FEF Assignment 2

Unsteady Linear convection

Hanna, John

1 Introduction

A problem that is faced by finite element method is the linear unsteady convection equation. Discretization in space and time is required and they affect each other. The solution is governed by the characteristics therefore, the methods used should have some directionality to follow the solution around the characteristics. The methods presented in this report are all discretized using the classical Galerkin method, the time discretization classifies them into Lax-Wendroff, Crank-Nicholson, 3rd order Taylor-Galerkin method. The Lumped mass matrix is used in some of the methods. The results of each method is shown and a discussion and conclusion are drawn from these results.

2 Problem description and summary of the methods

The differential equation with given initial and boundary conditions that is been solved is given as:

$$u_t + au_x = 0 \quad x \in (0,1), \quad t \in (0, 0.6)$$
$$u(x,0) = u_0, \quad x \in (0,1)$$
$$u(0,t) = 0, \quad t \in (0, 0.6)$$
$$u_0 = \begin{cases} 0.5(1 + cos(\pi(x - x_0)/\sigma)) & |x - x_0| \leq \sigma \\ 0 & otherwise \end{cases}$$

where u is the transported quantity, a is the convection velocity (a=1), $x_0 = 0.2$ and $\sigma = 0.12$.

The Taylor-Galerkin methods are based on the Taylor expansion series. Then any existing time derivatives are substituted from the differential equation. Lax-Wendroff method is basically a 2^{nd} order Taylor-Galerkin method. Crank-Nicholson is one of the θ family methods where $\theta = 0.5$.
In all the methods, the classical Galerkin method is used to discretize space. Lumped mass matrices are also applied to the Lax-Wendroff and Crank-Nicholson methods defined as

$$\mathbf{M_{ii}} = \sum_{j=1}^{n} \mathbf{M_{ij}}$$

3 Results and Discussion

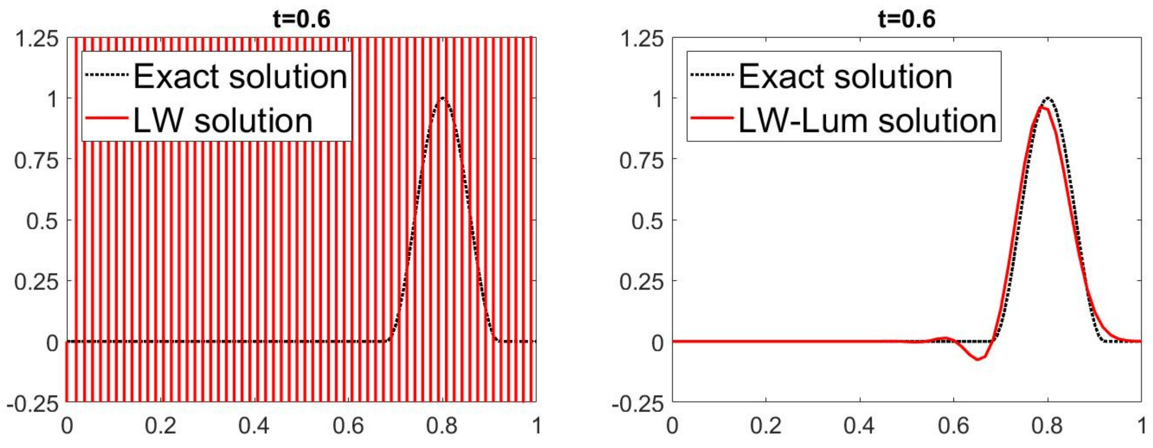

Figure 1: Lax-Wendroff with normal and lumped mass matrix C=0.72

The above graphs show the Lax-Wendroff solution with consistent mass matrix and with lumped mass matrix for Courant number $C = 0.72$. It is clear that with the normal mass matrix, the solution is completely unstable, that's because the solution is only stable for $C < 0.577$. However, when the lumped mass matrix is used the solution is stable up to $C = 1$.

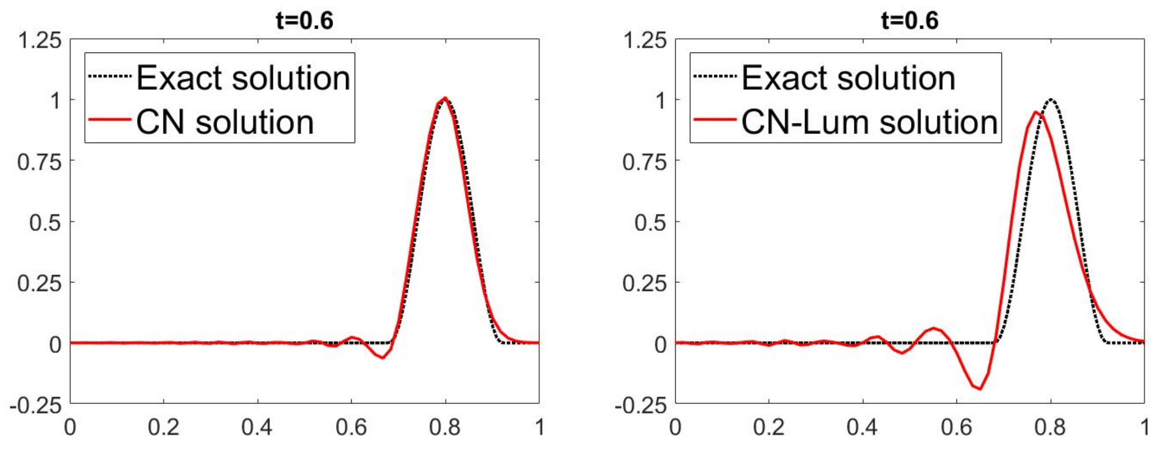

Figure 2: Crank-Nicholson with normal and lumped mass matrix C=0.72

The above graphs show the Crank-Nicholson solution with consistent mass matrix and with lumped mass matrix for Courant number $C = 0.72$. Both of them show stable solution since the CN is unconditionally stable. Using the normal mass matrix provides more accurate results than the lumped mass matrix. That's because using the normal formulation provides 4^{th} order accuracy, while using the lumped mass matrix provides only 2^{nd} order accuracy.

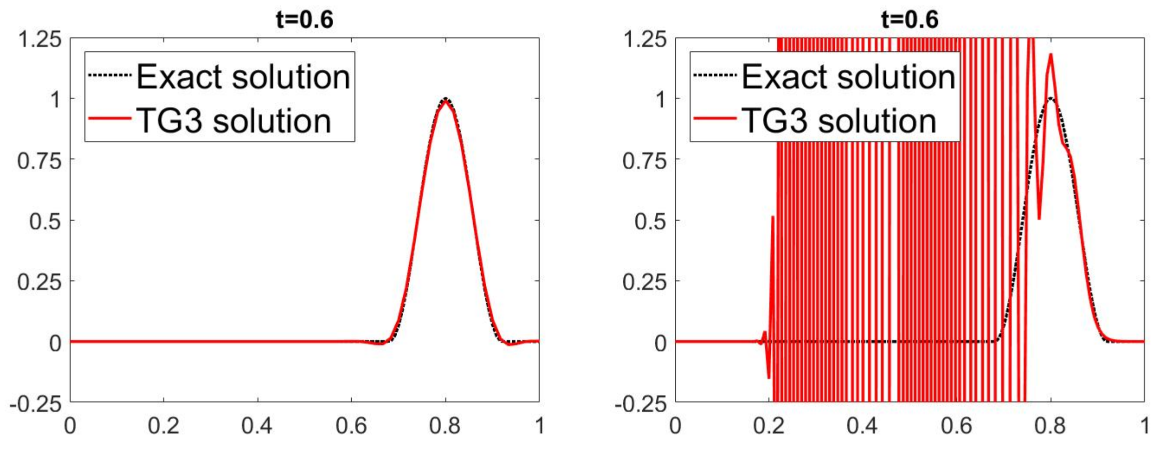

Figure 3: TG3 C=0.72 and TG3 C=1.44

The above graphs show the 3^{rd} order Taylor-Galerkin solution for $C = 0.72$ and $C = 1.44$. The TG3 provides 3^{rd} order accuracy but it's conditionally stable. The TG3 method is only stable with $C < 1$; that's why the solution is stable with $C = 0.72$ and unstable with $C = 1.44$.

4 Conclusion

To conclude, LW method is conditionally stable, therefore it's recommended to use lumped mass matrix for high Courant numbers. CN method is unconditionally stable, therefore it's advised to use the normal mass matrix to provide more accuracy. TG3 is a conditionally stable method; it has higher range of stability than LW, however for Courant numbers from 1 to 2, lumped mass matrix will provide stability.

5 Appendix(MATLAB codes)

5.1 Lumped Mass Matrix Calculation

```
Mlum=zeros(2,2);
if method == 2 || method == 4
    for i=1:2
        for j=1:2
            Mlum(i,i)=Mlum(i,i)+Me(i,j)
        end
    end
end
Me=Mlum;
```

5.2 Calculating A and B matrices for all methods

```
function [A,B,methodName] = System(method,M,K,C,a,dt)

switch method
    case 1 % Lax-Wendroff + Galerkin
        A = M;
        B = -a*dt*C - dt^2/2*a^2*K;
        methodName = 'LW';

    case 2 % Lax-Wendroff with lumped mass matrix + Galerkin
        A = M;
        B = -a*dt*C - dt^2/2*a^2*K;
        methodName = 'LW-Lum';

    case 3 % Crank-Nicolson + Galerkin
        A = M + 1/2*a*dt*C;
        B = -a*dt*C;
        methodName = 'CN';

    case 4 % Crank-Nicolson with lumped mass matrix + Galerkin
        A = M + 1/2*a*dt*C;
        B = -a*dt*C;
        methodName = 'CN-Lum';

    case 5 %Taylor-Garlekin 3rd order
        A = M + dt^2/6*a^2*K;
        B = -a*dt*C - dt^2/2*a^2*K;
        methodName = 'TG3';

    otherwise
        error('not available method')
end
```

Unsteady nonlinear convection problem

1 Introduction

A problem that is faced by finite element method is the nonlinear unsteady convection equation. A famous problem is the burger's equation which will be studied here. The main issue in this problem is the possibility of having discontinuous solution generated from continuous initial conditions. A diffusion term is added with very low viscosity to avoid singularities in the solution. The classical Galerkin method is used for space discretization. Forward and backward Euler are used for time discretization; the latter needs iterative methods to be solved.

2 Problem description and summary of the methods

The differential equation with given initial conditions that is been solved is given as:

$$u_t + uu_x = \epsilon u_{xx} \quad x \in (0,4), \quad t \in (0,4)$$
$$u(x,0) = u_0, \quad x \in (0,1)$$
$$u(0,t) = 1, \quad u(4,t) = 0, \quad t \in (0,4)$$
$$u_0 = \begin{cases} 1 - \frac{x}{3} & x < 3 \\ 0 & x \geq 3 \end{cases}$$

The problem is spatially discretized using the Galerkin method leading to:

$$\mathbf{M}\dot{\mathbf{U}} + \mathbf{C}(\mathbf{U})\mathbf{U} + \epsilon\mathbf{K}\mathbf{U} = 0$$

Using the forward Euler method will produce linear system of equations that can be solved explicitly for each time step. However, using backward Euler will produce non linear system of equations due to the dependency of the convection matrix C on U. This system can be solved iteratively using Picard or Newton-Rhapson methods.

3 Results and Discussion

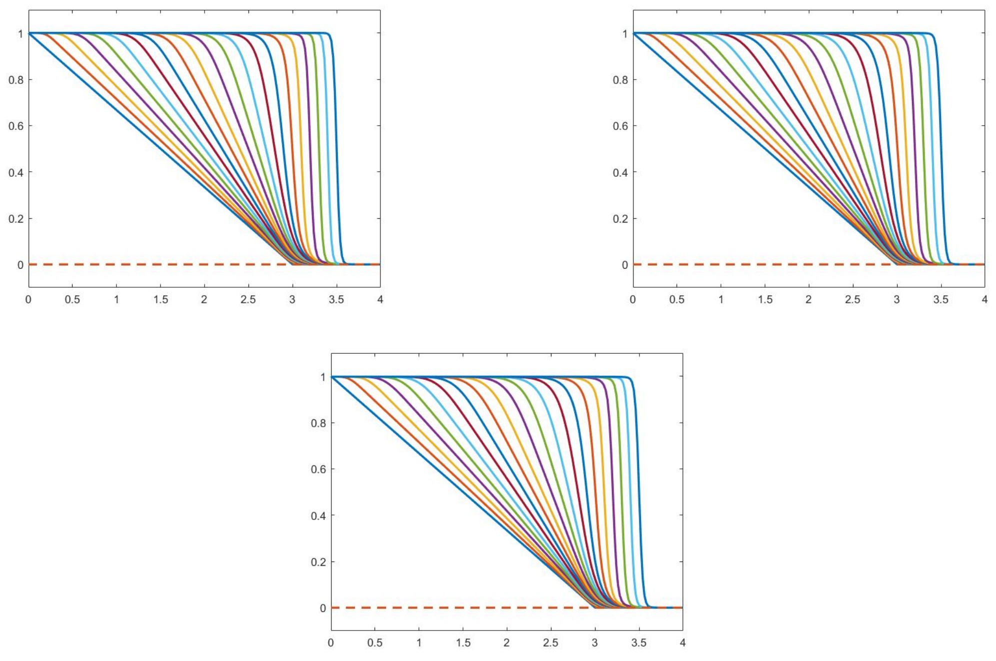

Figure 1: Explicit, Picard, and Newton-Rhapson methods ($\epsilon = 0.01, dt = 0.005$)

The above 3 graphs are for explicit, Picard, and Newton-Rhapson methods, respectively. The viscosity is equal to 0.01 and with 0.005 time step. The results of all the methods are stable and almost equivalent.

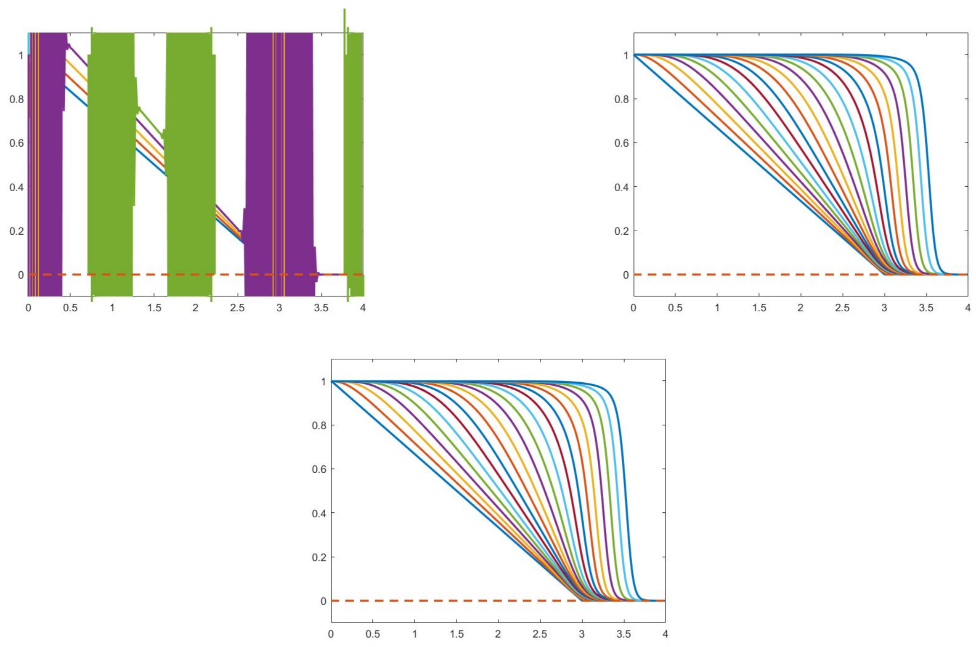

Figure 2: Explicit, Picard, and Newton-Rhapson methods ($\epsilon = 0.01, dt = 0.1$)

The above 3 graphs are for explicit, Picard, and Newton-Rhapson methods, respectively. The viscosity is equal to 0.01 and with 0.1 time step. The results of both implicit methods are stable while the explicit one show instability. This means that the explicit method doesn't behave well using large time steps. That is because the Euler method is first order convergent so very small time steps are required to achieved stability.

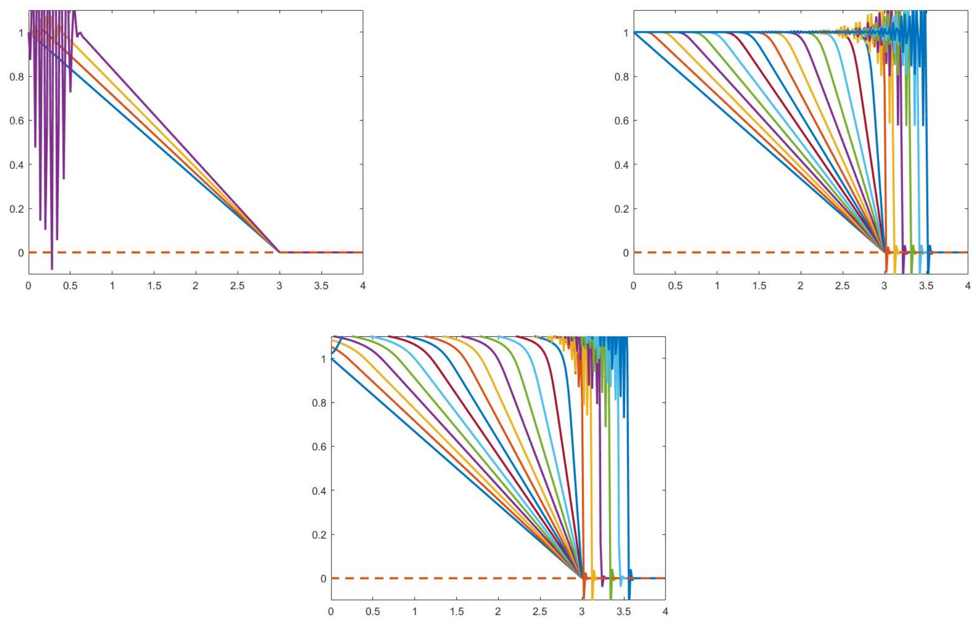

Figure 3: Explicit, Picard, and Newton-Rhapson methods ($\epsilon = 0.0001, dt = 0.005$)

The above 3 graphs are for explicit, Picard, and Newton-Rhapson methods, respectively. The viscosity is equal to 0.0001 and with 0.005 time step. All the methods show instability in the solution. That's mainly due to the domination of the convection over the diffusion which produces instabilities. With very low viscosity, singularities start to appear showing the existence of a discontinuity in the solution.

4 Conclusion

To conclude, Burger's equation can be solved using implicit and explicit methods. The implicit methods will need to be solved using iterative methods such as Picard or Newton-Rhapson. The explicit method is stable for low time steps but instability appears at large time steps. The implicit methods behave well at large and small time steps. All the methods show instability when using very low diffusion coefficient.

5 Appendix(Newton-Rhapson MATLAB code)

```
for n = 1:nTimeSteps
    U0 = U(:,n);      error_U = 1;      k = 0;
    while (error_U > 0.5e-5) && k < 20
        C = computeConvectionMatrix(X,T,U0);
        f = (M+At*C+E*At*K)*U0 - M*U(:,n);
        J = M + 2*At*C + E*At*K;
        U1 = U0-inv(J)*f;          error_U = norm(U1-U0)/norm(U1);
        U0 = U1; k = k+1;
    end
    U(:,n+1) = U1;
```