UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Swansea University Prifysgol Abertawe

CENTRALE NANTES

Universität Stuttgart

MSc in Computational Mechanics

Finite Elements in Fluids

MATLAB Assignment 2:

Unsteady convection and non-linear hyperbolic problems

Submitted By:
Mario Alberto Méndez Soto

Submitted To:
Prof. Pablo Saez
Prof. Antonio Huerta

Spring Semester, 2019

1 Propagation of a steep front - unsteady convection

The propagation of a steep front is a physical phenomenon governed by the unsteady convection equation and the following boundary and initial conditions:

$$\begin{cases} u_t + a u_x = 0 & x \in (0,1), t \in (0, 0.6] \\ u(x,0) = u_0 & x \in (0,1) \\ u(0,t) = 1 & t \in (0, 0.6] \end{cases} \tag{1}$$

$$u_0 = \begin{cases} 1 & \text{if } x \leq 0.2 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

For the given problem, the solution will be computed using a convection velocity $a = 1$. Moreover, the discretization in time and space will be $\Delta x = h = 2 \cdot 10^{-2}$ and $\Delta t = 1.5 \cdot 10^{-2}$, respectively. Thus, the Courant number equals 0.75.

The initial code given had a fully-operational implementation of the Crank-Nicolson time scheme with a Galerkin formulation for the space discretization. Firstly, the *FEM_matrices.m* file was modified to include the computation of the lumped mass matrix using the following expression:

$$\mathbf{M^L} : M_{ij}^L = \begin{cases} \int_\Omega N_i N_j d\Omega & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{3}$$

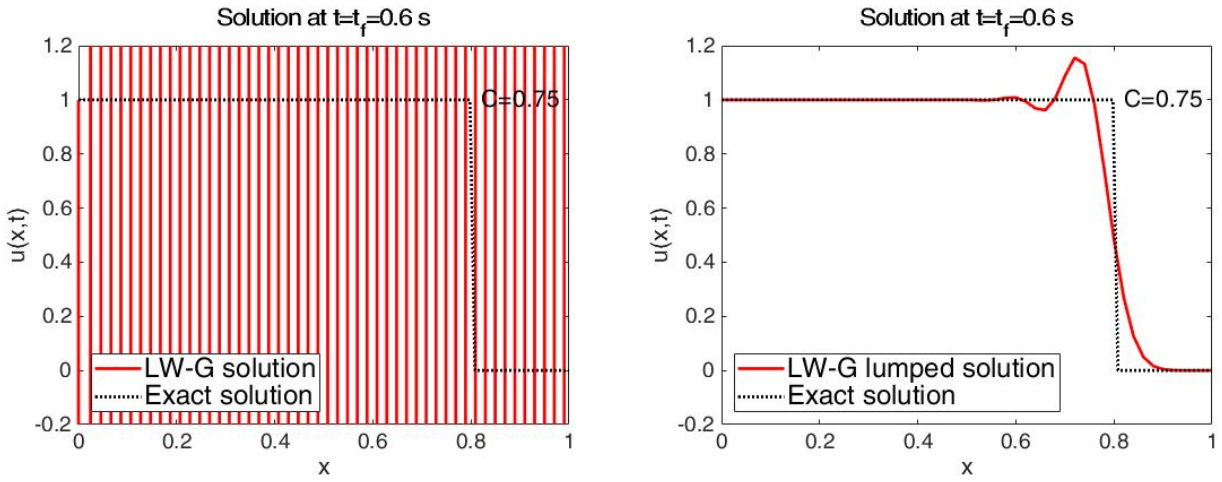
Using linear elements with an isoparametric formulation, the following matrix definitions are introduced:

$$\begin{aligned} \mathbf{M}: &\quad M_{ij} = \int_\Omega N_i N_j d\Omega && \text{Consistent mass matrix} \\ \mathbf{C}: &\quad C_{ij} = \int_\Omega N_i (\mathbf{a} \cdot \nabla N_j) d\Omega && \text{Convection matrix} \\ \mathbf{K}: &\quad K_{ij} = \int_\Omega (\nabla N_i \cdot \nabla N_j) d\Omega && \text{Stiffness matrix} \end{aligned}$$

Using the latter definitions, it is possible to write down the corresponding matricial expressions for different time-discretization schemes after the Galerkin formulation is implemented:

Crank-Nicolson	$\left(\frac{1}{\Delta t}\mathbf{M} - \theta\mathbf{C}\right)\mathbf{\Delta u} = \mathbf{f} + \mathbf{C u^n}$
Crank-Nicolson with lumped mass matrix	$\left(\frac{1}{\Delta t}\mathbf{M^L} - \theta\mathbf{C}\right)\mathbf{\Delta u} = \mathbf{f} + \mathbf{C u^n}$
Lax-Wendroff (TG2)	$\frac{1}{\Delta t}\mathbf{M}\mathbf{\Delta u} = \mathbf{f} + \mathbf{C u^n} - \frac{\Delta t}{2}\|\mathbf{a}\|^2 \mathbf{K u^n}$
Lax-Wendroff with lumped mass matrix	$\frac{1}{\Delta t}\mathbf{M^L}\mathbf{\Delta u} = \mathbf{f} + \mathbf{C u^n} - \frac{\Delta t}{2}\|\mathbf{a}\|^2 \mathbf{K u^n}$
Third-order Taylor-Galerkin (TG3)	$\left(\frac{1}{\Delta t}\mathbf{M} + \frac{\Delta t}{6}\|\mathbf{a}\|^2\mathbf{K}\right)\mathbf{\Delta u} = \mathbf{f} + \mathbf{C u^n} - \frac{\Delta t}{2}\|\mathbf{a}\|^2 \mathbf{K u^n}$

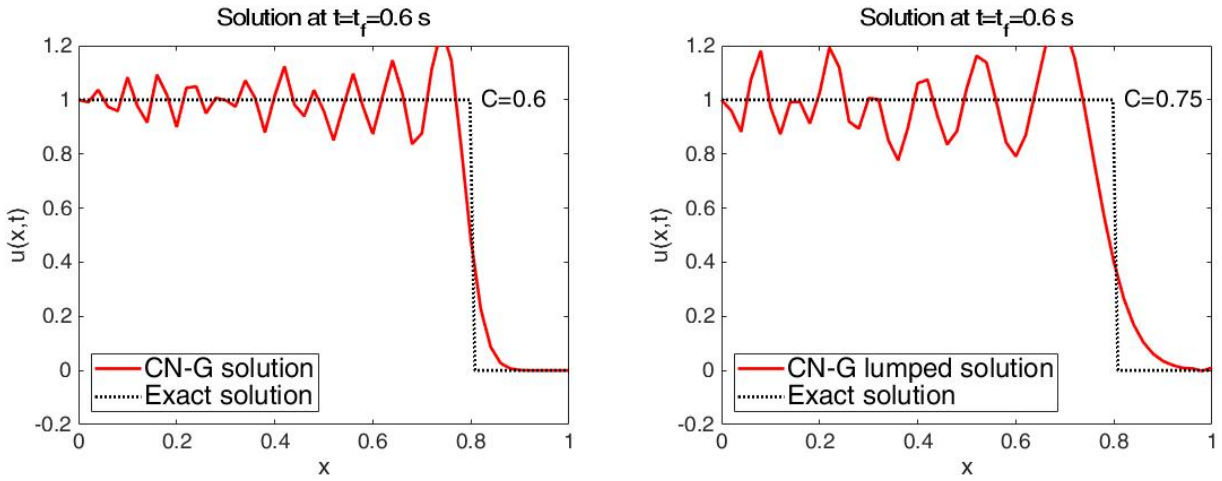
Subsequently, the file *System.m* was modified to include the previously introduced matricial expressions for the different schemes. The results obtained for the TG2 formulation are depicted in Figure (1). As mathematically predicted, TG2 presents with catastrophic instabilities when $C \geq \frac{\sqrt{3}}{3} \approx 0.577$. Nevertheless, the stability range expands when using a lumped mass matrix (1b) instead of a consistent one (1a), even though Courant number is the same in both cases.



(a) Solution with consistent mass matrix (b) Solution with lumped mass matrix

Fig. 1 – Solution of equation (1) using a Law-Wendroff Galerkin implementation (TG2)

On the other hand, as it can be noted in Figure (2), for the same values of the Courant number within the stability range, the results are more accurate if a mass consistent matrix is used (2a). In the case of the Crank-Nicolson formulation, even though both solutions are stable (with only local non-divergent instabilities) the error is greater when using a lumped mass matrix (2b).



(a) Solution with consistent mass matrix (b) Solution with lumped mass matrix

Fig. 2 – Solution of equation (1) using a Crank-Nicolson implementation

The solution of the problem using a TG3 formulation is represented in Figure (3). Since by definition the method has third-order accuracy in time, the associated error is smaller in comparison with TG2 and CN with either a consistent or a lumped mass matrix. Moreover, since $C \leq 1$ the scheme is stable.

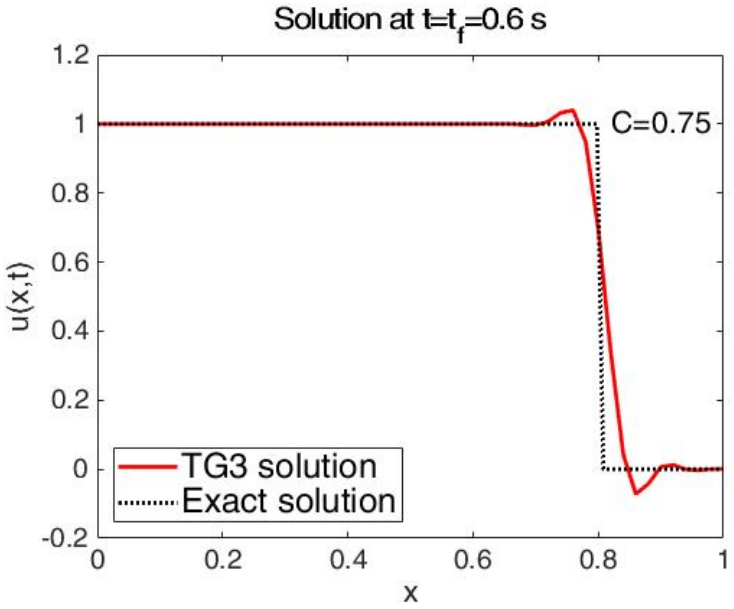


Fig. 3 – Solution of equation (1) using a Third-order Taylor Galerkin implementation

2 Burgers' equation - nonlinear hyperbolic equation

Burgers' equation is a non-linear hyperbolic equation that can be expressed as follows in a convective or non-convective form (see equation (4)).

$$\begin{cases} u_t + \left(\frac{u^2}{2}\right)_x = 0 & x \in (a,b), t \in (0,\infty) \\ u(x,0) = u_0(x) & x \in (a,b) \end{cases} \quad \begin{cases} u_t + uu_x = 0 & x \in (a,b), t \in (0,\infty) \\ u(x,0) = u_0(x) & x \in (a,b) \end{cases} \tag{4}$$

For the given case, the domain is defined in the interval $[0, 4]$ and the initial condition is depicted in Figure (4).

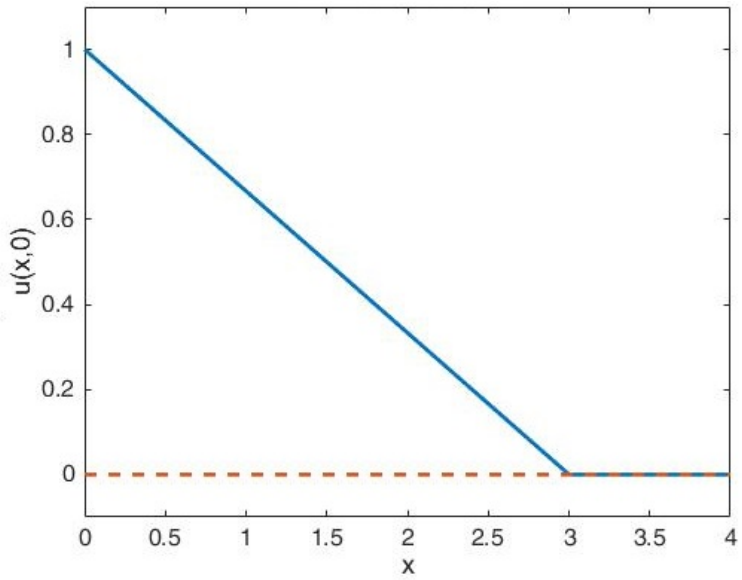


Fig. 4 – Initial condition for equation (4)

The correct solution of Burgers' equation can be determined by the *vanishing viscosity* approach. Thus, the physically compliant weak solution of the inviscid Burger's equation (4) corresponds to the solution of the viscous Burgers' equation (see equation (5))) when the added viscosity tends to zero. In such a manner, the inviscid case (4) is seen as a special case of:

$$u_t^\epsilon + u^\epsilon u_x^\epsilon = \epsilon u_{xx} \tag{5}$$

A FEM Galerkin discretization of equation (5) using $u(x,t) \approx u_h(x,t) = \sum_j N_j(x)u_j(t)$ and $w(x) = N_i(x)$ produces the following matricial system of equations:

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

Using a time discretization with a θ-family scheme, the following schemes are possible:

- Forward Euler

$$\mathbf{M}\frac{\mathbf{U}^{n+1}-\mathbf{U}^{n}}{\Delta t}+\mathbf{C}(\mathbf{U}^{n})\mathbf{U}^{n}+\epsilon\mathbf{K}\mathbf{U}^{n}=0$$
$$\mathbf{M}\mathbf{U}^{n+1}=\Big(\mathbf{M}-\Delta t\Big(\mathbf{C}(\mathbf{U}^{n})+\epsilon\mathbf{K}\Big)\Big)\mathbf{U}^{n}$$

- Backward Euler

$$\mathbf{M}\frac{\mathbf{U}^{n+1}-\mathbf{U}^{n}}{\Delta t}+\mathbf{C}(\mathbf{U}^{n+1})\mathbf{U}^{n+1}+\epsilon\mathbf{K}\mathbf{U}^{n+1}=0$$
$$\Big(\mathbf{M}+\Delta t\Big(\mathbf{C}(\mathbf{U}^{n+1})+\epsilon\mathbf{K}\Big)\Big)\mathbf{U}^{n+1}=\mathbf{M}\mathbf{U}^{n}$$

The initial code given had fully-operational implementations for the solution of the Burgers's equation using an Forward Euler and Backward Euler scheme (with Picard method as the iterative implementation). The main task was to include a code that uses the Newton-Raphson iterative method for the solution of non-linear systems of equations, in this case for the Backward Euler scheme.

To implement the Newton-Raphson method for time step $n+1$, we define the following equation to be solved:

$$\mathbf{f}(\mathbf{U}^{n+1})=\mathbf{0}$$

with $\mathbf{f}(\mathbf{U})$ defined as:

$$\mathbf{f}(\mathbf{U})=\Big(\mathbf{M}+\Delta t\mathbf{C}(\mathbf{U})+\epsilon\Delta t\mathbf{K}\Big)\mathbf{U}-\mathbf{M}\mathbf{U}^{n}$$

Using the Newton-Raphson, method the initial guess, for a given time step $n+1$, is taken to be the solution at the previous time step:

$${}^{0}\mathbf{U}^{n+1}=\mathbf{U}^{n}$$

Then, solution of the equation at time $n+1$ after k iterations will be computed as:

$${}^{k+1}\mathbf{U}^{n+1}={}^{k}\mathbf{U}^{n+1}-\mathbf{J}^{-1}({}^{k}\mathbf{U}^{n+1})\mathbf{f}({}^{k}\mathbf{U}^{n+1})$$

where matrix $\mathbf{J}$ is the Jacobian of the system of equations and is defined as: $\mathbf{J}=\dfrac{d\mathbf{f}}{d\mathbf{U}}$.

Since $\mathbf{C}(\mathbf{U})$ is a function that depends linearly on $\mathbf{U}$ (after being integrated with the corresponding shape functions). The derivative with respect to $\mathbf{U}$ of the expression $\mathbf{C}(\mathbf{U})\mathbf{U}$ will be of the form $2\mathbf{C}(\mathbf{U})$. Thus, the Jacobian for the particular given problem will be:

$$\mathbf{J}(\mathbf{U})=\frac{d\mathbf{f}}{d\mathbf{U}}=\mathbf{M}+2\Delta t\mathbf{C}(\mathbf{U})+\epsilon\Delta t\mathbf{K}$$

Thus, the computation of the matrix $\mathbf{C}(\mathbf{U})$ will be performed in each iteration k in the Newton-Raphson implementation.

Using the previous result, a new function *burgers_imNR.m* was created for the implementation of the Backward Euler scheme with a Newton-Raphson method. The following cite contains the main cycle that allows the implementation of the method within the function mentioned above:

```
for n = 1:nTimeSteps
    U0 = U(:,n); % solution at time time step n
    error_U = 1; k = 0;
    while (error_U > 0.5e-5) & & k < 20 % Target error and maximum number of iterations
        C = computeConvectionMatrix(X,T,U0);
        F = (M + At*C+At*E*K)*U0-M*U(:,n);% Function F(U)
        J = M+2*At*C+At*E*K;% Jacobian
        U1 = U0-J  F; % Result of k iteration
        error_U = norm(U1-U0)/norm(U1); % Iteration error
        U0 = U1; % U update
        k = k+1;
    end U(:,n+1) = U1;
end
```

The solution of Burgers' equation in the domain $[1, 4]$ is depicted in the Figure (6). The final time is set to be $t_f = 4$. For the computations, space-time discretization with $h = 0.02$ and $\Delta t = 0.005s$ were used.

Although under the given conditions of space-time discretizations, the methods seem to perform similarly and the solutions are equally accurate (see Figure (5)). It is worth mentioning that whereas the convergence of Picard's has a linear behavior, Newton Raphson converges with a quadratic law. As to the computational effort, Picard's method requires the solution of a single linear system per iteration while Newton-Raphson needs to firstly compute both $\mathbf{J}$, $\mathbf{F}$ and then solve the linear system associated.

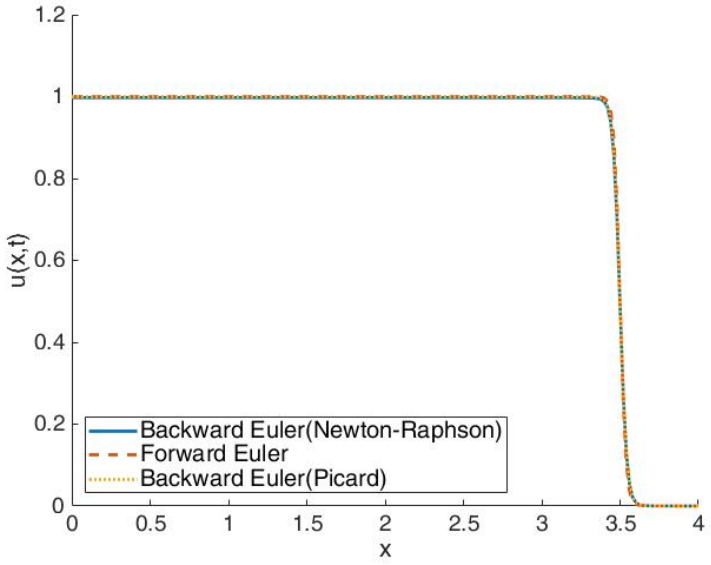
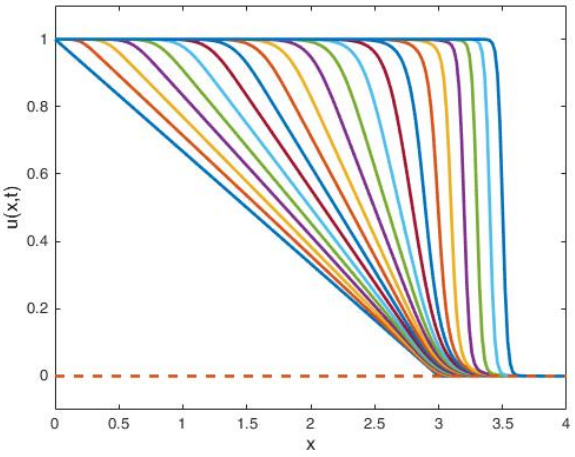
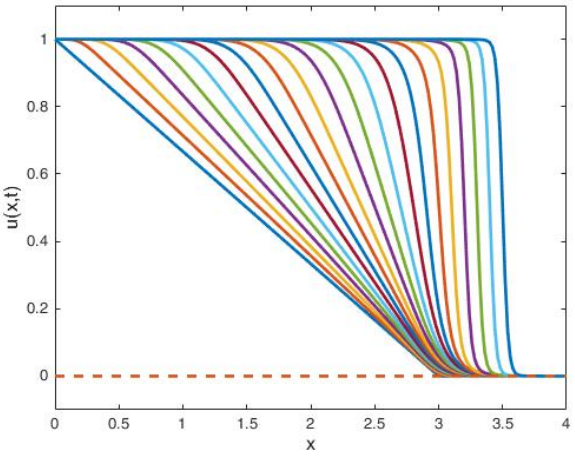


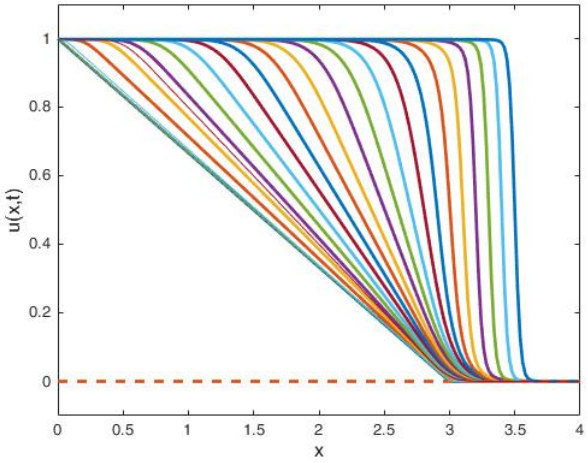
Fig. 5 – Result of equation (4) at $t = t_f = 4s$ using different techniques



(a) Explicit scheme (Forward Euler)



(b) Implicit scheme (Picard's Method)



(c) Implicit scheme (Newton-Raphson Method)

Fig. 6 – Solution of equation (4) using a explicit and implicit schemes