fe = fe + (N_ig+tau*((ax*Nx+ay*Ny)-nu*(Nxx+Nyy)))'*(f_ig*dvolu);
```

It should be mention that the following part should also be added to this code in order to define τ which is the stabilization parameter:

```
%GLS
Pe = a*h/(2*nu);
tau_p = h*(1 + 9/Pe^2)^(-1/2)/(2*a);
disp(strcat('Recommended stabilization parameter = ',num2str(tau_p)));
tau = cinput('Stabilization parameter',tau_p);
if isempty(tau)
    tau = tau_p;
end
```

According to the formulation of GLS method, it is obvious that we need the second derivative of the shape functions with respect to x and y however only the first derivative has been included in the code. Therefore, the function "ShapeFunc.m" should be modified for all cases. In the case of quadrilateral elements we have:

```
if elem == 0
    if p == 1
        N    = [(1-xi).*(1-eta)/4, (1+xi).*(1-eta)/4, (1+xi).*(1+eta)/4, (1-xi).*(1+eta)/4];
        Nxi  = [(eta-1)/4, (1-eta)/4, (1+eta)/4, -(1+eta)/4];
        Neta = [(xi-1)/4, -(1+xi)/4,   (1+xi)/4,  (1-xi)/4 ];
        N2xi = zeros(4,4);
        N2eta = zeros(4,4);
```

Second derivative of the shape functions with respect to the natural coordinates for polynomial degree one.

Then, for quadrilateral element, for polynomial of degree two we have the shape functions as a function of the natural coordinates defined as:

```
elseif p == 2
    N    = [xi.*(xi-1).*eta.*(eta-1)/4, xi.*(xi+1).*eta.*(eta-1)/4, ...
        xi.*(xi+1).*eta.*(eta+1)/4, xi.*(xi-1).*eta.*(eta+1)/4, ...
        (1-xi.^2).*eta.*(eta-1)/2,  xi.*(xi+1).*(1-eta.^2)/2,   ...
        (1-xi.^2).*eta.*(eta+1)/2,  xi.*(xi-1).*(1-eta.^2)/2,   ...
        (1-xi.^2).*(1-eta.^2)];
    Nxi  = [(xi-1/2).*eta.*(eta-1)/2,   (xi+1/2).*eta.*(eta-1)/2, ...
        (xi+1/2).*eta.*(eta+1)/2,   (xi-1/2).*eta.*(eta+1)/2, ...
        -xi.*eta.*(eta-1),          (xi+1/2).*(1-eta.^2),   ...
        -xi.*eta.*(eta+1),          (xi-1/2).*(1-eta.^2),   ...
        -2*xi.*(1-eta.^2)];
    Neta = [xi.*(xi-1).*(eta-1/2)/2,    xi.*(xi+1).*(eta-1/2)/2, ...
        xi.*(xi+1).*(eta+1/2)/2,    xi.*(xi-1).*(eta+1/2)/2, ...
        (1-xi.^2).*(eta-1/2),       xi.*(xi+1).*(-eta),   ...
        (1-xi.^2).*(eta+1/2),       xi.*(xi-1).*(-eta),   ...
        (1-xi.^2).*(-2*eta)];
    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*(1-eta.^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*(1-xi.^2)];
else
    error('not available interpolation degree')
end
```

The changes are made as the following in case of triangular elements:

```
elseif elem == 1
    if p == 1
        N = [1-xi-eta, xi, eta];
        Nxi = [-ones(size(xi)), ones(size(xi)), zeros(size(xi))];
        Neta = [-ones(size(xi)), zeros(size(xi)), ones(size(xi))];
        N2xi = zeros(size(xi));
        N2eta = zeros(size(xi));
```

Second derivative of the shape functions with respect to the natural coordinates for triangular element.

```
elseif p == 2
    N = [ xi*(2*xi-1), eta*(2*eta-1), (1-xi-eta)*(2*(1-xi-eta)-1),...
        4*xi*eta, 4*eta*(1-xi-eta), 4*(1-xi-eta)*xi];
    Nxi = [ 4*xi - 1, 0, 4*eta + 4*xi - 3, 4*eta, -4*eta, 4 - 8*xi - 4*eta];
    Neta = [ 0, 4*eta - 1, 4*eta + 4*xi - 3, 4*xi, 4 - 4*xi - 8*eta, -4*xi];
    N2xi = [ 4*ones(size(xi)), 0*ones(size(xi)), 4*ones(size(xi)), 0*ones(size(xi)),...
             0*ones(size(xi)), -8*ones(size(xi))];
    N2eta = [  0*ones(size(xi)), 4*ones(size(xi)), 4*ones(size(xi)), 0*ones(size(xi)),...
              -8*ones(size(xi)), 0*ones(size(xi))];
else
    error('not available interpolation degree')
end
```

The boundary conditions should also be changed in terms of the dimension since we are dealing with 2 dimensional problem. The changes are made in “main.m” as the following:

```
%%%%%%%%%%%%Neuman
%nodesDir0 = [nodes_x0( X(nodes_x0,2) <= 0.2 ); nodes_y0]; %% B.C default

%%%%%%%%%%%%Dirichlet
nodesDir0 = [nodes_x0( X(nodes_x0,2) <= 0.2 ); nodes_y0; nodes_x1; nodes_y1 ];
```

```
% BOUNDARY CONDITIONS
% Boundary conditions are imposed using Lagrange multipliers
if p == 1
    nodes_y0 = [1:nx+1]';                          % Nodes on the boundary y=0
    nodes_x1 = [2*(nx+1):nx+1:(ny+1)*(nx+1)]' ;    % Nodes on the boundary x=1
    nodes_y1 = [ny*(nx+1)+nx:-1:ny*(nx+1)+1]' ;    % Nodes on the boundary y=1
    nodes_x0 = [(ny-1)*(nx+1)+1:-(nx+1):nx+2]';    % Nodes on the boundary x=0
elseif p == 2
    nodes_y0 = [1:2*nx+1]';                          % Nodes on the boundary y=0
    nodes_x1 = [2*(2*nx+1):2*nx+1:(2*ny+1)*(2*nx+1)]' ;    % Nodes on the boundary x=1
    nodes_y1 = [2*ny*(2*nx+1)+2*nx:-1:2*ny*(2*nx+1)+1]' ;    % Nodes on the boundary y=1
    nodes_x0 = [(2*ny-1)*(2*nx+1)+1:-(2*nx+1):2*nx+2]';    % Nodes on the boundary x=0
end
```

In order to add the reaction term to the solution, a function called “ReactionTerm.m” is defined and the corresponding terms related to the reaction are added to the equations as the variable *“g-ig”*.

Results and Discussion for two-dimensional steady equation

The results for GLS method are shown below for two different types of problems:

Type 1: Convection-reaction dominated case with ||a||=0.5, nu=10^-2, sigma=1

Type 2: Reaction dominated case with ||a||=0.001, nu=10^-2, sigma=1

It should be mentioned that the figures taken for both cases of quadrilateral and triangular elements of 1st and 2nd order elements are the same since all kinds work perfectly for element numbers such as 50 or 100.

The following figures are results of element numbers of 50 and shows a comparison between the two types of the problems.

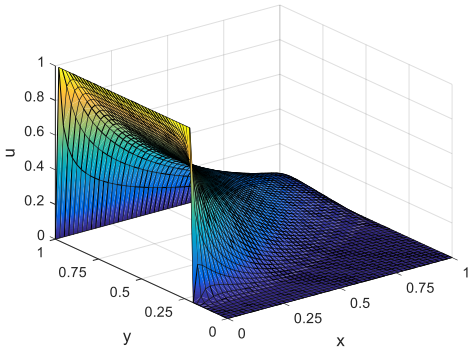


Figure 33 GLS, Quadrilateral of 1st order, problem type 1

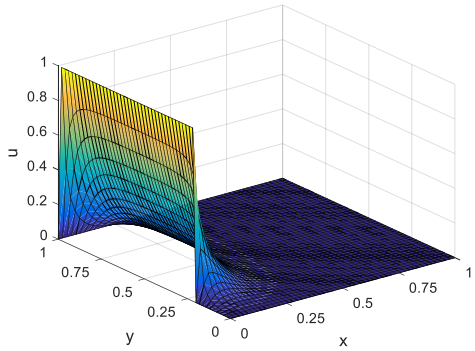


Figure 34 GLS, Quadrilateral of 1st order, problem type 2

Two Dimensional Unsteady problem

The goal of this part of the assignment is to implement a higher order method to solve two-dimensional unsteady problem. The method chosen is the forth order two step method with the following formula:

$$\tilde{u} = u^n + \frac{1}{3}\Delta t\, u_t^n + \frac{1}{12}\Delta t^2\, u_{tt}^n,$$
$$u^{n+1} = u^n + \Delta t\, u_t^n + \frac{1}{2}\Delta t^2\, \tilde{u}_{tt}.$$

After substituting the time derivatives of u based on the main equation that we have and applying Galerkin weighted residual method we will end up with the formulation containing mass, convection and stiffness matrices that can be applied to the program as the following:

```
elseif meth == 8
    % 1st step implementation with delta u = containing u_tilde
    A1 = M;
    B1 = (1/3)*C*dt - (1/12)*K*dt^2 - (1/3)*Mo*dt + (1/12)*Co*dt^2;
    f1 = (1/3)*v1*dt + (1/12)*v2*dt^2 - (1/12)*vo*dt^2;
    % 2nd step implementation with dealta u = containing u_n+1
    A2 = M;
    B2 = C*dt - Mo*dt; %containing u
    f2 = v1*dt + 0.5*v2*dt^2 - 0.5*vo*dt^2; %containing s
    C2 = -0.5*K*dt^2 + 0.5*Co*dt^2; %containing u_tilde
```

in which the parameters are defined as the following:

$C = a\nabla w\, u$ $\qquad$ $K = a^2\nabla w\, \nabla u$
$v_1 = ws$ $\qquad$ $v_2 = a\, \nabla w\, s$
$C_0 = a.n\, w.a\, \nabla u$ $\qquad$ $V_0 = a.n.ws$
$M_0 = a.n.wu$

Other methods to solve 2D unsteady were already implemented. The results of applying two step 4th order method are obtained for three different velocity fields defined as below:

Field 1	Field 2	Field 3
$v(x,y) = (-y,x)$	$v(x,y) = (1,0)$	$v(x,y) = (\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}})$

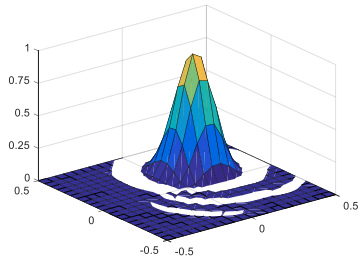


Figure 35 2S-4th order method for velocity field 1

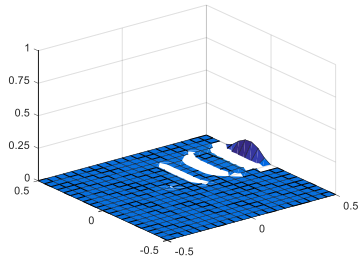


Figure 36 2S-4th order method for velocity field 2

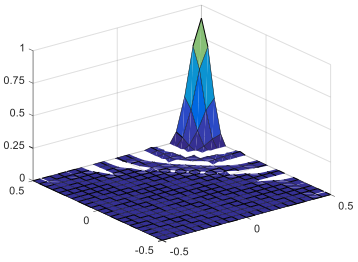


Figure 37 2S-4th order method for velocity field 3

The difference between the three velocity fields is the way the problem is behaving and the equation is improving through time. The figures shown above are related to the last moment of a whole revolution.

Propagation of cosine profile

Second order Lax-Wendroff finite element method

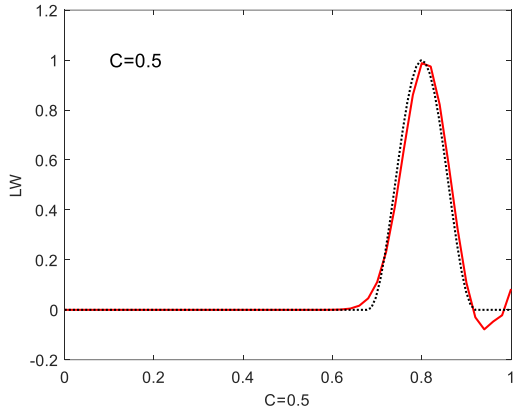


Figure 38 2nd order Lax-wendroff for Courant=0.5

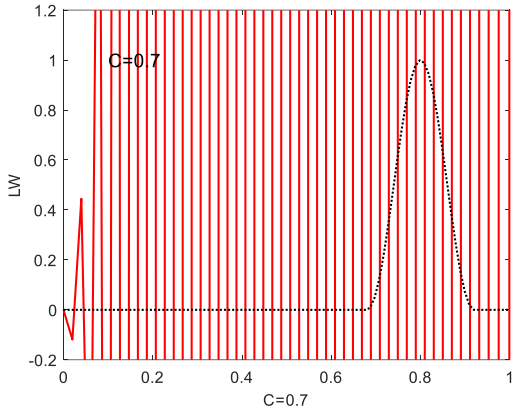


Figure 39 2nd order Lax-wendroff for Courant=0.7

As we know the stability criterion for Lax-Wendroff is $C^2 < \frac{1}{3}$. Figure 38 and Figure 39 confirm this fact that for courant numbers larger than 0.57 the method starts to have oscillations.

Third-order explicit Taylor-Galerkin method

The stability criterion for this method is $C^2 < 1$. In the following we see if the results obtained from the program confirm this fact.

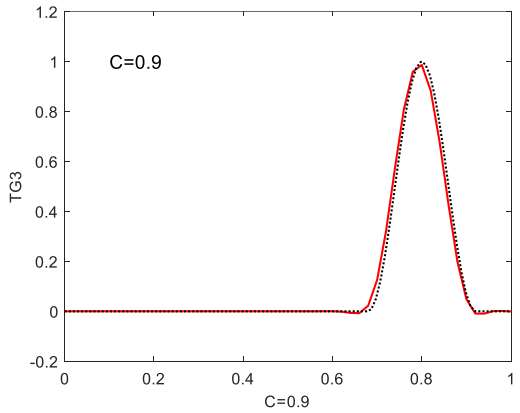


Figure 40 TG3 for Courant=0.9

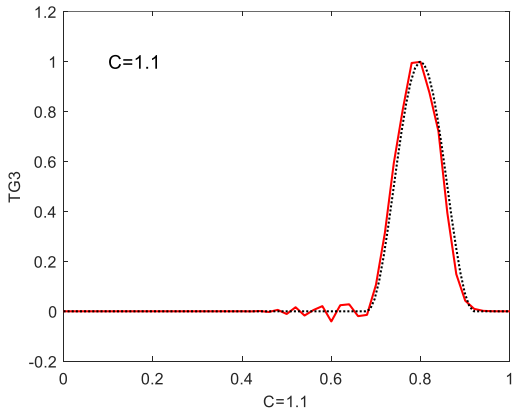


Figure 41 TG3 for Courant=1.1

As we can see in the figures for Courant number of C=1.1 which is crossing the stability region the method starts to show oscillations. Now if we increase the courant number we will see more accuracy as below:

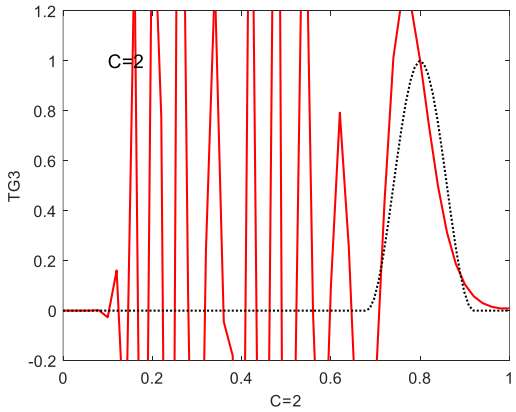


Figure 42 TG3 for Courant=2

Second-order Crank-Nicolson finite element method

Second order crank Nicolson is unconditionally stable. So it is expected that for all values of Courant number the result show good stability.

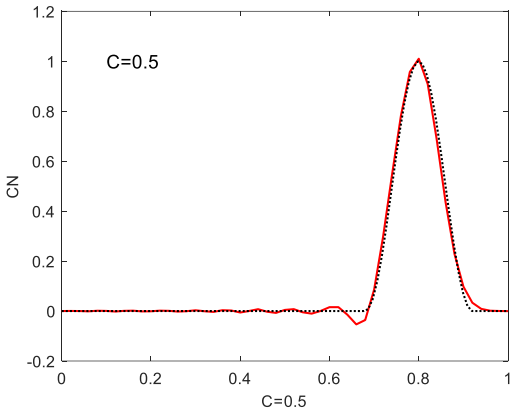


Figure 43 Crank-Nicolson for Courant=0.5

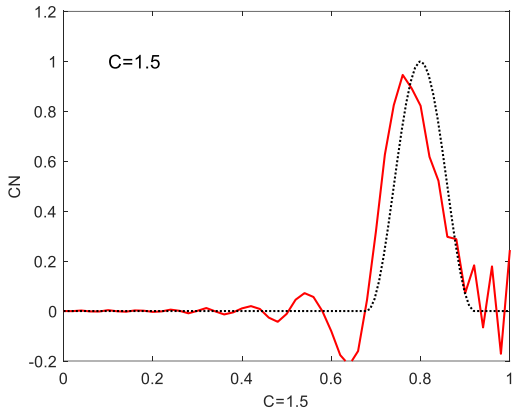


Figure 44 Crank Nicolson for Courant=1.5

As it can be seen the results show stability. However as the courant number is increased the accuracy decreases. We can obtain higher accuracy by decreasing the time step.

Fourth-order implicit Taylor-Galerkin method

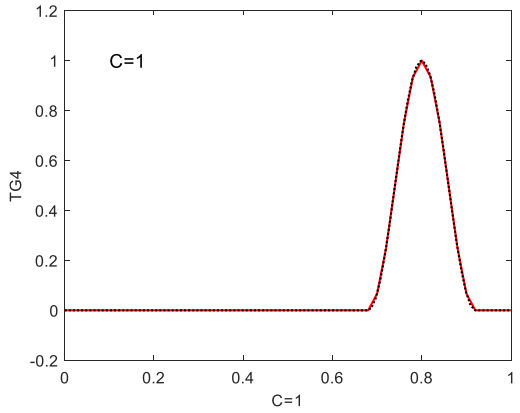


Figure 45 4th order TG for Courant=1

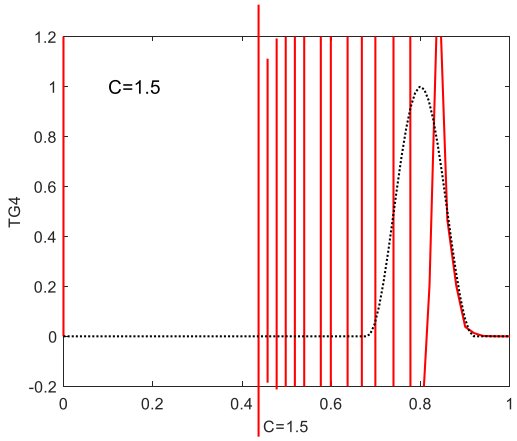


Figure 46 4th order TG for Courant=1.5

The forth order TG implicit method shows great accuracy for Courant number equal to 1 as it can be seen from the figure. However, when it comes to larger courant numbers it is seen that it starts to have oscillations and not giving good results.

To sum up the results of cosine propagation profile, it can be seen that for smaller values of Courant number the 4^{th} order implicit taylor galerkin shows good results. And for larger Courant numbers the Crank Nicolson behaves better than TG3. Among all, the Lax-Wendroff of second order appears to have less accuracy and less acceptable results compared to other methods and the exact solution.

Propagation of Steep front

In order to implement this part we shall only change the condition for u0 as the following based on the definition of the problem:

```
for i=1:numnp
    dist = xnode(i)-x0;
    if xnode(i) <= x0
        u(i,1) = 1;
    end
end
```

Then, the crank-nicolson scheme in time and Galerkin in space has been studied to check the accuracy of the results. The results are shown below. As it can be seen in Figure 47, the method for this problem shows high inaccuracy and oscillations but it is still stable while the solution is not going to infinite by the passing of time. When we study Crank-Nicolson in time discretization and GLS in space discretization. The result is shown below.

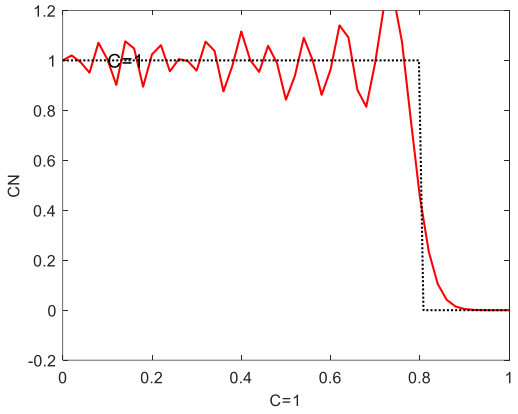


Figure 47 CN with Courant=1 for Steep Front problem

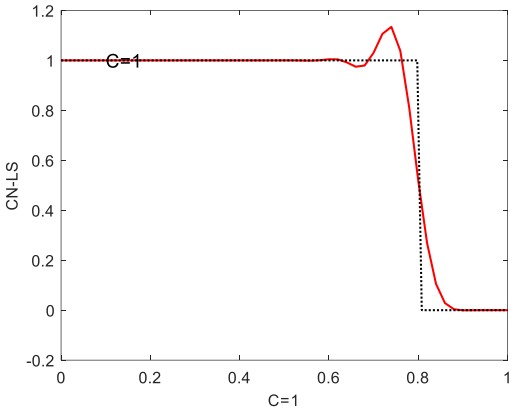


Figure 48 VN in time GLS in space for Courant=1

As the results show we have great accuracy and stability for this method. So when it comes to using CN in time discretization it is better to use GLS for space rather than Galerkin method.

The Gaussian Hill

Study on Viscosity

This study has been done for different values of diffusion coefficient using method of time discretization R22 and GLS for space discretization. As the results of Figure 49 and Figure 50 show, when increasing the value of diffusion coefficient the amplitude of the solution decreases. This is in accordance with the theory since increasing the diffusion yields to higher energy dissipation and smaller amplitude.

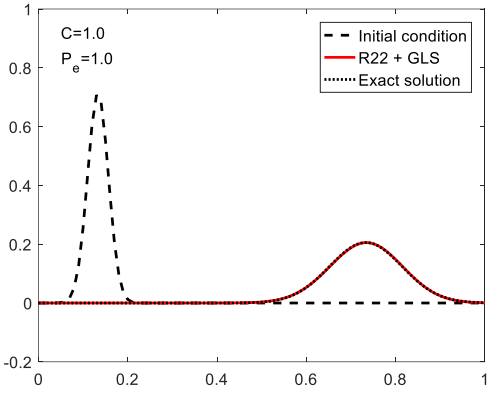


Figure 49 nu=0.005

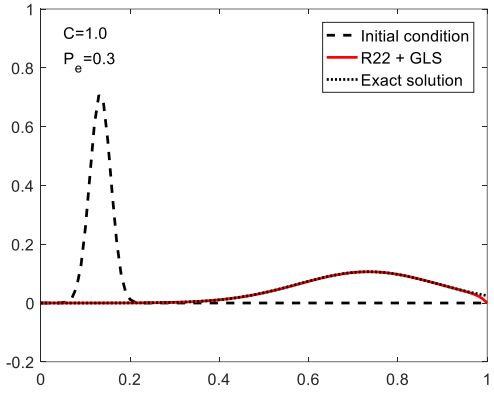


Figure 50 nu=0.02

Implementation of Adam-Bashforth

In order to implement the adam bashforth method, we shall use the formulation provided by the book and then apply the Galerkin weighted residual method and do the integration by parts. The main equation of Adam-bashforth is:

$$u^{n+1} = u^n + \frac{\Delta t}{2}\left(3u_t^n - u_t^{n-1}\right),$$

Note that this method is not self-starting therefore another method has to be used to compute the solution at the first time step and then continue with Adam-Bashforth. To implement this method we have to define another function in which we define the formulation of this method and also another method for the first step:

```
% Second step Adam-Bashforth
T = 0;
sn = 3/2;
sn0 = -1/2;
[n,m] = size(T);
Id = eye(n,m);
```

```
% Loop to compute the transient solution
for i=2:nstep
    auxn = dt*(-Kt*Sol(:,i) + Mf);
    auxn0 = dt*(-Kt*Sol(:,i-1) + Mf);
%     F=[];
    for i = 1:n
        F = [F(105:end); sn(i)*auxn + sn0(i)*auxn0];
    end
    F = [F;bccd*0];
    dc  = U\(L\F);
    dc = reshape(dc(1:n*npoin),npoin,n);
    c = c + sum(dc,2);
    Sol = [Sol c];
end
```

Everything keeps to be the same, only it has to be considered that the "W" for this method is equal to zero and "w" takes two values that are multiplied by the solution of the previous step and two steps before respectively.

The stabilization criterion for Adam-Bashforth is half of that for Euler. The following formula is defined for Euler:

$$\begin{cases} \text{if } P_e \leq \sqrt{3}, & C \leq P_e/3, \\ \text{if } P_e > \sqrt{3}, & C \leq 1/P_e. \end{cases}$$

Therefore for Adam-Bashforth for Pe less than 1.7 for Courants less than Pe/6 the method is stable. The same reasoning is done for higher values of Pe.

If we set the Pe=1, then for Courants less than 1/6 the method should be stable. For C=0.1 the result is shown below which shows great accuracy when staying in stability region.

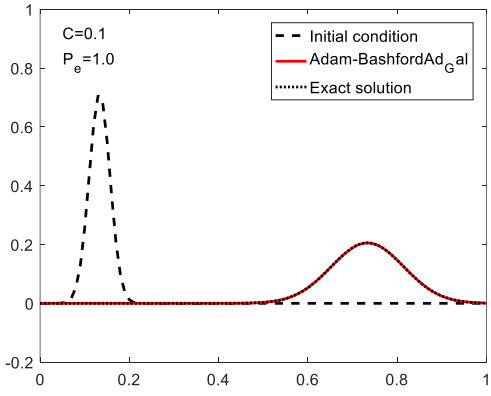


Figure 51 Adam-Bashforth

Now if we try it with R22 method in time and Galerkin in space we see that in the same condition of being in stable region both methods show great results. However it should be kept in mind that pade R22 is implicit and Adam-Bashforth is explicit which is considered a positive point for this method.

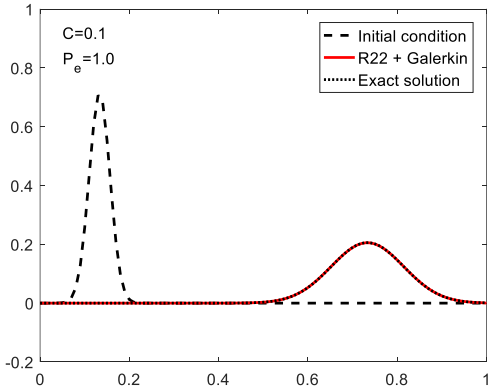


Figure 52 Pade R22

However for larger values of Courant, the R22 still shows good results but Adam-Bashforth starts to have inaccuracy and oscillation.

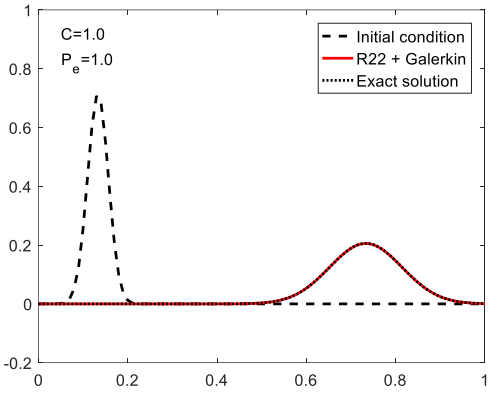


Figure 53 R22 for C=1

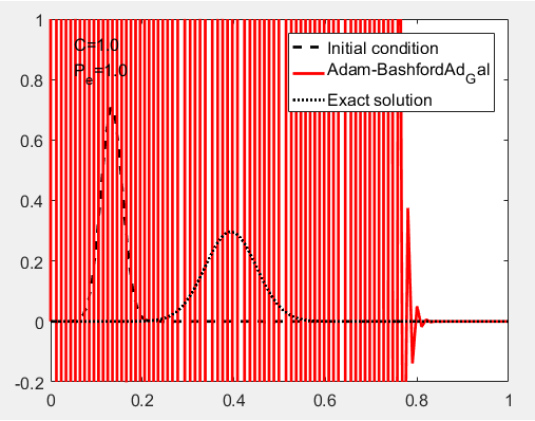


Figure 54 Adam-Bashforth for C=1