Project 4

Due on Monday, June 9th, 2017 (Three weeks)

Consider the following time-dependent equation:

$$\frac{\partial u}{\partial t} = \epsilon^2 \frac{\partial^2 u}{\partial x^2} + (1-u^2)u + f(x,t), \quad x \in [-1,1],\ t > 0 \tag{1}$$

with boundary conditions $u(-1,t) = -1$, $u(1,t) = 1$, and initial condition

$$u(x,0) = \cos\left(5\pi \frac{x-1}{2}\right).$$

The function f is to be chosen later. Discretizing the right hand side with second order centered differences, and defining

$$D_+D_-u_i = \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}, \tag{2}$$

we get

$$\left(\epsilon^2 \frac{\partial^2 u}{\partial x^2} + (1-u^2)u\right)(x_i) = \epsilon^2 D_+D_-u_i + (1-u_i^2)u_i \tag{3}$$

Consider the following fully implicit discretization:

$$\frac{3u_i^{n+1} - 4u_i^n + u_i^{n-1}}{2\Delta t} = \epsilon^2 D_+D_-u_i^{n+1} + (1-(u_i^{n+1})^2)u_i^{n+1} + f(x_i, t_{n+1}) \tag{4}$$

1. Show that the numerical scheme is second order accurate in time and space.

2. Choose an appropriate function $f(x,t)$ so that $u(x,t) = \cos(t)\cos(5\pi(x-1)/2)$ is the actual solution, with boundary conditions $u(-1,t) = -\cos(t)$, and $u(1,t) = \cos(t)$.

3. Using the solution computed in the previous part, show that the numerical scheme is second order in time and space by implementing the method, using Newton's method to solve the nonlinear system of equations, using $\epsilon = 1$.

4. Now set $f = 0$, and consider the original boundary and initial conditions. Solve the equation up to time $T = 1$. Plot the solution at time

intervals $\Delta T = 0.1$, so that you can get an idea of the actual evolution. Use $\epsilon = 0.005$. Solve the equation for several values of Δx and Δt to make sure your results don't change anymore when changing the grid size, or the time step. Plot the results for the different values to illustrate the convergence of the method.

Solution.

1. Show that the numerical scheme is second order accurate in time and space.

Consider the fully implicit numerical scheme as follows,

$$\frac{3u_i^{n+1} - 4u_i^n + u_i^{n-1}}{2\Delta t} = \epsilon^2 D_+ D_- u_i^{n+1} + (1 - (u_i^{n+1})^2)u_i^{n+1} + f(x_i, t_{n+1})$$

i.e.,

$$\begin{aligned}\frac{3u_i^{n+1} - 4u_i^n + u_i^{n-1}}{2\Delta t} &= \epsilon^2 \frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{\Delta x^2} \\ &+ (1 - (u_i^{n+1})^2)u_i^{n+1} + f(x_i, t_{n+1}).\end{aligned}$$

The local truncation error at (x_i, t_{n+1}) is given as

$$\begin{aligned}\tau_i^{n+1} = &\frac{3u(x_i, t_{n+1}) - 4u(x_i, t_n) + u(x_i, t_{n-1})}{2\Delta t} \\ &- \epsilon^2 \frac{u(x_{i+1}, t_{n+1}) - 2u(x_i, t_{n+1}) + u(x_{i-1}, t_{n+1})}{\Delta x^2} \\ &- (1 - u^2(x_i, t_{n+1}))u(x_i, t_{n+1}) - f(x_i, t_{n+1}).\end{aligned} \tag{5}$$

Using Taylor series expansions around (x_i, t_{n+1}) we have

$$u(x_i, t_n) = u(x_i, t_{n+1}) - \Delta t\, u_t(x_i, t_{n+1}) + \frac{(\Delta t)^2}{2} u_{tt}(x_i, t_{n+1}) + O((\Delta t)^3) \tag{6}$$

$$u(x_i, t_{n-1}) = u(x_i, t_{n+1}) - 2\Delta t\, u_t(x_i, t_{n+1}) + \frac{(2\Delta t)^2}{2} u_{tt}(x_i, t_{n+1}) + O((\Delta t)^3) \tag{7}$$

$$\begin{aligned}u(x_{i+1}, t_{n+1}) = &u(x_i, t_{n+1}) + \Delta x\, u_x(x_i, t_{n+1}) + \frac{(\Delta x)^2}{2} u_{xx}(x_i, t_{n+1}) \\ &+ \frac{(\Delta x)^3}{3!} u_{xxx}(x_i, t_{n+1}) + O((\Delta x)^4)\end{aligned} \tag{8}$$

$$u(x_{i-1},t_{n+1}) = u(x_i,t_{n+1}) - \Delta x\, u_x(x_i,t_{n+1}) + \frac{(\Delta x)^2}{2} u_{xx}(x_i,t_{n+1}) - \frac{(\Delta x)^3}{3!} u_{xxx}(x_i,t_{n+1}) + O((\Delta x)^4) \quad (9)$$

Substituting (6), (7), (8) and (9) in (5) we get

$$\tau_i^{n+1} = u_t(x_i,t_{n+1}) + O((\Delta t)^2) - \epsilon^2 u_{xx}(x_i,t_{n+1}) + O((\Delta x)^2) - (1-u^2(x_i,t_{n+1}))u(x_i,t_{n+1}) - f(x_i,t_{n+1}).$$

Since u is the exact solution, we have

$$\frac{\partial u}{\partial t} = \epsilon^2 \frac{\partial^2 u}{\partial x^2} + (1-u^2)u + f(x,t),$$

so, we have

$$\tau_i^{n+1} = O((\Delta t)^2 + (\Delta x)^2).$$

Thus, the fully implicit numerical scheme is consistent and it is second order accurate in time and space. If we choose $\Delta t = \Delta x$ then the scheme will be second order overall in both space and time. If we make $\Delta t \ll \Delta x$ then it is wasteful in the sense that the dominant error will be $O(\Delta x^2)$.

2. Choose an appropriate function $f(x,t)$ so that $u(x,t) = \cos(t)\cos(5\pi(x-1)/2)$ is the actual solution, with boundary conditions $u(-1,t) = -\cos(t)$, and $u(1,t) = \cos(t)$.

   Consider $u(x,t) = \cos(t)\cos(5\pi(x-1)/2)$ solves the equation:

$$\frac{\partial u}{\partial t} = \epsilon^2 \frac{\partial^2 u}{\partial x^2} + (1-u^2)u + f(x,t), \quad x \in [-1,1],\ t > 0$$

   with boundary conditions $u(-1,t) = -\cos(t)$, and $u(1,t) = \cos(t)$. Then,

$$\begin{aligned} f(x,t) &= \frac{\partial u}{\partial t} - \epsilon^2 \frac{\partial^2 u}{\partial x^2} - (1-u^2)u \\ &= -\sin(t)\cos(\frac{5\pi(x-1)}{2}) + \left(\frac{25\epsilon^2\pi^2}{4} - 1\right)\cos(t)\cos(\frac{5\pi(x-1)}{2}) \\ &+ \cos^3(t)\cos^3(\frac{5\pi(x-1)}{2}). \end{aligned}$$

3. Using the solution computed in the previous part, show that the numerical scheme is second order in time and space by implementing the method, using Newton's method to solve the nonlinear system of equations, using $\epsilon = 1$.

   To construct the algorithm that led to an appropriate fixed-point method in the one dimensional case, we found a function ϕ with the property that

   $$g(x) = x - \phi(x)f(x)$$

   gives quadratic convergence to the fixed point p of the function g. From this condition Newton's method evolved by choosing $\phi(x) = \frac{1}{f'(x)}$, assuming that $f'(x) \neq 0$.

   A similar approach in the n dimensional case involves a matrix

   $$\mathbf{A}(\mathbf{x}) = \begin{pmatrix} a_{11}(\mathbf{x}) & a_{12}(\mathbf{x}) & \cdots & a_{1n}(\mathbf{x}) \\ a_{21}(\mathbf{x}) & a_{22}(\mathbf{x}) & \cdots & a_{2n}(\mathbf{x}) \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1}(\mathbf{x}) & a_{n2}(\mathbf{x}) & \cdots & a_{nn}(\mathbf{x}) \end{pmatrix}$$

   where each of the entries $a_{ij}(\mathbf{x})$ is a function from $\mathbb{R}^n$ into $\mathbb{R}$. This requires that $\mathbf{A}(\mathbf{x})$ be found so that

   $$\mathbf{G}(\mathbf{x}) = \mathbf{x} - \mathbf{A}(\mathbf{x})^{-1}\mathbf{F}(\mathbf{x})$$

   gives quadratic convergence to the solution of $\mathbf{F}(\mathbf{x}) = \mathbf{0}$, assuming that $\mathbf{A}(\mathbf{x})$ is nonsingular at the fixed point $\mathbf{p}$ of $\mathbf{G}$.

   Define the matrix $\mathbf{J}(\mathbf{x})$ by

   $$\mathbf{J}(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \frac{\partial f_1}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \frac{\partial f_2}{\partial x_1}(\mathbf{x}) & \frac{\partial f_2}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_2}{\partial x_n}(\mathbf{x}) \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_n}{\partial x_1}(\mathbf{x}) & \frac{\partial f_n}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_n}{\partial x_n}(\mathbf{x}) \end{pmatrix}$$

   An appropriate choice for $\mathbf{A}(\mathbf{x})$ is, consequently, $\mathbf{A}(\mathbf{x}) = \mathbf{J}(\mathbf{x})$. The function $\mathbf{G}$ is defined by

   $$\mathbf{G}(\mathbf{x}) = \mathbf{x} - \mathbf{J}(\mathbf{x})^{-1}\mathbf{F}(\mathbf{x})$$

and the functional iteration procedure evolves from selecting $\mathbf{x}^{(0)}$ and generating, for $k \geq 1$,

$$\mathbf{x}^{(k)} = \mathbf{G}(\mathbf{x}^{(k-1)}) = \mathbf{x}^{(k-1)} - \mathbf{J}(\mathbf{x}^{(k-1)})^{-1}\mathbf{F}(\mathbf{x}^{(k-1)}).$$

This is called Newton's method for nonlinear systems, and it is generally expected to give quadratic convergence, provided that a sufficiently accurate starting value is known and that $\mathbf{J}(\mathbf{p})^{-1}$ exists. The matrix $\mathbf{J}(\mathbf{x})$ is called the Jacobian matrix.

A weakness in Newton's method arises from the need to compute and invert the matrix $\mathbf{J}(\mathbf{x})$ at each step. In practice, explicit computation of $\mathbf{J}(\mathbf{x})^{-1}$ is avoided by performing the operation in a two step manner. First, a vector $\mathbf{y}$ is found that satisfies $\mathbf{J}(\mathbf{x}^{(k-1)})\mathbf{y} = -\mathbf{F}(\mathbf{x}^{(k-1)})$. Then the new approximation, $\mathbf{x}^{(k)}$, is obtained by adding $\mathbf{y}$ to $\mathbf{x}^{(k-1)}$. Algorithm for Newton's Method is as follows.

To approximate the solution of the nonlinear system $\mathbf{F}(\mathbf{x}) = \mathbf{0}$ given an initial approximation $\mathbf{x}$:

```
INPUT   number $n$ of equations and unknowns; initial approximation $\mathbf{x} = (x_1, \ldots, x_n)^T$,
        tolerance ($TOL$); maximum number of iterations $N$.

OUTPUT  approximate solution $\mathbf{x} = (x_1, \ldots, x_n)^T$ or a message that the number
        of iterations was exceeded.

Step 1  Set $k = 1$.
Step 2  While ($k \leq N$) do Steps $3 - 7$.
Step 3  Calculate $\mathbf{F}(\mathbf{x})$ and $\mathbf{J}(\mathbf{x})$, where $\mathbf{J}(\mathbf{x})_{ij} = (\frac{\partial f_i(\mathbf{x})}{\partial x_j})$ for $1 \leq i, j \leq n$.
Step 4  Solve the $n \times n$ linear system $\mathbf{J}(\mathbf{x})\mathbf{y} = -\mathbf{F}(\mathbf{x})$.
Step 5  set $\mathbf{x} = \mathbf{x} + \mathbf{y}$.
Step 6  If $\|\mathbf{y}\| <$ TOL then OUTPUT($\mathbf{x}$);(the procedure was successful), STOP.
Step 7  Set $k = k + 1$.
Step 8  OUTPUT (Maximum number of iterations exceeded); (The procedure
        was unsuccessful).STOP.
```

One option for solving this nonlinear system is to use the Newton's method. Typically one applies the method by solving the linear system

$$\mathbf{J}(\mathbf{x}^{(k)})(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = -\mathbf{F}(\mathbf{x}^{(k)}).$$

A disadvantage to Newton's method is that the Jacobian changes at each iteration so a different linear system must be solved for each k. The advantage is that if it converges, it typically converges quadratically so rapid convergence is achieve. If explicit formulas for the partial derivatives are not available then finite differences can be used to estimate the derivatives.

Bringing the known terms to the right hand side of the equation, using $\epsilon = 1$, and letting $\lambda = \frac{\Delta t}{\Delta x^2}$ produces the difference scheme

$$\begin{aligned} -2\lambda u_{i+1}^{n+1} + (3+4\lambda)u_i^{n+1} - 2\lambda u_{i-1}^{n+1} - 2\Delta t(1-(u_i^{n+1})^2)u_i^{n+1} \\ = 4u_i^n - u_i^{n-1} + 2\Delta t f_i^{n+1}. \end{aligned} \tag{10}$$

Note that this is a nonlinear difference equation due to the term $(1 - (u_i^{n+1})^2)u_i^{n+1}$, so the difference equations cant be written as a linear system $\mathbf{Ax} = \mathbf{f}$. Consequently we must use the above Newton's method to linearize this problem and we expect that we will have to perform an iteration at each time step to solve these nonlinear equations. To use the Newton method we must be able to compute the Jacobian. Recall that Newtons method will converge in one iteration for a linear system so in that case the Jacobian is just the same as the coefficient matrix. This means that all of the linear terms in our equation will be the same as in the linear case.

The $(k-1)$-step iteration Jacobian matrix for system (10) is that

$$\mathbf{J}(\mathbf{x}^{(\mathbf{k-1})}) = \left(\begin{array}{ccccc} 3+4\lambda-2\Delta t[1-3(x_0^{(k-1)})^2] & -2\lambda & 0 & \cdots & 0 \\ -2\lambda & 3+4\lambda-2\Delta t[1-3(x_1^{(k-1)})^2] & -2\lambda & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & -2\lambda & 3+4\lambda-2\Delta t[1-3(x_m^{(k-1)})^2] & -2\lambda \\ 0 & \cdots & 0 & -2\lambda & 3+4\lambda-2\Delta t[1-3(x_{m+1}^{(k-1} \end{array}\right.$$

To solve the above difference scheme using the Newton's Method, we enclose the code as follows,

```
%
% project 4
% full implicit scheme
% Question 3 newton_method
clear,clc
intervals=2*10^4;k=1/10;
% parameters
m=intervals-1;
h=2/(m+1);
r=k/(h^2);
T=1;
% spatial mesh
x=(-1:h:1)';
% temporal mesh
nsteps=floor(T/k);
% initial condition
u=zeros(m+2,nsteps+1);
f=zeros(m+2,nsteps+1);
F=zeros(m,nsteps+1);
u(2:m+1,1)=cos(5*pi*(x(2:m+1)-ones(m,1))/2);
u(2:m+1,2)=cos(k)*cos(5*pi*(x(2:m+1)-ones(m,1))/2);
% boundary condition
u(1,:)=-cos((0:nsteps)*k)';
u(m+2,:)=cos((0:nsteps)*k)';
% f(x,t)
f(1:m+2,3:nsteps+1)=-cos(5*pi*(x-1)/2)*sin((2:nsteps)*k)+...
(25*pi^2/4-1)*cos(5*pi*(x-1)/2)*cos((2:nsteps)*k)...
+(cos(5*pi*(x-1)/2)).^3*(cos((2:nsteps)*k)).^3;
% preceed the iterations
for n=2:nsteps
F(1,n)=-2*r*u(3,n)+(3+4*r)*u(2,n)-...
2*r*u(1,n+1)-2*k*(1-(u(2,n)).^2).*(u(2,n))...
-4*u(2,n)+u(2,n-1)-2*k*f(2,n+1);
F(m,n)=-2*r*u(m+2,n+1)+(3+4*r)*u(m+1,n)-...
2*r*u(m,n)-2*k*(1-(u(m+1,n)).^2).*(u(m+1,n))...
-4*u(m+1,n)+u(m+1,n-1)-2*k*f(m+1,n+1);
F(2:m-1,n)=-2*r*u(4:m+1,n)+(3+4*r)*u(3:m,n)-...
2*r*u(2:m-1,n)-2*k*(ones(m-2,1)-(u(3:m,n)).^2).*(u(3:m,n))...
-4*u(3:m,n)+u(3:m,n-1)-2*k*f(3:m,n+1);
% Jacobi matrix
a=-2*r*ones(m,1);
```

```
b(1:m)=(3+4*r)*ones(m,1)-...
    2*k*(ones(m,1)-3*(u(2:m+1,n)).^2);
J=spdiags([a,b',a],[-1,0,1],m,m);
% next time step
u(2:m+1,n+1)=u(2:m+1,n)+J\(-F(1:m,n));
end
%
% plot numerical solution
t=0:k:nsteps*k;
figure(1)
[T,X]=meshgrid(t,x);
mesh(T,X,u),hold on,
xlabel('t'),ylabel('x'),zlabel('u')
title('numerical solution,Xie Changjian,2017/6/10')
% exact solution
uk=zeros(m+2,nsteps+1);
uk(1:m+2,1:nsteps+1)=cos(5*pi*(x-1)/2)...
    *cos((0:nsteps)*k);
% plot actual solution
figure(2)
[T,X]=meshgrid(t,x);
mesh(T,X,uk),hold on,
xlabel('t'),ylabel('x'),zlabel('u')
title('exact solution,Xie Changjian,2017/6/10')
% plot the last time step solution
figure(3)
plot(x,u(:,nsteps+1),'k--'),hold on,grid on
plot(x,uk(:,nsteps+1),'k.-'),hold on
legend('num','exact'),hold on
xlabel('x'),ylabel('u'),
title('the last time step iteration'),
% find the error
format long
err=norm(u(:,nsteps+1)-uk(:,nsteps+1),inf)
figure(4)
% the final time step solution, find the order of err
dt=[1/10 1/20 1/40 1/80 1/160]; %dx=2/10^4;
err=[3.030871473064245*10^(-4) 7.641035586880562*10^(-5)...
```

```
    1.916788730826902*10^(-5) 4.829755135271618*10^(-6)...
    1.244429739699626*10^(-6)];
loglog(dt,err,'k*-'),hold on,grid on
xlabel('\Deltat (log)'),ylabel('err (log)')
title('the order of err w.r.t \Deltat'),
% slope_dt=1.9839954399069232859886824371642
% intercept_dt=-3.5374209204112583293522220628802
% dt data fitting
r=sum((log(dt)).^2);
s=sum(log(dt));
t=sum(log(dt).*(log(err)));
p=sum(log(err));
slope_dt=(p*s-5*t)/(s^2-5*r);
% Cx=d
C=[sum((log(dt)).^2),sum(log(dt));sum(log(dt)),5];
d=[sum((log(dt)).*(log(err)));sum(log(err))];
xxx=C\d;
slope_dt=xxx(1);
intercept_dt=xxx(2);
slope_dt=vpa(slope_dt),intercept_dt=vpa(intercept_dt)
figure(5)
dx=[2/40 2/80 2/120 2/160 2/200];%dt=1/10^4;
err=[0.012386398597796 0.003090139709516...
    0.001374115855245 7.725349703339157*10^(-4)...
    4.942946686625760*10^(-4)];
loglog(dx,err,'k-s'),hold on,grid on
xlabel('\Deltax (log)'),ylabel('err (log)')
title('the order of err w.r.t \Deltax'),
% slope_dx=2.0012961525522112715691491757752
% intercept_dx=1.6038148252019415185287698477623
% dt data fitting
r=sum((log(dx)).^2);
s=sum(log(dx));
t=sum(log(dx).*(log(err)));
p=sum(log(err));
slope_dx=(p*s-5*t)/(s^2-5*r);
% Cx=d
C=[sum((log(dx)).^2),sum(log(dx));sum(log(dx)),5];
```

```
d=[sum((log(dx)).*(log(err)));sum(log(err))];
xx=C\d;
slope_dx=xx(1);
intercept_dx=xx(2);
slope_dx=vpa(slope_dx),intercept_dx=vpa(intercept_dx)
```

The numerical solution is the following result,

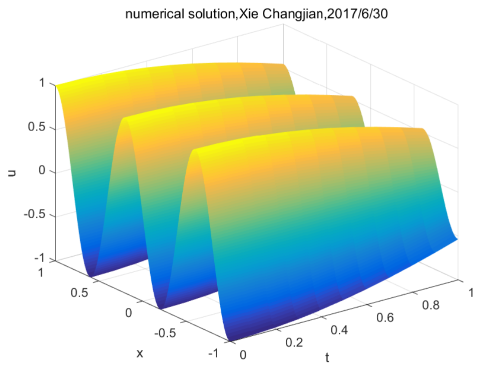

Figure 1: Using the space interval $\Delta x = 10^{-4}$, the time interval $\Delta t = 0.1$. Do a plot of the numerical solution using Newton's method, up to time $T = 1$.

We can also do the actual solution as follows,

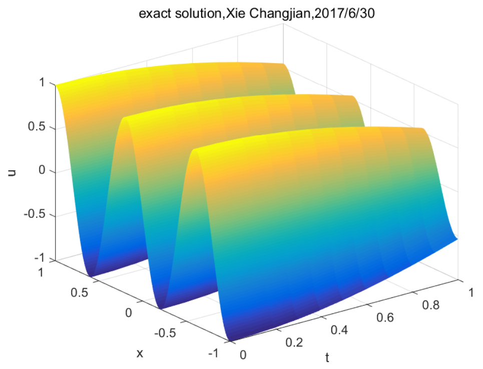

Figure 2: The actual solution $u(x,t) = \cos(t)\cos(5\pi\frac{(x-1)}{2})$.

From the calculation, we can also deserve the order of space and time using the grid encryption, i.e., if we know that the error is $e(\Delta x, \Delta t) = O(\Delta x^p + \Delta t^q)$. In order to find the spatial discrete format convergence order p, we adopt grid successive encryption method. When we use different sizes of grid computing, we can get the error function with the grid size of the relationship, we use the step $(\Delta x, \Delta t), \frac{1}{2}(\Delta x, \Delta t), \frac{1}{4}(\Delta x, \Delta t)$ to compute the result. We deserve $e(\Delta x), e(\frac{1}{2}\Delta x), e(\frac{1}{4}\Delta x)$, due to $e(\Delta x) = C(\Delta x^p)$, then

$$\frac{e(\Delta x)}{e(\frac{1}{2}\Delta x)} = \frac{C\Delta x^p}{C(\frac{\Delta x}{2})^p},$$

so, we estimate the accurate p as follows,

$$p = \frac{\log(e(\Delta x)/e(\frac{1}{2}\Delta x))}{\log(2)}.$$

we take $\Delta x = 0.1, \Delta t = 10^{-4}$, we deserve the error $e(\Delta x) = 0.050568895517941$, then we take $\Delta x = \frac{0.1}{2}, \Delta t = 10^{-4}$, then, the error is $e(\frac{\Delta x}{2}) = 0.012386398597796$. Then, we deserve that

$$p = \frac{\log(e(\Delta x)/e(\frac{1}{2}\Delta x))}{\log(2)} = 2.029493490464663 \approx 2.$$

we take $\Delta x = 10^{-4}, \Delta t = 0.1$, we deserve the error $e(\Delta t) = 3.030871473064245 \times 10^{-4}$, then we take $\Delta x = 10^{-4}, \Delta t = \frac{0.1}{2}$, then, the error is $e(\frac{\Delta t}{2}) = 7.641035586880562 \times 10^{-5}$. Then, we deserve that

$$q = \frac{\log(e(\Delta t)/e(\frac{1}{2}\Delta t))}{\log(2)} = 1.987892590009570 \approx 2.$$

Then the numerical scheme is second order in time and space.

In order to observe clearly, we list the table and give the log-log plot of the error as the function of the grid size or the time step as follows,

For fixed $\Delta x = \frac{2}{10^4}$, the relationship between the error and the time step is as follows,

Δt	$\frac{1}{10}$	$\frac{1}{20}$	$\frac{1}{40}$	$\frac{1}{80}$	$\frac{1}{160}$
err	3.03087×10^{-4}	7.641×10^{-5}	1.9167887×10^{-5}	4.82×10^{-6}	1.24×10^{-6}

For fixed $\Delta t = \frac{1}{10^4}$, the relationship between the error and the grid size is as follows,

Δx	$\frac{2}{40}$	$\frac{2}{80}$	$\frac{2}{120}$	$\frac{2}{160}$	$\frac{2}{200}$
err	0.012386	0.0030901397	0.001374	7.7253497×10^{-4}	4.9429×10^{-4}

For fixed $\Delta x = \frac{2}{10^4}$, the log-log plot of the error as the function of grid size is as follows,

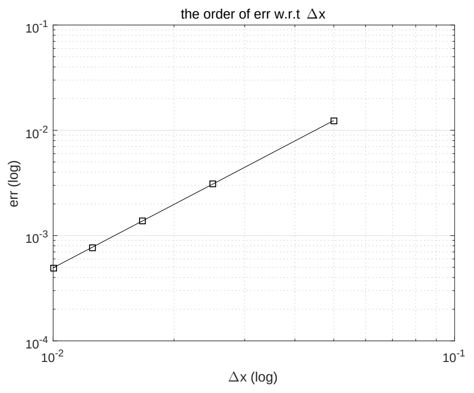

Figure 3: Do a log-log plot of the error and the grid size.

For fixed $\Delta t = \frac{1}{10^4}$, the log-log plot of the error as the function of time step is as follows,

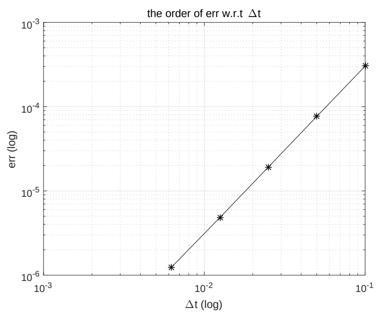

Figure 4: Do a log-log plot of the error and the time step.

4. Consider the difference scheme as follows,

$$\frac{3u_i^{n+1}-4u_i^n+u_i^{n-1}}{2\Delta t}=\epsilon^2 D_+D_-u_i^{n+1}+(1-(u_i^{n+1})^2)u_i^{n+1}+f(x_i,t_{n+1})$$

i.e.,

$$-2\lambda\epsilon^2u_{i+1}^{n+1}+(3+4\lambda\epsilon^2)u_i^{n+1}-2\lambda\epsilon^2u_{i-1}^{n+1}-2\Delta t(1-(u_i^{n+1})^2)u_i^{n+1}=4u_i^n-u_i^{n-1}+2\Delta t f_i^{n+1}.$$

Now we set $f=0$, and consider the original boundary and initial conditions, i.e., we consider the boundary conditions $u(-1,t)=-1$, $u(1,t)=1$, and initial condition

$$u(x,0)=\cos\left(5\pi\frac{x-1}{2}\right).$$

Solve the equation up to time $T=1$. We Plot the solution at time intervals $\Delta T=0.1$, so that you can get an idea of the actual evolution. Here, we use $\epsilon=0.005$ and we solve the equation for several values of Δx and Δt to make sure our results don't change anymore when changing the grid size, or the time step. We plot the results for the different values to illustrate the convergence of the method.

The matlab code is as follows,

```
%
% project 4
% full implicit scheme
% Question 4
clear,clc
intervals=10^4;k=1/10;
% parameters
m=intervals-1;
h=2/(m+1);
r=k/(h^2);
T=1;
epsilon=0.005;
% spatial mesh
x=(-1:h:1)';
% temporal mesh
```

```
nsteps=floor(T/k);
% initial condition
u=zeros(m+2,nsteps+1);
F=zeros(m,nsteps+1);
u(2:m+1,1)=cos(5*pi*(x(2:m+1)-ones(m,1))/2);
u(2:m+1,2)=cos(5*pi*(x(2:m+1)-ones(m,1))/2);
% boundary condition
u(1,:)=-ones(nsteps+1,1);
u(m+2,:)=ones(nsteps+1,1);
% preceed iterations
for n=2:nsteps
F(1,n)=-2*r*(epsilon)^2*u(3,n)+(3+4*r*(epsilon)^2)*u(2,n)-...
2*r*(epsilon)^2*u(1,n+1)-2*k*(1-(u(2,n)).^2).*(u(2,n))...
-4*u(2,n)+u(2,n-1);
F(m,n)=-2*r*(epsilon)^2*u(m+2,n+1)+(3+4*r*(epsilon)^2)*u(m+1,n)-...
2*r*(epsilon)^2*u(m,n)-2*k*(1-(u(m+1,n)).^2).*(u(m+1,n))...
-4*u(m+1,n)+u(m+1,n-1);
F(2:m-1,n)=-2*r*(epsilon)^2*u(4:m+1,n)+(3+4*r*(epsilon)^2)*u(3:m,n)-...
2*r*(epsilon)^2*u(2:m-1,n)-2*k*(ones(m-2,1)-(u(3:m,n)).^2).*(u(3:m,n))...
-4*u(3:m,n)+u(3:m,n-1);
% Jacobi matrix
a=-2*r*(epsilon)^2*ones(m,1);
b(1:m)=(3+4*r*(epsilon)^2)*ones(m,1)-...
    2*k*(ones(m,1)-3*(u(2:m+1,n)).^2);
J=spdiags([a,b',a],[-1,0,1],m,m);
% next time step
u(2:m+1,n+1)=u(2:m+1,n)+J\(-F(1:m,n));
end
%
% plot numerical solution
t=0:k:nsteps*k;
figure(1)
[T,X]=meshgrid(t,x);
mesh(T,X,u),hold on,
xlabel('t'),ylabel('x'),zlabel('u')
title('numerical solution,Xie Changjian,2017/6/28')
```

the result for $\Delta t = 0.1$ is as follows,

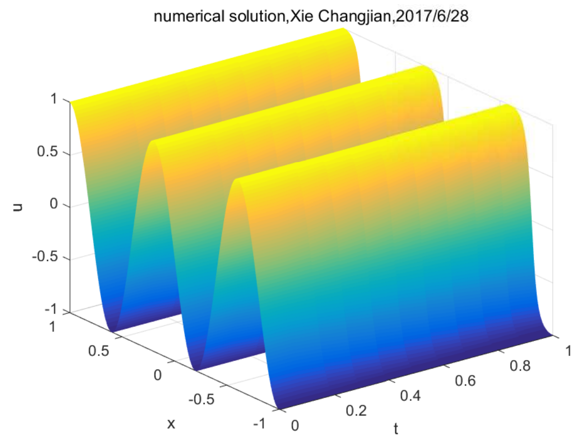

Figure 5: The solution at the time interval $\Delta t = 0.1$. Do a plot of the numerical solution up to time $T = 1$.

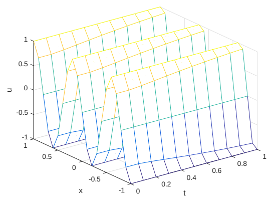

Figure 6: The grid size is $\Delta x = \frac{2}{20}$, the time step $\Delta t = \frac{1}{10}$.

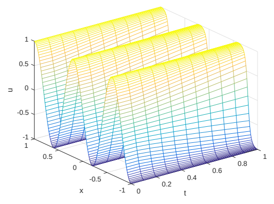

Figure 7: The grid size is $\Delta x = \frac{2}{200}$, the time step $\Delta t = \frac{1}{10}$.

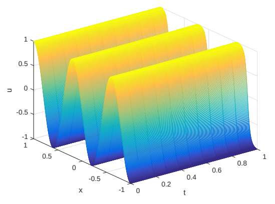

Figure 8: The grid size is $\Delta x = \frac{2}{2000}$, the time step $\Delta t = \frac{1}{10}$.

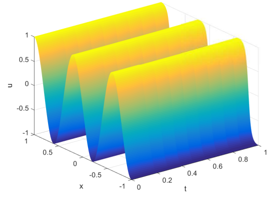

Figure 9: The grid size is $\Delta x = \frac{2}{2\times 10^4}$, the time step $\Delta t = \frac{1}{10}$.

From these above four figures, we know the result don't change anymore when changing the grid size.

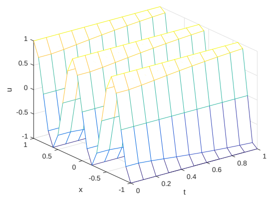

Figure 10: The grid size is $\Delta x = \frac{2}{20}$, the time step $\Delta t = \frac{1}{10}$.

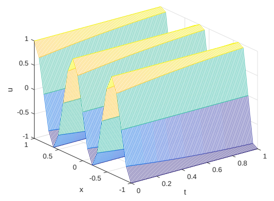

Figure 11: The grid size is $\Delta x = \frac{2}{20}$, the time step $\Delta t = \frac{1}{100}$.

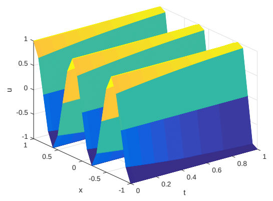

Figure 12: The grid size is $\Delta x = \frac{2}{20}$, the time step $\Delta t = \frac{1}{10^3}$.

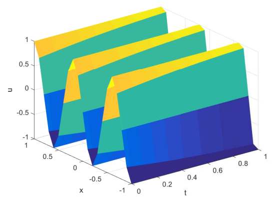

Figure 13: The grid size is $\Delta x = \frac{2}{20}$, the time step $\Delta t = \frac{1}{10^4}$.

From these above four figures, we know the result don't change anymore when changing the time step.

In conclusion, we deserve that the full implicit scheme is unconditionally stable, then, it's convergent to the exact solution. As we all know, the Newton's method is quickly convergent by the rate of square. We can obtain the actual solution by the mesh refinement. Then, we compare the numerical solution with the exact one. As $k \to \infty$,

$$\frac{\|u^{(k+1)} - u_{\text{exact}}\|_\infty}{\|u^{(k)} - u_{\text{exact}}\|_\infty^2}$$

is exsist.

Then, the accuracy of the above numerical scheme is high, and we can think of any other scheme for this above famous equation, and do some discussions.