Introduction to Scientific Computing Matlab Project

Monica Conte and Riley Kavanagh

6^{th} Jun, 2023

Introduction

This project's main aim is to use numerical techniques to solve partial differential equations (PDE), such as diffusion and diffusion-reaction equations. These equations are very useful for describing many different physical systems from heat dissipation and wave motion, to financial modelling, but our study won't focus on a particular type of physical system, leaving the model widely applicable in many fields.

The project is structured as follows: in the first section we describe the *finite-difference* method (FDM) [1] which is used throughout the entire work, in section Exercise 4.1 we solve the diffusion equation using three different methods to discretize the time component of our solution, in section Exercise 4.2 we solve the diffusion/reaction equation using a method called IMEX for the time variable, in sections Exercise 6.1 and Exercise 6.2 we solve the systems of the previous sections where we modify the spatial derivative in the PDE to make it fractional and in the last section we compare one of the results in Exercise 6.1 with another type of approximation of the fractional spatial derivative.

Finite-Difference Method

Let's start with a one-dimensional scalar function $f(x)$ with periodic boundary conditions, i.e. $f(0) = f(1)$. Our goal is to obtain estimations for the derivatives of f, from Taylor expansions:

$$f(x + \Delta x) = f(x) + \Delta x \frac{df(x)}{dx} + \frac{1}{2}\Delta x^2 \frac{d^2 f(x)}{dx^2} + \dots,$$
$$f(x - \Delta x) = f(x) - \Delta x \frac{df(x)}{dx} + \frac{1}{2}\Delta x^2 \frac{d^2 f(x)}{dx^2} + \dots,$$

this gives the forward and backward finite-difference

$$\text{forward:} \quad \frac{df(x)}{dx} = \frac{f(x + \Delta x) - f(x)}{\Delta x} - \frac{1}{2}\Delta x \frac{df^2}{dx^2} + \mathcal{O}(\Delta x^2),$$
$$\text{backward:} \quad \frac{df(x)}{dx} = \frac{f(x) - f(x - \Delta x)}{\Delta x} + \frac{1}{2}\Delta x \frac{df^2}{dx^2} + \mathcal{O}(\Delta x^2).$$

Note that the linear correction has opposite signs in these two cases (this also gives us the Euler Forward (EF) and Euler Backward (EB) integration techniques that we will use later), so taking

their sum will give an estimation with a quadratic error, called the central finite-difference

$$\frac{df(x)}{dx} = \frac{1}{2}\left(\frac{f(x+\Delta x) - f(x-\Delta x)}{\Delta x}\right) + \mathcal{O}(\Delta x^2).$$

A similar calculation gives the second and third order central finite-difference derivative

$$\frac{df^2(x)}{dx^2} = \frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x)}{\Delta x^2} + \mathcal{O}(\Delta x^2)$$
$$\frac{df^3(x)}{dx^3} = \frac{1}{2}\left(\frac{f(x+2\Delta x) - 2f(x+\Delta x) + 2f(x-\Delta x) - f(x-2\Delta x)}{\Delta x^3}\right) + \mathcal{O}(\Delta x^2).$$

Now we can convert these to central finite-derivative matrices, by discritizing the variable x using intervals of length Δx, as a consequence also the function f assumes discrete values and we obtain:

$$\frac{d^n\vec{f}}{dx^n} = \mathcal{D}_n\vec{f} + \mathcal{O}(\Delta x^2)$$

$$\mathcal{D}_2 = \frac{1}{\Delta x^2}\begin{bmatrix} -2 & 1 & 0 & . & . & . & 1 \\ 1 & -2 & 1 & & & & \\ 0 & 1 & -2 & 1 & & & \\ . & & 1 & -2 & 1 & & \\ . & & & & & & \\ . & & & & & & \\ 1 & & & & & & \end{bmatrix}, \quad \mathcal{D}_3 = \frac{1}{2\Delta x^2}\begin{bmatrix} 0 & -2 & 1 & . & . & -1 & 2 \\ 2 & 0 & -2 & 1 & & & -1 \\ -1 & 2 & 0 & -2 & 1 & & \\ 0 & -1 & 2 & 0 & -2 & 1 & \\ . & & & & & & \\ . & & & & & & \\ 1 & & & & & & \end{bmatrix}.$$

These derivative-matrices allow us to treat the spatial-derivative operator in our heat equations as a linear mapping of a discitized space. This turns our PDE into an Ordinary Differential Equation (ODE), which we can solve either analytically or numerically using, for example, the EF or EB methods.

Exercise 4.1

In this section we consider a one space-dimensional heat equation, with $x \in [0,1]$ and $t \in [0,T]$ (we use $T = 0.1$ in this section), defined by the following set of conditions

$$\begin{aligned} \frac{\partial u}{\partial t} &= \frac{\partial^2 u}{\partial x^2}, \\ u(x,0) &= \sin(\pi x), \\ u(0,t) &= u(1,t) = 0. \end{aligned}$$

This is a great initial differential equation to consider as there is a known exact solution (which can be found via separation of variables): $u(x,t) = e^{-\pi^2 t}\sin(\pi x)$, which can be compared to the results we get via our numerical techniques as a check of our methods. As described in the Introduction, we will use the Method-of-Lines technique to convert our PDE to an ODE, but recall that we also have fixed boundary conditions that state $u_1(t) = u_{N_x}(t) = 0$, thus the top and bottom row of our matrix needs to be 0. All together we can write a new linear space-discrete heat equation

$$\dot{\vec{u}}(t) = \mathcal{D}_2\vec{u}(t), \qquad \mathcal{D}_2 = \frac{1}{\Delta x^2}\begin{bmatrix} 0 & . & . & . & & \\ 1 & -2 & 1 & & & \\ 0 & 1 & -2 & 1 & & \\ . & & 1 & -2 & 1 & \\ . & & & & & \\ . & & & & & 0 \end{bmatrix},$$

which can be solved using many known techniques. In this case there is again an exact solution $\vec{u} = sin(\pi x)e^{\mathcal{D}_2 t}$, where the exponential of a matrix is defined via the Taylor expansion of the normal exponential function $e^{\mathcal{D}_2 t} = \mathcal{I} + \mathcal{D}_2 t + \frac{1}{2}(\mathcal{D}_2)^2 t^2 + \ldots$, with $\mathcal{I}$ the identity matrix. We call this the "EXPM" method, for the matrix-exponent function in Matlab.

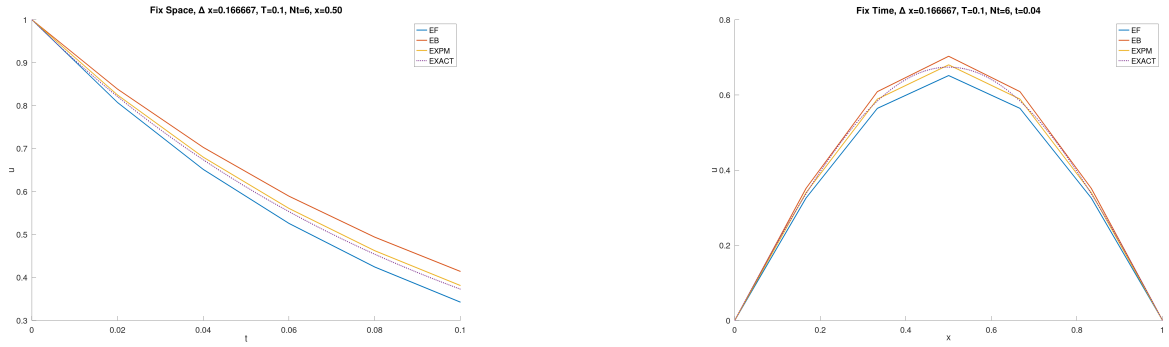
Additionally, we consider the Euler-Forward (EF) and the Euler-Backward (EB) numerical integration techniques, which districtize the time-dimension (with $\Delta t = T/(N_t - 1)$ similar to what we did for x) and use a forward/backward looking projector in the *guess* for the temporal-derivative:

$$\text{EF:} \quad \vec{u}^{n+1} = (\mathcal{I} + \Delta t\mathcal{D}_2)\vec{u}^n,$$

$$\text{EB:} \quad \vec{u}^{n} = (\mathcal{I} - \Delta t\mathcal{D}_2)\vec{u}^{n+1}.$$

In our code, we use a left matrix division $\vec{u}^{n+1}(\mathcal{I} - \Delta t\mathcal{D}_2)\backslash\vec{u}^n$ to correctly compute the EB technique, though this does involve a computationally expensive step. For EF, we need to be careful with our time interval size, as if we choose an interval too large, the result becomes unstable. We need to use a ratio $\Delta t/\Delta x^2 \leq 0.5$ in order to ensure this numerical stability.

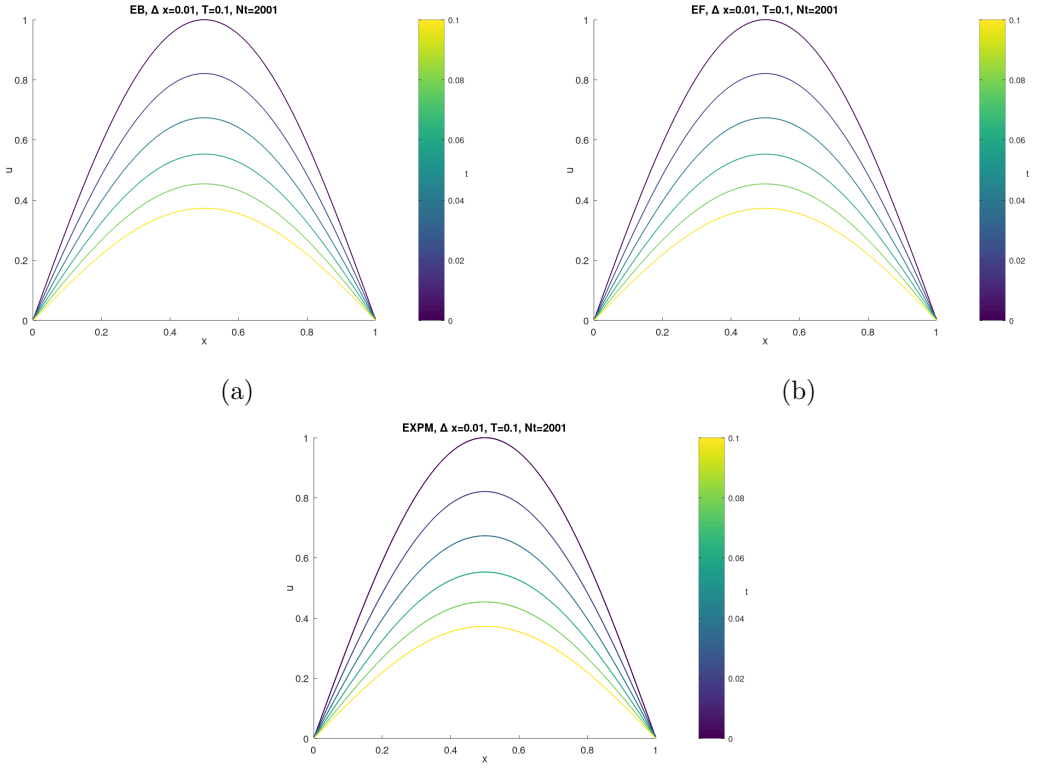
In Figure (1a), for a fixed value of $x = 0.33$ (and Figure (1b) for a fixed value $t = 0.5$) we see how these different methods of Integration compare to each other, first using large interval sizes $\Delta x = 0.1667$ and $N_t = t$ to make their differences more visible. We note that the EF method underestimates the value for $u(x,t)$ while EB overestimates it. This can be seen from them over- and under-estimating the slope, though if we had a function with an increasing slope, this would be flipped. EXPM on the other hand seems to agree with the exact solution much better.



(a) (b)

Figure 1: Solutions $u(x,t)$ using EF, EB, EXPM and the Exact solution, as a function of (a) space, fixing $t = 0.5$ and (b) time, fixing $x = 0.33$. Both use a space-step size of $\Delta x = 0.1667$ and 6 time steps.

In Figure (2b)-c, we consider a much lower interval size of $\Delta x = 0.01$ and $N_t = 2001$ for various fixed values of t. We see that all three methods give the same resulting diffusion of u. As we are considering fixed boundaries, this can be interpreted as permanent sink absorbing energy from the system, while if we had open boundaries we might expect the 'heat' to diffuse down to a uniform, non-zero value.



(a) (b)

(c)

Figure 2: Solution $u(x,t)$ to $\dot{\vec{u}}(t) = \mathcal{D}_2\vec{u}(t)$ found by using the three different numerical techniques: in (a) EB, in (b) EF and in (c) EXPM.

Exercise 4.2

In this section, we numerically treat the non-linear Fisher PDE, a reaction-diffusion equation constructed from the diffusion equation of Exercise 4.1 by adding a non-linear term. The set of equations we consider is the following:

$$\begin{aligned} \frac{\partial u}{\partial t} &= d\frac{\partial^2 u}{\partial x^2} + \gamma u\,(1-u)\,, \\ u(x,0) &= e^{-50(x-1/2)^2}, \\ u(0,t) &= u(1,t) = 0, \end{aligned}$$

where, once again, we restrict the time interval to $t \in [0,T]$ (here we use $T=1$) and space interval to $x \in [0,1]$. The parameter d describes diffusion while the parameter γ quantifies the contribution of the non-linear term to the PDE, in our simulations we set them to $d = 0.001$ and $\gamma = 5$. These types of equations are usually used to study population growth and dynamics; if u is interpreted as the concentration of a population, we can look at the first term of the PDE as the term governing the diffusion of the population and the second as the term controlling growth called the *reaction term*. This is a quadratic term that reaches its maximum at $u(x,t) = 0.5$ and goes to zero at the $u(x,t) = 0$ and $u(x,t) = 1$.

We proceed in a similar way as Exercise 4.1, we first use the Method-of-Lines to discretize the spatial component obtaining the equation

$$\dot{\vec{u}}(t) = d\mathcal{D}_2\vec{u}(t) + \vec{f}(\vec{u}),$$

where we define the function $\vec{f}(\vec{u}) = \gamma\vec{u}(t)\left(1-u(\vec{t})\right)$ for simplicity of notation. At this point, we apply the "IMEX" method to discretize the time variable. This method combines EF and EB in the sense that the evolution in time is at a certain time-step n is written as:

$$\frac{\vec{u}^{n+1}-\vec{u}^{n}}{\Delta t} = d\mathcal{D}_2\vec{u}^{n+1} + \vec{f}(\vec{u}^{n}),$$

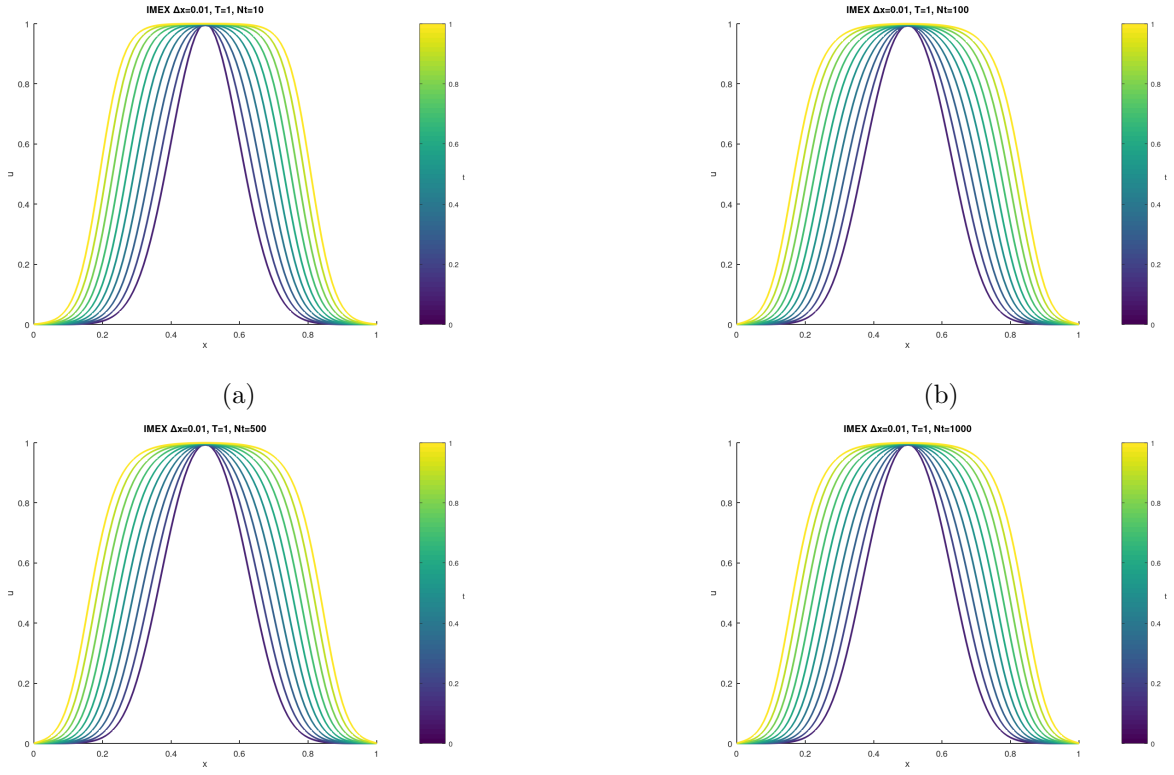
from which we can derive the recursive relation

$$\mathcal{A}\vec{u}^{n+1} = \vec{g}(\vec{u}^{n}), \tag{0.1}$$

where we defined the matrix $\mathcal{A} = \mathcal{I} - d\Delta t\mathcal{D}_2$ and the function $\vec{g}(\vec{u}^{n}) = \vec{u}^{n} + \Delta t\vec{f}(\vec{u}^{n})$. Starting from the initial condition, this recursive relation is implemented at each successive time-step by computing the inverse of the matrix $\mathcal{A}$ and applying it to $\vec{g}$, function of the previous time-step.

In order to assess the ideal time-step size, we plotted the solution $u(x,t)$ obtained with the method just described as a function of space for a set of times. Figure (3) shows this behaviour for four different time-step sizes, in each figure the behaviour at different times is indicated by a colour bar. We observe that after time-step $\Delta t = 0.002$ the shape of the solution doesn't change, we can therefore conclude that this is a reasonable choice. From these figures, it is also possible to understand the effect of the reaction term by comparing the solutions to those obtained in the case with only diffusion: the solution doesn't decrease in time, meaning that the reaction term is able to generate population at a higher rate than the population's diffusion rate.

Moreover, we studied the behaviour of the solution as a function of time for a set of space points after setting $\Delta t = 0.002$. The symmetry of the solution in space around $x = 0.5$ observed

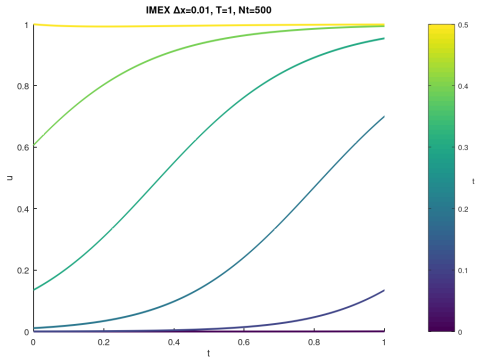


(a) (b)

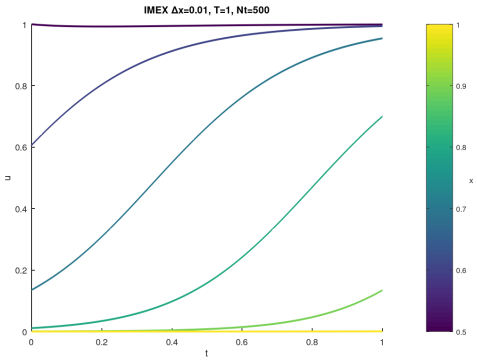
(c) (d)

Figure 3: Solution $u(x,t)$ as a function of the position for a series of times and for four different step sizes: in (a) $\Delta t = 0.1$, in (b) $\Delta t = 0.01$, in (c) $\Delta t = 0.002$ and in (d) $\Delta t = 0.001$. After time-step $\Delta t = 0.002$ the solutions don't change.

in Figure (3) is also observable in the plot as a function of time, as we can see in Figure (4). The colour bars indicate different space-points in the range $x \in [0,1]$: the solution is symmetric around $x = 0.5$ and the boundary conditions are correctly satisfied.



(a)



(b)

Figure 4: Solution $u(x,t)$ as a function of the time for a series of space-points and $\Delta t = 0.002$. In (a) the behaviour in time for $x \in [0, 0.5]$ and in (b) the behaviour in time for $x \in [0.5, 1]$.

Exercise 6.1

This section deals with a similar situation as Exercise 4.1, but now we introduce the *fractional*-derivative. We use finite-difference matrices with periodic boundary conditions (so we can use the unaltered matrices from the Introduction) for our derivative-matrices. We can now apply normal matrix operators to solve the fractional heat equation:

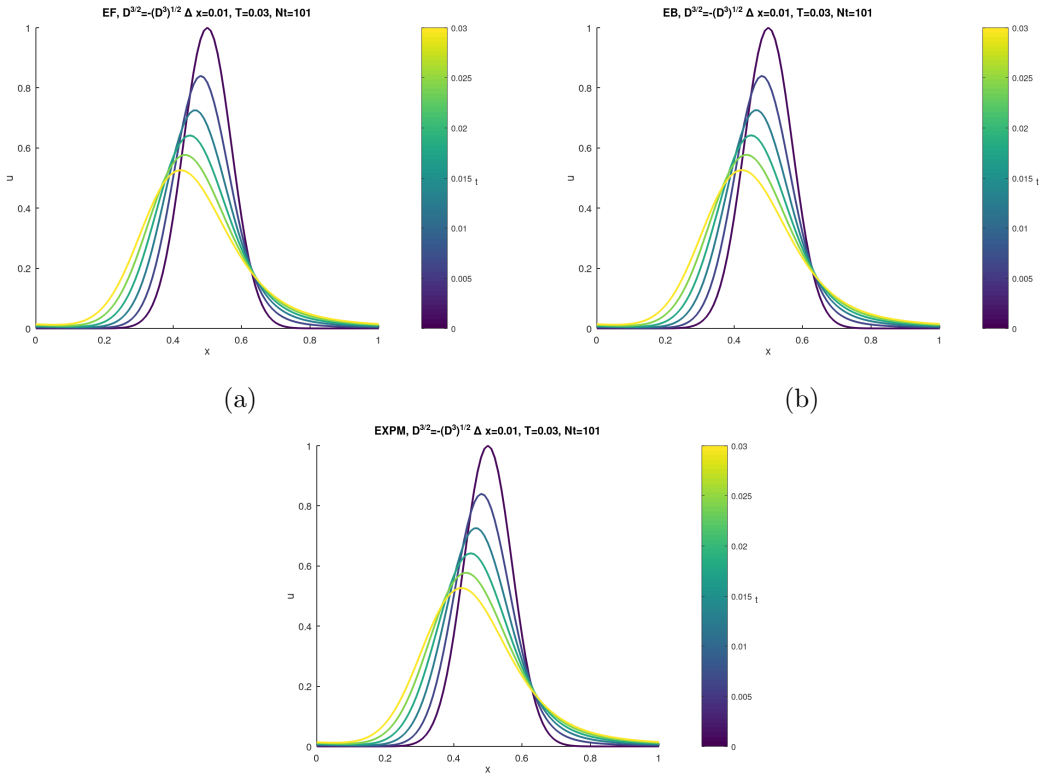
$$\frac{\partial u}{\partial t} = \mathcal{D}_{3/2} u, \tag{0.2}$$

$$u(x,0) = e^{-100\left(x-\frac{1}{2}\right)}. \tag{0.3}$$

As before, we use the Method of Lines, and consider 3 different finite-difference matrices to give separate ODE forms of this problem, all with the same initial and boundary conditions

$$\dot{\vec{u}}(t) = -\sqrt{D_3}\vec{u}(t)$$
$$\dot{\vec{u}}(t) = -\sqrt{-D_3}\vec{u}(t)$$
$$\dot{\vec{u}}(t) = -\frac{1}{\sqrt{2}}\left(\sqrt{D_3} + \sqrt{-D_3}\right)\vec{u}(t)$$

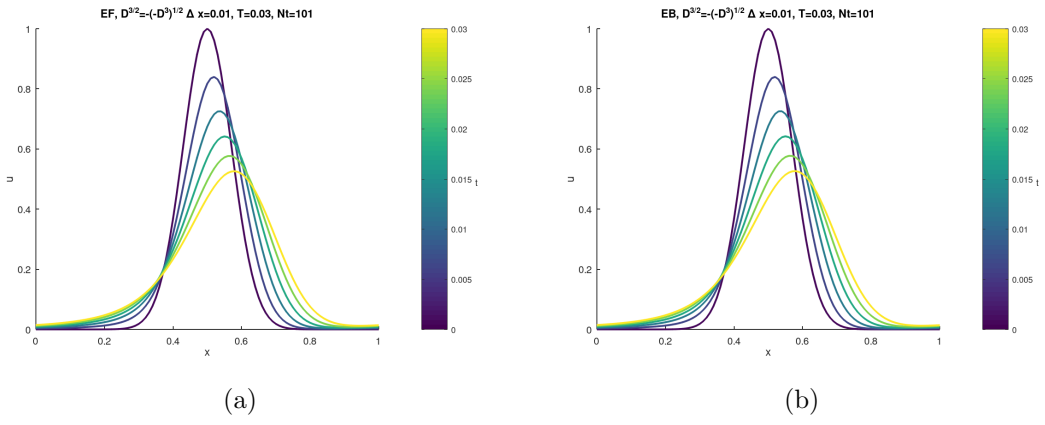
Now we want to see how EXPM, EF and EB solve these systems and how their behaviour is similar/different. We use a spatial-step of $\Delta x = 0.01$, $N_t = 101$ and a total time of $T = 0.03$.



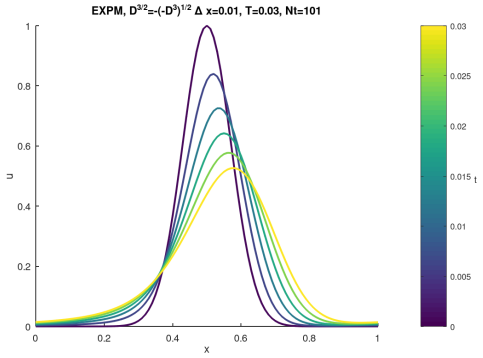
(a) (b)

(c)

Figure 5: Solutions for the $\dot{\vec{u}}(t) = -\sqrt{D_3}\vec{u}(t)$, using (a) EF, (b) EB, (c) EXPM as our Integration techniques



(a) (b)



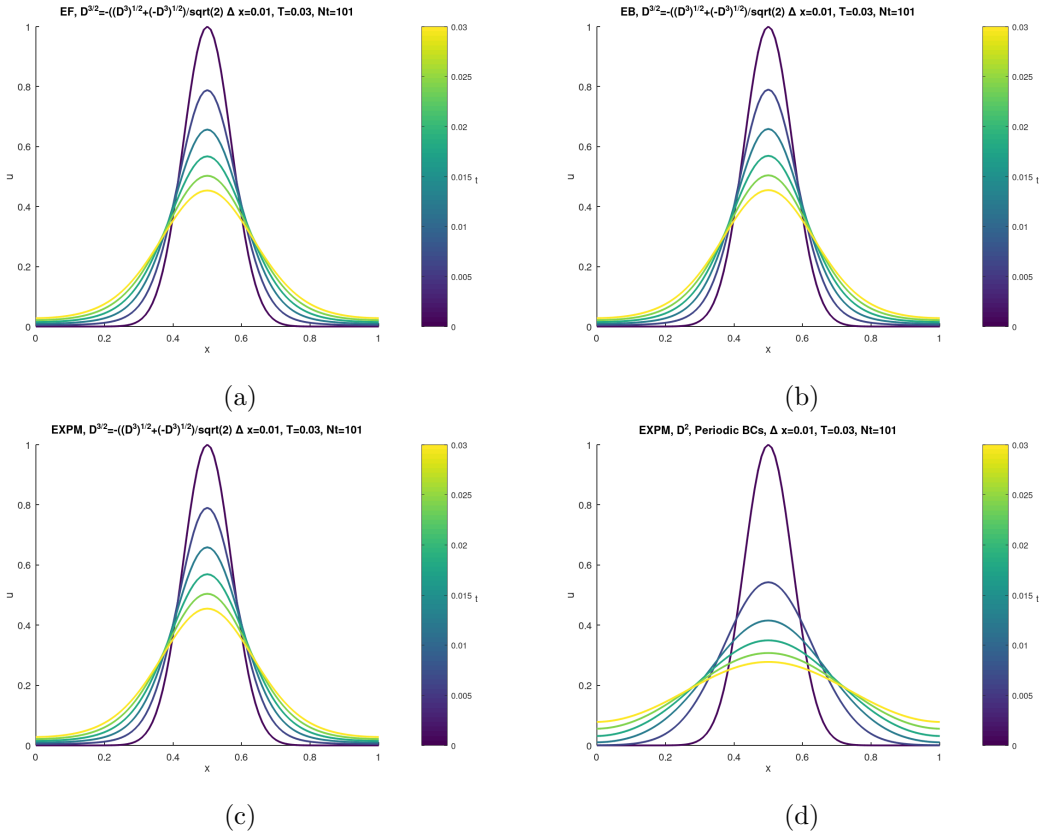
(c)

Figure 6: Solutions for the $\dot{\vec{u}}(t) = -\sqrt{-D_3}\vec{u}(t)$, using (a) EF, (b) EB, (c) EXPM as our Integration techniques

In Figure (5), we plot the solution using $\mathcal{D}_{3/2} = -\sqrt{\mathcal{D}_2}$. We see that the system diffuses down with a left leaning tilt, while in Figure (6) for $\mathcal{D}_{3/2} = \mathcal{D}_{3/2} = -\sqrt{-\mathcal{D}_2}$ there is a right leaning tilt. This can be interpreted similar to the behaviour of the forward and backward Euler integration techniques with their estimations either side on the central exact solution.

In Figure (7)a-c., using $-\frac{1}{\sqrt{2}}\left(\sqrt{D_3} + \sqrt{-D_3}\right)$, we see a symmetric decay, again similar to the central integration technique, having no bias either way. In all of the methods the difference between the EXPM, EF and EB methods for the time integration are minimal for such a low space and time step size.

We also compare to Figure (7)d. where we consider D_2 with periodic boundaries conditions (thus we now do not set the first and final row to 0). Here we see that the higher order derivative causes the diffusion to occur much quicker, as might be expected given the initial PDE, where a higher order derivative gives faster decay.



(a) (b)

(c) (d)

Figure 7: Solutions for the $\dot{\vec{u}}(t) - \frac{1}{\sqrt{2}}\left(\sqrt{D_3} + \sqrt{-D_3}\right)\vec{u}(t)$, using (a) EF, (b) EB, (c) EXPM as our Integration techniques. In (d), the solution for $\dot{\vec{u}}(t) = -D_2\vec{u}(t)$ using periodic boundary conditions and EXPM as the Integration technique

Exercise 6.2

In this exercise, we look at the non-linear version of the fraction PDE studied in the previous section. The set of equations defining this PDE are

$$\frac{\partial u}{\partial t} = d\mathcal{D}_{3/2}u + \gamma u(1-u),$$
$$u(x,0) = e^{-50(x-1/2)^2},$$

and we impose periodic boundary conditions (PBC), meaning that $u_{N_x+1}(t) = u_1(t)$. We define space in the interval $x \in [0,1]$, time $t \in [0,1]$ and we fix the parameters $\Delta x = 0.01$, $T = 1$, 0.0001 and $\gamma = 5$.

Discretizing the differential equation in space and using Riesz's definition of fractional derivative we get the following equation

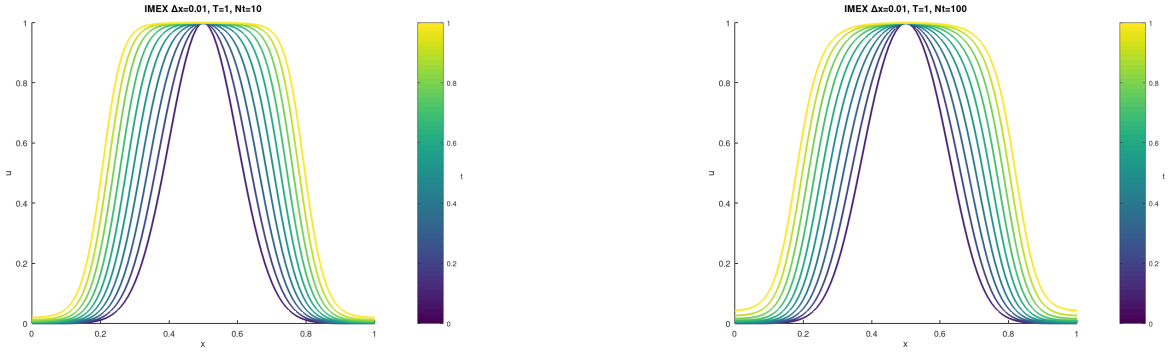
$$\dot{\vec{u}}(t) = -\frac{d}{\sqrt{2}}\left(\sqrt{\mathcal{D}_3} + \sqrt{-\mathcal{D}_3}\right)\vec{u}(t) + \vec{f}(\vec{u})$$

to which we apply the IMEX method introduced in Exercise 6.1 and implement the recursive relation in equation (0.1) where, in this case, the definition of the matrix on the left-side of the equation is $\mathcal{A} = -\frac{d}{\sqrt{2}}\left(\sqrt{\mathcal{D}_3} + \sqrt{-\mathcal{D}_3}\right)$.

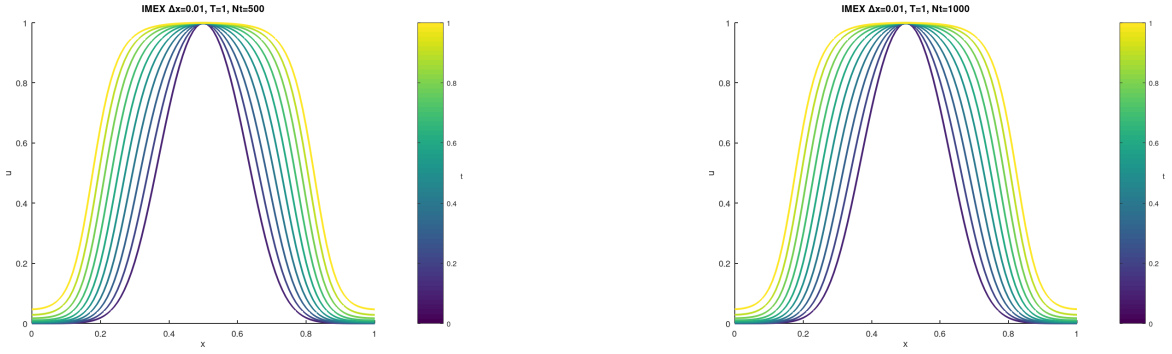
As we did in Exercise 4.2, we studied the behaviour of the solution in space and time separately. For the case of the solution in space, we plot the solution for different values of time and different time-step size, as shown in Figure (8). From these figures, we can conclude that using $\Delta t = 0.002$ is a reasonable choice since the solution doesn't change if we decrease the time-step even further.

After having set the time-step, we compared the solution in space for a set of times with the solution obtained by substituting the fractional derivative with a second-order derivative, but maintaining PBC. The comparison is shown in Figure (9) and we notice that the difference is at the boundaries: the solution from the fractional derivative doesn't go to zero as the solution approaches the space boundaries, meaning that the fractional derivative influences the balance diffusion/reaction in such a way that as time approaches 1 the population is able to diffuse through the boundaries. We would get a similar behaviour for the second-order derivative if we wait a longer time, as we can see in Figure (10), which shows the behaviour for the range $t \in [0, 1.3]$.

Finally, we look at the time dependence of the solution in Figure (11) for some space points in the range $[0,1]$. As expected, the solution is symmetric around $x = 0.5$. The difference we notice with respect to the case of second order derivative with open boundary conditions studied in Exercise 4.2 is when time approaches the maximum time, as we can infer from the plot in Figure (8).

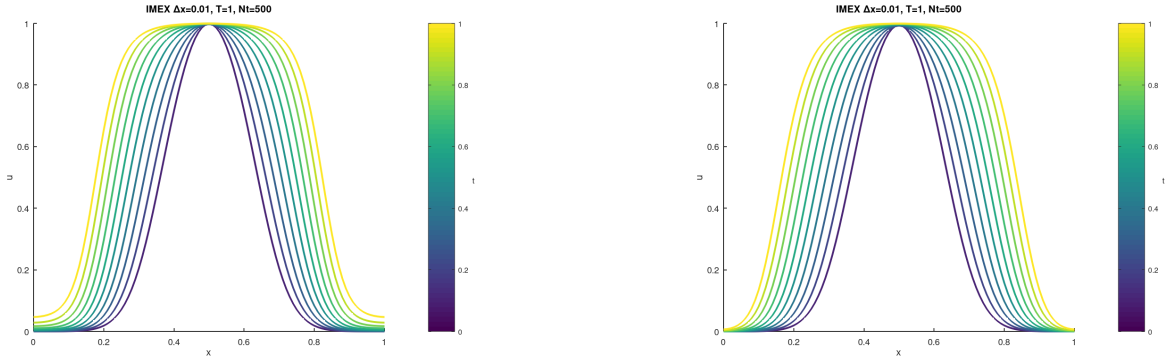


(a) (b)



(c) (d)

Figure 8: Solution $u(x,t)$ of the fractional reaction/diffusion equation with PBC as a function of the position for a series of times and for four different step sizes: in (a) $\Delta t = 0.1$, in (b) $\Delta t = 0.01$, in (c) $\Delta t = 0.002$ and in (d) $\Delta t = 0.001$. After time-step $\Delta t = 0.002$ the solutions don't change.



(a) (b)

Figure 9: Comparison of the solution $u(x,t)$ as a function of the time for a series of times and $\Delta t = 0.002$. In (a) the solution to the fractional reaction/diffusion equation with PBC and in (b) the solution to the second-order reaction/diffusion equation with PBC.

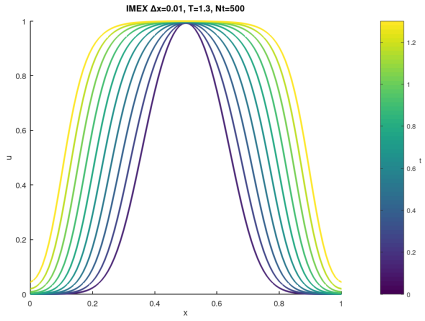
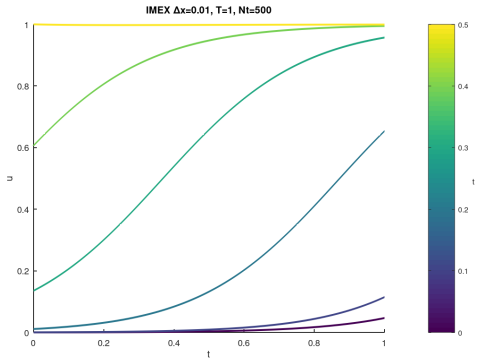
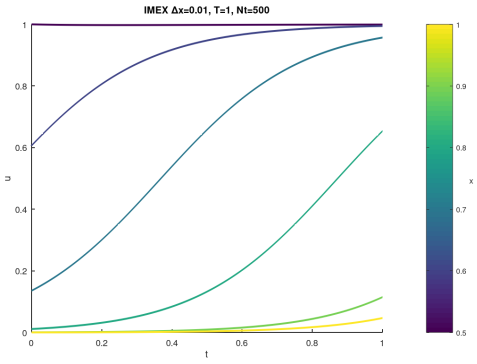


Figure 10: Solution $u(x,t)$ of the second-order reaction/diffusion equation with PBC as a function of the time for a series of time, $\Delta t = 0.002$ and time interval $t \in [0, 1.3]$.



(a)



(b)

Figure 11: Solution $u(x,t)$ as a function of the time for a series of space-points and $\Delta t = 0.002$. In (a) the behaviour in time for $x \in [0, 0.5]$ and in (b) the behaviour in time for $x \in [0.5, 1]$.

Comparison

In this section, we compare the solution to the fractional linear differential equation of Exercise 6.1 part (c) with the solution obtained from solving the equation

$$\dot{\vec{v}}(t) = -\sqrt{\sqrt{(-\mathcal{D}_2)^3}}\vec{v}$$

with the same initial and boundary conditions and final time T. The comparison is made by computing the modulus of the difference of the solutions at the final time $||\vec{u}(T) - \vec{v}(T)||$ and it has been done for different values of Δx. The result of this study is shown in Figure (12) where we plot the difference for a range $\Delta x \in [0.001, 0.1]$ in a logarithmic scale and we also list the values in Table (1).

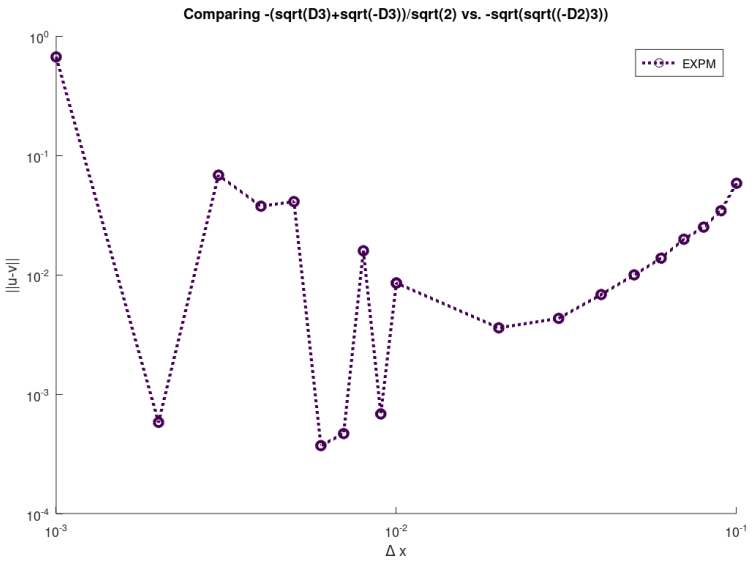


Figure 12: Modulus of the difference of the vectors $u(T)$ and $v(T)$ as a function of space step size in a log-log scale.

We observe that the difference between the two solutions decreases as we decrease the size of the step in space Δx up until a value of $\Delta x \sim 0.02$, after which the graph oscillates and we can not infer any more information about its behaviour. The initial decrease can be explained if we consider that a smaller Δx implies a smaller error in the discretization of the space derivative and, therefore, both solutions get closer to the exact solution. The reason for the instability we found at small values of Δx could be connected to the way the matrices $\mathcal{D}_2$ and $\mathcal{D}_3$ are processed by Matlab since, as the space interval decreases, they assume very large values which could, in turn, cause discrepancies in the two ways the matrices are treated.

Δx	$\|\|\vec{u}(T) - \vec{v}(T)\|\|$
0.001	0.667127
0.002	0.000580745
0.003	0.0682732
0.004	0.037505
0.005	0.040982
0.006	0.000370672
0.007	0.000467586
0.008	0.0158763
0.009	0.000682862
0.01	0.00852768
0.01	0.00852768
0.02	0.00359188
0.03	0.00431092
0.04	0.00682593
0.05	0.00997324
0.06	0.0137657
0.07	0.0198307
0.08	0.025061
0.09	0.0344026
0.1	0.0583467

Table 1: Values for the Comparison

A Code Files

A.1 ex41.m

```
1;


%% Exact Solution for Problem 4.1
function u=HeatEquationExact(x,t)
    u=exp(-(pi^2)*t).*sin(pi*x);
endfunction



%%%%%%%%%%%%%% Initial Values
% Here we define our global values, the total time, the number of time steps
% and the space interval size
T=0.1;
Nt=6;
delta_x=1/6;
% Nt=21;
% delta_x=0.1;
delta_t=T/(Nt-1);

% We use these to generate the number of space steps, then define a 'continuous' and
% discrete vector for space and time
Nx=int32(1+(1/delta_x));
x_continuous=linspace(0,1);
t_continuous=linspace(0,T)
t=linspace(0,T,Nt);
x=linspace(0,1,Nx);


%%%%%%%%%%%%%% Stability check
% Here we check the stability for the EF method; not this only warns when the
% code is unstable, it will still run
stability=delta_t/(delta_x)^2
if stability>0.5
    printf("EF Unstable, stability ratio: %.1f\n", stability)
end




%%%%%%%%%%%%%% Defining the finite-differnce Matrices
% Here we define the finite-differnce matrix we want to use in this problem. This consists of
% -2 along the diagonal and 1's on the two 1-off diagonals. We also use fied boundary values,
% so the first and final rows need to be zerod. For later convience the Periodic condisiton is
% also included
der_a=-2;    % Diagonal
der_b=1;     % Right-1-off Diagonal
der_c=1;     % Left-1-off Diagonal
der_d=0;     % Left-2-off Diagonal
der_f=0;     % left-2-off Diagonal
D2 = diag(der_a*ones(1,Nx)) + diag(der_b*ones(1,Nx-1),1) +diag(der_f*ones(1,Nx-2),2)+diag(der_d*
% D2(1,Nx)=der_c;      % P.B.C.s
% D2(Nx,1)=der_b;      % P.B.C.s
D2(1,:)=0;           % F.B.C.s
```

```
D2(Nx,:)=0;          % F.B.C.s
D2=D2 / delta_x^2;



%%%%%%%%%%%%%% Initial Conditions
% We initialise our three integration method solution matrices.
u_expm=zeros(Nx,Nt);
u_eulfor=zeros(Nx,Nt);
u_eulback=zeros(Nx,Nt);
u_initial=sin(pi*x);
u_eulfor(:,1)=u_initial;
u_eulback(:,1)=u_initial;



%%%%%%%%%%%%%% Integration Steps
% EXPM - Solve the ODE system analytically via the EXPM
for i=1:Nt
    u_expm(:,i)=(expm(t(i).*D2)*u_initial')';
end

% Euler Forward - Solve using the EF integration techniques
for i=1:Nt-1
    u_eulfor(:,i+1) = ((eye(Nx)+delta_t.*D2)*u_eulfor(:,i));
endfor

% Euler Backward - Solve using the EB integration techniques
for i=1:Nt-1
    u_eulback(:,i+1) = ((eye(Nx)-delta_t.*D2)\u_eulback(:,i));
endfor


%%%%%%%%%%%%%% Plotting
% Some of this is not used, and is keep as archival for future usage if required
% Also note the plots require a folder Ex41 to be present.
listoftimes=1:round((Nt-1)/5):Nt
listofspace=1:round((Nx-1)/5):Nx
h=get(gcf, "currentaxes");
set(h, "fontsize", 12, "linewidth", 1);
clf
hold on
labels=[''];
for ti=1:size(listoftimes)(2)
    tau=listoftimes(ti);
    filename=sprintf("Ex41/Ex41eulfor-delx=%d.png", delta_x );

    % Plot EF solution
    plot(x, u_eulfor(:,tau))
    labels=[labels;sprintf("t=%d",t(tau))];


    title(sprintf('Euler␣Forward:␣\\Delta␣x=%d,␣T=%d,␣Nt=%d', delta_x, T, Nt))
    legend(labels=labels)
    axis([0 1 0 1])

    saveas(1, filename)
```

```
end
% Similar Plots bellow
% clf
% hold on
% labels=[''];
% for ti=1:size(listoftimes)(2)
%     tau=listoftimes(ti);
%     filename=sprintf("Ex41/Ex41eulback-delx=%d.png", delta_x );
%
%     plot(x, u_eulback(:,tau))
%     labels=[labels;sprintf("t=%d",t(tau))];
%
%
%     title(sprintf('Euler Backward: \\Delta x=%d, T=%d, Nt=%d', delta_x, T, Nt))
%     legend(labels=labels)
%     axis([0 1 0 1])
%
%     saveas(1, filename)
% end
%
% clf
% hold on
% labels=[''];
% for ti=1:size(listoftimes)(2)
%     tau=listoftimes(ti);
%     filename=sprintf("Ex41/Ex41expm-delx=%d.png", delta_x);
%
%     plot(x, u_expm(:,tau))
%     labels=[labels;sprintf("t=%d",t(tau))];
%
%
%     title(sprintf('Expm: \\Delta x=%d, T=%d, Nt=%d', delta_x, T, Nt))
%     legend(labels=labels)
%     axis([0 1 0 1])
%
%     saveas(1, filename)
% end


%%%%%%%%%%%%% Colour Plotting
%%% EXPM Colorplot
clf
hold on
cmap=colormap();
for ti=1:size(listoftimes)(2)
    tau=listoftimes(ti);
    colour_i=round(1+(tau*63/Nt));
    plot(x, u_expm(:,tau),'Color', color=cmap(colour_i,:),'linewidth',2)
    axis([0 1 0 1])
end
xlabel('x')
ylabel('u')
h=get(gcf, "currentaxes");
set(h, "fontsize", 12, "linewidth", 1);
cbar=colorbar()
caxis([0,T])
ylabel(cbar,'t','Rotation',0)
title(sprintf('EXPM, \\Delta x=%d, T=%d, Nt=%d', delta_x, T, Nt));
filename=sprintf('Ex41/Ex41-EXPM-multi-t-delx=%d.png',delta_x);
```

```
print(1,filename, '-dpng','-S1200,800')

% Similar plots below
%%% EF Colorplot
% clf
% hold on
% cmap=colormap();
% for ti=1:size(listoftimes)(2)
%     tau=listoftimes(ti);
%     colour_i=round(1+(tau*63/Nt));
%     plot(x, u_eulfor(:,tau),'Color', color=cmap(colour_i,:),'linewidth',2)
%     axis([0 1 0 1])
% end
% xlabel('x')
% ylabel('u')
% h=get(gcf, "currentaxes");
% set(h, "fontsize", 12, "linewidth", 1);
% cbar=colorbar()
% caxis([0,T])
% ylabel(cbar,'t','Rotation',0)
% title(sprintf('EF, \\Delta x=%d, T=%d, Nt=%d', delta_x, T, Nt));
% filename=sprintf('Ex41/Ex41-EF-multi-t-delx=%d.png',delta_x);
% print(1,filename, '-dpng','-S1200,800')
%

%%% EB Colorplot
% clf
% hold on
% cmap=colormap();
% for ti=1:size(listoftimes)(2)
%     tau=listoftimes(ti);
%     colour_i=round(1+(tau*63/Nt));
%
%     plot(x, u_eulback(:,tau),'Color', color=cmap(colour_i,:),'linewidth',2)
%
% end
% xlabel('x')
% ylabel('u')
% axis([0 1 0 1])
% h=get(gcf, "currentaxes");
% set(h, "fontsize", 12, "linewidth", 1);
% cbar=colorbar()
% caxis([0,T])
% ylabel(cbar,'t','Rotation',0)
% title(sprintf('EB, \\Delta x=%d, T=%d, Nt=%d', delta_x, T, Nt));
% filename=sprintf('Ex41/Ex41-EB-multi-t-delx=%d.png',delta_x);
% print(1,filename, '-dpng','-S1200,800')
%

%%% Fix Time
labels=['']
for ti=2:size(listoftimes)(2)
    clf
    hold on
    labels=[''];
    tau=listofspace(ti)
    plot(x, u_eulfor(:,tau),'linewidth',2)
    labels=[labels;"EF"];
```

```
    plot(x, u_eulback(:,tau),'linewidth',2)
    labels=[labels;"EB"];

    plot(x, u_expm(:,tau),'linewidth',2)
    labels=[labels;"EXPM"];

    plot(x_continuous ,HeatEquationExact(x_continuous,t(tau)),":",'linewidth',2)
    labels=[labels;"EXACT"];

    filename=sprintf("Ex41/Ex41-fixtime-delx=%d-x=%.2f.png", delta_x, x(tau));
    title(sprintf('Fix Time, \\Delta x=%d, T=%d, Nt=%d, t=%.02f', delta_x, T, Nt, t(tau)))
    legend(labels=labels)
    xlabel('x')
    ylabel('u')

    print(1,filename, '-dpng','-S1200,800')
end

%%% Fix Space
labels=['']
for ti=2:size(listoftimes)(2)
    clf
    hold on
    labels=[''];
    tau=listofspace(ti)
    plot(t, u_eulfor(tau,:),'linewidth',2)
    labels=[labels;"EF"];

    plot(t, u_eulback(tau,:),'linewidth',2)
    labels=[labels;"EB"];

    plot(t, u_expm(tau,:),'linewidth',2)
    labels=[labels;"EXPM"];

    plot(t_continuous ,HeatEquationExact(x(tau),t_continuous),":",'linewidth',2)
    labels=[labels;"EXACT"];

    filename=sprintf("Ex41/Ex41-fixspace-delx=%d-t=%.2f.png", delta_x, x(tau));
    title(sprintf('Fix Space, \\Delta x=%d, T=%d, Nt=%d, x=%.02f', delta_x, T, Nt, x(tau)))
    legend(labels=labels)
    xlabel('t')
    ylabel('u')

    print(1,filename, '-dpng','-S1200,800')
end
```

A.2 ex42.m

```
% -------------- FUNCTIONS --------------

    % Defining g(u, delta_t, gamma) function to be called in the recursive relation
    1;

    function g_vec=g_funct(u, delta_t, gamma_val)

        g_vec = u + delta_t * gamma_val*(u.*(1-u));

    endfunction

% -------------- MAIN CODE --------------

    % Defining parameters of the simulation: tmax = maximum time, delta_x = space step-size
    % d_val = diffusion coefficient, gamma_val = reaction coefficient

    t_max = 1;
    delta_x = 0.01;
    d_val = 0.001;
    gamma_val = 5;

    % Find number of steps in space N and create vector x with values in space in [0,1]

    N = 1 + 1/delta_x
    x = linspace(0,1,N);

    % Define matrix D (=D2) that computes second order time deivative with PBC

    a = -2;
    b = 1;
    c = 1;
    D = diag(a*ones(1,N)) + diag(b*ones(1,N-1),1) + diag(c*ones(1,N-1),-1);
    D(1,:)=0;
    D(N,:)=0;

    D = D/(delta_x^2);

    % Impose initial condition

    u_initial = zeros(N,1);

    for i=1:N
        u_initial(i) = exp(-50*(x(i)-1/2)^2);
    end

    % Fix number of time-steps t_interval and compute vector t with values of time in [0,t_max]

    t_interval = 200;
    t = linspace(0,t_max,t_interval);
    delta_t = t_max/(t_interval-1);

    % Implement recursive relation in time to find the solution u_approx

    u_approx=zeros(N,size(t)(2));

        u_approx(:,1) = u_initial;
```

```
        for j=1:(size(t)(2)-1)

            A = eye(N) - d_val*delta_t*D;

            u_approx(:,j+1) = A\g_funct(u_approx(:,j), delta_t, gamma_val);

        end


% -------------- PLOTS --------------

    % Plot for a bunch of different times with colour map

    listoftimes = [];

    time_intervals = 10;

    delta_tint = floor(t_interval/time_intervals)

    listoftimes(1) = delta_tint;

    for i=2:time_intervals;
        listoftimes(i) = listoftimes(i-1) + delta_tint;
    end

    clf
    hold on

        cmap=colormap();
    for ti=1:size(listoftimes)(2)
        tau=listoftimes(ti)
        colour_i=round(1+(tau*63/(t_interval)));
        plot(x, u_approx(:,tau),'Color', color=cmap(colour_i,:),'linewidth',2)
        axis([0 1 0 1])
    end

    xlabel('x');
    ylabel('u');
    h=get(gcf, "currentaxes");
    set(h, "fontsize", 12, "linewidth", 1);
    cbar=colorbar();
    caxis([0,t_max]);
    ylabel(cbar,'t','Rotation',0);
    title(sprintf('IMEX \\Deltax=%d, T=%d, Nt=%d', delta_x, t_max, t_interval));
    filename=sprintf('Ex42multit_Nt=%d.png',t_interval);
    print(1,filename, '-dpng','-S600,400')

    % Plot for a bunch of different space points with colour map

    clf
    hold on

    cmap=colormap();
    listofspace=1
    for xi=1:6

        colour_i=round((listofspace*63/(N/2)))
        plot(t, u_approx(listofspace,:),'Color', color=cmap(colour_i,:),'linewidth',2)
```

```
    axis([0 1 0 1])

    listofspace = listofspace + (N-1)/10
end

xlabel('t');
ylabel('u');
h=get(gcf, "currentaxes");
set(h, "fontsize", 12, "linewidth", 1);
cbar=colorbar();
caxis([0,0.5]);
ylabel(cbar,'t','Rotation',0);
title(sprintf('IMEX␣\\Deltax=%d,␣T=%d,␣Nt=%d', delta_x, t_max, t_interval));
filename=sprintf('Ex42multix_Nt=%d_first1.png',t_interval);
print(1,filename, '-dpng','-S600,400')


clf
hold on

cmap=colormap();

listofspace = 51

for xi=1:6

    colour_i=round(((listofspace-50)*63/(N/2)))
    plot(t, u_approx(listofspace,:),'Color', color=cmap(colour_i,:),'linewidth',2)

    axis([0 1 0 1])

    listofspace = listofspace + (N-1)/10
end

xlabel('t');
ylabel('u');
h=get(gcf, "currentaxes");
set(h, "fontsize", 12, "linewidth", 1);
cbar=colorbar();
caxis([0.5,1]);
ylabel(cbar,'x','Rotation',0);
title(sprintf('IMEX␣\\Deltax=%d,␣T=%d,␣Nt=%d', delta_x, t_max, t_interval));
filename=sprintf('Ex42multix_Nt=%d_second.png',t_interval);
print(1,filename, '-dpng','-S600,400')
```

A.3 ex61.m

```
1;


%%%%%%%%%%%%%% Initial Values
% Here we define our global values, the total time, the number of time steps
% and the space interval size
T=0.03;
Nt=101;
delta_x=0.01;
delta_t=T/(Nt-1);


% We use these to generate the number of space steps, then define a
% discrete vector for space and time
Nx=int32(1+(1/delta_x));
t=linspace(0,T,Nt);
x=linspace(0,1,Nx);

%%%%%%%%%%%%%% Stability check
% Here we check the stability for the EF method; not this only warns when the
% code is unstable, it will still run
stability=delta_t/(delta_x)^(3/2);
if stability>0.5
    printf("EF Unstable, stability ratio: %.1f\n", stability)
end




%%%%%%%%%%%%%% Defining the finite-differnce Matrices
% Here we define the finite-differnce matrix we want to use in this problem.
% For later convience the Periodic condisiton is
% also included
% D3
% This consists of 0 along the diagonal,  2 's on the1-off diagonal and   1  2-off diagonals.
a=0;     % Diagonal
b=-1;    % Right-1-diagonal
c=1;     % Left-1-diagonal
d=-1/2;  % Left-2-diagonal
f=1/2;   % Right-2-diagonal
D3 = diag(a*ones(1,Nx)) + diag(b*ones(1,Nx-1),1) +diag(f*ones(1,Nx-2),2)+diag(d*ones(1,Nx-2),-2)
% B.C.
D3(1,Nx)=c;
D3(1,Nx-1)=d;
D3(2,Nx)=d;
D3(Nx,1)=b;
D3(Nx,2)=f;
D3(Nx-1,1)=f;
D3;
D3=D3/(delta_x^3);

% (a)
D3over2=-sqrtm(D3);
%
% (b)
D3over2=-sqrtm(-D3);
```

```
% (c)
D3over2=-(sqrtm(D3) + sqrtm(-D3))/sqrt(2);


%% D2
% This consists of -2 along the diagonal and 1's on the two 1-off diagonals.
der_a=-2;    % Diagonal
der_b=1;     % Right-1-off Diagonal
der_c=1;     % Left-1-off Diagonal
der_d=0;     % Left-2-off Diagonal
der_f=0;     % left-2-off Diagonal
D2 = diag(der_a*ones(1,Nx)) + diag(der_b*ones(1,Nx-1),1) +diag(der_f*ones(1,Nx-2),2)+diag(der_d*
D2(1,Nx)=der_c;
D2(Nx,1)=der_b;
D2=D2 / delta_x^2;
D2Fixed=D2;
D2Fixed(1,:)=0;
D2Fixed(Nx,:)=0;

% (Comparison)
% D3over2=D2


%%%%%%%%%%%%%% Initial Conditions
% We initialise our three integration method solution matrices.
u_initial=zeros(Nx,1);
for i=1:Nx
    u_initial(i)=exp(-100*(x(i)-0.5)^2);
end
u_expm=zeros(Nx,Nt);
u_eulfor=zeros(Nx,Nt);
u_eulback=zeros(Nx,Nt);
u_eulfor(:,1)=u_initial;
u_eulback(:,1)=u_initial;


%%%%%%%%%%%%%% Integration Steps
% EXPM - Solve the ODE system analytically via the EXPM
for i=1:Nt
    u_expm(:,i)=(expm(t(i).*D3over2)*u_initial)
end

% Euler Forward - Solve using the EF integration techniques
for i=1:Nt-1
    u_eulfor(:,i+1)=(eye(Nx)+delta_t.*D3over2)*u_eulfor(:,i);
end

% Euler Backward - Solve using the EB integration techniques
for i=1:Nt-1
    A=eye(Nx)-delta_t.*D3over2;
    u_eulback(:,i+1)=A\u_eulfor(:,i);
end



%%%%%%%%%%%%%% Plotting
% Some of this is not used, and is keep as archival for future usage if required
% Also note the plots require a folder Ex61 to be present.
%%% Colorplot EXPM
```

```
clf
hold on
listoftimes=1:(Nt-1)/5:Nt;
cmap=colormap();
for ti=1:size(listoftimes)(2)
    tau=listoftimes(ti);
    colour_i=round(1+(tau*63/Nt));
    plot(x, u_expm(:,tau),'Color', color=cmap(colour_i,:),'linewidth',2)
    axis([0 1 0 1])
end
xlabel('x')
ylabel('u')
h=get(gcf, "currentaxes");
set(h, "fontsize", 12, "linewidth", 1);
cbar=colorbar();
caxis([0,T])
ylabel(cbar,'t','Rotation',0)

% title(sprintf('EXPM, D^{3/2}=-(D^3)^{1/2} \\Delta x=%d, T=%d, Nt=%d', delta_x, T, Nt));
% filename=sprintf('Ex61/Ex61-EXPM-posDer-multi-t-delx=%d.png',delta_x);
% title(sprintf('EXPM, D^{3/2}=-(-D^3)^{1/2} \\Delta x=%d, T=%d, Nt=%d', delta_x, T, Nt));;
% filename=sprintf('Ex61/Ex61-EXPM-minusDer-multi-t-delx=%d.png',delta_x);
% title(sprintf('EXPM, D^{3/2}=-((D^3)^{1/2}+(-D^3)^{1/2})/sqrt(2) \\Delta x=%d, T=%d, Nt=%d', d
% filename=sprintf('Ex61/Ex61-EXPM-mixedDer-multi-t-delx=%d.png',delta_x);
%

title(sprintf('EXPM,␣D^2,␣Periodic␣BCs,␣\\Delta␣x=%d,␣T=%d,␣Nt=%d', delta_x, T, Nt));
filename=sprintf('Ex61/Ex61-EXPM-2Der-PBCs-multi-t-delx=%d.png',delta_x);


print(1,filename, '-dpng','-S600,400')


%%% Colorplot EF
% clf
% hold on
% cmap=colormap();
% for ti=1:size(listoftimes)(2)
%     tau=listoftimes(ti);
%     colour_i=round(1+(tau*63/Nt));
%     plot(x, u_eulfor(:,tau),'Color', color=cmap(colour_i,:),'linewidth',2)
%     axis([0 1 0 1])
% end
% xlabel('x')
% ylabel('u')
% h=get(gcf, "currentaxes");
% set(h, "fontsize", 12, "linewidth", 1);
% cbar=colorbar();
% caxis([0,T])
% ylabel(cbar,'t','Rotation',0)

% title(sprintf('EF, D^{3/2}=-(D^3)^{1/2} \\Delta x=%d, T=%d, Nt=%d', delta_x, T, Nt));
% filename=sprintf('Ex61/Ex61-EF-posDer-multi-t-delx=%d.png',delta_x);
% title(sprintf('EF, D^{3/2}=-(-D^3)^{1/2} \\Delta x=%d, T=%d, Nt=%d', delta_x, T, Nt));
% filename=sprintf('Ex61/Ex61-EF-minusDer-multi-t-delx=%d.png',delta_x);
% title(sprintf('EF, D^{3/2}=-((D^3)^{1/2}+(-D^3)^{1/2})/sqrt(2) \\Delta x=%d, T=%d, Nt=%d', del
% filename=sprintf('Ex61/Ex61-EF-mixedDer-multi-t-delx=%d.png',delta_x);
%
% print(1,filename, '-dpng','-S600,400')
```

```
%%% Colorplot EB
% clf
% hold on
% cmap=colormap();
% for ti=1:size(listoftimes)(2)
%     tau=listoftimes(ti);
%     colour_i=round(1+(tau*63/Nt));
%     plot(x, u_expm(:,tau),'Color', color=cmap(colour_i,:),'linewidth',2)
%     axis([0 1 0 1])
% end
% xlabel('x')
% ylabel('u')
% h=get(gcf, "currentaxes");
% set(h, "fontsize", 12, "linewidth", 1);
% cbar=colorbar();
% caxis([0,T])
% ylabel(cbar,'t','Rotation',0)
%
% title(sprintf('EB, D^{3/2}=-(D^3)^{1/2} \\Delta x=%d, T=%d, Nt=%d', delta_x, T, Nt));
% filename=sprintf('Ex61/Ex61-EB-posDer-multi-t-delx=%d.png',delta_x);
% title(sprintf('EB, D^{3/2}=-(-D^3)^{1/2} \\Delta x=%d, T=%d, Nt=%d', delta_x, T, Nt));
% filename=sprintf('Ex61/Ex61-EB-minusDer-multi-t-delx=%d.png',delta_x);
% title(sprintf('EB, D^{3/2}=-((D^3)^{1/2}+(-D^3)^{1/2})/sqrt(2) \\Delta x=%d, T=%d, Nt=%d', del
% filename=sprintf('Ex61/Ex61-EB-mixedDer-multi-t-delx=%d.png',delta_x);
%
% print(1,filename, '-dpng','-S600,400')

% listoftimes=1:(Nt-1)/10:Nt;
% clf
% hold on
% for i=1:10
%     tau=listoftimes(i)
%     plot(x,u_eulfor(:,tau));
% end
% title("\\Delta x=0.01")
% h=get(gcf, "currentaxes");
% set(h, "fontsize", 12, "linewidth", 1);
% print(1,'diff-openbound-D3over2.png', '-dpng','-S1200,800')
%
```

A.4 ex62.m

```
% -------------- FUNCTIONS --------------

    % Defining g(u, delta_t, gamma) function to be called in the recursive relation
    1;

    function g_vec=g_funct(u, delta_t, gamma_val)

        g_vec = u + delta_t * gamma_val*(u.*(1-u));

    endfunction

% -------------- MAIN CODE --------------

    % Defining parameters of the simulation: tmax = maximum time, delta_x = space step-size
    % d_val = diffusion coefficient, gamma_val = reaction coefficient

    t_max = 1;
    delta_x = 0.01;
    d_val = 0.001;
    gamma_val = 5;

    % Find number of steps in space N and create vector x with values in space in [0,1]

    N = 1 + 1/delta_x
    x = linspace(0,1,N);

    % Define matrix D3 that computes third order time deivative with OBC and compute the
    % square roots with sqrtm function

    a = 0.5;
    b = -1;
    c = 1;
    D3= diag(a*ones(1,N-2),2) + diag(b*ones(1,N-1),1) + diag(c*ones(1,N-1),-1) + diag((-a)*ones(
    D3 = D3/(delta_x^3)
    D3sq = sqrtm(D3)
    D3sqmin = sqrtm(-D3)

    % Define matrix D (=D2) that computes second order time deivative with OBC

    a = -2;
    b = 1;
    c = 1;
    D2 = diag(a*ones(1,N)) + diag(b*ones(1,N-1),1) + diag(c*ones(1,N-1),-1);
    D2(1,N)=1;
    D2(N,1)=1;
    D2 = D2/(delta_x)^2;

    % Impose initial condition

    u_initial = zeros(N,1);

    for i=1:N
        u_initial(i) = exp(-50*(x(i)-1/2)^2);
    end

    % Fix number of time-steps t_interval and compute vector t with values of time in [0,t_max]
```

```
    t_interval = 500;
    t = linspace(0,t_max,t_interval)
    delta_t = t_max/(t_interval-1)

    % Implement recursive relation in time to find the solution u_approx

    u_approx=zeros(N,size(t)(2));

    u_approx(:,1) = u_initial;

    for j=1:(size(t)(2)-1);

        % Solution with D3 and OBC

            % A = -d_val*(D3sq+D3sqmin)/sqrt(2);

            % A = eye(N) - delta_t*A;

        % Solution with D2 and OBC

            A = eye(N) - delta_t*d_val*(D2);

        u_approx(:,j+1) = A\g_funct(u_approx(:,j), delta_t, gamma_val);

    end

% -------------- PLOTS --------------

    % Plot for a bunch of different times with colour map

    listoftimes = [];

    time_intervals = 10;

    delta_tint = floor(t_interval/time_intervals)

    listoftimes(1) = delta_tint;


    for i=2:time_intervals;
        listoftimes(i) = listoftimes(i-1) + delta_tint;
    end

    listoftimes
    size(listoftimes)(2)

    clf
    hold on

        cmap=colormap();
    for ti=1:size(listoftimes)(2)
        tau=listoftimes(ti)
        colour_i=round(1+(tau*63/(t_interval)));
        plot(x, u_approx(:,tau),'Color', color=cmap(colour_i,:),'linewidth',2)
        axis([0 1 0 1])
    end

    xlabel('x');
```

```
ylabel('u');
h=get(gcf, "currentaxes");
set(h, "fontsize", 12, "linewidth", 1);
cbar=colorbar();
caxis([0,t_max]);
ylabel(cbar,'t','Rotation',0);
title(sprintf('IMEX \\Deltax=%d, T=%d, Nt=%d', delta_x, t_max, t_interval));
filename=sprintf('plots_6.2/Ex62multit_Nt=%d_D2.png',t_interval);
% title(sprintf('EXPM, D^{3/2}=-(-D^3)^{1/2}) \\Delta x=%d, T=%d, Nt=%d', delta_x, T, Nt));
% filename=sprintf('Ex61/Ex61-EXPM-minusDer-multi-t-delx=%d.png',delta_x);
% title(sprintf('EXPM, D^{3/2}=-((D^3)^{1/2}+(-D^3)^{1/2})/sqrt(2) \\Delta x=%d, T=%d, Nt=%d
% filename=sprintf('Ex61/Ex61-EXPM-mixedDer-multi-t-delx=%d.png',delta_x);
print(1,filename, '-dpng','-S600,400')


% Print in time for a bunch of space points

clf
hold on

cmap=colormap();
listofspace=1
for xi=1:6

    colour_i=round((listofspace*63/(N/2)))
    plot(t, u_approx(listofspace,:),'Color', color=cmap(colour_i,:),'linewidth',2)

    axis([0 1 0 1])

    listofspace = listofspace + (N-1)/10
end

xlabel('t');
ylabel('u');
h=get(gcf, "currentaxes");
set(h, "fontsize", 12, "linewidth", 1);
cbar=colorbar();
caxis([0,0.5]);
ylabel(cbar,'t','Rotation',0);
title(sprintf('IMEX \\Deltax=%d, T=%d, Nt=%d', delta_x, t_max, t_interval));
filename=sprintf('plots_6.2/Ex62multix_Nt=%d_first1_D2.png',t_interval);
print(1,filename, '-dpng','-S600,400')


clf
hold on

cmap=colormap();

listofspace = 51

for xi=1:6

    colour_i=round(((listofspace-50)*63/(N/2)))
    plot(t, u_approx(listofspace,:),'Color', color=cmap(colour_i,:),'linewidth',2)

    axis([0 1 0 1])

    listofspace = listofspace + (N-1)/10
```

```
    end

    xlabel('t');
    ylabel('u');
    h=get(gcf, "currentaxes");
    set(h, "fontsize", 12, "linewidth", 1);
    cbar=colorbar();
    caxis([0.5,1]);
    ylabel(cbar,'x','Rotation',0);
    title(sprintf('IMEX \\Deltax=%d, T=%d, Nt=%d', delta_x, t_max, t_interval));
    filename=sprintf('plots_6.2/Ex62multix_Nt=%d_second_D2.png',t_interval);
    print(1,filename, '-dpng','-S600,400')
```

A.5 comp.m

```
1;



%%%%%%%%%%%%% Initial Values
% Here we define our global values, the total time, the number of time steps
% and the space interval size
% Here is various methods to up the list of delta x's
% to loop through

%%% Set number of steps and uper and lower bounds
% Nsteps=100;
% delta_xList=linspace(0.01,0.1,Nsteps);

% Similar but with two different linspaces, alowing us to probe across a wider range of delta x'
delta_xList=[linspace(0.001,0.01,10),linspace(0.01,0.1,10)];
Nsteps=size(delta_xList)(2);
differenceEXPM=zeros(1,Nsteps);
for j=1:Nsteps
    delta_x=delta_xList(j)

% delta_x=0.01;
    T=0.03;
    Nt=81;
    delta_t=T/(Nt-1);
    stability=delta_t/(delta_x)^(3/2);
    if stability>0.5
        printf("EF Unstable, stability ratio: %.1f\n", stability)
    end

    Nx=int32(1+(1/delta_x));
    t=linspace(0,T,Nt);
    x=linspace(0,1,Nx);



%%%%%%%%%%%%% Defining the finite-differnce Matrices
%     Here we define the finite-differnce matrix we want to use in this problem.
%     For later convience the Periodic condisiton is
%     also included
%     D3
%     This consists of 0 along the diagonal,  2 's on the1-off diagonal and   1  2-off diagonals
    a=0;    % Diagonal
    b=-1;   % Right-1-diagonal
    c=1;    % Left-1-diagonal
    d=-1/2; % Left-2-diagonal
    f=1/2;  % Right-2-diagonal
    D3 = diag(a*ones(1,Nx)) + diag(b*ones(1,Nx-1),1) +diag(f*ones(1,Nx-2),2)+diag(d*ones(1,Nx-2)
    % B.C.
    D3(1,Nx)=c;
    D3(1,Nx-1)=d;
    D3(2,Nx)=d;
    D3(Nx,1)=b;
    D3(Nx,2)=f;
    D3(Nx-1,1)=f;
    D3;
    D3=D3/(delta_x^3);
```

```
    D3over2=-(sqrtm(D3) + sqrtm(-D3))/sqrt(2);


    %% D2
    % This consists of -2 along the diagonal and 1's on the two 1-off diagonals.
    der_a=-2;    % Diagonal
    der_b=1;     % Right-1-off Diagonal
    der_c=1;     % Left-1-off Diagonal
    der_d=0;     % Left-2-off Diagonal
    der_f=0;     % left-2-off Diagonal
    D2 = diag(der_a*ones(1,Nx)) + diag(der_b*ones(1,Nx-1),1) +diag(der_f*ones(1,Nx-2),2)+diag(de
    D2(1,Nx)=der_c;
    D2(Nx,1)=der_b;
    D2=D2 / delta_x^2;
    D2Fixed=D2;
    D2Fixed(1,:)=0;
    D2Fixed(Nx,:)=0;

    D23over4=-sqrtm(sqrtm((-D2)^3));






%%%%%%%%%%%%%% Initial Conditions
%   We initialise our three integration method solution
%   matrices.
    u_initial=zeros(Nx,1);
    for i=1:Nx
        u_initial(i)=exp(-100*(x(i)-0.5)^2);
    end;


%%%%%%%%%%%%%% Final Conditions
%   We find the vectors at the final T
    uofT=expm(t(end).*D3over2)*u_initial;
    vofT=expm(t(end).*D23over4)*u_initial;
%   Compute their differeces
    differ=uofT - vofT;
    differenceEXPM(j)=sqrt(differ'*differ);

end

%%%%%%%%%%%%%% Plotting
% Some of this is not used, and is keep as archival for future usage if required
% Also note the plots require a folder Comp to be present.

% Difference Plotting
clf
hold on
cmap=colormap
loglog(delta_xList, differenceEXPM, 'Color', color=cmap(1,:), ':o', 'linewidth', 2)
legend(labels=['EXPM'])
xlabel('\Delta x')
ylabel('||u-v||')
title('Comparing -(sqrt(D3)+sqrt(-D3))/sqrt(2) vs. -sqrt(sqrt((-D2)3))')
print(1,'Comp/Comp-diff-EXPM.png', '-dpng','-S600,400')
```

```
%%% Colorplot EXPM
% Plotting of u(x,t)
clf
hold on
listoftimes=1:(Nt-1)/5:Nt;
cmap=colormap();
for ti=1:size(listoftimes)(2)
    tau=listoftimes(ti);
    colour_i=round(1+(tau*63/Nt));
    plot(x, u_expm(:,tau),'Color', color=cmap(colour_i,:),'linewidth',2)
    axis([0 1 0 1])
end
xlabel('x')
ylabel('u')
h=get(gcf, "currentaxes");
set(h, "fontsize", 12, "linewidth", 1);
cbar=colorbar();
caxis([0,T])
ylabel(cbar,'t','Rotation',0)
%
title(sprintf('EXPM,␣D^{3/2}=-((D^3)^{1/2}+(-D^3)^{1/2})/sqrt(2)␣\\Delta␣x=%d,␣T=%d,␣Nt=%d', del
filename=sprintf('Comp/Ex61-EXPM-mixedDer-multi-t-delx=%d.png',delta_x);

print(1,filename, '-dpng','-S600,400')



% Plotting of v(x,t)
clf
hold on
listoftimes=1:(Nt-1)/5:Nt;
cmap=colormap();
for ti=1:size(listoftimes)(2)
    tau=listoftimes(ti);
    colour_i=round(1+(tau*63/Nt));
    plot(x, v_expm(:,tau),'Color', color=cmap(colour_i,:),'linewidth',2)
    axis([0 1 0 1])
end
xlabel('x')
ylabel('v')
h=get(gcf, "currentaxes");
set(h, "fontsize", 12, "linewidth", 1);
cbar=colorbar();
caxis([0,T])
ylabel(cbar,'t','Rotation',0)

title(sprintf('EXPM,␣D^2,␣Periodic␣BCs,␣\\Delta␣x=%d,␣T=%d,␣Nt=%d', delta_x, T, Nt));
filename=sprintf('Comp/Comp-EXPM-2D3over4-PBCs-multi-t-delx=%d.png',delta_x);
print(1,filename, '-dpng','-S600,400')
```

References

[1] Zegeling, P. An Adaptive Grid Method for a Non-equilibrium PDE Model from Porous Media. *Journal Of Mathematical Study*. **48** pp. 187-198 (2015,6)