Danmarks Tekniske Universitet

02685 Assignment 03

Time-dependent Partial Differential Equations

Arturo Arranz Mateo (s160412)
Manuel Montoya Catala (s162706)

April 2017

Contents

1 Exercise 1: Purely parabolic model problem

1.1 The θ-scheme

For solving the one-dimensional unsteady heat diffusion problem the θ-scheme discretization is proposed:

$$\frac{U_j^{n+1} - U_j^n}{k} = \frac{\kappa}{h^2}((1-\theta)\delta_x^2 U_j^n + \theta\delta_x^2 U_j^{n+1}), \qquad \delta_x^2 U_j^n = U_{j-1}^n - 2U_j^n + U_{j+1}^n \tag{1}$$

where, k and h are the time and space step sizes respectively. Rearranging terms

$$\begin{aligned} U_j^{n+1} &= \frac{k\kappa}{h^2}((1-\theta)\delta_x^2 U_j^n + \theta\delta_x^2 U_j^{n+1}) + U_j^n \\ (1 - \mu\theta\delta_x^2)U_j^{n+1} &= (1 + \mu(1-\theta)\delta_x^2)U_j^n \end{aligned} \tag{2}$$

where $\mu = {}^{k\kappa}/_{h^2}$. This is a one-step method where next step, U^{n+1}, is calculated with the information from previous step, U^n, for all the space points. We can write it on matrix from,

$$\begin{aligned} (I - \theta\mu A_0)U^{n+1} &= (I + (1-\theta)\mu A_0)U^n + (g^{n+1} - g^n) \\ A_l U^{n+1} &= A_E U^n + (g^{n+1} - g^n) \end{aligned} \tag{3}$$

Where I is the identity matrix and

$$A_0 = \begin{bmatrix} 0 & 0 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 0 & 0 \end{bmatrix} \qquad U^n = \begin{bmatrix} U_0^n \\ U_1^n \\ \vdots \\ U_{m-1}^n \\ U_m^n \end{bmatrix} \qquad g^n = \begin{bmatrix} g_L^n \\ 0 \\ \vdots \\ 0 \\ g_R^n \end{bmatrix}$$

Dirichlet boundary conditions are included through the vector g^n, where $gL(t) = u(-1,t)$ and $gR(t) = u(1,t)$. The boundary conditions change over time.

1.2 FTCS-scheme analysis

If we set θ to 0 then we get the Forward Time and Central Space discrete system, where next step depends explicitly on terms from the current state.

$$\frac{U_j^{n+1} - U_j^n}{k} = \frac{\kappa}{h^2}(U_{j-1}^n - 2U_j^n + U_{j+1}^n) \tag{4}$$

1.2.1 Consistency

In order the study the consistency it is necessary to analyze the LTE:

$$\tau(x,t) = \frac{u(x,t+k) - u(x,t)}{k} - \frac{\kappa}{h^2}(u(x-h,t) - 2u(x,t) + u(x+h,t))$$

where u(x,t) represent the real value for the discrete point U_j^n. If we assume that the function u(x,t) is smooth we can Taylor expand

$$\tau(x,t) = \Big(u_t + \frac{1}{2}ku_{tt} + \frac{1}{6}k^2 u_{ttt} + \mathcal{O}(k^4)\Big) - \Big(\kappa u_{xx} + \frac{\kappa}{12}h^2 u_{xxxx} + \mathcal{O}(h^4)\Big)$$

the terms u_t and κu_{xx} are canceled and knowing $u_{tt} = \kappa^2 u_{xxxx}$ then the LTE is

$$\tau(x,t) = \Big(\frac{\kappa^2}{2}k - \frac{\kappa}{12}h^2\Big)u_{xxxx} + \mathcal{O}(k^2 + h^4) \tag{5}$$

The first term of the expanded LTE depends on k and h^2, hence the discrete system is $\mathcal{O}(k+h^2)$-consistent since LTE $\to 0$ as $k, h \to 0$. However, we can improve this by wisely choosing k and h in order to cancel the first LTE term and making the method $\mathcal{O}(k^2+h^4)$ consistent. If $\mu = \kappa k/h^2 = 1/6$

$$\begin{aligned} \tau(x,t) &= \Big(\frac{\kappa^2}{2}k - \frac{\kappa}{12}(6\kappa k)\Big)u_{xxxx} + \mathcal{O}(k^2 + h^4) \\ &= \mathcal{O}(k^2 + h^4) \end{aligned} \tag{6}$$

1.2.2 A-stability and Discrete maximum principle

The discrete FTCS-scheme (4) can be expressed in the matrix form as

$$U^{n+1} = (I + k\kappa A)U^n + g^n \tag{7}$$

where $A = 1/h^2 A_0$ and A_0 defined above. In order to achieve A-stability the impact of the past errors should not grow exponentially over time. This is true if the following condition is fulfilled

$$||I + k\kappa A|| \leq 1 \tag{8}$$

Knowing that the general form of the matrix $(I - k\kappa A)$ eigenvalues can be calculated as:

$$\lambda_p = 1 + \frac{2k\kappa}{h^2}(cos(p\pi h) - 1) \quad for \quad p = 1, 2, ...m,$$

and its L2-norm is equal to the spectral radius, since it is symmetric, it follows

$$||I + k\kappa A|| = max|\lambda_p| = max\Big|1 + \frac{2k\kappa}{h^2}(cos(p\pi h) - 1)\Big| \approx \Big|1 - \frac{4k\kappa}{h^2}\Big| \leq 1$$

From where **two criterion** can be impose to the grid parameters h,k in order to achieve **A-stability**

$$\frac{1}{2} \geq \frac{\kappa k}{h^2} \quad \& \quad \frac{-k\kappa}{h^2} \leq 0 \tag{9}$$

The second one is always fulfilled since $\kappa = 0.1$ and negative grid spaces are impossible.

The **maximum discrete criterion** states that if $\mu(1 - \theta) < 1/2$ then the solution is bounded by a lower and upper bound, and for any refinement path $(k, h) \to 0$ the θ-method will converge uniformly, provided the system is consistent.

$$\frac{\kappa k}{h^2}(1 - \theta) = \frac{\kappa k}{h^2} < 1/2 \tag{10}$$

The MDP criterion impose exactly the same constraints that the A-stability criterion.

1.2.3 Convergence

Lax Equivalence Theorem state that a linear method of the form (7) converges if and only if it is consistent and Lax–Richtmyer stable. The method (7) is Lax–Richtmyer stable if:

$$||I + k\kappa A|| \leq C_T \tag{11}$$

Consistency has been proved in the section 1.2.1. Moreover, if the conditions from previous section are fulfilled the system is not only Lax–Richtmyer stable but *strongly stable* since $C_T = 1$. Therefore, if the time step is set for example to $k = 0.4h^2/\kappa$ the **FTCS-scheme is convergent** at every point in the grid, since $E \to 0$ as $k \to 0$

1.3 Other schemes

Inserting in the θ-scheme (1) a new value $theta = 1/2 + h^2/(12k\kappa)$

$$\frac{U_j^{n+1} - U_j^n}{k} = \frac{\kappa}{h^2}\Big((\frac{1}{2} - \frac{h^2}{12k\kappa})\delta_x^2 U_j^n + (\frac{1}{2} + \frac{h^2}{12k\kappa})\delta_x^2 U_j^{n+1}\Big), \qquad \delta_x^2 U_j^n = U_{j-1}^n - 2U_j^n + U_{j+1}^n \tag{12}$$

1.3.1 Consistency

The LTE of the new scheme looks like

$$\begin{aligned}\tau(x,t) = \frac{u(x,t+k) - u(x,t)}{k} - \frac{\kappa}{h^2}\Big((\frac{1}{2} - \frac{h^2}{12k\kappa})(u(x-h,t) - 2u(x,t) + u(x+h,t) + ... \\ (\frac{1}{2} + \frac{h^2}{12k\kappa})(u(x-h,t+k) - 2u(x,t+k) + u(x+h,t+k))\Big)\end{aligned} \tag{13}$$

Again, assuming smoothness of u(x,t) and Taylor expanding,

$$\tau(x,t) = \Big(u_t + \frac{1}{2}ku_{tt} + \frac{1}{6}k^2u_{ttt}\mathcal{O}(k^4)\Big) - \frac{\kappa}{h^2}\Big((\frac{1}{2} - \frac{h^2}{12k\kappa})(h^2u_{xx} + \frac{1}{12}h^4u_{xxxx} + \mathcal{O}(h^4)\Big) + ... \\ (\frac{1}{2} + \frac{h^2}{12k\kappa})(h^2u_{xx} + h^2u_{xxt} + \mathcal{O}(h^4))\Big) \tag{14}$$

using the fact that $u_t = \kappa u_x x$ and $u_x xt = \kappa u_x xxx$ the LTE simplify to a $\mathcal{O}(h^2 + k^2)$ consistency scheme

$$\tau(x,t) = \Big(\frac{\kappa h^2}{2} - \frac{h^2}{12}\Big)u_{xxxx} + \frac{k^2}{6}u_{ttt} \quad ... \tag{15}$$

1.3.2 A-stability and Discrete maximum principle

In the new scheme (12) the next step can be calculated as

$$U^{n+1} = A_l^{-1}A_E U^n + A_l^{-1}(g^{n+1} - g^n) \tag{16}$$

in order to achieve A-stability $U^n \to 0$ as $kn \to \infty$. This is true if $|\lambda(A_E)/\lambda(A_l)| < 1$ and $|1/\lambda(A_l)| < 1$.

$$\frac{\lambda(A_E)}{\lambda(A_l)} = \Big|\frac{1 + (\frac{\kappa k}{h^2} - \frac{1}{6})(cos(ph\pi) - 1)}{1 - (\frac{\kappa k}{h^2} + \frac{1}{6})(cos(ph\pi) - 1)}\Big| < 1 \qquad \frac{1}{\lambda(A_l)} = \Big|\frac{1}{1 - (\frac{\kappa k}{h^2} + \frac{1}{6})(cos(ph\pi) - 1)}\Big| < 1 \tag{17}$$

this is always true since $k\kappa/h^2 \geq 1/6$ and the cos is bounded between +1 and -1.

1.3.3 Convergence

The method (16) is Lax–Richtmyer stable if:

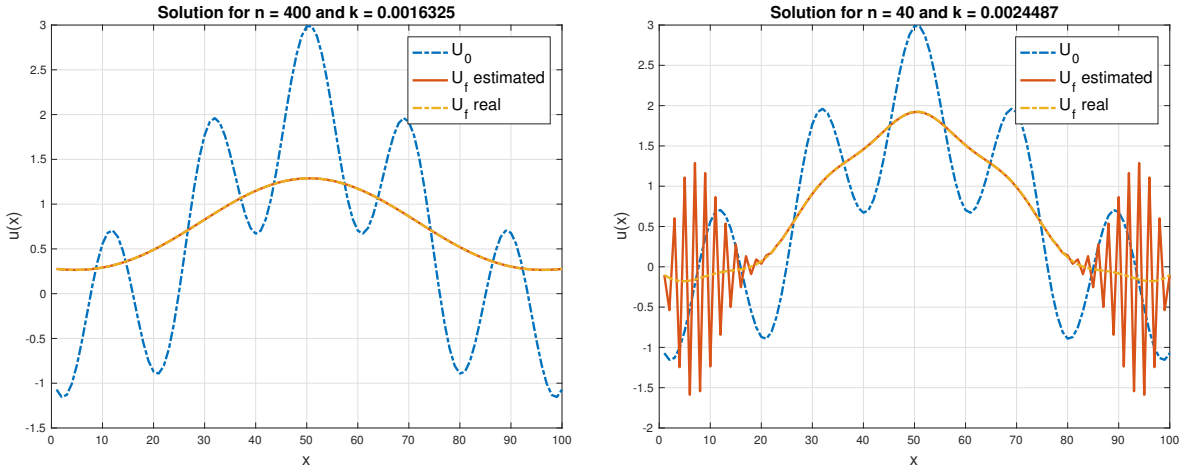
$$||A_l^{-1}A_E|| \leq C_T \tag{18}$$

For any choice of $\mu > 1/6$ it has been proved that all the eigenvalues are bounded by 1. Hence the matrix is bounded for by a constant C_T. In addition to the consistency of the scheme result in a convergent scheme i.e. $E \to 0$ as $k, h \to 0$

1.4 Simulations

1.4.1 FTCS results

The theory from FTCS-scheme predicted a LTE convergence rate of $\mathcal{O}(k+h^2)$ and $\mathcal{O}(k^2+h^4)$ for the special case of $\mu = 1/6$. The simulated results presented in Figure 1 agree with the predictions, where the blue line correspond to $k = 0.4h^2/\kappa$ and the red corresponds to the aforementioned special case.
The theory also predicted A-stability when the criterion (9) is fulfilled. The Figure 2a corresponds to a grid $k = 0.4h^2/\kappa$ which agree with the criterion while the Figure 2b corresponds to $k = 0.6h^2/\kappa$, which violate the criterion and becomes unstable after 40 iterstions.



(a) Stable grid

(b) Unstable grid

Figure 2: Solution for and stable and unstable grid

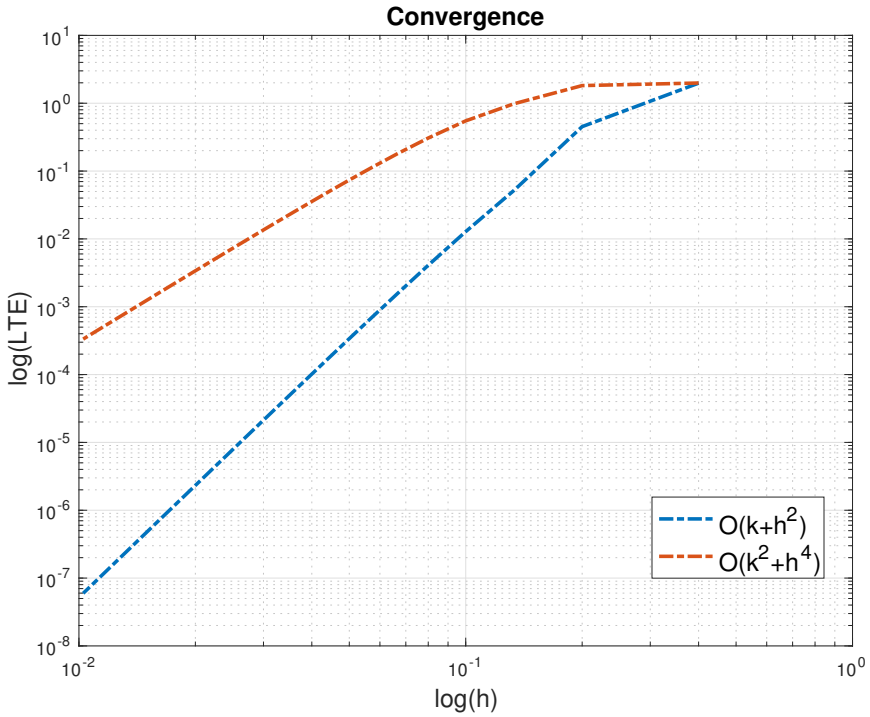


Figure 1: LTE convergence

1.4.2 Other scheme results

To simulate the scheme (12) the relation $k = 0.3h^2/\kappa$ is used in order to fulfill the restriction $\mu = 1/6$. In the Figure 3 a solution for certain k and h is shown.

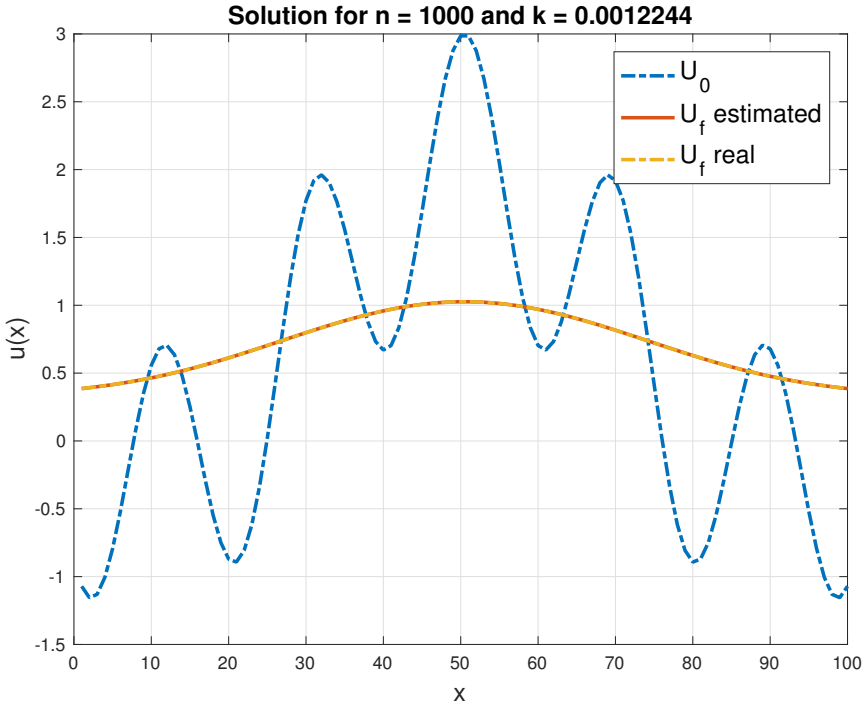
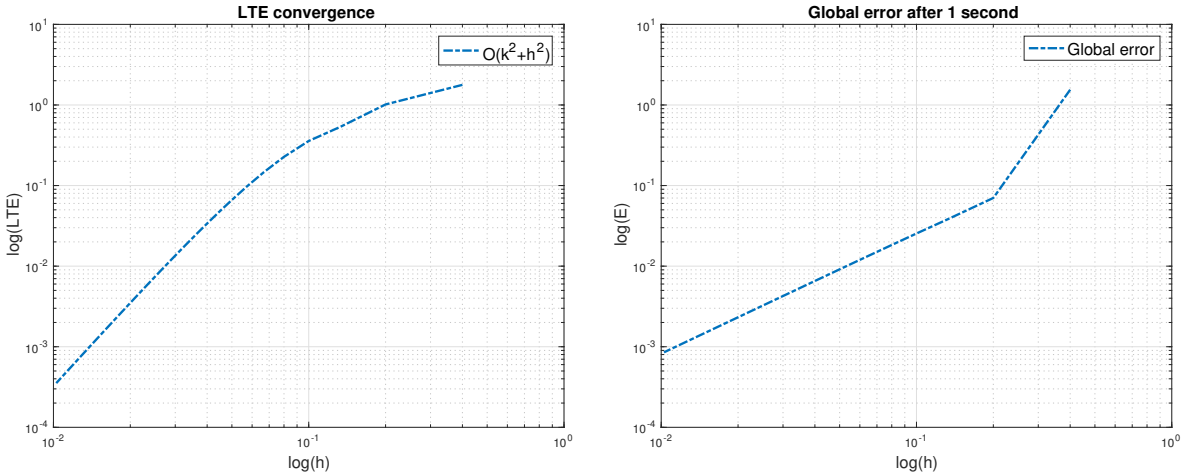


Figure 3: Solution over x

The theory also predicted A-stability when the criterion (9) is fulfilled. The Figure 2a corresponds to a grid $k = 0.4h^2/\kappa$ which agree with the criterion while the Figure 2b corresponds to $k = 0.6h^2/\kappa$, which violate the criterion and becomes unstable after 40 iterstions.

The theory predicted a LTE convergence rate of $\mathcal{O}(k^2 + h^2)$ which can be confirmed by the Figure 4a. The theory also stated that the scheme is convergent, which agree with the simulations form Figure 4. In the latter Figure the global error is calculated at $t = 1s$ as h ,and hence k, decrease.



(a) LTE convergence

(b) Global error convergence

Figure 4: Solution for and stable and unstable grid

2 Exercise 2: Purely hyperbolic model problemn

2.1 Upwind scheme stability

In order to prove this, we first express the differential equation using the Finite Differences of the Upwind scheme, in this case we basically apply Forward Euler to both time and space. Since $a > 0$ we choose the left uncentered approximation of the derivative:

$$\frac{U_j^{n+1} - U_j^n}{\triangle t} + a\frac{U_j^n - U_{j-1}^n}{\triangle x} = 0 \tag{19}$$

Renaming $h = \triangle x$ and $k = \triangle t$ and moving terms around we rewrite the expression as:

$$U_j^{n+1} - U_j^n = -\frac{ak}{h}(U_j^n - U_{j-1}^n) \tag{20}$$

This is the discrete version of the general linear equation for the time derivative of $U(t)$ which can be expressed as:

$$U^{'}(t) = A_\epsilon U(t) \tag{21}$$

In this case, we need to move terms arround to express A_ϵ in the desired standard form. We can rewrite the equation as:

$$U_j^{n+1} - U_j^n = -\frac{ak}{2h}(U_{j+1}^n - U_{j-1}^n) + \frac{ak}{2h}(U_{j+1}^n - 2U_j^n + U_{j-1}^n) \tag{22}$$

So we can rewrite A_ϵ as :

$$A_\epsilon = -\frac{a}{2h}\begin{bmatrix} 0 & 1 & & & -1 \\ -1 & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 0 & 1 \\ 1 & & & -1 & 0 \end{bmatrix} + \frac{\epsilon}{h^2}\begin{bmatrix} 0 & 0 & & & -1 \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ 1 & & & 0 & 0 \end{bmatrix}$$

Being in this case, just by matching terms, $\epsilon = ah/2$. The eigenvalues of this time continuous version are:

$$\lambda_p = -i\frac{a}{h}sin(2\pi ph) - \frac{a}{h}(1 - cos(2\pi ph)) \qquad p = 1, 2, ..., m+1$$

Which in the discrete version will take the form:

$$\lambda_p = \frac{ak}{h}(-1 - isin(2\pi ph) + cos(2\pi ph)) \qquad p = 1, 2, ..., m+1$$

Methods of this form are stable if:

$$|\frac{ak}{h}| \leq 1$$

and

$$-2 \leq -2\frac{\epsilon k}{h^2} = -\frac{ak}{h} \leq 0$$

Which means that $a > 0$ so the condition for stability is:

$$0 \leq \frac{ak}{h} \leq 1$$

This can also be proved by von Newman analisys, which is performed in the third section of this exercise.

2.2 Convergence of the scheme

The Matlab code for implementing the FTBS scheme is appended at the end of the document. All the graphs are generated with it.

Regarding the expected order of accurary. Due to the first degree approximation of the derivatives in both space and time, the order of the error is expected as:

$$E = O(k) + O(m)$$

Regarding the grid parameters k and h, for stability reasons, given $a = 0.5$ we have that:

$$k \leq 2h$$

So the time-delta value k must lower than twice the space-delta h, we must have at least the double space-delta as time-delta. In the following graph, we plot the real system function $u(x, t = t_i)$ and the estimation of the algorithm using $k = 0.016$ and $h = 0.01$. In red we can see the real function and in blue the estimated one.

We have plotted the real and estimated function $u(x, t = t_i)$ for several time instances $t_i = i \cdot 2.2$ at regular intervals of 2.2 (this means the ideal signal is displaced 1.1 wave-lenghts since $a = 0.5$), as time progresses, the lines are plotted more lightly. We can observe that:

- The function is being displaced to the right in the space as time goes by. This is because as time increases, the phase of the function decreases, so the signal is "delayed", it takes more distance to reach the same phase.
- There is an attenuation in the estimated signal, this attenuation increases with time. It seems there could be a delay of the estimated signal but it is too small to be observed now.
- We can see that there are 2 wavelengths inside the space. This is easy to see from the solution equation of $u(x, t)$ since its phase goes from $-2\pi + \phi$ to $2\pi + \phi$ in the spatial domain.

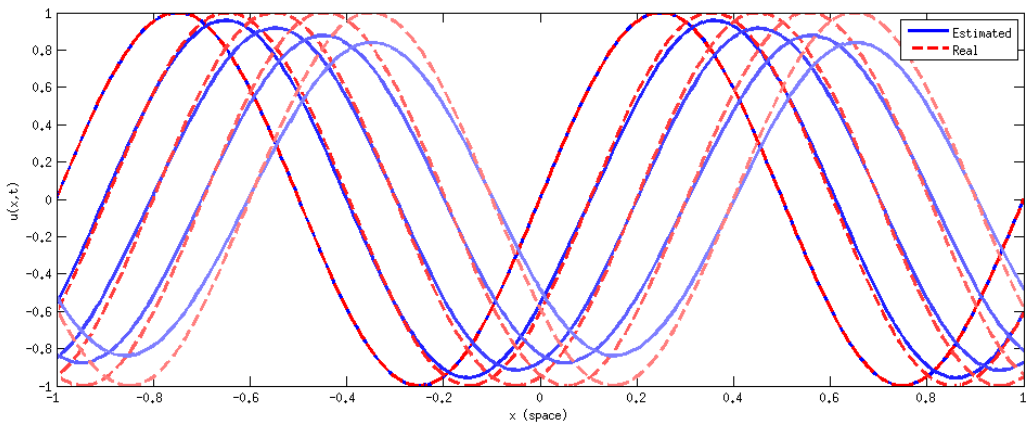


Figure 5: Estimated and real function of the hyperbolic system

The following graphs shows the infinity norm of the error in all of the points of the space, for different grid values of $h = \triangle x$. Each line corresponds to a different value of time-delta $k = \triangle t$. The values of h and k satisfy the stability conditions. As we can see the error decreases linearly with h. It also decreases with k, notice that in each realization, we perform 200 discrete time steps. If we performed for each k, the number of discrete steps required to be at the same time instant, we would get a higher error as we decrease k due to the increase in attenuation of the signal in the algorithm

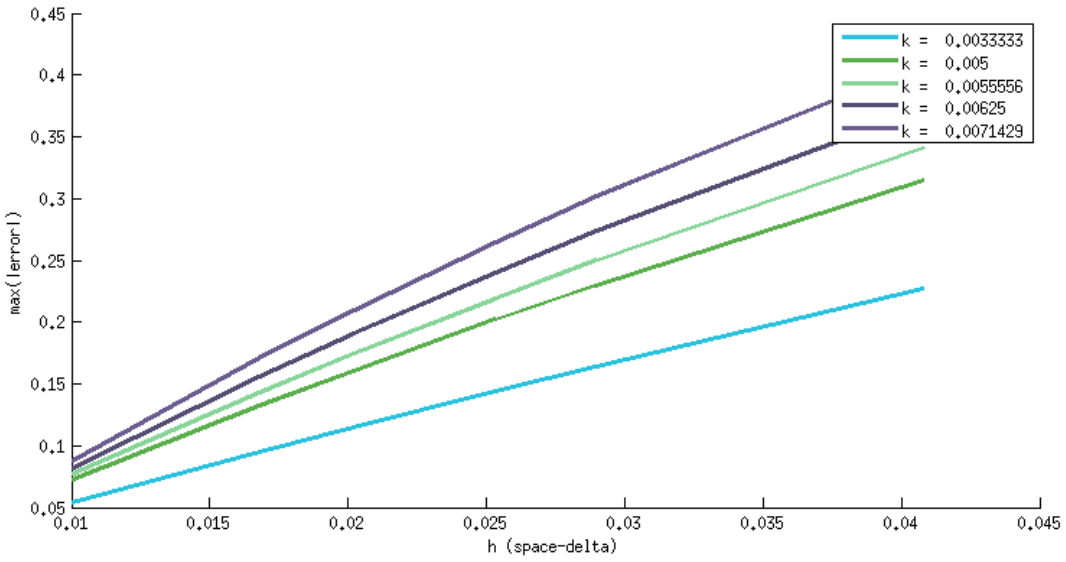


Figure 6: Convergence of the error for the hyperbolic system

2.3 Diffusion prediction

To determine the dispersion and amplitude error per time step we perform von-Neumann stability analysis, based on Fourier analysis of the system. We can express our system $u(x,t)$ as a linear combination of complex exponential functions.

$$u(x,t) = \sum_m e^{\alpha t} e^{ik_m x}$$

In our specific case we have:

$$u(x,t) = sin(2\pi(x-at)) = \frac{1}{2i}(e^{i2\pi(x-at)} - e^{-i2\pi(x-at)})$$

So we have 2 complex exponentials with wave-numbers $k_0 = 2\pi$ and $k_1 = -2\pi$. And the temporal coefficients are $\alpha = i2\pi a$.
Os course the initial state will be then:

$$u(x,0) = \frac{1}{2i}(e^{i2\pi x} - e^{-i2\pi x})$$

For a single exponential function, we have a linear system where the function at time $t + \triangle t$ is linearly obtained from the system at time t. We express $U(x,t) = e^{\alpha t} e^{ik_m x}$, and so $U(x, t+\triangle)$ has the form.

$$U(x,t+\triangle t) = e^{\alpha t + \triangle t} e^{ik_m x} = e^{\alpha t} e^{ik_m x} - a\frac{\triangle t}{\triangle x}[e^{\alpha t} e^{ik_m x} - e^{\alpha t} e^{ik_m (x-\triangle x)}]$$

If we divide by the signal $e^{\alpha t} e^{ik_m x}$ we obtain the transfer function:

$$\frac{u(x,t+\triangle t)}{u(x,t)} = e^{\alpha \triangle t} = 1 - a\frac{\triangle t}{\triangle x}[1 - e^{-ik_m \triangle x}]$$

So we have the temporal amplification factor of our algorithm to be:

$$A = 1 - a\frac{\triangle t}{\triangle x}[1 - e^{-ik_m \triangle x}]$$

The module and phase of this amplification factor will determine how the algorithm evolves with time. As we can see, the amplification factor depends on the step sizes and the wave-number of the exponential k_m.

In the ideal case, we see that this evolution would be $e^{\alpha \triangle t}$, the error of the algorithm after n time instances t would be:

$$Error(t = n \cdot \triangle t) = u(x, 0)[A^n - e^{\alpha n \cdot \triangle t}]$$

Using trigonometric equations, the module of A is:

$$|A| = \sqrt{1 - 2a\frac{\triangle t}{\triangle x}[1 - cos(k_m \triangle x)](1 - a\frac{\triangle t}{\triangle x})}$$

From here we could also obtain the stability properties given that the module of the amplification factor must be lower than 1. We see that the temporal amplication factor depends in both $\triangle x$ and $\triangle t$.

In our example, we have two conjugated exponentials, looking at the equation of the gain A, we can see that in this case, the gains of both exponentials will have the same module and conjugate phase.

$$A_0 = A_1^* = |A_0|e^{i\phi}$$

and so we have that the update of the algorithm for the sine signal that we have is:

$$U(x, n \cdot \triangle t) = \frac{1}{2i}(A_0^n e^{i2\pi x} - A_1^n e^{-i2\pi x}) = \frac{|A_0|^n}{2i}(e^{i2\pi x + n \cdot i\phi} - e^{-i2\pi x - n \cdot i\phi}) = |A_0|^n sin(2\pi x + n \cdot \phi)$$

The dispersion ϕ occurring at each iteration of the algorithm can be computed as $\phi = arg\{A\}$.

For the proposed parameters, $a = 0.5$, $h = \triangle x = 0.01$ and therefore $k = \triangle t = 0.016$. The phase (dispersion) error and amplitude (diffusion) error per time will be indicated next. We have a one step attenuation of $|A_0| = 0.9996842267$ and a step delay of $\triangle\phi = 3.9695e - 06rad$.

To compute the number of discrete time steps we need to take to achieve the 40 wave periods, we have that $u(x, t) = sin(2\pi(x - at))$, its difference in phase has to go from 0 to $40 \cdot 2\pi$ then $2\pi at$ has to go from 0 to 80π, which means t has to go from 0 to 80, which means that we need $80/0.016 = 5000$ discrete time steps.

The predicted attenuation will be:

$$|A_{end}| = |A_0|^{5000} = 0.2062$$

and the predicted delay is:

$$\triangle\phi_{end} = 5000 \cdot \triangle\phi = 0.0198rad$$

.

So we expect very little delay. The following graph is coherent with this results.

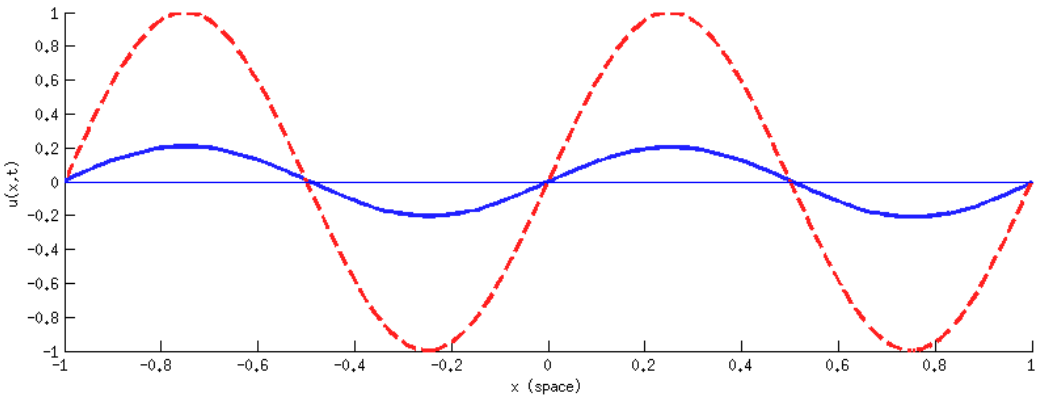


Figure 7: Real and estimated signal after 40 wavelengths

3 Exercise 3: Non-linear advection-diffusion equation

3.1 Numerical scheme

The Burgers equation can be rewritten as

$$u_t = \epsilon u_{xx} - u u_x \tag{23}$$

the following scheme can be applied

$$\frac{U_j^{n+1} - U_j^n}{k} = \frac{\epsilon}{h^2}(U_{j-1}^n - 2U_j^n + U_{j+1}^n) - U_j^n \frac{U_j^n - U_{j-1}^n}{h} \tag{24}$$

and its LTE can be demonstrated to be $\mathcal{O}(k+h)$-consistent

$$\begin{aligned}\tau(x,t) &= \left(u_t + \frac{1}{2}k u_{tt} + \frac{1}{6}k^2 u_{ttt} + \mathcal{O}(k^4)\right) - \left(\epsilon u_{xx} + \frac{\epsilon}{12}h^2 u_{xxxx} + \mathcal{O}(h^4)\right) + \left(u u_{xx} + \frac{1}{6}h^2 u u_{xxx} + \mathcal{O}(h^4)\right) \\ &= \left(\frac{1}{2}k u_{tt} + \frac{1}{6}k^2 u_{ttt} + \mathcal{O}(k^4)\right) - \left(\frac{\epsilon}{12}h^2 u_{xxxx} + \mathcal{O}(h^4)\right) + \left(\frac{1}{2}h u u_{xxx} + \mathcal{O}(h^3)\right)\end{aligned} \tag{25}$$

3.2 Scheme simulations and results

The following figure shows the estimated function for the interval $t = 0$ to $t = 1$, for different values of ϵ. The grid parameters are $h = 0.02, k = 0.001$. Regarding stability, our solver is stable for values of $k < \alpha \cdot h^2$ being α a constant. Also the viscosity ϵ affects the stability, the bigger ϵ, the more points we need in the grid to make the system stable. In the graph we can see how increasing the viscosity generates more smooth graphs.

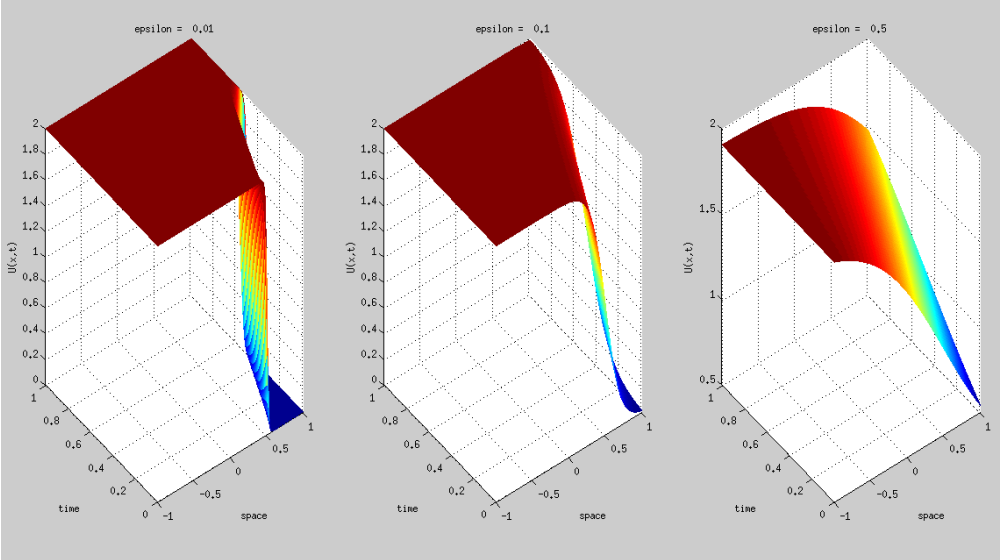


Figure 8: Estimated function for different values of epsilon

The LTE converges as in Figure 8 for $\epsilon = 0.01$.

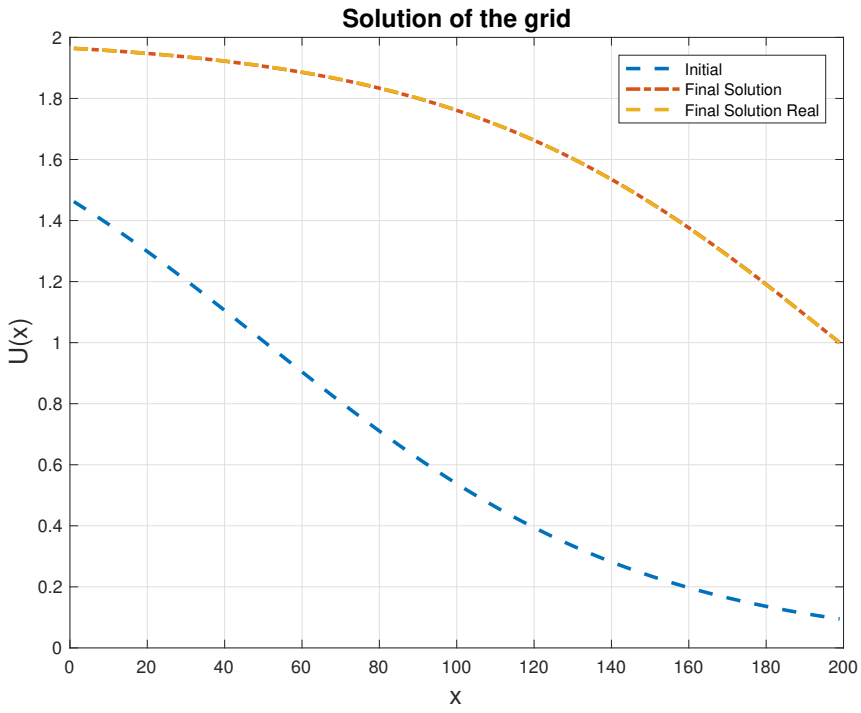


Figure 9: Real and estimated solution after t=1.5 seconds

In the 9 the signal has being calculated after 1.5 seconds and it can be seen that coincide with the real one.

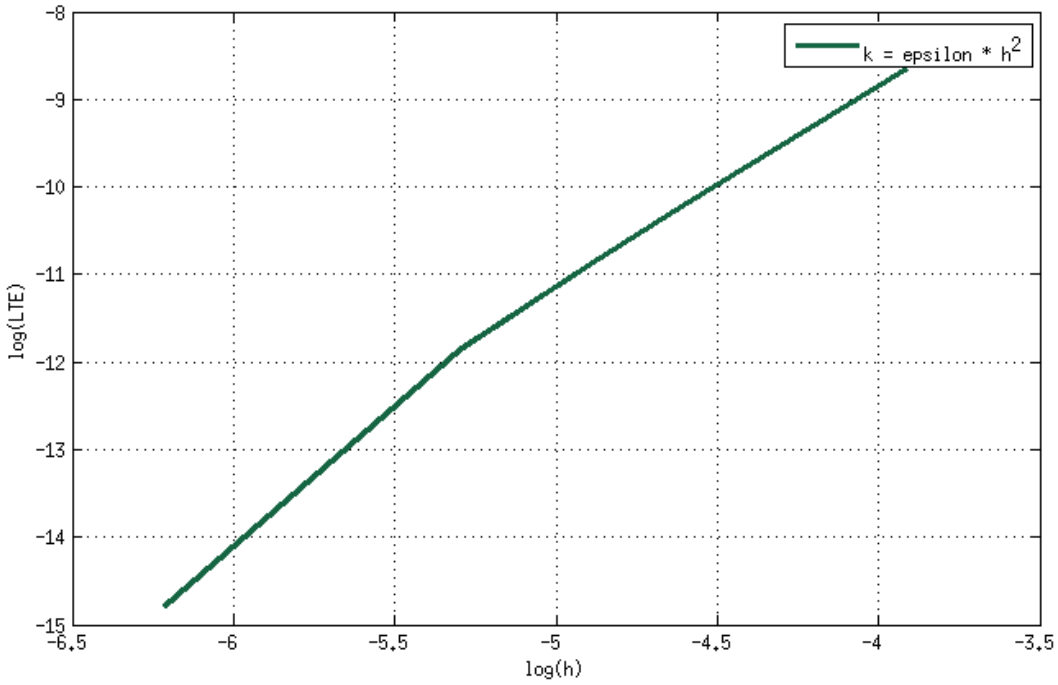


Figure 10: Estimated function for different values of epsilon

3.3 Estimate of $u_x|_{x=0}$ at time $t = 1.6037/\pi$

The next graph shows the system function for different values of ϵ. For the *epsilon* $= 0.01/\pi$ specified, we obtain a spacial derivative of -80.729 at $x = 0$, this is far from the expected value, but looking at the

derivative vector, the peak happens at $x = -0.03$, being $u_x|_{x=0} = -152.8$. Which could mean that the system has a small delay that causes the miss match.

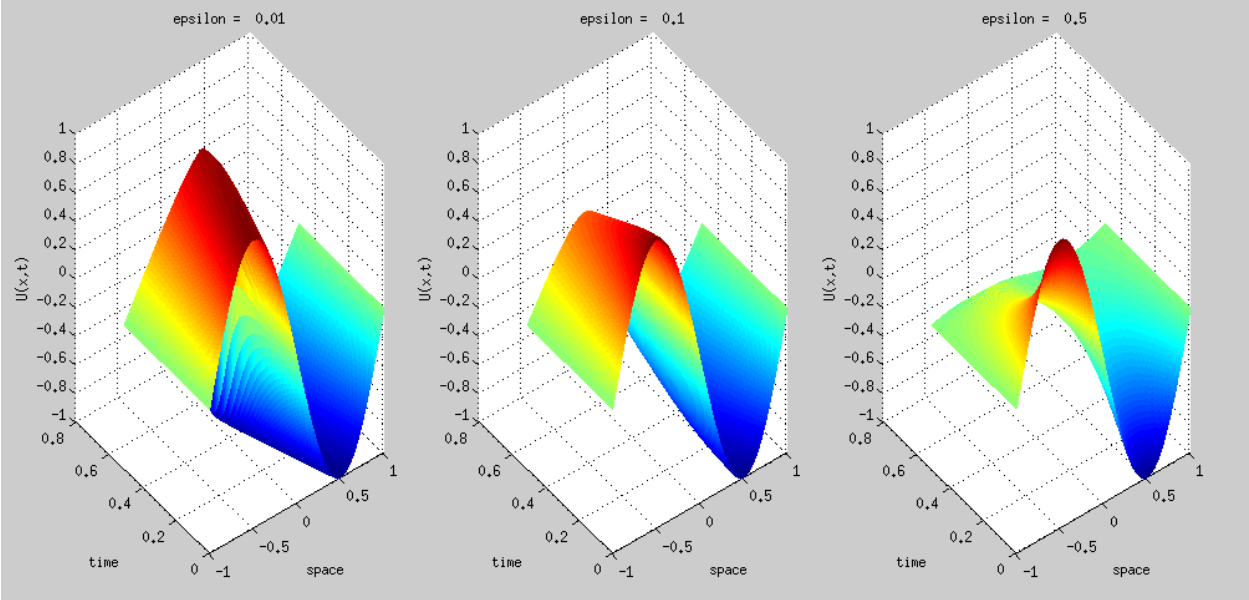


Figure 11: Estimated function for different values of epsilon

References

[1] Randall J. LeVeque *Finite Difference Methods for Ordinary and Partial Differential Equations*

A Code files

A.1 Problem 1

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%  Exercise 1: Parabolic equations   %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add subscripts
addpath('functions');
%% Forward time Central Space
%model paramters
kappa = 0.1;

% Space grid
m = 100;
h = 2/(m-1);
x = linspace(-1,1,m);
%% Forward time Central Space
%model paramters
theta = 0;
% to fulfil the stability criterion k > 0.5*h^2/kappa
k = 0.4*h^2/kappa;      % good k
% k = h^2/(6*kappa);      % extremely good k
% k = 0.6*h^2/kappa;        % wrong k

%Solving the system
steps = 4000;
Un = solver(m,k,h,theta,x,steps);
Uf = realValue(m,x,steps*k);
U0 = realValue(m,x,0);

%Convergence
k_ex = @(h,kappa) h^2/(6*kappa);
k_good = @(h,kappa) 0.4*h^2/kappa;
k_wrong = @(h,kappa) 0.6*h^2/kappa;
m_List = 6:5:200;
[LTE_List,h_List,k_List] = LTEconvergence(m_List,kappa,...
    theta,k_ex);
[LTE_List1,h_List1,k_List1] = LTEconvergence(m_List,kappa,...
    theta,k_good);

%%%%% Plotting

%Solution after time
figure(1)
lw = 2;
plot(U0,'-.','LineWidth',lw)
hold on
    plot(Un,'LineWidth',lw)
    plot(Uf,'-.','LineWidth',lw)
    leg = legend('U_0','U_f estimated','U_f real');
    set(leg,'FontSize',14);
    ste = num2str(steps);
    kstring = num2str(k);
    title(['Solution for n = ' ste ' and k = ' kstring],'FontSize',14);
    ylabel('u(x)','FontSize',14)
```

```
    xlabel('x','FontSize',14)
    grid on
hold off

% LTE convergence

figure(2)
lw = 2;
loglog(h_List1,LTE_List,'-.','LineWidth',lw)
hold on
    loglog(h_List,LTE_List1,'-.','LineWidth',lw)
    leg = legend('O(k+h^2)','O(k^2+h^4)');
    set(leg,'FontSize',14);
    title('Convergence','FontSize',14);
    ylabel('log(LTE)','FontSize',14)
    xlabel('log(h)','FontSize',14)
    grid on
hold off

%% Other scheme
%mu = kappa*k/h^2 > 1/6 -->
k = 10*h^2/kappa;
theta = 1/2 + h^2/(12*kappa*k);
mu = kappa*k/h^2;

%Solving the system
steps = 1000;
Un = solver(m,k,h,theta,x,steps);
Uf = realValue(m,x,steps*k);
U0 = realValue(m,x,0);

%Convergence
m_List = 6:5:200;
k_func = @(h,kappa) 0.3*h^2/kappa;
[LTE_List,h_List,k_List] = LTEconvergence(m_List,kappa,theta,k_func);

%Solving the system
m_List = 6:5:200;
[E_List,h_List1,k_List1] = GlobalError(m_List,kappa,theta);


%%%%% Plotting
%Solution after time
figure(1)
lw = 2;
plot(U0,'-.','LineWidth',lw)
hold on
    plot(Un,'LineWidth',lw)
    plot(Uf,'-.','LineWidth',lw)
    leg = legend('U_0','U_f estimated','U_f real');
    set(leg,'FontSize',14);
    ste = num2str(steps);
    kstring = num2str(k);
    title(['Solution for n = ' ste ' and k = ' kstring],'FontSize',14);
    ylabel('u(x)','FontSize',14)
    xlabel('x','FontSize',14)
    grid on
hold off

% LTE convergence
```

```
figure(2)
lw = 2;
loglog(h_List,LTE_List,'-.','LineWidth',lw)
hold on
    title('LTE convergence ','FontSize',14);
    leg = legend('O(k^2+h^2)');
    set(leg,'FontSize',14);
    ylabel('log(LTE)','FontSize',14)
    xlabel('log(h)','FontSize',14)
    grid on
hold off

% Global error Convergence

figure(3)
lw = 2;
loglog(h_List,E_List,'-.','LineWidth',lw)
hold on
    title('Global error after 1 second ','FontSize',14);
    leg = legend('Global error');
    set(leg,'FontSize',14);
    ylabel('log(E)','FontSize',14)
    xlabel('log(h)','FontSize',14)
    grid on
hold off
```

```
function [Un] = solver(m,k,h,theta,x,steps)

%system matrices
[Al,Ae] = sysMatrices(m,k,h,theta);
U0 = realValue(m,x,0);

%Solving the system
Un = U0;
for i = 1:steps
    tn = k*(i-1);
    tn1 = k*i;
    gn = boundaries(m,x(1),x(end),tn);
    gn1 = boundaries(m,x(1),x(end),tn1);
    %calculate next step
    Un1 = Al\(Ae*Un + (gn1-gn));

    %update
    Un = Un1;
end

end
```

```
function U0 = realValue(m,x,t)
%%% Calculate real u(x,t) values

%system constants
kappa =0.1;
Q = 3;
alpha = [1,4,16];
ai = 1;
bi = 0;

U0 = zeros(m,1);

for j=1:m
    for i=1:Q
```

```
        U0(j) = U0(j) + (ai*cos(alpha(i)*x(j))+bi*sin(alpha(i)*x(j)))* ...
            exp(-kappa*alpha(i)^2*t);
    end
end

end

function [LTE_List,h_List,k_List] = LTEconvergence(m_List,kappa,theta,kfunc)

LTE_List = zeros(1,length(m_List));
h_List = zeros(1,length(m_List));
k_List = zeros(1,length(m_List));
steps = 1;

for i = 1:length(m_List)
    h = 2/(m_List(i)-1);
    k = kfunc(h,kappa);
    x = linspace(-1,1,m_List(i));
    Un = solver(m_List(i),k,h,theta,x,steps);
    Uf = realValue(m_List(i),x,steps*k);
    LTE_List(i) = norm(Un-Uf);
    h_List(i) = h;
    k_List(i) = k;
end
end

function [E_List,h_List,k_List] = GlobalError(m_List,kappa,theta)

E_List = zeros(1,length(m_List));
h_List = zeros(1,length(m_List));
k_List = zeros(1,length(m_List));
time = 1;

for i = 1:length(m_List)
    h = 2/(m_List(i)-1);
    k = 0.3*h^2/kappa;
    x = linspace(-1,1,m_List(i));
    steps = round(time/k);
    Un = solver(m_List(i),k,h,theta,x,steps);
    Uf = realValue(m_List(i),x,steps*k);
    E_List(i) = norm(Un-Uf);
end
end

function g = boundaries(m,x0,xend,t)
%%% Calculate boundary values

%system constants
kappa =0.1;
Q = 3;
alpha = [1,4,16];
ai = 1;
bi = 0;

%bounraies
gr = 0;
gl = 0;
for i=1:Q
    gl = gl + (ai*cos(alpha(i)*x0)+bi*sin(alpha(i)*x0))* ...
        exp(-kappa*alpha(i)^2*t);
    gr = gr + (ai*cos(alpha(i)*xend)+bi*sin(alpha(i)*xend))* ...
        exp(-kappa*alpha(i)^2*t);
```

```
end

g = zeros(m,1);
g(1) = gl;
g(end) = gr;

end

function [Al,Ae] = sysMatrices(m,k,h,theta)
%%% Calculate system matrices

if(theta<0 || theta>1)
    disp('Invalid theta');
    return
end


%model paramter
kappa = 0.1;
mu = k*kappa/h^2;

%A0 matrix construction
mid = -2.*ones(m,1);
sides = ones(m,1);
A0 = spdiags([sides mid sides],[-1,0,1],m,m);
A0(1,1) = 0; A0(1,2) = 0; A0(end,end) = 0; A0(end,end-1) = 0;

I = speye(m);
Al = (I-mu*theta*A0);
Ae = (I+mu*(1-theta)*A0);


end
```

A.2 Problem 2

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%  Exercise 2.2: Hyperbolic equations   %%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add subscripts
addpath('functions');

clc;
clear all;
close all;

%%
% time and space grids

figure(1)

m_list = [50, 70, 80, 100, 120,200]; % space

n_list = [140, 160, 180, 200,300]; % time

for in = 1:length(n_list) % time

    Error_abs = [];
    k_s = [];
    h_s = [];
```

```
    for im = 1:length(m_list)  % space
        n = n_list(1 + length(n_list) - in);
        tspan = [0 1];
        k = (tspan(2)-tspan(1))/n;
        k_s = [ k_s k];

        m = m_list(1 + length(m_list) -im);
        h = 2/(m-1);
        h_s = [ h_s h];

        x = linspace(-1,1,m);

        %model paramters
        a = 0.5;
        c = a;
        f = 0;

        %system matrices

        U0 = sin(2*pi*x);

        %Solving the system
        Un = U0;

        nplots = 1;
        n_errors_seen = 1;
        for i = 1:200   % 200
            tn = k*(i-1);
            tn1 = k*i;
            gn = boundaries(m,x(1),x(end),tn);
            gn1 = boundaries(m,x(1),x(end),tn1);

            %calculate next step
            Un1 = Un;
            N_elem = length(Un1);
            for j = 2:(N_elem-1)
                Un1(j) = Un(j) - (c*k/h)*(Un(j) - Un(j-1));
            end
            Un1(1) = Un(1) - (c*k/h)*(Un(1) - Un(N_elem-1));
            Un1(N_elem) = Un(N_elem) - (c*k/h)*(Un(N_elem) - Un(N_elem-1));

            %update
            Un = Un1;

        end
        exact_solution = sin(2*pi*(x -c*tn1));

        Error = abs(exact_solution - Un1);

        Error_abs = [Error_abs max(Error)];
    end

    hold on

    %% Check stability

    caca = sum(k > 2*h_s);
    if (caca == 0)
        plot(h_s, Error_abs, 'color', rand(1, 3), 'DisplayName', strjoin({'k = ', num2str(k)}), 'Lin
    end
end
```

```
xlabel('m (space-delta)' ,'FontSize', 50)
ylabel('max(|error|)', 'FontSize', 50)
legend('show')
hold off


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%  Exercise 2.3: Hyperbolic equations   %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add subscripts
addpath('functions');

clc;
clear all;
close all;

%%
% time and space grids

h = 0.01;
m =  + 2/h;
x = linspace(-1,1,m);

k = h*1.6;
tspan = [0 1];
n = (tspan(2)-tspan(1))/k;

%model paramters
a = 0.5;
c = a;
f = 0;

%system matrices

U0 = sin(2*pi*x);

%Solving the system
Un = U0;

A1 = 1 - (c*k/h)*(1 - exp(sqrt(-1)* 2* pi * h));
A1 = A1^1;
A1_mod = abs(A1);
A1_phase = angle(A1);
phase_ideal = 2 * pi * k * a;

A1_f = A1^(40*n);
A1_fmod = abs(A1_f );
A1_fphase = angle(A1_f );
phase_ideal_f = rem(2 * pi * k * a * (40*n), 2\pi);
total_delat = A1_fphase - phase_ideal_f;

figure(1)
% pl = plot(x,Un, 'color',[0 0 1],'LineStyle','-', 'LineWidth', 3);
% hold on
% exact_solution = sin(2*pi*(x -c*0));
% pl = plot(x,exact_solution, 'color',[1 0 0],'LineStyle','--', 'LineWidth', 3);
% hold on

legend('Estimated','Real')
xlabel('x (space)' ,'FontSize', 50)
ylabel('u(x,t)', 'FontSize', 50)
```

```
Errors = {};
Error_abs = [];
nplots = 1;
n_errors_seen = 1;

for i = 1:80*n
    tn = k*(i-1);
    tn1 = k*i;
    % i
    %calculate next step
    Un1 = Un;
    N_elem = length(Un1);
    for j = 2:(N_elem-1)
        Un1(j) = Un(j) - (c*k/h)*(Un(j) - Un(j-1));
    end
    Un1(1) = Un(1) - (c*k/h)*(Un(1) - Un(N_elem));
    Un1(N_elem) = Un(N_elem) - (c*k/h)*(Un(N_elem) - Un(N_elem-1));

    %Un1(N_elem) = Un1(1)

    %update
    Un = Un1;
    if i >= nplots*(n)*80
%         subplot(3,1,nplots + 1);

   %     Ux = A1_fmod * sin(2*pi*x + A1_fphase);
        hold on
        plot(x,Un1, 'color',[nplots/8 nplots/8 1] ,'LineStyle','-', 'LineWidth', 3)

        hold on
        exact_solution = sin(2*pi*(x -c*tn1));
        plot(x,exact_solution,'color',[1 nplots/8 nplots/8],'LineStyle','--', 'LineWidth', 3)
        nplots = nplots + 1;

        % Compute error
        Error = abs(exact_solution - Un1);
        Errors{n_errors_seen} = Error;
        Error_abs = [Error_abs norm(Error)];
        n_errors_seen = n_errors_seen + 1;
    end
end

plot ([-1 1],[0,0])

A1 = 1 - (c*k/h)*(1 - exp(sqrt(-1)* 2* pi * h));
A1 = A1^1;
A1_mod = abs(A1);
A1_phase = angle(A1);
phase_ideal = 2 * pi * k * a;

A1_f = A1^(80*n);
A1_fmod = abs(A1_f );
A1_fphase = angle(A1_f );
phase_ideal_f = rem(2 * pi * k * a * (40*n), 2\pi);
total_delat = A1_fphase - phase_ideal_f;


delay = A1_phase - phase_ideal;
A2 = 1 - (c*k/h)*(1 - exp(-sqrt(-1)* 2* pi * h));
A2 = A2^1;
A2_mod = abs(A2);
```

```
A2_phase = angle(A2);

den = exp(sqrt(-1)* 2* pi * h) - exp(- sqrt(-1)* 2* pi * h);
num = A1 * exp(sqrt(-1)* 2* pi * h) - A2 * exp(- sqrt(-1)* 2* pi * h);
num2 = exp(sqrt(-1)* 2* pi * (h - a*k)) -  exp(- sqrt(-1)* 2* pi * (h - a*k));

Total_gain = num / den;
Total_gain_mod = norm(Total_gain);
Total_gain_angle = angle(Total_gain);


figure()
plot(sin(2*pi*x - 0.5))
hold on
plot(sin(2*pi*x))
```

A.3 Problem 3

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%  Exercise 3: Hyperbolic equations   %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add subscripts
addpath('functions');

clc;
clear all;
close all;

%%
% time and space grids

h = 0.01;
m = ceil(-1 + 2/h);
x = linspace(-1,1,m);

k = 0.2*h^2/epsilon;

%model paramters
epsilon = 0.5;

time_process = 1.5; % Number of seconds to simulate
Nsim = round(time_process/k);
t = zeros(1,Nsim);
for i = 1:Nsim
    t(i) = k*i;
end

% Set the solution data
Un = zeros(m,Nsim);
% func1 = @(x) -sin(pi*x);
% Un(1,:) = zeros(1,Nsim);
% Un(m,:) = zeros(1,Nsim);
% Un(:,1) = func1(x);
func = @(x,t,epsilon) -tanh(x+0.5-t/(2*epsilon))+1;
Un(1,:) = func(-1,t,epsilon);
Un(m,:) = func(1 ,t,epsilon);
Un(:,1) = func(x,0,epsilon);
Uf = func(x,t(end),epsilon);

for i = 2:Nsim
    tn = k*(i-1);
```

```
    tn1 = k*i;
    %calculate next step
    for j = 2:(m-1)
        Un(j,i) = (k*epsilon/h^2 )*(Un( j-1,i-1) - ...  %% 1
            2*Un(j, i - 1) + Un(j + 1, i-1)) + Un(j, i - 1) -...  %% 2
             k/h*Un(j,i-1) * (Un(j,i-1)-Un(j-1,i-1));   %% 3
    end
end

figure(1)
lw = 2;
plot(Un(:,1),'--','LineWidth',lw)
hold on
    plot(Un(:,end),'-.','LineWidth',lw)
    plot(Uf,'--','LineWidth',lw)
    legend('Initial','Final Solution','Final Solution Real')
    title('Solution of the grid','FontSize',14);
    ylabel('U(x)','FontSize',14)
    xlabel('x','FontSize',14)
    grid on
hold off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%  Exercise 3: Hyperbolic equations   %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add subscripts
addpath('functions');

clc;
clear all;
close all;

%%
% time and space grids

h = 0.001;
m = ceil(-1 + 2/h);
x = linspace(-1,1,m);

k = 0.00005;
tspan = [0 1];
n = ceil((tspan(2)-tspan(1))/k);

%model paramters
epsilon = 0.9/pi;
epsilons = [0.01/pi, 0.1,0.5];
n_graph = 1;
figure()
for ie = 1:n_graph
    epsilon = epsilons(ie);
    %system matrices

    % U0 = sysfunc_3_t0(x, epsilon);
    % U1 = sysfunc_3(x, 1, epsilon);
    %
    % plot (x, U0);
    % hold on
    % plot (x, U1);
    %
    % figure();
    %
```

```
    % tmax = 0.1; % Number of seconds

    time_process = 1.6037/pi; % Number of seconds to simulate
    Nsim = ceil(n * time_process); % Number of iterations of the algo
    t = linspace(0,time_process,Nsim);

    %% Set the solution data
    Un = zeros(m,Nsim);
    Un(:,1) = sysfunc_3(x, 0 ,epsilon)';   % t = 0 conditions

    Un(1,:) = sysfunc_3(-1,0, epsilon);   % Boundary conditions
    Un(m,:) = sysfunc_3(1 ,t,epsilon);

    for i = 2:Nsim
        tn = k*(i-1);
        tn1 = k*i;
        %calculate next step
        for j = 2:(m-1)
            Un(j,i) = (k*epsilon/h^2 )*(Un( j-1,i-1) - ...  %% 1
                2*Un(j, i - 1) + Un(j + 1, i-1)) + Un(j, i - 1) -...  %% 2
                 k/h*Un(j,i-1) * (Un(j,i-1)-Un(j-1,i-1));   %% 3
        end
    end

    %% Plotting of the solution
%     subplot(1,n_graph,ie)
%     [T ,X ] = meshgrid(t ,x);
%     mesh( X, T, Un);
%     title(strjoin({'epsilon = ', num2str(epsilon)}))
%     ylabel('time')
%     xlabel('space')
%     zlabel('U(x,t)')
end

 ultimo = Un(:,Nsim);
 Derivatives = diff(ultimo)/h;
 Derivatives = (-ultimo + [ultimo(3:end)' 0 0 ]')/(2*h);
 derivative = Derivatives(floor(m/2));


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%  Exercise 3: Hyperbolic equations   %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add subscripts
addpath('functions');

clc;
clear all;
close all;

%%


%model paramters
epsilon = 0.01;
epsilons = [0.01/pi, 0.1,0.5];
n_graph = 1;

Errors = {};
Error_abs = [];
nplots = 0;
```

```
n_errors_seen = 1;

hs = [0.02, 0.01, 0.0075, 0.005, 0.002,0.001];
nshow = 5;
for ih = 1:nshow
    % time and space grids

    h = hs(ih);
    m = ceil(-1 + 2/h);
    x = linspace(-1,1,m);

    k =  epsilon * h^2;

    tspan = [0 1];
    n = ceil((tspan(2)-tspan(1))/k);
    %system matrices

    % U0 = sysfunc_3_t0(x, epsilon);
    % U1 = sysfunc_3(x, 1, epsilon);
    %
    % plot (x, U0);
    % hold on
    % plot (x, U1);
    %
    % figure();
    %
    % tmax = 0.1; % Number of seconds

    time_process = 2*k; % Number of seconds to simulate
    Nsim = ceil(n * time_process); % Number of iterations of the algo


    t = linspace(0,time_process,Nsim);

    n_errors_seen = 1;
    %% Set the solution data
    Un = zeros(m,Nsim);
    Un(:,1) = sysfunc_3(x, 0 ,epsilon)';   % t = 0 conditions

    Un(1,:) = sysfunc_3(-1,0, epsilon);   % Boundary conditions
    Un(m,:) = sysfunc_3(1 ,t,epsilon);

    for i = 2:Nsim
        tn = k*(i-1);
        tn1 = k*i;
        %calculate next step
        for j = 2:(m-1)
            Un(j,i) = (k*epsilon/h^2 )*(Un( j-1,i-1) - ...  %% 1
                2*Un(j, i - 1) + Un(j + 1, i-1)) + Un(j, i - 1) -...  %% 2
                 k/h*Un(j,i-1) * (Un(j,i-1)-Un(j-1,i-1));   %% 3
        end

    end
    exact_solution = sysfunc_3(x,tn, epsilon);

    % Compute error
    Error = exact_solution' - Un(:,end);
    Errors{n_errors_seen} = Error;
    Error_abs = [Error_abs norm(Error)];
    n_errors_seen = n_errors_seen + 1;
end
```

```
figure()

plot(log(hs(1:nshow)), log(Error_abs), 'color', rand(1, 3), 'DisplayName', 'k = epsilon * h *h', 'Li

xlabel('log(h)' ,'FontSize', 50)
ylabel('log(LTE)', 'FontSize', 50)
legend('show')
hold off
grid on
 ultimo = Un(:,Nsim);
 Derivatives = diff(ultimo)/h;
 Derivatives = (-ultimo + [ultimo(3:end)' 0 0 ]')/(2*h);
 derivative = Derivatives(floor(m/2));


function [ out ] = sysfunc_3( x,t,epsilon)
     out = -tanh((x - 0.5 -t )/(2*epsilon)) +1;
     % out = -sin(pi * x );
end
```