CALCOLO SCIENTIFICO

Project 2

Student
Pranav KASELA
Roll no. 866261

May 5, 2018

Contents

Exercise 1

We are considering the following problem :

$$\begin{cases} -u''(x) = f(x) & x \in (0,1) \\ u(0) = u(1) = 0 \end{cases} \tag{$\star$}$$

for $f(x) = 1$, $f(x) = x$ and $f(x) = e^x$. We will compute the exact solution of $(\star)$ in the form of the convolution of the Green's function with $f(x)$:

$$u(x) = x \int_0^1 (1-y)f(y)dy - \int_0^x (x-y)f(y)dy \tag{1}$$

We define

$$\alpha(x) = \int_0^x f(y)dy, \qquad \beta(x) = \int_0^x yf(y)dy$$

using the linearity of the integral in (1) we have

$$u(x) = x \int_0^1 f(y)dy - x \int_0^1 yf(y)dy - x \int_0^x f(y)dy + \int_0^x yf(y)dy$$

Substitute $\alpha(x)$ and $\beta(x)$

$$u(x) = x \cdot (\alpha(1) - \beta(1)) - x \cdot \alpha(x) + \beta(x)$$

To approximate numerically $u(x)$ we start by approximating $\alpha(x)$ and $\beta(x)$ using the composite trapezoidal method :

$$\alpha(x_{i+1}) \approx \alpha_{i+1} = \alpha_i + \frac{h}{4}(f_i + 2f_{i+1/2} + f_{i+1})$$

Where $f_i = f(x_i)$ and $f_{i+1/2} = f(x_{i+1/2})$.This is a recursive method of composite trapezoid and we proceed by calculating only the area between x_i and x_{i+1} and adding the area calculated till x_i thus finding the area between $x_0 = 0$ and x_{i+1} and since $\int_0^0 f(x)dx = 0$ we put $\alpha_0 = 0$.
By the same fashion we approximate $\beta(x_{i+1})$ but with the integrand $x \cdot f(x)$ thus

$$\beta(x_{i+1}) \approx \beta_{i+1} = \beta_i + \frac{h}{4}(x_i f_i + 2x_{i+1/2} f_{i+1/2} + x_{i+1} f_{i+1})$$

Using the Green's convolution we have the approximation of $u(x)$

$$u_i = x_i(\alpha_{n+1} - \beta_{n+1}) + \beta_i - x_i\alpha_i$$

for $i = 1, ..., n$ and the boundary conditions translate as $u_0 = u_{n+1} = 0$.
Now let's see the numerical implementation in MATLAB and compare it with the exact solution for three different functions (we will be using n=16 nodes):

f(x)=1

In the case of $f(x) = 1$ using the convolution we have

$$\alpha(x) = x, \qquad \beta(x) = \frac{x^2}{2}$$

so the exact solution is

$$u(x) = x \cdot (1 - \frac{1}{2}) - x \cdot x + \frac{x^2}{2} = \frac{x}{2} - \frac{x^2}{2}$$

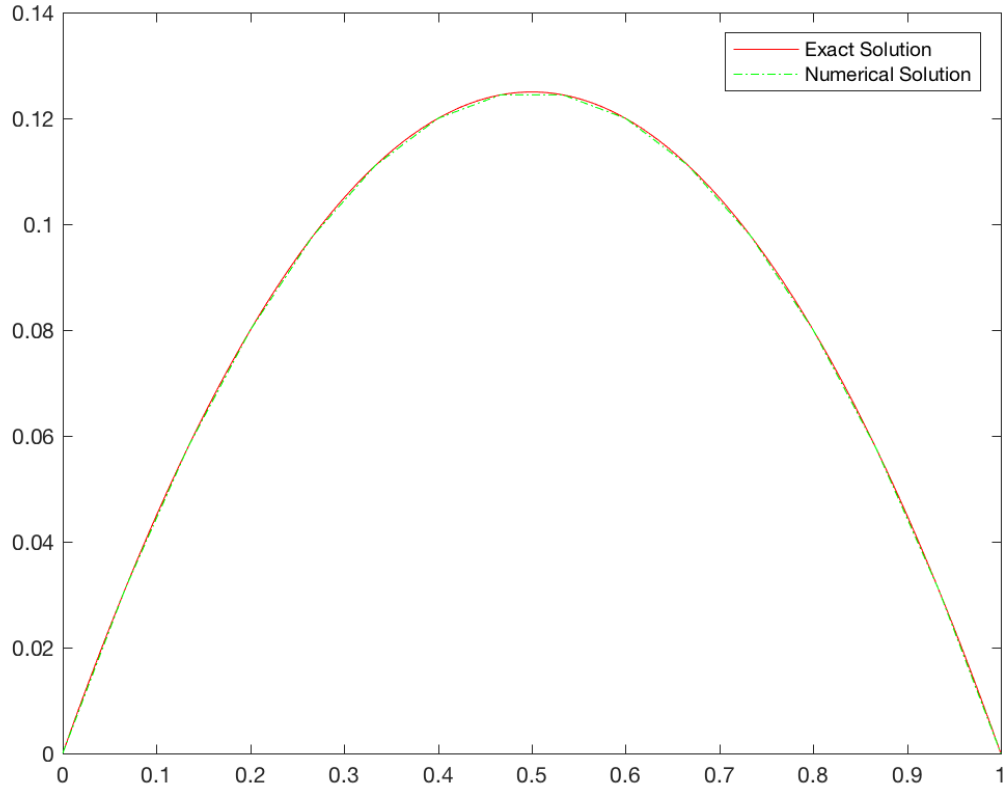


Figure 1: Plot of the exact solution and the numerical solution for $f(x) = 1$

f(x)=x

With the $f(x) = x$ in the same way we find that

$$\alpha(x) = \frac{x^2}{2}, \qquad \beta(x) = \frac{x^3}{3}$$

and the exact solution

$$u(x) = \frac{x - x^3}{6}$$

Now let's see the numerical implementation

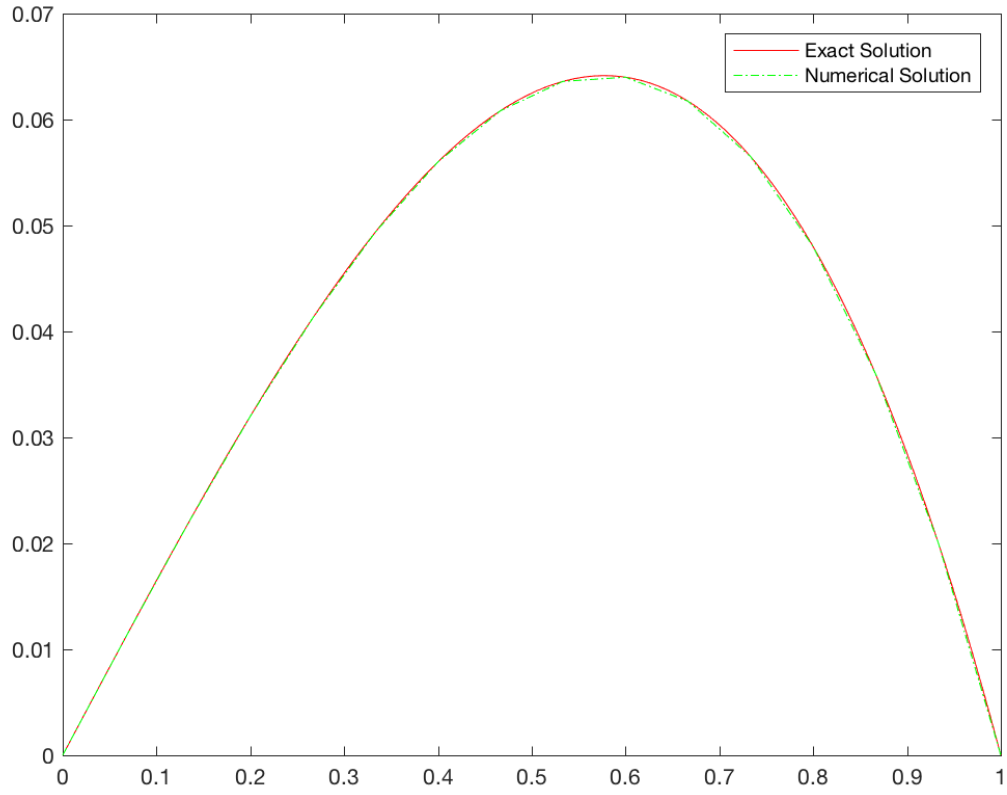


Figure 2: Plot of the exact solution and the numerical solution for $f(x) = x$

$\mathbf{f(x)}=e^x$

For $f(x) = e^x$ we will find the solution also with a standard procedure : the general solution of $u''(x) = 0$ is $c_1 + c_2x$ and a particular solution of $(\star)$ is $-e^x$ thus the solution of the problem is

$$u(x) = c_1 + c_2x - e^x$$

we impose the ICs

$$\begin{cases} u(0) = 0 \Rightarrow c_1 - 1 = 0 & \Rightarrow c_1 = 1 \\ u(1) = 0 \Rightarrow c_1 + c_2 - e = 0 & \Rightarrow c_2 = e - 1 \end{cases}$$

and we obtain

$$u(x) = 1 + (e - 1)x - e^{-x}$$

or we could use again the Green's convolution obtaining the same result with

$$\alpha(x) = e^x - 1 \qquad \beta(x) = e^x(x - 1) + 1$$

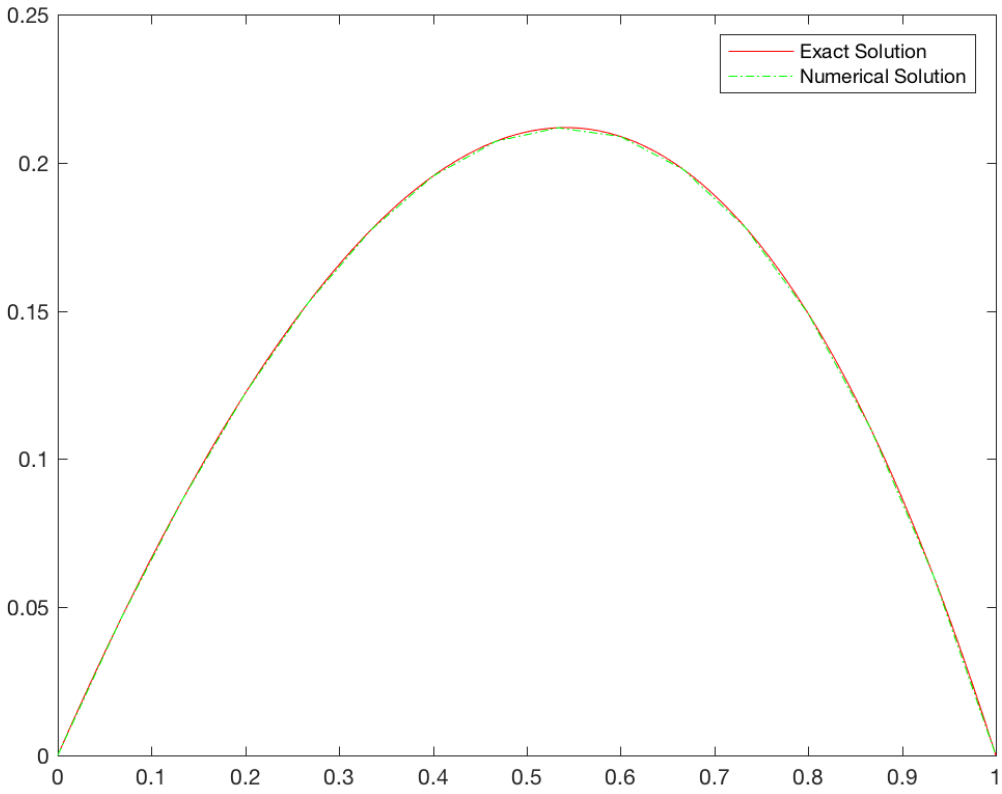


Figure 3: Plot of the exact solution and the numerical solution for $f(x) = e^x$

It can be seen that even with only 16 nodes (the internal nodes are 14) we have such a good approximation of the solution, if we were to increase the nodes we would expect the error to go down, this is true in the case of $f(x) = e^x$ but in the other two cases we see that the error increases as the number of nodes increases(probably because of the machine error since the error in these two cases reach order of $10^{-16} - 10^{-17}$). Now we put in a table the values of the error for different number of nodes for $f(x) = e^x$ and do a log plot of it to see the order of the truncation error.

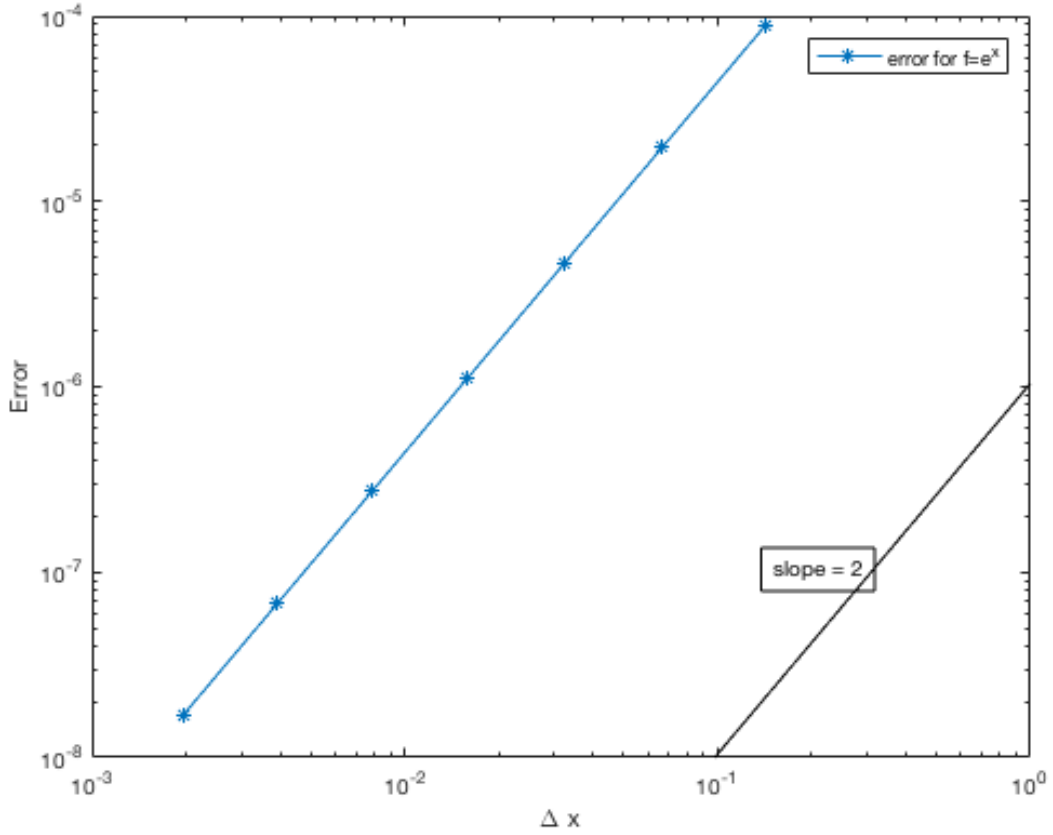


Figure 4: Plot of the exact solution and the numerical solution for $f(x) = e^x$

numer of nodes	8	16	32	64	128	256	512
error	$8.97 \cdot 10^{-}5$	$1.96 \cdot 10^{-}5$	$4.59 \cdot 10^{-}6$	$1.11 \cdot 10^{-}6$	$2.73 \cdot 10^{-}7$	$6.78 \cdot 10^{-}8$	$1.69 \cdot 10^{-}8$

We can conclude from the log plot that the order of error for $f(x) = e^x$ is Δx^2 where Δx is the distance between the nodes.

Exercise 2

We study the following problem

$$\begin{cases} \dfrac{\partial u}{\partial t} - \dfrac{\partial}{\partial x}(\varepsilon(x)\dfrac{\partial u}{\partial x}) = f(t,x) \quad \text{in} \quad Q = (0,1)\times(0,1), \\ u(t,0) = 0, \\ u(t,1) = 1, \\ u(0,x) = x + \sin(\pi x) \end{cases} \tag{$*$}$$

where $\varepsilon(x) = 2+\cos(\pi x)$, we compute $f(x)$ such as $u(t,x) = x+\sin(\pi x)e^{-t}$ is the exact solution using $(*)$ we have that this $u(t,x)$ satisfies the BCs and ICs and

$$\frac{\partial u}{\partial t}(t,x) = -\sin(\pi x)e^{-t}$$

$$\frac{\partial u}{\partial x}(t,x) = 1 + \pi\cos(\pi x)e^{-t}$$

$$\frac{\partial}{\partial x}(\varepsilon(x)\frac{\partial u}{\partial x}(t,x)) = \frac{\partial \varepsilon}{\partial x}\frac{\partial u}{\partial x} + \varepsilon\frac{\partial^2 u}{\partial x^2} = (-\pi\sin(\pi x))(1+\pi\cos(\pi x)e^{-t}) + (2+\cos(\pi x))(-\pi^2\sin(\pi x)e^{-t})$$

putting all together and using $(*)$ we find $f(t,x)$

$$f(t,x) = e^{-t}\sin(\pi x)(\pi e^t + 2\pi^2\cos(\pi x) + 2\pi^2 - 1)$$

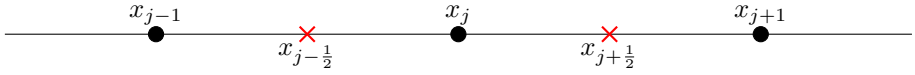
Now let's start with the discretisation of the problem with centered finite differences, we will use the following time discretisation scheme with $\theta \in [0,1]$:

$$\frac{\boldsymbol{u}^{(k+1)} - \boldsymbol{u}^{(k)}}{\Delta t} = \theta A\boldsymbol{u}^{(k+1)} + (1-\theta)A\boldsymbol{u}^{(k)} + \theta\boldsymbol{f}^{(k+1)} + (1-\theta)\boldsymbol{f}^{(k)}$$

the centered finite differences scheme for the space variable x of $(*)$ which is a non linear problem is :

$$\frac{\partial}{\partial x}(\varepsilon\frac{\partial u}{\partial x}) \approx \frac{\varepsilon_{j+1/2}(u^t_{j+1} - u^t_j) - \varepsilon_{j-1/2}(u^t_j - u^t_{j-1/2})}{\Delta x^2} \quad \forall t \in (0,1)$$

so the matrix A is tridiagonal matrix having as elements $(\varepsilon_{j+1/2}, -\varepsilon_{j+1/2} - \varepsilon_{j-1/2}, \varepsilon_{j-1/2})$ where $\varepsilon_{j\pm1/2} = \varepsilon(x_{j\pm1/2})$ where $x_{j\pm1/2}$ is the node between x_j and $x_{j\pm1}$.



So the matrix A for n nodes is :

$$A = \begin{pmatrix} -\varepsilon_{1+1/2} - \varepsilon_{1-1/2} & \varepsilon_{1+1/2} & 0 & 0 & \dots & 0 \\ \varepsilon_{2-1/2} & -\varepsilon_{2+1/2} - \varepsilon_{2-1/2} & \varepsilon_{2+1/2} & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & \varepsilon_{n-1-1/2} & -\varepsilon_{n-1+1/2} - \varepsilon_{n-1-1/2} & \varepsilon_{n-1+1/2} \\ 0 & \dots & 0 & 0 & \varepsilon_{n-1/2} & -\varepsilon_{n+1/2} - \varepsilon_{n-1/2} \end{pmatrix}$$

We observe that the matrix A is independent from t because ε is independent and since the values of $u_0^t = u_0^{t+1} = u(0,t)$ and $u_n^t = u_n^{t+1} = u(1,t)$ are known $\forall t$ due to the BCs, the coefficients of the nodes x_0 and x_n does not appear in the matrix A, we will add them as a known term $\boldsymbol{b}(\theta) = [\theta\varepsilon_{1/2}u_0^{t+1} + (1-\theta)\varepsilon_{1/2}u_0^t,\, 0, \dots,\, 0,\, \theta\varepsilon_{n+1/2}u_n^{t+1} + (1-\theta)\varepsilon_{n+1/2}u_n^t]$. Now returning on the time discretization we note that for $\theta = 0$ the method is

$$\frac{\boldsymbol{u}^{(k+1)} - \boldsymbol{u}^{(k)}}{\Delta t} = \frac{A\boldsymbol{u}^{(k)}}{\Delta x^2} + \boldsymbol{f}^{(k)} + \boldsymbol{b}(0)$$

which is forward difference in time (the right hand side depends on t_k so the derivate is a forward derivative in time) and is centered in space scheme.
For $\theta = 1$ the method is

$$\frac{\boldsymbol{u}^{(k+1)} - \boldsymbol{u}^{(k)}}{\Delta t} = \frac{A\boldsymbol{u}^{(k+1)}}{\Delta x^2} + \boldsymbol{f}^{(k+1)} + \boldsymbol{b}(1)$$

which is backward difference in time (the right hand side depends on t_{k+1} so the derivate is a backward derivative in time) and is centered in space scheme.
The only θ for which the method is totally explicit is when the right hand side does not depend on t_{k+1} iff $\theta = 0$ in this case we need the find the condition of convergence, assume $f = 0$, n_t and n_x are the number of nodes of t and x respectively, the method explicitly is

$$\frac{u_j^{k+1} - u_j^k}{\Delta t} = \frac{\varepsilon_{j+1/2}(u_{j+1}^k - u_j^k) - \varepsilon_{j-1/2}(u_j^k - u_{j-1/2}^k)}{\Delta x^2}$$

now we linearize the problem near a point (x_0, t_0) setting $\varepsilon_0 = \varepsilon(x_0)$ and proceeding with discrete separation of variable finding the condition $1 - 4\varepsilon_0 \frac{\Delta t}{\Delta x^2} \leq 1$ thus

$$\frac{\varepsilon_0 \Delta t}{\Delta x^2} \leq \frac{1}{2} \Leftrightarrow \Delta t \leq \frac{\Delta x^2}{2\varepsilon_0}$$

with is only a local condition for a neighbourhood of (x_0, t_0) but we note that

$$\varepsilon(x) = 2 + \cos(\pi x) \leq 2 + 1 = 3 \qquad \forall (x,t) \in Q$$

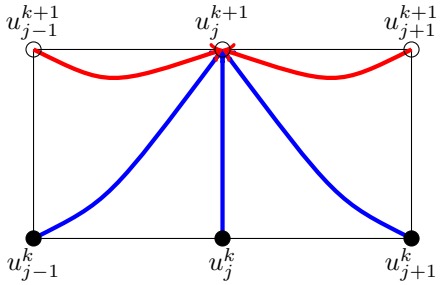
so by choosing $\varepsilon_0 = 3$ we find a global condition for the stability:

$$\Delta t \leq \frac{\Delta x^2}{2\varepsilon_0} = \frac{\Delta x^2}{6}$$

for all other θ the method is implicit so unconditionally stable.

Implementation for $\theta = \frac{1}{2}$

For $\theta = \frac{1}{2}$ the method is half dependent on forward derivative and half on backward derivative for time. the computational molecule is :



In the molecule the red lines indicate the implicit dependence and the blue lines indicate the explicit dependence, and the filled elements are the one known instead the ones empty are unknown when calculating the value of u_j^{k+1}. There are some "special" elements near the boundary for which the value of either u_{j-1}^{k+1} or u_{j+1}^{k+1} is known because these are the elements of the boundary.
Due to all the discussion above for $(\boldsymbol{x}, t_{k+1})$ we have that $u(\boldsymbol{x}, t_{k+1}) = \boldsymbol{u}^{k+1}$ is calculated with the following scheme :

$$(\frac{\boldsymbol{I}}{\Delta t} - \frac{\boldsymbol{A}}{2\Delta x^2})\boldsymbol{u}^{k+1} = (\frac{\boldsymbol{I}}{\Delta t} + \frac{\boldsymbol{A}}{2\Delta x^2})\boldsymbol{u}^{k} + \frac{\boldsymbol{f}(\boldsymbol{x}, t_{k+1})}{2} + \frac{\boldsymbol{f}(\boldsymbol{x}, t_k)}{2} + \boldsymbol{b}(\frac{1}{2})$$

where the vector $\boldsymbol{b}(\frac{1}{2})$ is the vector of knows boundary elements we discussed before, if we call $\tilde{\boldsymbol{A}}$ the matrix on the left hand side and $\tilde{\boldsymbol{b}}$ the term on the right hand side we have a linear system

$$\tilde{\boldsymbol{A}}\boldsymbol{u}^{k+1} = \tilde{\boldsymbol{b}}$$

which can be solved using any linear system solver, we will use the incorporated method in MATLAB $\boldsymbol{u}^{k+1} = \tilde{\boldsymbol{A}} \backslash \tilde{\boldsymbol{b}}$; notice that the matrix $\tilde{\boldsymbol{A}}$ is non singular, one can see it through the fact that it's eigenvalues are positive non zero, so it can be inverted thus it has non trivial solution. Since the method is implicit we have that the stability of the solution is guaranteed for every choice of Δt and Δx even though we lose accuracy for big values of Δt or Δx.
We will use $\Delta t = \Delta x = 0.05$ (21 nodes) for the following mesh plots, in which we can see the precision of the method, it has a error of 0.0019. Another thing is that the solution

$$u(x, t) = x + \sin(\pi x)e^{-t} \to x \qquad \text{for } t \to +\infty$$

which is nearly reached at $t = 5$.

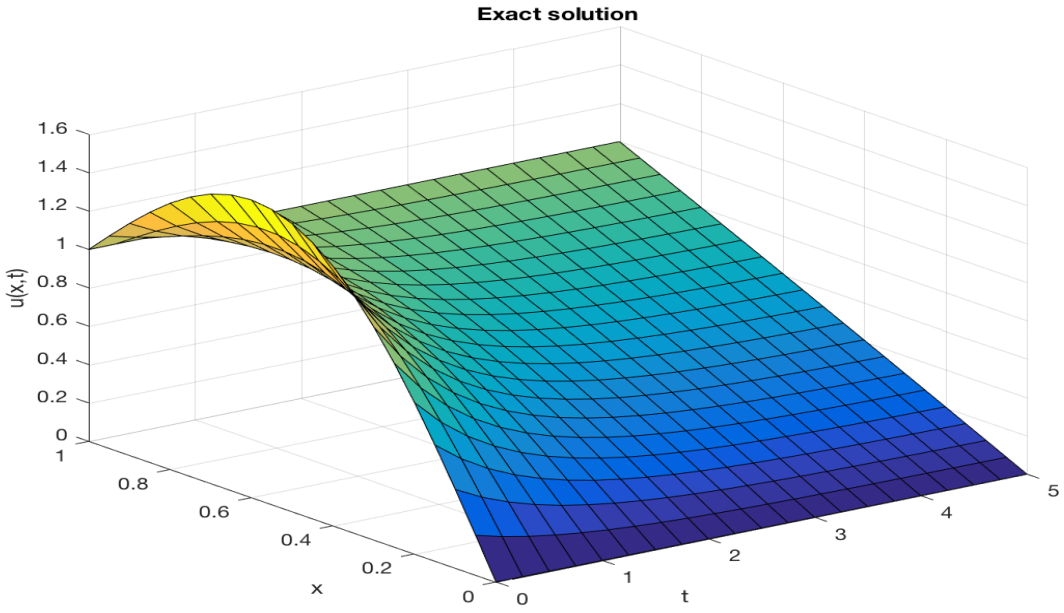


Figure 5: Surface plot of the exact solution of the problem $(*)$

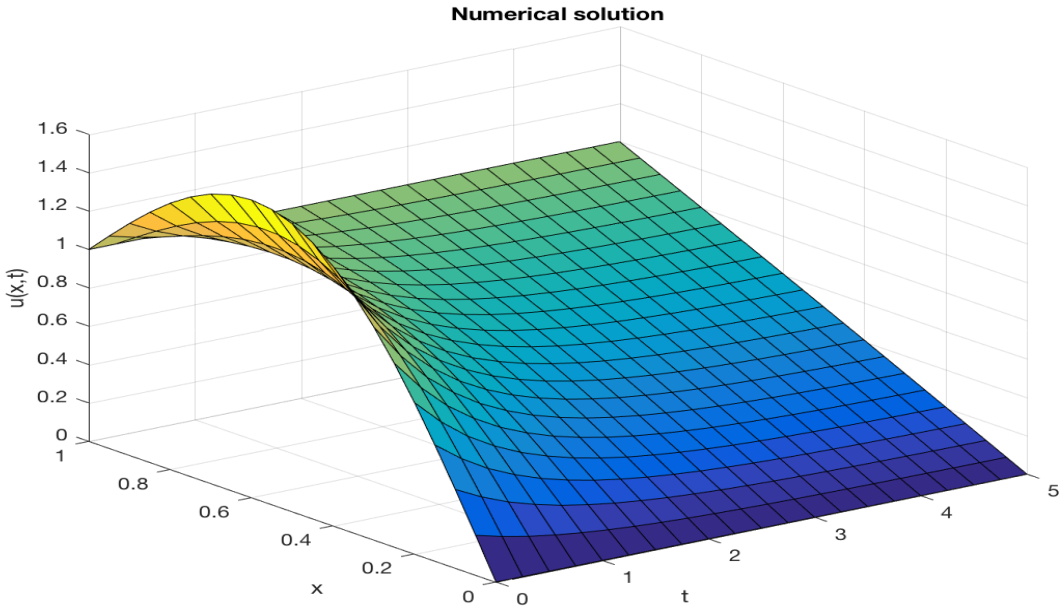


Figure 6: Surface plot of the numerical solution of the problem $(*)$

Error of the Scheme

We will now see the order of the truncation error, first in Δx and then is Δt, to make sure one variable's error does not influence the other we will fix one of them and choose different values for the other and do a log plot, and for the one fixed we will choose a low value so even that error is negligible. First we choose $\Delta t = 2.5 \cdot 10^{-3}$ and we put in a table the various Δx and the relative errors.

Δx	$2 \cdot 10^{-2}$	$1 \cdot 10^{-2}$	$5 \cdot 10^{-3}$	$2.5 \cdot 10^{-3}$	$1.25 \cdot 10^{-3}$
Error	$3 \cdot 10^{-4}$	$7.5 \cdot 10^{-5}$	$1.87 \cdot 10^{-5}$	$4.67 \cdot 10^{-6}$	$1.15 \cdot 10^{-6}$

Now we do a log plot to find the slope of the error line thus approximating the error order

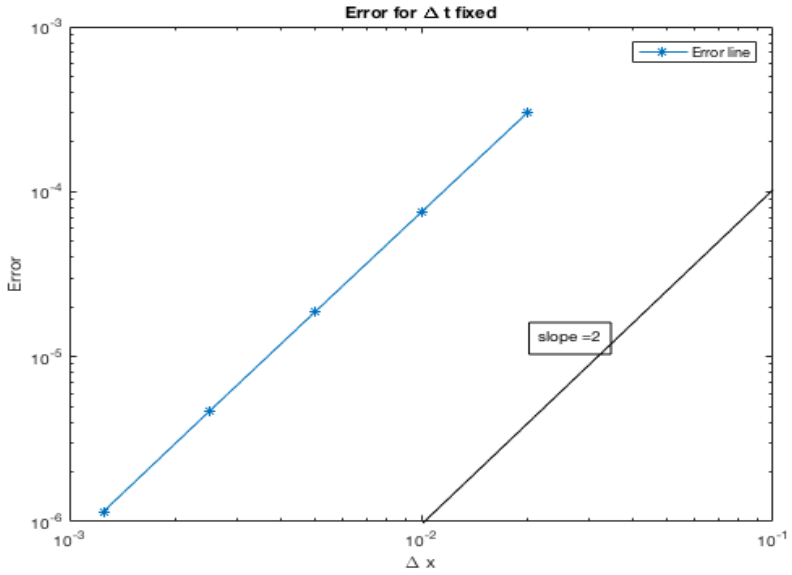


Figure 7: log plot of error with Δt fixed

The truncation error is of order Δx^2 which was to expected since we used finite difference scheme which itself has an error of order 2.

Now let's see truncation error order for Δt, we fix $\Delta x = 5 \cdot 10^{-4}$ and in the table we put all the Δt with their errors.

Δt	$1 \cdot 10^{-1}$	$5 \cdot 10^{-2}$	$2.5 \cdot 10^{-2}$	$1.25 \cdot 10^{-2}$	$6.25 \cdot 10^{-3}$	$3.125 \cdot 10^{-3}$
Error	$3.1 \cdot 10^{-5}$	$9.6 \cdot 10^{-6}$	$2.27 \cdot 10^{-6}$	$4.64 \cdot 10^{-7}$	$7.12 \cdot 10^{-8}$	$1.55 \cdot 10^{-7}$

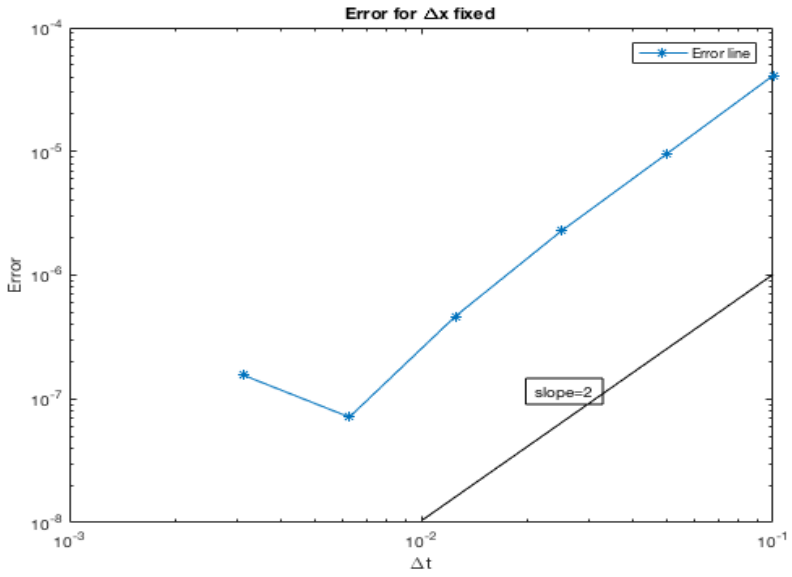


Figure 8: log plot of error with Δx fixed

Now let's comment the Figure 8, what we can see is that until $\Delta t = 1.25 \cdot 10^{-2}$ the method seems to have order 2 (error= $O(\Delta t^2)$) and for $\Delta t = 6.25 \cdot 10^{-3}$ we start seeing a different slope and for $\Delta t = 3.125 \cdot 10^{-3}$ the error increases this is due to the fact that Δt is very close to Δx so the error of Δx is no more negligible, this is the reason for that "strange" bump at $\Delta t = 6.25 \cdot 10^{-3}$ and for further reduction of Δt the error will start increasing, so we can conclude that if we for a moment we forget about Δx (i.e $\Delta x << \Delta t$) this scheme has a truncation error of order 2 in Δt (error = $O(\Delta t^2)$).

Exercise 3

In the following system of diffusion equation for the population u and v in spacial domain $\Omega = (0,1)$,

$$\begin{cases} u_t = u_{xx} + uM(u,v) \\ v_t = v_{xx} + vN(u,v) \end{cases} \tag{$\star$}$$

we consider them with null Neumann condition at the boundary,

$$u_x(0,t) = u_x(1,t) = 0, \quad v_x(0,t) = v_x(1,t) = 0 \tag{1}$$

We will use the scheme in Exercise 2 with $\theta = 0$

$$\begin{cases} \dfrac{u_j^{k+1} - u_j^k}{\Delta t} = \dfrac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{\Delta x^2} + u_j^k(M(u_j^k, v_j^k)) \\ \dfrac{v_j^{k+1} - v_j^k}{\Delta t} = \dfrac{v_{j+1}^k - 2v_j^k + v_{j-1}^k}{\Delta x^2} + v_j^k(N(u_j^k, v_j^k)) \end{cases}$$

$\forall j = 1, \ldots, n-1$ and $k = 1, \ldots, m$,where n are the number of nodes in space and m are the number of nodes in time and $u_0\,, v_0$ and $u_n\,, v_n$ will be found with the BCs, we explicitly find u_k^{k+1} and v_k^{k+1}

$$\begin{cases} u_j^{k+1} = u_j^k + \Delta t \cdot \dfrac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{\Delta x^2} + \Delta t \cdot u_j^k(M(u_j^k, v_j^k)) \\ v_j^{k+1} = v_j^k + \Delta t \cdot \dfrac{v_{j+1}^k - 2v_j^k + v_{j-1}^k}{\Delta x^2} + \Delta t \cdot v_j^k(N(u_j^k, v_j^k)) \end{cases}$$

the explicitness of the method implies the condition of stability: $\Delta t \leq \dfrac{\Delta x^2}{2}$.
The system $(\star)$ is a diffusion equation for both u and v so this avoids any kind of overpopulation concentrated in one small region of Ω and the terms ,$uM(u,v)$ and $vN(u,v)$, depends on the population itself so by the choice of M and N we can decide, for example, if the two population lives in symbiosis or if one is a predator and the other is a prey.

Note : In MATLAB we will use $\Delta x = 0.05$ and $\Delta t = 0.001$ to guarantee the stability. While the final moment (t_f) may vary from example to example.For the Neumann condition $u_x(0,t) \approx (u_1^t - u_0^t)/\Delta x = 0$ if we were to translate it as $u_0^t = u_1^t$ same for $u_x(1,t) = 0$ ($\Rightarrow u_n^t = u_{n-1}^t$), then we would lose accuracy that's why we use the ghost nodes and maintain our accuracy ($u_{-1}^t = u_1^t$ and $u_{n+1}^t = u_{n-1}^t$ and now solving as if we had n+2 nodes in x) same for v.

Now we choose $M(u,v) = 1 - v$ and $N(u,v) = u - 1$, we note that the solution $u = 0$, $v = 0$ are the trivial solution but they are a equilibrium, so is $u = 1$, $v = 1$, if the ICs of u and v are near 0, $u_{in} = \varepsilon_x$ and $v_{in} = \varepsilon_y$ with ε_x and ε_y are small numbers then the populations start oscillating rapidly reaching high numbers and then returning back to very near to zero as we can see in Figure 9 (for the plot we used $\varepsilon_x = 0.001$ and $\varepsilon_y = 0.01\,\forall x \in \Omega$), and the peak's value increases with time:

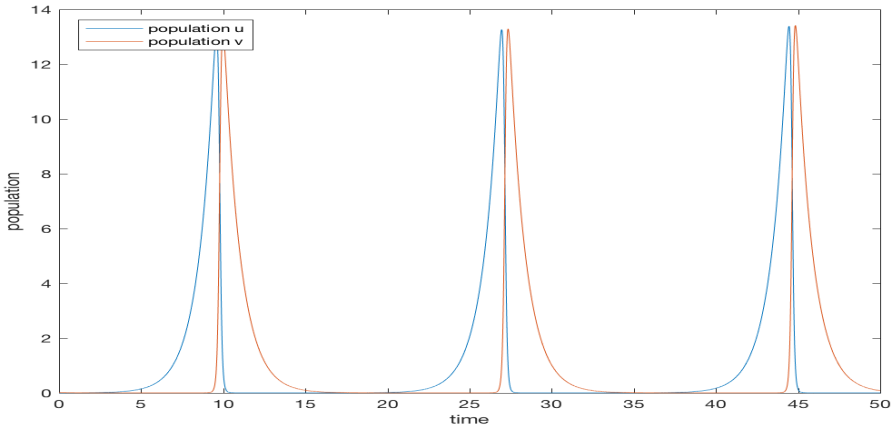


Figure 9: plot with ICs near zero

If we put the ICs near 1, $u_{in} = 1 \pm \varepsilon_x$ and $v_{in} = 1 \pm \varepsilon_y$ with ε_x and ε_y are small numbers then the populations start oscillating but not as much as near zero but still the maximum and minimum population increases with time (Figure 10) probably due to numerical errors since this solution should be a stable solution. We chose $u_{in} = 1.001$ and $v_{in} = 0.99\,\forall x \in \Omega$.

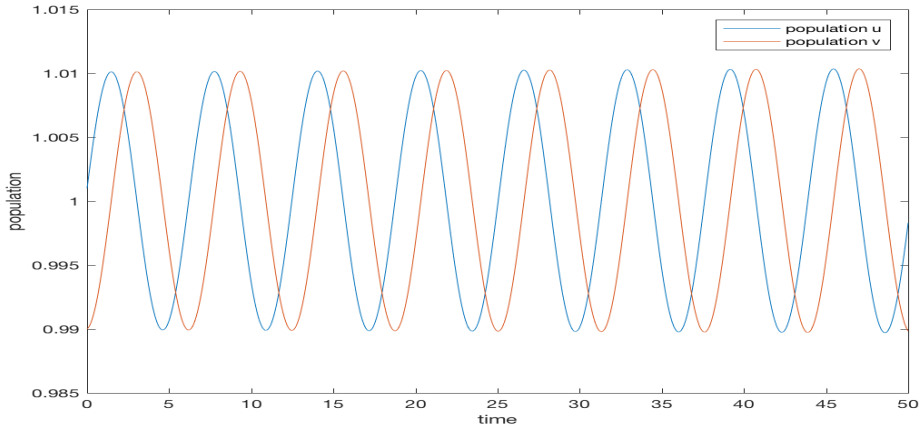


Figure 10: plot with ICs near One

Now if we assume $u = 0$ and $v \neq 0$ so the second equation of $(\star)$ becomes $v_t = v_{xx} - v$ whose solution has an exponential decay, we can see it through our MATLAB implementation Figure 11:

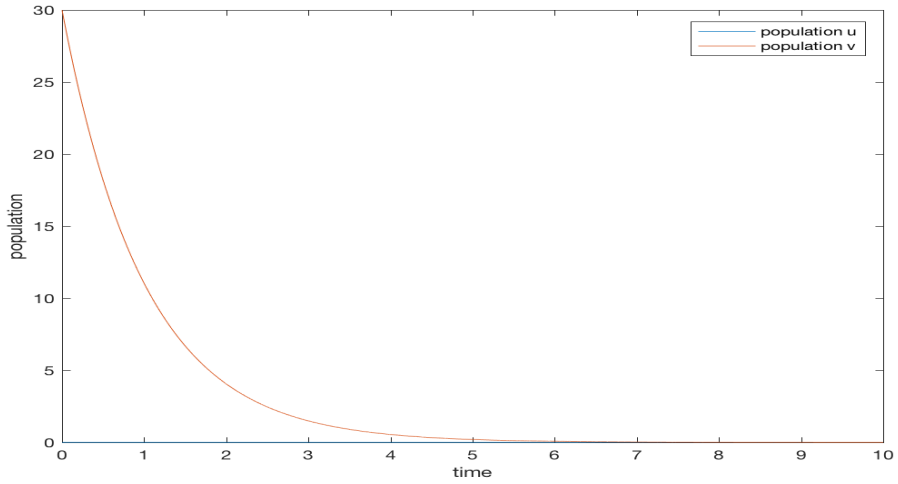


Figure 11: plot with $u = 0$ and $v_{in} = 30 \, \forall x \in \Omega$

But if we assume $u \neq 0$ and $v = 0$, the second equation of $(\star)$ becomes $u_t = u_{xx} + u$ whose solution has an exponential growth, let's see it through our MATLAB implementation Figure 12:

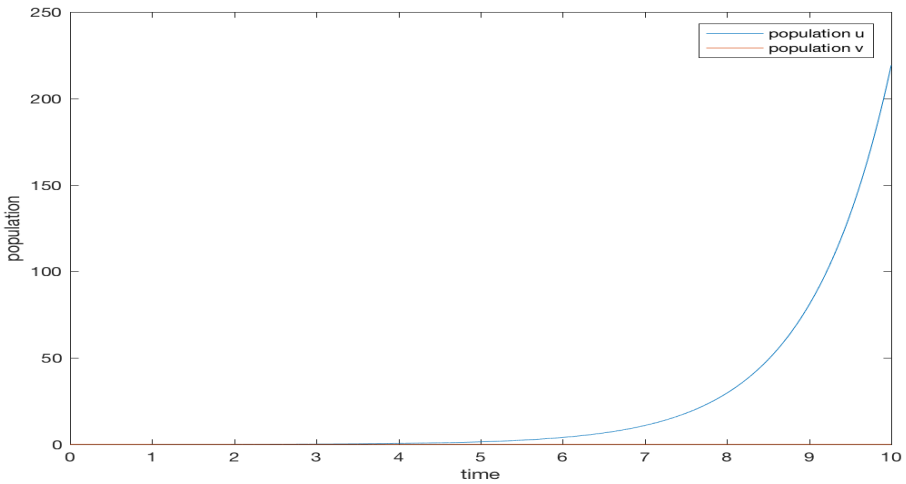


Figure 12: plot with $v = 0$ and $u_{in} = 0.01 \, \forall x \in \Omega$

There two examples explains that the growth of the population v needs the population u, on

the contrary the growth of the population u is slowed and reduced due to the population v, this observation is also verified in the first two examples, by looking the phase difference in the Figure 9 and 10.
Another observation that confirms our precedent observation is that if we fix u for a moment and if at a certain time $v > 1$ then the source/well $u(1-v)(< 0)$ helps the population u to decrease, on the other hand if $0 < v < 1$ then the source/well helps in the growth and if we fix v for a moment and if at a certain time $u > 1$ then the source/well $v(u-1)(> 0)$ helps the population v to grow, on the other hand if $0 < u < 1$ then the source/well helps in the shrinking of the population v.
Let's consider a different case in which the population u is already present and the population v does not exist in Ω and we manually introduce a certain amount of v on both border so

$$v_{in} = \begin{cases} 0 & \text{if } 0 < x < 1 \\ 2 & \text{if } x = 0, 1 \end{cases}$$

while $u_{in} = 1 \, \forall x \in \Omega$. The equation $(\star)$ diffuses the population v over Ω rapidly and then after a little bit it is as if v and u were constant and they start competing for survival. We do a 3D plot now to see the behaviour over time in Figure 13 :

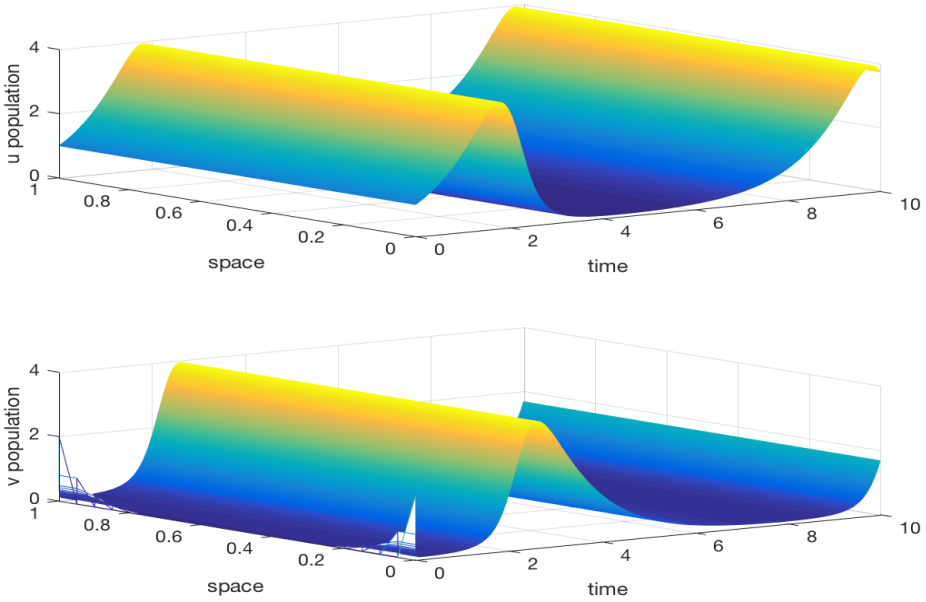


Figure 13: plot with the introduction of the species v and $u_{in} = 1 \, \forall x \in \Omega$

We can see that the behaviour of the population remain oscillatory. This is true even if the ICs of the two populations were not constant, in fact, even if the ICs were not constant the diffusive property flattens the ICs so after some time it becomes nearly a constant. We can see it with an example let's take $u_{in} = -x^3 + \frac{3}{2}x^2$ and $v_{in} = \cos^2(\pi x)$ (we take $\cos^2$ so that v is not negative).

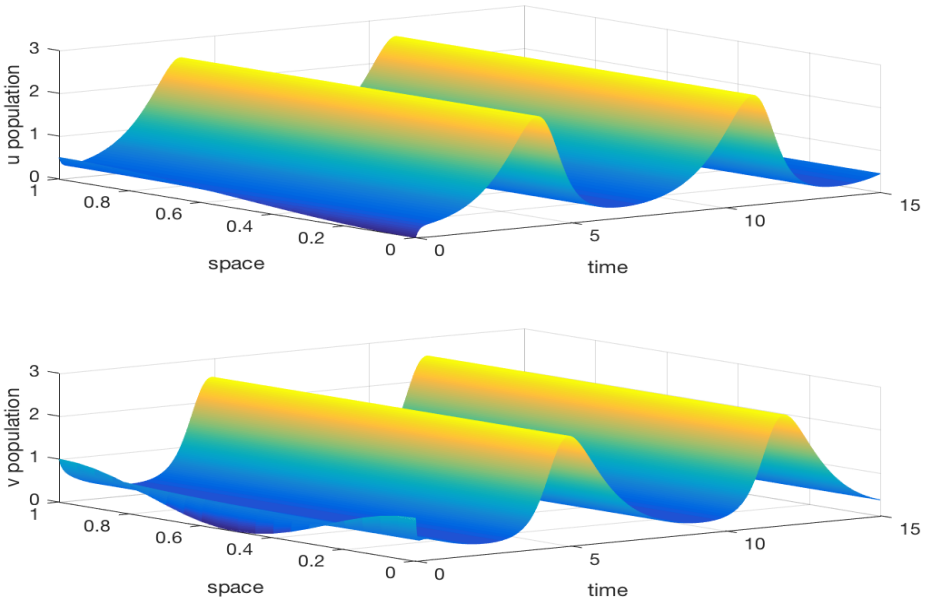


Figure 14: plot with $v_{in} = \cos^2(\pi x)$ and $u_{in} = -x^3 + \frac{3}{2}x^2$

In Figure 14 we can see the usual behaviour of the problem $(\star)$, the two population if not both zero or one will start following this oscillatory behaviour with a phase between due to the fact that v need u to grow before growing himself and during this process the number of u falls so after a bit also the number of v falls, at this point the environment is favourable for the growth of u and this process repeats over and over again.
The fact that in Figure 14, u and v has become constant after a certain period can be seen in Figure 15 where we plot some section of the Figure 14.

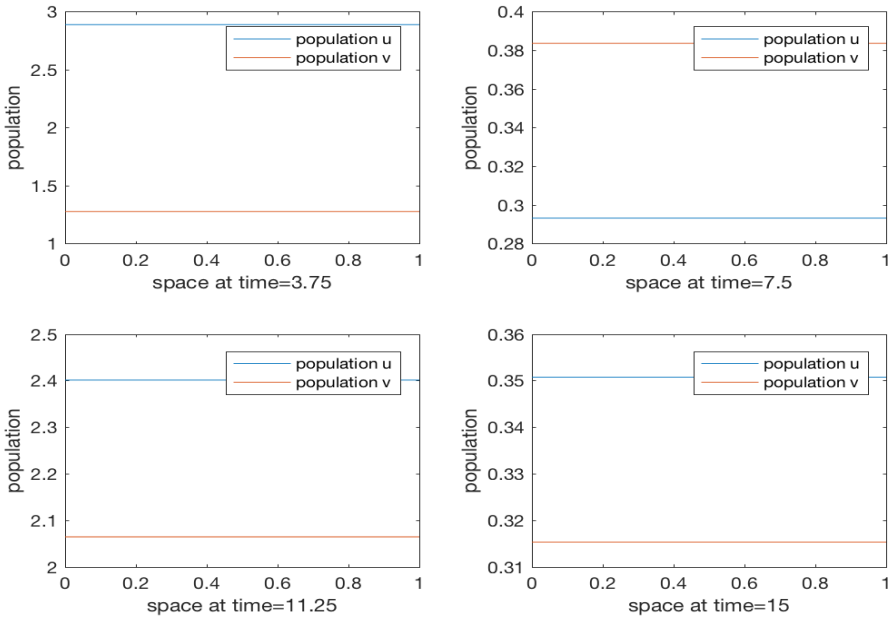


Figure 15: Sub plot of Figure 14 at different times

MATLAB Codes

Exercise 1 Code

```
%% Exercise 1
clear all, close all

xi=0; xf=1;
nn=[15];%thus total nodes are 16
%% Problem f(x)=1

alpha_ex=@(x) x;
beta_ex=@(x) x.^2/2;
f=@(x) 1;
u_ex=@(x) (x-x.^2)/2;

%% Problem f(x)=x

% alpha_ex=@(x) x.^2/2;
% beta_ex=@(x) x.^3/3;
% f=@(x) x;
% u_ex=@(x) (x-x.^3)/6;

%% Problem f(x)=exp(x)

% alpha_ex=@(x) exp(x)-1;
% beta_ex=@(x) exp(x).*(x-1)+1;
% f=@(x) exp(x);
% u_ex=@(x) 1+(exp(1)-1)*x-exp(x);

%% main
alpha(1)=0;
beta(1)=0;
x(1)=0;
u(1)=0;

```

```
z=linspace(0,1);
for k=1:numel(nn)
    n=nn(k);
    h=(xf-xi)/n;
    hh(k)=h;
    for i=1:n

        x(i+1)=x(i)+h;
        x12=x(i)+h/2;
        alpha(i+1)=alpha(i) + h/4*(f(x(i))+2*f(x12)+f(x(i+1)));
        beta(i+1)=beta(i)+h/4*(x(i)*f(x(i))+2*x12*f(x12)+x(i+1)*f(x(i
            +1)));
    end

    for i=1:n
        u(i+1)=x(i+1)*(alpha(end)-beta(end))+beta(i+1)-x(i+1)*alpha(i
            +1);
    end

    % plot(z,alpha_ex(z),'r'), hold on
    % plot(x,alpha,'g')
    %
    % figure(2), plot(z,beta_ex(z),'r'), hold on
    % plot(x,beta,'g')

    plot(z,u_ex(z),'r'), hold on
    plot(x,u,'g-.')
    legend('Exact Solution','Numerical Solution')
    %print -dpng Solf_exp(x) %to save the image
    err(k)=max(abs(u_ex(x)-u));
end
close all
loglog(hh,err,'-*')
legend('error for f=e^x')
xlabel('\Delta x')
ylabel('Error')
```

Exercise 2 Code

```
%% Exercise 2
clear all, close all
%% Problem details
xi=0; xf=1;
theta=1/2;
ti=0; tf=1;

```

```
epsilon=@(x) 2+cos(pi*x);
uex=@(x,t) x+sin(pi*x).*exp(-t);
uIC=@(x) x+sin(pi*x);
u0=0; uL=1; %\forall t
%NNx=[50 100 200 400 800];
NNy=[10 20 40 80 160 320];
for l=1:numel(NNy)
%Nx=NNx(l); %node on x
Nx=2000;
Ny=NNy(l); %nodes on t
nx=Nx-1; %internal nodes
h=(xf-xi)/Nx; %delta x
%hh(l)=h;
ht=(tf-ti)/Ny; %delta t
hh(l)=ht;
clear u
x=linspace(xi,xf,Nx+1);
t=linspace(ti,tf,Ny+1);
u(:,1)=uIC(x(2:end-1));
%% calculation of f(x,t)
% syms u(x,t) eps(x)
%
% u=x+sin(pi*x).*exp(-t);
% eps=2+cos(pi*x);
%
% ut=diff(u,'t');
% ux=diff(u,'x');
% f=ut-diff(eps.*ux,'x');
% f=simplify(f);
f=@(x,t) exp(-t).*sin(pi*x).*(pi*exp(t) + 2*pi^2*cos(pi*x) + 2*pi^2 -
    1);
    %% create matrix A (indipendent from time)
    A=zeros(nx);

    for j=2:nx-1
        xm12=(j*h-h/2); %node x_{j-1/2}
        xp12=(j*h+h/2); %node x_{j+1/2}
        A(j,j-1)=-epsilon(xm12);
        A(j,j)=(epsilon(xm12)+epsilon(xp12));
        A(j,j+1)=-epsilon(xp12);
    end

    %first row of matrix A
    xm12=h/2; xp12=3/2*h;
    A(1,1)=epsilon(xm12)+epsilon(xp12);
    A(1,2)=-epsilon(xp12);
```

```
53
54      %last row of matrix A
55      xm12=xf-3/2*h; xp12=xf-h/2;
56      A(end,end)=epsilon(xm12)+epsilon(xp12);
57      A(end,end-1)=-epsilon(xm12);
58
59      A=A/h^2;
60
61      Atilde=eye(nx)/ht+theta*A;
62
63  %% calculate u
64  for k=2:numel(t)
65      %% known term b
66
67      fk=zeros(nx,1);
68      for i=1:nx
69          fk(i)=f(i*h,t(k-1));
70      end
71
72      fk1=zeros(nx,1);
73      for i=1:nx
74          fk1(i)=f(i*h,t(k));
75      end
76
77      u_remaining=zeros(nx,1);
78      u_remaining(1)=(1-theta)*u0/h^2*epsilon(xm12)+theta*u0/h^2*epsilon(
            xm12);
79      u_remaining(end)=(1-theta)*uL/h^2*epsilon(xp12)+theta*uL/h^2*
            epsilon(xp12);
80
81      btilde = (eye(nx)/ht-(1-theta)*A)*u(:,k-1)+theta*fk1+(1-theta)*fk+
            u_remaining;
82
83      %% calculate u_{k+1}
84      u(:,k)=Atilde\btilde;
85  end
86  %% plot of u
87
88   u=[u0*ones(1,Ny+1);u;uL*ones(1,Ny+1)];
89   x=xi:h:xf; t=ti:ht:tf;
90   [T,X]=meshgrid(t,x);
91   surf(T,X,uex(X,T))
92   title('Exact solution')
93   xlabel('t')
94   ylabel('x')
95   zlabel('u(x,t)')
```

```
 %print -dpng Ex2ExSol
 figure, surf(T,X,u)
  title('Numerical solution')
 xlabel('t')
 ylabel('x')
 zlabel('u(x,t)')
 %print -dpng Ex2NumSol


 error(l)=max(max(abs(u-uex(X,T))));
end

figure, loglog(hh,error,'-*')
xlabel('\Deltat')
ylabel('Error')
title('Error for \Deltax fixed')
legend('Error line')
```

Exercise 3 Code

```
%% Exercise 3
clear all, close all

xi=0; xf=1;
dx=0.05;
x=xi:dx:xf;

ti=0; tf=15;
dt=0.8*0.00125;
t=ti:dt:tf;


%ICs
%u_in= 1*(ones(numel(x),1));
u_in=-x.^3'+(3/2)*x.^2';
%v_in= 0*(ones(numel(x),1));
v_in=cos(pi*x).^2';

%   v_in(1)=2;
%   v_in(end)=2;

u=zeros(numel(t),numel(x));
v=zeros(numel(t),numel(x));

u(1,:)=u_in;
v(1,:)=v_in;
```

```
for k=1:numel(t)-1
    for j=2:numel(x)-1
        u(k+1,j)=u(k,j)+dt/dx^2*(u(k,j-1)-2*u(k,j)+u(k,j+1))+dt*(u(k,j)
            -u(k,j)*v(k,j));
        v(k+1,j)=v(k,j)+dt/dx^2*(v(k,j-1)-2*v(k,j)+v(k,j+1))+dt*(-v(k,j
            )+u(k,j)*v(k,j));
    end
    u(k+1,1)=u(k,1)+dt/dx^2*(2*u(k,2)-2*u(k,1))+dt*u(k,1)*(1-v(k,1));
    u(k+1,end)=u(k,end)+dt/dx^2*(2*u(k,end-1)-2*u(k,end))+dt*u(k,end)
        *(1-v(k,end));
    v(k+1,1)=v(k,1)+dt/dx^2*(2*v(k,2)-2*v(k,1))+dt*v(k,1)*(u(k,1)-1);
    v(k+1,end)=v(k,end)+dt/dx^2*(2*u(k,end-1)-2*u(k,end))+dt*v(k,end)*(
        u(k,end)-1);
end

%% mesh plot
[T,X]=meshgrid(t,x);
figure, subplot(2,1,1)
mesh(T,X,u')
xlabel('time')
ylabel('space')
zlabel('u population')
subplot(2,1,2),
mesh(T,X,v')
xlabel('time')
ylabel('space')
zlabel('v population')
%print -dpng PopNonConst

 Nt=ceil((tf-ti)/dt);

figure,
plot(x,u(Nt/Nt,:))
hold on, plot(x,v(Nt/Nt,:))
figure,
subplot(2,2,1)
plot(x,u(0.25*Nt,:))
hold on, plot(x,v(0.25*Nt,:))
legend('population u','population v')
ylabel('population')
xlabel('space at time=3.75')


subplot(2,2,2)
plot(x,u(0.50*Nt,:))
hold on, plot(x,v(0.50*Nt,:))
```

```
69 legend('population u','population v')
70 ylabel('population')
71 xlabel('space at time=7.5')
72
73 subplot(2,2,3)
74 plot(x,u(0.75*Nt,:))
75 hold on, plot(x,v(0.75*Nt,:))
76 legend('population u','population v')
77 ylabel('population')
78 xlabel('space at time=11.25')
79
80
81 subplot(2,2,4)
82 plot(x,u(Nt,:))
83 hold on, plot(x,v(Nt,:))
84 legend('population u','population v')
85 ylabel('population')
86 xlabel('space at time=15')
87
88 %print -dpng PopNonConstAtTime
89
90 %% iteractive versione of plot
91 % time=1:numel(t);
92 % figure,
93 % for i=1:numel(time)
94 %     plot(x,u(time(i),:)), hold on
95 %     plot(x,v(time(i),:)),hold off
96 %     legend('population u','population v')
97 %     pause(0.0000001) %increase the value inside for more speed
98 %     axis([xi xf 0 2])
99 % end
100
101 %% plot on on space x=0.5 (useful for contant population on x)
102 % figure,plot(t,u(:,ceil((xf-xi)/(2*dx)))),
103 % hold on, plot(t,v(:,ceil((xf-xi)/(2*dx)))),
104 % legend('population u','population v')
105 % ylabel('population')
106 % xlabel('time')
107 % print -dpng PopV=0
```