Universitat Polytechnica de Catalunya
MSc Computational Mechanics
Spring 2018

Finite Elements in Fluids

FEFmatlab2

Due 02/04/2018

Alexander Keiser

1 Implementation of Leap-Frog Method

We will first begin by developing the formulation for the leap-frog method that we can plug into the provided codes. This derivation can be seen below.

→ Leap frog method

$$\frac{U^{n+1} - U^{n-1}}{2\Delta t} = U_t^n = \cancelto{0}{S^n} - a\nabla U^n$$

the source term S^n is zero

$$\frac{U^{n+1}}{2\Delta t} = \frac{U^{n-1}}{2\Delta t} - a\nabla U^n$$

now, introducing the galerkin formulation

$$\int w \frac{U^{n+1}}{2\Delta t} = \int w \frac{U^{n-1}}{2\Delta t} - a\int w\nabla U^n$$

integrating by parts gives

$$(-a)\int w\nabla U^n = (-a)\left(\cancelto{0}{\left[wU^n\right]_{\Gamma_N}} - \left[U^n\nabla w\right]_{\Gamma_{out}}\right)$$

now, rewriting $U^{n+1} = \sum N_\gamma U^{n+1}$, $U^n = \sum N_\gamma U^n$, & $U^{n-1} = \sum N_\gamma U^{n-1}$

with $w = N_i$ gives us ...

$$\frac{N_i N_\gamma}{2\Delta t} U^{n+1} = \frac{N_i N_\gamma}{2\Delta t} U^{n-1} + a N_\gamma N_{xi} U^n$$

and we know that $M = N_i N_\gamma$ $C = N_{xi} N_\gamma$ $K = N_{xi} N_{x\gamma}$

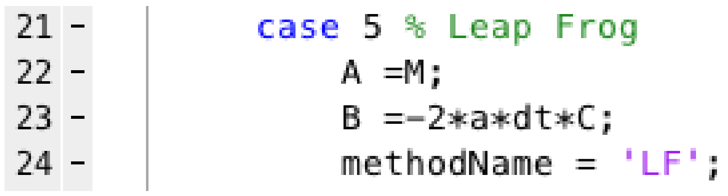
$$\frac{M}{2\Delta t} du = a C^T U^n = -a C U^n$$

$$\boxed{M\, du = -2\Delta t\, a\, C U^n}$$

$A = M$

$B = -2\Delta t a C$

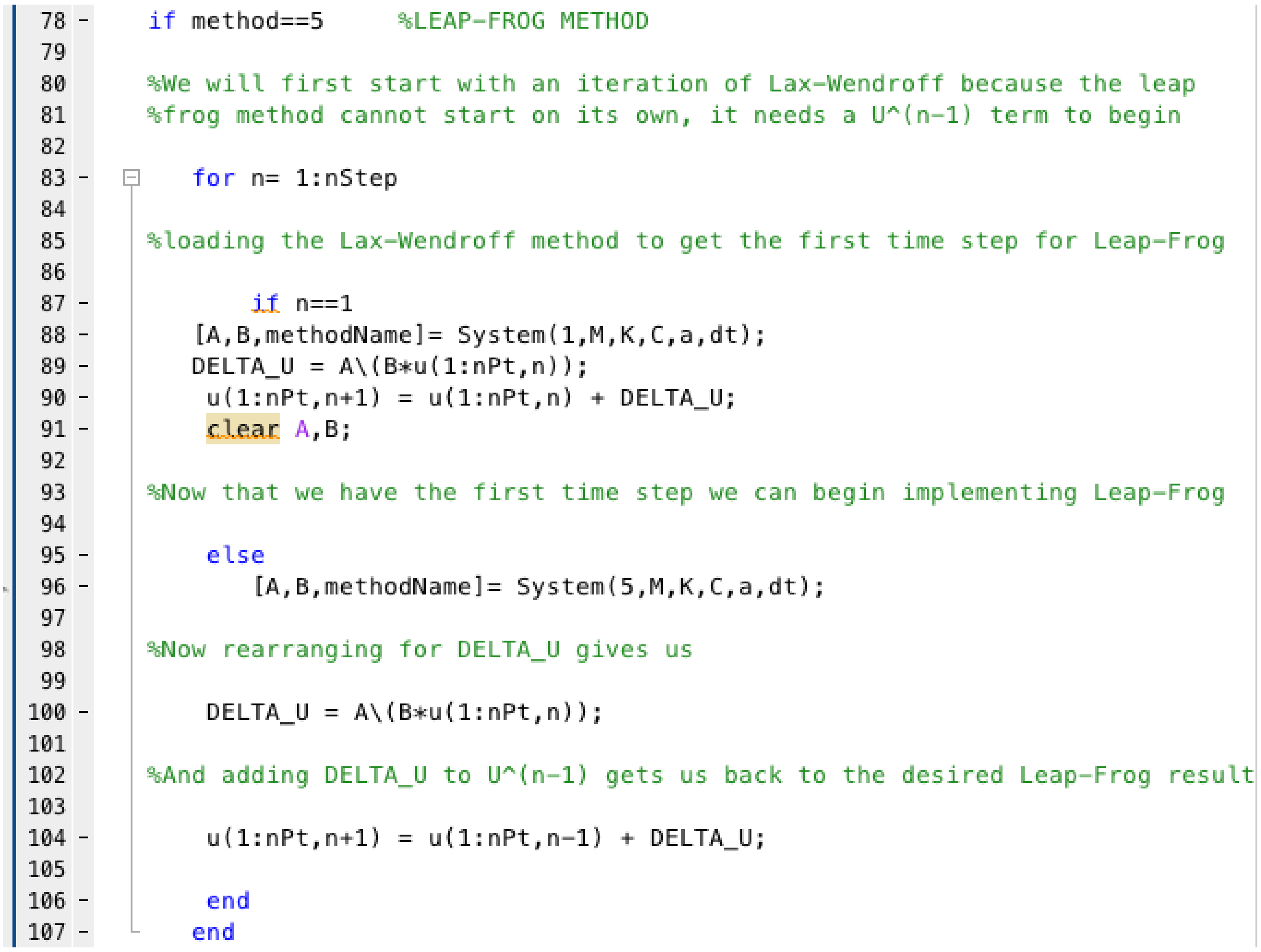
Using the above derivation, we can make the first addition to the codes to successfully implement the leap-frog method. This can be seen on the next page in figure 1.



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

Figure 1: Code Modification to System.m for Leap-Frog

In the above figure, we have successfully implemented matrices A and B from our derivation on the previous page into the provided Matlab file System.m. We will now modify the code in Main.m to finish the implementation. This will be done in 2 steps since the Leap-Frog method is not able to be started by its self. This code and relevant comments can be seen below.



```
if method==5     %LEAP-FROG METHOD

%We will first start with an iteration of Lax-Wendroff because the leap
%frog method cannot start on its own, it needs a U^(n-1) term to begin

   for n= 1:nStep

%loading the Lax-Wendroff method to get the first time step for Leap-Frog

       if n==1
   [A,B,methodName]= System(1,M,K,C,a,dt);
   DELTA_U = A\(B*u(1:nPt,n));
    u(1:nPt,n+1) = u(1:nPt,n) + DELTA_U;
    clear A,B;

%Now that we have the first time step we can begin implementing Leap-Frog

    else
       [A,B,methodName]= System(5,M,K,C,a,dt);

%Now rearranging for DELTA_U gives us

    DELTA_U = A\(B*u(1:nPt,n));

%And adding DELTA_U to U^(n-1) gets us back to the desired Leap-Frog result

    u(1:nPt,n+1) = u(1:nPt,n-1) + DELTA_U;

    end
   end
```

Figure 2: Code Modification to Main.m for Leap-Frog

Now that we have implemented the Leap Frog method, it is time to analyze the plots for this method with varying Courant numbers and comment on the results. This can be seen on the next page.

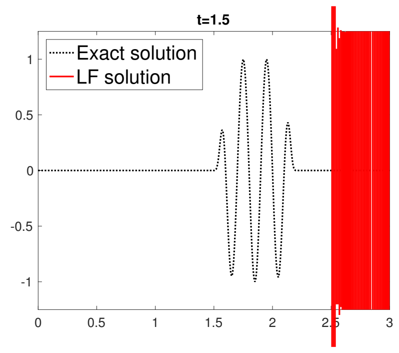


Figure 3a: Courant Number = 2

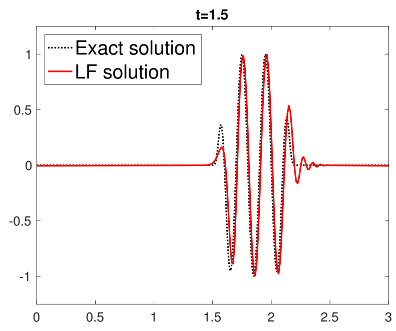


Figure 3b: Courant Number = 0.5

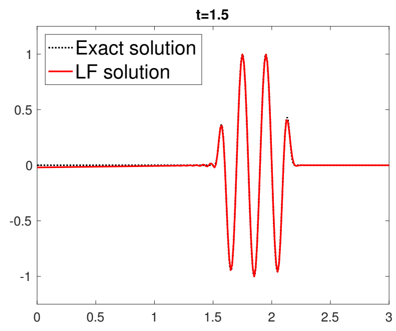


Figure 3c: Courant Number = 0.125

Here are the results from running problem 2 with the leap frog method at varying Courant numbers by changing the number of time steps. Figure 3a was run with 60 time steps resulting in a Courant number of 2. As expected, this plot exhibits high inaccuracy and instability since the Courant number is much higher than allowable. Figure 3b was run at the default number of 240 time steps resulting in a Courant number of 0.5. This Courant number is within the threshold for stability but produces some small unwanted oscillations and inaccuracies as time goes on at the first and last oscillations of the exact solution. Figure 3c was run with 960 time steps resulting in a Courant number of 0.125. This Courant number is very low so the high degree of accuracy and stability exhibited by the method in the figure is expected. In conclusion, the implemented leap frog method was done correctly and performs as expected.

2 Implementation of Third Order Taylor Galerkin Method

We will begin this section by developing the Third Order Taylor Galerkin Method. This can be seen below.

Third order Taylor galerkin

$$\frac{u(t^{n+1})-u(t^n)}{\Delta t} = u_t(t^n) + \frac{1}{2}\Delta t\, u_{tt}(t^n) + \frac{1}{6}\Delta t^2 u_{ttt}(t^n) + \cancelto{0}{O(\Delta t^3)}$$

with $u_t = \cancelto{0}{S} - a\nabla u^n$

$$u_{tt} = \cancelto{0}{S_t} - a\nabla \cancelto{0}{S} + (a\nabla)^2 u^n$$

$$u_{ttt} = \cancelto{0}{S_{tt}} - a\nabla \cancelto{0}{S_t} + (a\nabla)^2 \frac{(u^{n+1}-u^n)}{\Delta t}$$

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

$$\left[1 - \frac{\Delta t^2}{6}(a\nabla)^2\right]\frac{\Delta u}{\Delta t} = -a\nabla u^n + \frac{1}{2}\Delta t (a\nabla)^2 u^n$$

Applying Galerkins

$$\int w \frac{\Delta u}{\Delta t} - \frac{\Delta t^2 a^2}{6}\int w\nabla^2 \frac{(\Delta u)}{\Delta t} = -a\int w\nabla u^n + \frac{\Delta t a^2}{2}\int w\nabla^2 u^n$$

Integrating by parts...

$$-\frac{\Delta t^2 a^2}{6\Delta t}\int w\nabla^2(\Delta u) = \frac{\Delta t a^2}{6}\left[\nabla w \nabla(\Delta u)\right]_{\Gamma_{out}}$$

$$-a\int w\nabla u^n = a\left[u^n \nabla w\right]_{\Gamma_{out}}$$

$$\frac{\Delta t a^2}{2}\int w\nabla^2 u^n = -\frac{\Delta t a^2}{2}\left[\nabla w \nabla u^n\right]_{\Gamma_{out}}$$

And substituting...

$$\frac{1}{\Delta t} N_i N_j (\Delta u) + \frac{\Delta t a^2}{6} N_{xi} N_{xj} (\Delta u) = a N_j N_{xi} - \frac{\Delta t a^2}{2} N_{xi} N_{xj}$$

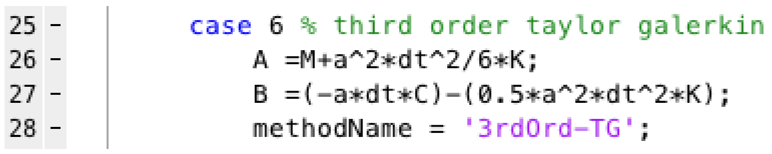
$$\left[M + \frac{\Delta t^2 a^2}{6} K\right](\Delta u) = -a\Delta t C u^n - \frac{1}{2}\Delta t^2 a^2 K u^n$$

$$\boxed{\left[M + \frac{\Delta t^2 a^2}{6} K\right](\Delta u) = -\left(a\Delta t C + \frac{1}{2}\Delta t^2 a^2 K\right)u^n}$$

$$A = M + \frac{\Delta t^2 a^2}{6} K$$

$$B = -\left(a\Delta t C + \frac{1}{2}\Delta t^2 a^2 K\right)$$

Now that we have developed the method, It is time to implement it in the codes provided. This implementation can be seen on the next page in the following figure.



```
case 6 % third order taylor galerkin
    A =M+a^2*dt^2/6*K;
    B =(-a*dt*C)-(0.5*a^2*dt^2*K);
    methodName = '3rdOrd-TG';
```

Figure 4: Code Modification to System.m for Third Order Taylor Galerkin method

Here we have implemented the A and B matrices that were derived on the previous page into the System.m file. No additional modifications are necessary and no code must be added to Main.m to successfully implement the method. Below in the following figures we will test this method for various Courant Numbers and analyze the resulting plots.

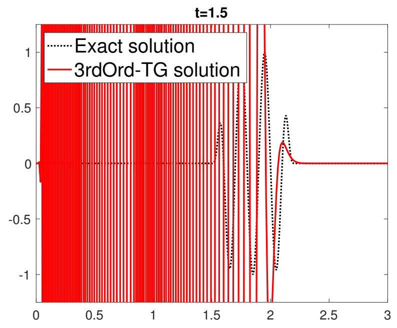


Figure 4a: Courant Number = 2

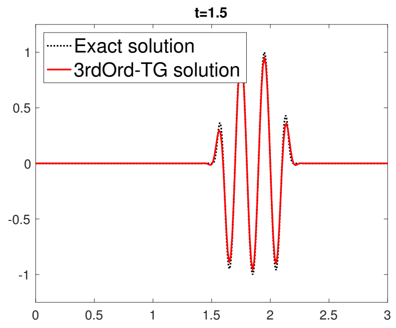


Figure 4b: Courant Number = 0.5

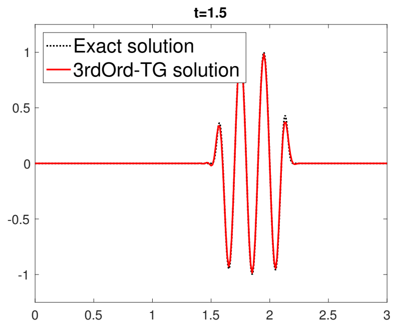


Figure 4c: Courant Number = 0.125

Here are the results from running problem 2 with the Third Order Taylor Galerkin method at varying Courant numbers by changing the number of time steps. Figure 4a was run with 60 time steps resulting in a Courant number of 2. This plot exhibits high inaccuracy and instability since the Courant number is much higher than its allowable value of $C^2 \leq 1$. Figure 4b was run at the default number of 240 time steps resulting in a Courant number of 0.5. This Courant number is within the threshold for stability of $C^2 \leq 1$ and therefore behaves accurately and exhibits stability. It is worth noting that there are some slight inaccuracies at the first and last oscillation peaks of the exact solution. Figure 4c was run with 960 time steps resulting in a Courant number of 0.125. This Courant number is very low so the high degree of accuracy and stability exhibited by the method in the figure is expected. In conclusion, the implemented Third Order Taylor Galerkin method is correct and performs as expected.

3 Implementation of Third Order, Two Step Taylor Galerkin Method

We will begin this section by developing the Third Order Two Step Taylor Galerkin Method. Step 1 can be seen below, and Step 2 can be seen on the following page.

STEP ONE

$$\bar{U}^n = U^n + \frac{1}{3}\Delta t U_t^n + \alpha \Delta t^2 U_{tt}^n$$

with $U_t^n = -a\nabla U^n$

$U_{tt}^n = (a\nabla)^2 U^n$

rewriting as..

$$\bar{U}^n = U^n - \frac{1}{3}\Delta t a \nabla U^n + \alpha \Delta t^2 a^2 \nabla^2 U^n$$

applying galerkins...

$$\int w\bar{U}^n = \int wU^n - \frac{\Delta t a}{3}\int w\nabla U^n + \alpha \Delta t^2 a^2 \int w\nabla^2 U^n$$

integrating by parts...

$$-\frac{\Delta t a}{3}\int w\nabla U^n = \frac{\Delta t a}{3}\left[U^n \nabla w\right]$$

$$\alpha \Delta t^2 a^2 \int w \nabla^2 U^n = -\alpha \Delta t^2 a^2 \left[\nabla w \nabla U^n\right]$$

and substituting..

$$N_i N_j \bar{U}^n = N_i N_j U^n + \frac{a\Delta t}{3} N_j N_{xi} U^n - \alpha \Delta t^2 a^2 N_{xi} N_{xj} U^n$$

$$M\bar{U}^n = MU^n - \frac{1}{3}a\Delta t\, C U^n - \alpha \Delta t^2 a^2 K U^n$$

$$\boxed{M\,\overline{\Delta U} = -\left(\frac{1}{3}a\Delta t\, C + \alpha \Delta t^2 a^2 K\right)U^n}$$

$A = M$

$B = -\left(\frac{1}{3}a\Delta t\, C + \alpha \Delta t^2 a^2 K\right)$

Now that we have developed the first step of the method, we will proceed to develop the second step on the following page.

STEP TWO

$$U^{n+1} = U^n + \Delta t\, U_t^n + \frac{1}{2}\Delta t^2 \tilde{U}_{tt}^n$$

we know: $U_t^n = \cancelto{0}{S^n} - a\nabla U^n$

$$\tilde{U}_{tt}^n = \cancelto{0}{S_t^n} - \cancelto{0}{a\nabla S_n} + (a\nabla)^2 \tilde{U}^n$$

from Step 1 we know: $\tilde{U}^n = U^n + \frac{1}{3}\Delta t\left(-a\nabla U^n\right) + \alpha \Delta t^2 (a\nabla)^2 U^n$

so we can say...

$$\tilde{U}_{tt}^n = (a\nabla)^2 \tilde{U}^n = (a\nabla)^2\left[U^n + \frac{1}{3}\Delta t\left(-a\nabla U^n\right) + \alpha\Delta t^2 (a\nabla)^2 U^n\right]$$

$$\tilde{U}_{tt}^n = (a\nabla)^2 U^n + 0 + 0$$

these terms are zero because orders of 3 or higher go to zero

$$\therefore\ \tilde{U}_{tt}^n = (a\nabla)^2 U^n$$

and plugging back into our original equation gives us...

$$U^{n+1} = U^n - \Delta t\, a\nabla U^n + \frac{1}{2}a^2\Delta t^2\, \nabla^2 U^n$$

$$\int w U^{n+1} = \int w U^n - a\Delta t \int w\nabla U^n + \frac{1}{2}a^2\Delta t^2 \int w\nabla^2 U^n$$

applying galerkins

integrating by parts...

$$-a\Delta t\int w\nabla U^n = +a\Delta t\left[U^n \nabla w\right]_{\Gamma_{out}}$$

$$\frac{1}{2}a^2\Delta t^2 \int w\nabla^2 U^n = -\frac{1}{2}a^2\Delta t^2\left[\nabla w \nabla U^n\right]_{\Gamma_{out}}$$

Finally..

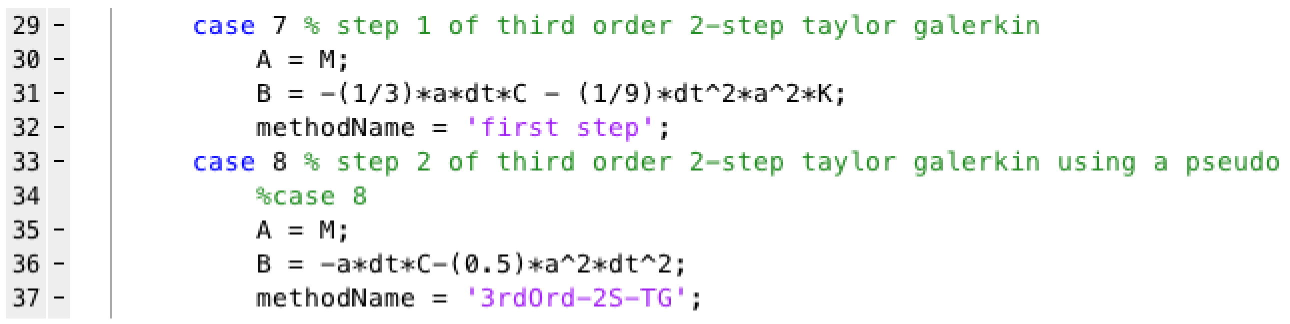
$$N_i N_j U^{n+1} = N_i N_j U^n + \Delta t\, a\, N_j N_{xi} U^n - \frac{1}{2}a^2\Delta t^2 N_{xi} N_{xj} U^n$$

Rewriting.

$$\boxed{M\Delta U = -a\Delta t\, C U^n - \frac{1}{2}a^2\Delta t^2 K U^n}$$

$$A = M$$
$$B = -\left(a\Delta t\, C - \frac{1}{2}a^2\Delta t^2 K\right)$$

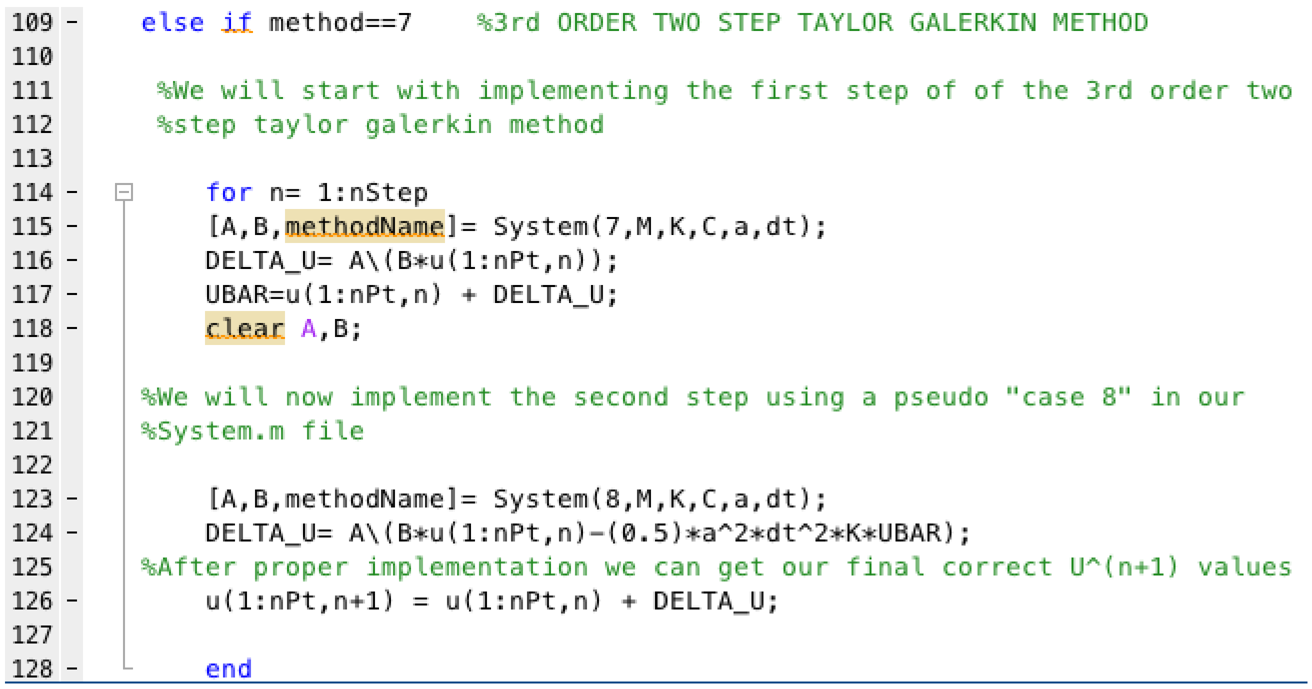
Now that we have developed both steps of the method, It is time to implement these steps in the codes provided. These implementations can be seen on the following pages. It is worth noting that the value is given and is $\alpha = (1/9)$.



```
    case 7 % step 1 of third order 2-step taylor galerkin
        A = M;
        B = -(1/3)*a*dt*C - (1/9)*dt^2*a^2*K;
        methodName = 'first step';
    case 8 % step 2 of third order 2-step taylor galerkin using a pseudo
        %case 8
        A = M;
        B = -a*dt*C-(0.5)*a^2*dt^2;
        methodName = '3rdOrd-2S-TG';
```

Figure 5: Code Modification to System.m for Third Order, Two Step Taylor Galerkin method

Here we have implemented all the A and B matrices that were derived on the previous pages into the System.m file. It is worth noting that 2 steps are being used and that the values of A and B are cleared and overwritten before moving on to the second step.



```
else if method==7    %3rd ORDER TWO STEP TAYLOR GALERKIN METHOD

 %We will start with implementing the first step of of the 3rd order two
 %step taylor galerkin method

    for n= 1:nStep
    [A,B,methodName]= System(7,M,K,C,a,dt);
    DELTA_U= A\(B*u(1:nPt,n));
    UBAR=u(1:nPt,n) + DELTA_U;
    clear A,B;

%We will now implement the second step using a pseudo "case 8" in our
%System.m file

    [A,B,methodName]= System(8,M,K,C,a,dt);
    DELTA_U= A\(B*u(1:nPt,n)-(0.5)*a^2*dt^2*K*UBAR);
%After proper implementation we can get our final correct U^(n+1) values
    u(1:nPt,n+1) = u(1:nPt,n) + DELTA_U;

    end
```

Figure 6: Code Modification to Main.m for Third Order, Two Step Taylor Galerkin method

On the following page we will once again plot this method with different values of the Courant number by varying the amount of time steps. We will subsequently analyze and comment on the results.

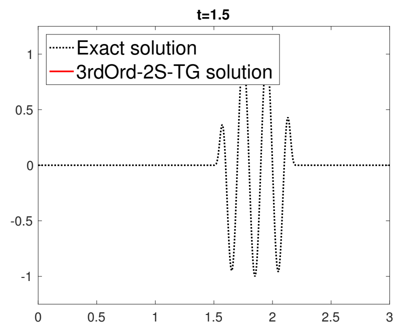


Figure 7a: Courant Number = 2

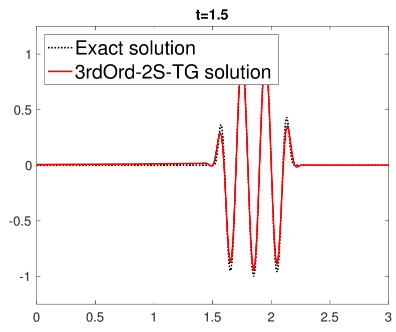


Figure 7b: Courant Number = 0.5

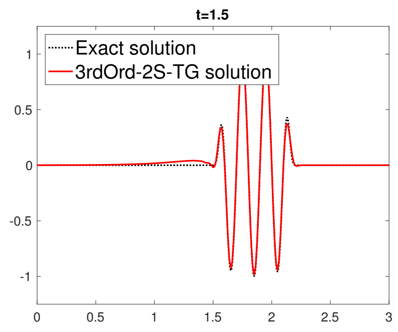


Figure 7c: Courant Number = 0.125

Here are the results from running problem 2 with the Third Order, Two Step Taylor Galerkin method at varying Courant numbers by changing the number of time steps. Figure 7a was run with 60 time steps resulting in a Courant number of 2. This plot exhibits such high inaccuracy and instability, that the plot is not even visible relative to the exact solution. It is worth noting that as the code was plotting, the oscillations were visible and appeared to grow with time. Since the Courant number is much higher than its allowable value of $C^2 \leq (3/4)$, this behavior is not out of the ordinary and is to be expected. Figure 7b was run at the default number of 240 time steps resulting in a Courant number of 0.5. This Courant number is within the threshold for stability of $C^2 \leq (3/4)$ and therefore behaves accurately and exhibits stability. It is worth noting that there are some slight inaccuracies at the first and last oscillation peaks of the exact solution similar to how the Third Order, Single Step Method performed. Figure 7c was run with 960 time steps resulting in a Courant number of 0.125. This Courant number is very low so the high degree of accuracy and stability exhibited by the method in the figure is expected. In conclusion, the implemented Third Order, Two Step Taylor Galerkin method is correct and performs as expected.