```
% MATH3036 Coursework 2
% NAME: JAKE DENTON
```

Question 1 File Checklist

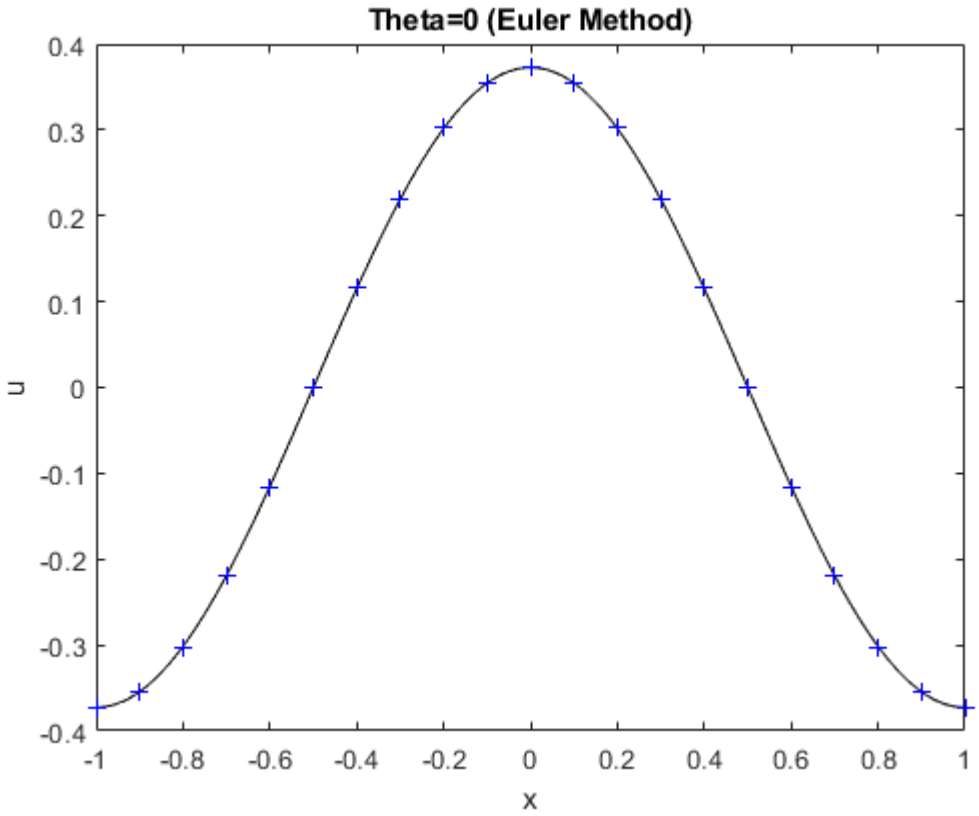
The files diffusion1d.m and Coursework2Script.m are required for this question.

Question 1a) Theta=0

The first three sections of code call the diffusion1d.m function, which produces a plot with the approximations and the true curve at t=T along with the value of the 2-norm error, for theta=0,0.5,1 respectively.

```
[~,~]=diffusion1d(0,0.1,1,20,100,1,1,1);
title('Theta=0 (Euler Method)');
```

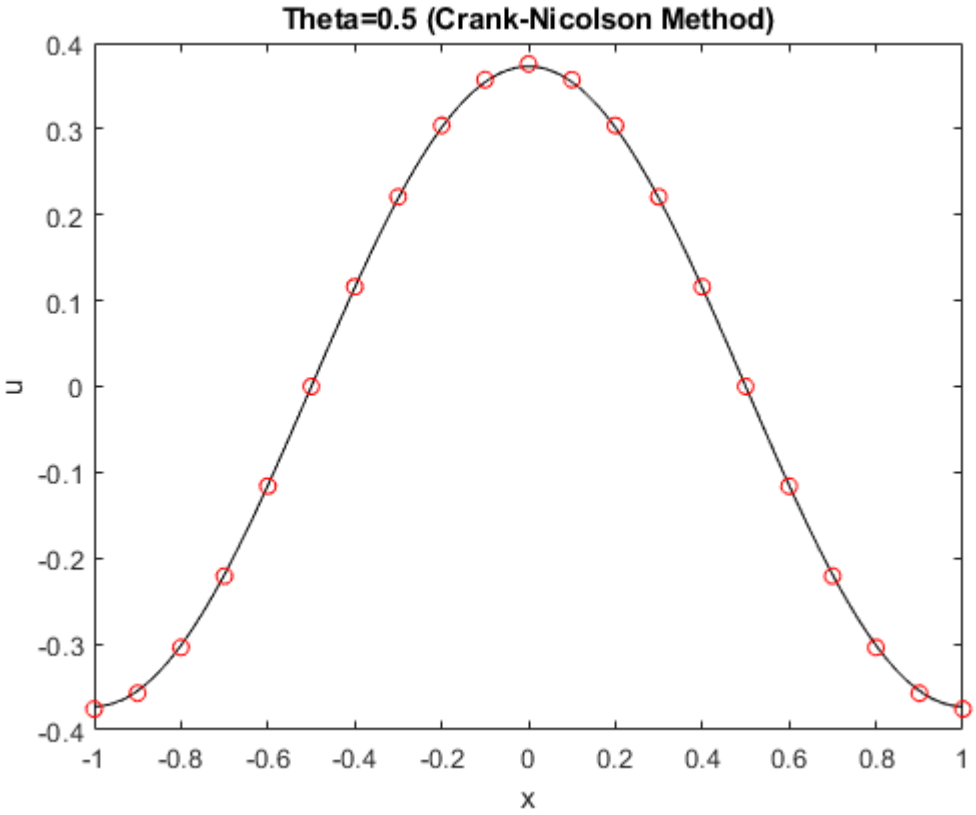
```
The 2-norm error for theta=0 is 0.0012797
```



Question 1a) Theta=0.5

```
[~,~]=diffusion1d(0.5,0.1,1,20,100,1,2,2);
title('Theta=0.5 (Crank-Nicolson Method)');
```

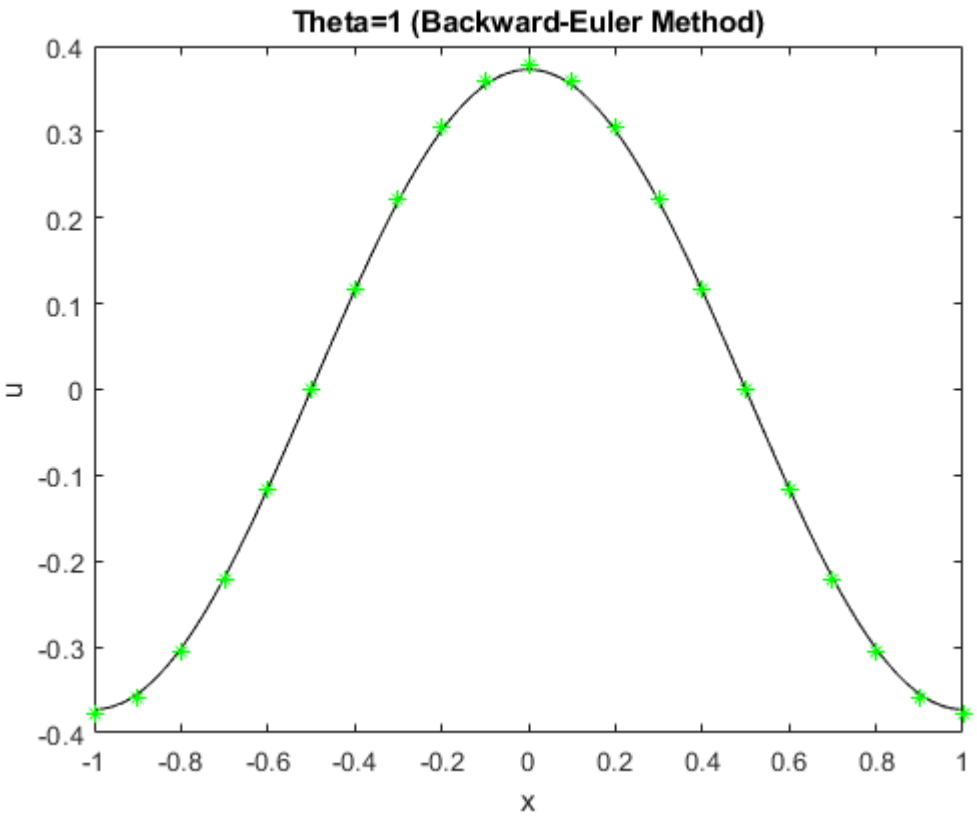
```
The 2-norm error for theta=0.5 is 0.0031724
```



Question 1a) Theta=1

```
[~,~]=diffusion1d(1,0.1,1,20,100,1,3,3);
title('Theta=1 (Backward-Euler Method)');
```

```
The 2-norm error for theta=1 is 0.0050557
```



Question 1b)i)

Three tables for the first refinement path with different theta are given below. For theta=0, the error does not decrease uniformly and for larger numbers of sub-intervals the error diverges. We showed in the lectures that the Euler method in time has a refinement path dt<=h^2/(2*kappa), so for larger Nx/Nt in the table, this inequality isn't satisfied and the approximation is unstable. For theta=0.5,1 the approximations are stable as shown in the table. For theta=1 the method is first order accurate as h approaches 0, whilst for theta=0.5 the method is second order accurate (using the error ratios and the fact h is halved between each row). The additional stability is due to the equations solved at each step being implicit (the Euler method is explicit).

```
ErrorEM=zeros(5,1); %Initialise vectors
ErrorBM=zeros(5,1);
ErrorCN=zeros(5,1);
SpaceIntervals=zeros(5,1);
TimeIntervals=zeros(5,1);
Nx=20; %Initialise number of sub-intervals
Nt=100;
for i=1:5 %For loop finds the errors using each theta for each number of sub-intervals
    [eTEM,~]=diffusion1d(0,0.1,1,Nx,Nt,0,1);
    [eTBM,~]=diffusion1d(1,0.1,1,Nx,Nt,0,1);
    [eTCN,~]=diffusion1d(0.5,0.1,1,Nx,Nt,0,1);
    ErrorEM(i)=eTEM;
    ErrorBM(i)=eTBM;
    ErrorCN(i)=eTCN;
    SpaceIntervals(i)=Nx;
    TimeIntervals(i)=Nt;
    Nx=2*Nx; %For this refinement scheme, when Nx is doubled Nt should also be doubled
    Nt=2*Nt;
```

```
end
TEM=table(SpaceIntervals,TimeIntervals,ErrorEM); %Create table for theta=0
TEM.Properties.VariableNames={'Nx','Nt','ErrorTheta0'};
disp(TEM);
ratioBM=zeros(5,1); %We want to see the ratios for the other values of theta
ratioCN=zeros(5,1);
ratioBM(1)=1; %This first element can be ignored, simply so the vectors have same length in
table
ratioCN(1)=1;
for j=1:4
    ratioBM(j+1)=ErrorBM(j)/ErrorBM(j+1);
    ratioCN(j+1)=ErrorCN(j)/ErrorCN(j+1);
end
TBM=table(SpaceIntervals,TimeIntervals,ErrorBM,ratioBM);
TBM.Properties.VariableNames={'Nx','Nt','ErrorTheta1','ErrorRatio'};
disp(TBM);
TCN=table(SpaceIntervals,TimeIntervals,ErrorCN,ratioCN);
TCN.Properties.VariableNames={'Nx','Nt','ErrorThetaHalf','ErrorRatio'};
disp(TCN);
```

Nx	Nt	ErrorTheta0
20	100	0.0012797
40	200	0.00015485
80	400	0.00026824
160	800	Inf
320	1600	NaN

Nx	Nt	ErrorTheta1	ErrorRatio
20	100	0.0050557	1
40	200	0.0017014	2.9716
80	400	0.00065018	2.6168
160	800	0.00027574	2.358
320	1600	0.00012563	2.1949

Nx	Nt	ErrorThetaHalf	ErrorRatio
20	100	0.0031724	1
40	200	0.00077442	4.0965
80	400	0.00019126	4.0491
160	800	4.752e-05	4.0248
320	1600	1.1843e-05	4.0124

Question 1b)ii)

On this refinement path, when h is halved, dt is quartered. The three tables below show that, as expected, we have stable approximations for all three values of theta. The error ratios approach 4 in all three cases, and the errors are very small in magnitude. The theta=0 method is the only one which reaches a magnitude of 10^(-6) for these values of Nx/Nt, and has the lowest error of the three for every fixed Nx/Nt. These error ratios are expected since a second-order centred difference method was used for each approximation, and this second-order is inherited by each method.

```
ErrorEM2=zeros(5,1); %Initialise vectors
ErrorBM2=zeros(5,1);
ErrorCN2=zeros(5,1);
SpaceIntervals2=zeros(5,1);
TimeIntervals2=zeros(5,1);
Nx=20; %Initialise number of sub-intervals
Nt=100;
for i=1:5 %For loop finds the errors using each theta for each number of sub-intervals
    [eTEM2,~]=diffusion1d(0,0.1,1,Nx,Nt,0,1);
    [eTBM2,~]=diffusion1d(1,0.1,1,Nx,Nt,0,1);
    [eTCN2,~]=diffusion1d(0.5,0.1,1,Nx,Nt,0,1);
    ErrorEM2(i)=eTEM2;
    ErrorBM2(i)=eTBM2;
    ErrorCN2(i)=eTCN2;
    SpaceIntervals2(i)=Nx;
    TimeIntervals2(i)=Nt;
    Nx=2*Nx; %For this refinement scheme, when Nx is doubled Nt should be quadrupled
    Nt=4*Nt;
end
ratioEM2=zeros(5,1);
ratioBM2=zeros(5,1);
ratioCN2=zeros(5,1);
ratioEM2(1)=1;
ratioBM2(1)=1;
ratioCN2(1)=1;
for j=1:4
    ratioEM2(j+1)=ErrorEM2(j)/ErrorEM2(j+1);
    ratioBM2(j+1)=ErrorBM2(j)/ErrorBM2(j+1);
    ratioCN2(j+1)=ErrorCN2(j)/ErrorCN2(j+1);
end

TEM1=table(SpaceIntervals2,TimeIntervals2,ErrorEM2,ratioEM2);
TEM1.Properties.VariableNames={'Nx','Nt','ErrorTheta0','ErrorRatio'};
disp(TEM1);
TBM1=table(SpaceIntervals2,TimeIntervals2,ErrorBM2,ratioBM2);
TBM1.Properties.VariableNames={'Nx','Nt','ErrorTheta1','ErrorRatio'};
disp(TBM1);
TCN1=table(SpaceIntervals2,TimeIntervals2,ErrorCN2,ratioCN2);
TCN1.Properties.VariableNames={'Nx','Nt','ErrorThetaHalf','ErrorRatio'};
disp(TCN1);
```

Nx	Nt	ErrorTheta0	ErrorRatio
20	100	0.0012797	1
40	400	0.00031065	4.1194
80	1600	7.6615e-05	4.0547
160	6400	1.9029e-05	4.0262
320	25600	4.7421e-06	4.0128

Nx	Nt	ErrorTheta1	ErrorRatio
20	100	0.0050557	1
40	400	0.0012388	4.0813
80	1600	0.00030622	4.0453
160	6400	7.6102e-05	4.0238

```
    320    25600    1.8968e-05       4.0122

    Nx      Nt      ErrorThetaHalf    ErrorRatio
    ___    _____    ______________    __________

     20      100       0.0031724            1
     40      400      0.00077499       4.0935
     80     1600      0.00019144       4.0483
    160     6400      4.7567e-05       4.0246
    320    25600      1.1855e-05       4.0124
```

Question 1b) General Kappa

We might want to apply these methods to the diffusion problem with a different value of kappa. It was mentioned previously that the inequality that dt must satisfy for the Euler method is dt<=h^2/(2*kappa). Clearly, if kappa is made larger, the size of this interval shrinks and we can easily get an unstable approximation. To investigate this, I set kappa equal to 2 which produced the table below. The theta=0 approximation diverges straight away whilst the theta=0.5,1 approximations showed similar behaviour to before. In this way, the theta=0 approximation is only reliable for small values of kappa with dt proportional to h^2 as a refinement path, whereas the implicit methods are reliable for larger magnitudes of kappa.

```
ErrorEMKappa=zeros(4,1); %Initialise vectors
ErrorBMKappa=zeros(4,1);
ErrorCNKappa=zeros(4,1);
SpaceIntervalsKappa=zeros(4,1);
TimeIntervalsKappa=zeros(4,1);
Nx=20; %Initialise number of sub-intervals
Nt=100;
for i=1:4 %For loop finds the errors using each theta for each number of sub-intervals
    [eTEMk,~]=diffusion1d(0,2,1,Nx,Nt,0,1);
    [eTBMk,~]=diffusion1d(1,2,1,Nx,Nt,0,1);
    [eTCNk,~]=diffusion1d(0.5,2,1,Nx,Nt,0,1);
    ErrorEMKappa(i)=eTEMk;
    ErrorBMKappa(i)=eTBMk;
    ErrorCNKappa(i)=eTCNk;
    SpaceIntervalsKappa(i)=Nx;
    TimeIntervalsKappa(i)=Nt;
    Nx=2*Nx; %For this refinement scheme, when Nx is doubled Nt should also be doubled
    Nt=4*Nt;
end
TKappa=table(SpaceIntervalsKappa,TimeIntervalsKappa,ErrorEMKappa,ErrorBMKappa,ErrorCNKappa);
TKappa.Properties.VariableNames={'Nx','Nt','ErrorTheta0','ErrorTheta1','ErrorThetaHalf'};
disp(TKappa);
```

```
    Nx      Nt      ErrorTheta0     ErrorTheta1     ErrorThetaHalf
    ___    _____    ___________     ___________     ______________

     20      100     3.027e+67       1.5216e-08       2.9176e-10
     40      400            NaN      1.8249e-09       1.0211e-10
     80     1600            NaN      3.7846e-10       2.6933e-11
    160     6400            NaN      9.0057e-11       6.7948e-12
```

Question 1b) Which method should be used?

From the investigations above, the method which should be used largely depends on the problem. The theta=0 method requires the h^2 refinement path, so many more time-steps are required for a stable approximation, and it is also unstable for values of kappa that aren't small (unless a huge number of time-steps are used). However, despite being the only explicit method, it runs in a similar time to the implicit methods as these involve working with a tridiagonal matrix, so the number of calculations required to solve these systems is O(n). For these reasons, this method should only be used if kappa is small and many time steps can be used. If the number of time-steps is restricted to a smaller range and high accuracy is still required, or kappa is not small, the theta=0.5 method should be used. This method is reliable on both refinement paths investigated and is stable for large kappa (kappa=5,10 were also investigated). It also gives an error that is in general smaller than the theta=1 (backward-Euler) method.

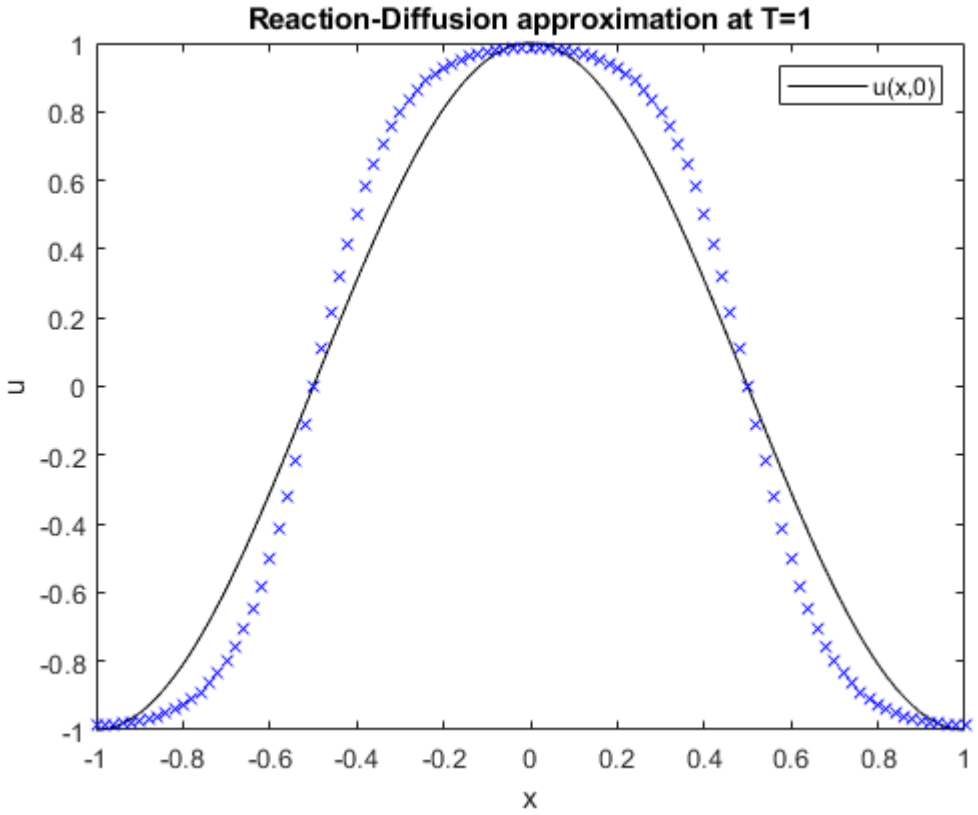
Question 2 File Checklist

The files reaction_diffusion1d.m, Coursework2Script.m and FindMaxDt.m (for the theta=0 part of 2b)) are used in this question.

Question 2a) Approximation at T=1

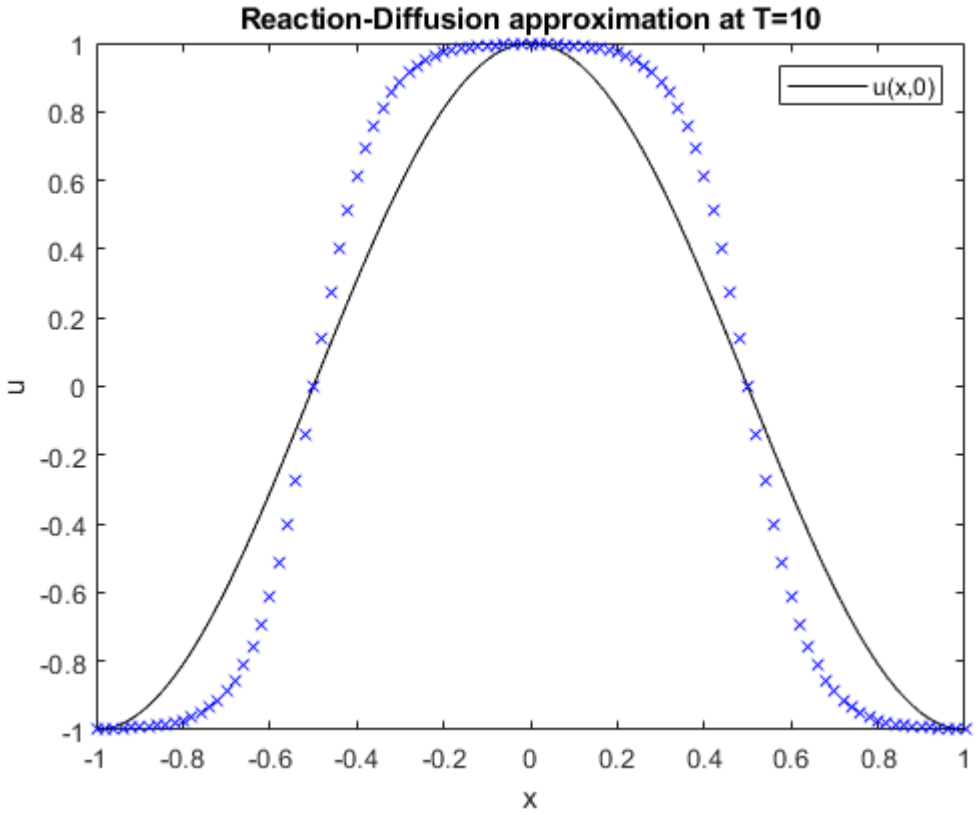
The next three sections produce plots of the u approximation for the reaction-diffusion equation for differing final times T=1,10,20. The u approximations are found by Backward-Euler method (or theta=1), and kappa=0.01. The second and third plots appear to be very similar, so a for loop is used to check that the space approximations are in fact different for these final times T. They are in fact very close, the minimum absolute change is of magnitude 10^(-15) and the maximum is of magnitude 10^(-7).

```
[~]=reaction_diffusion1d(1,0.01,1,100,10,1,4); %calls function to produce plot
title('Reaction-Diffusion approximation at T=1');
legend('u(x,0)','Location','northeast');
```



Question 2a) Approximation at T=10

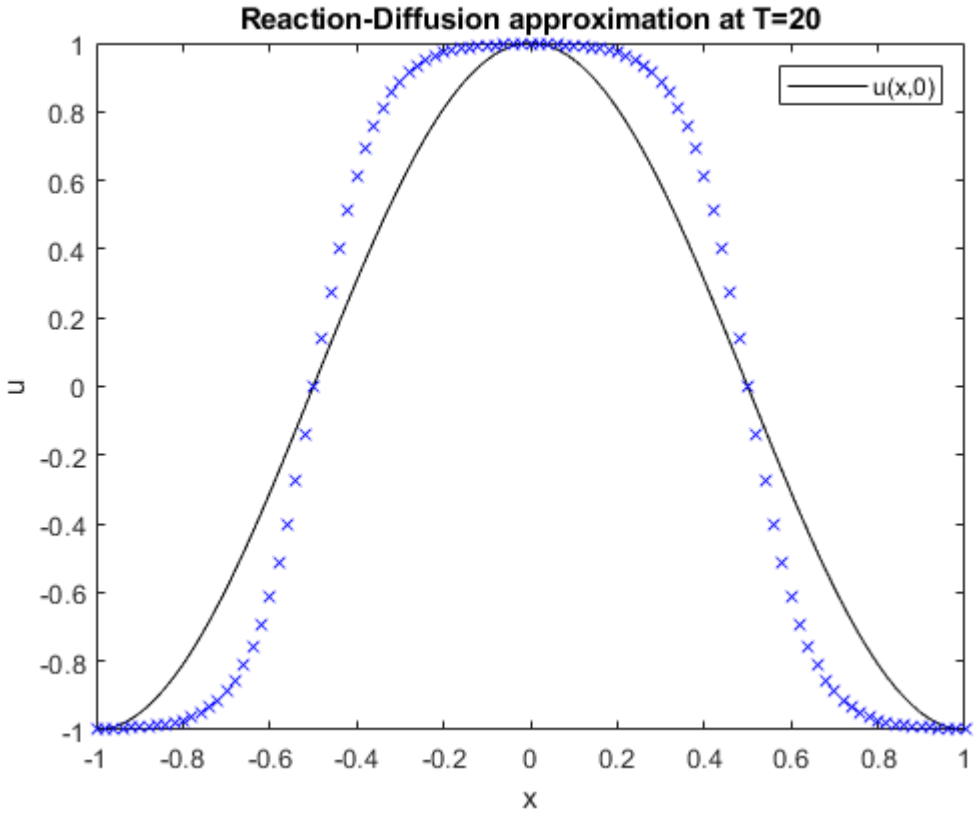
```
[~]=reaction_diffusion1d(1,0.01,10,100,100,1,4);
title('Reaction-Diffusion approximation at T=10');
legend('u(x,0)','Location','northeast');
```



Question 2a) Approximation at T=20

```
[u20]=reaction_diffusion1d(1,0.01,20,100,200,1,4);
title('Reaction-Diffusion approximation at T=20');
legend('u(x,0)','Location','northeast');
[u10]=reaction_diffusion1d(1,0.01,10,100,100,0,1); %obtain approx. at T=10 to check if equal
change=u20-u10; %difference between space points
T20Equal=0;
for i=1:101 %for loop checks if the value of the approximation at each space point x is the
same for T=10,20
    if change(i)==0
        T20Equal=T20Equal+1;
        disp(['Element ',num2str(i),' is the same for both times T']);
    else
    end
end
disp(['There are ',num2str(T20Equal),' space approximations in common between the T=10,T=20
vectors']);
```

```
There are 0 space approximations in common between the T=10,T=20 vectors
```



Question 2b) Theta=0,Kappa=1e-2,1e-4

This section of code produces tables which show the minimum number of time-steps required to obtain an approximation (MinimumNtForApprox) and then the value for Nt near to this minimum which gives sensible approximations i.e. the shape of the plot is similar to what we obtained in Q2a).The MinimumNtForApprox values are found using the function FindMaxDt which takes values of theta, kappa, Nx and an initial Nt value and using these applies a bisection-type method to give the lowest value of Nt which gives a vector of approximations that does not contain NaN values. Even equipped with these values, the approximations produced using them may not be sensible which is shown via the plot produced by this section, which doesn't resemble the shape we had at all. Due to this, a second number for the time steps Nt is found by trial and error by looking slightly above the values given by FindMaxDt which does give a sensible shape in approximation. The code section following this produces a table for the smaller values of kappa. The number of time-steps suggested lead to sensible plots as shown for the specific case of kappa=1e-6,Nx=3200. These values were also found using trial and error, but FindMaxDt did not need to be called as Nt was small for Nx=800,1600.

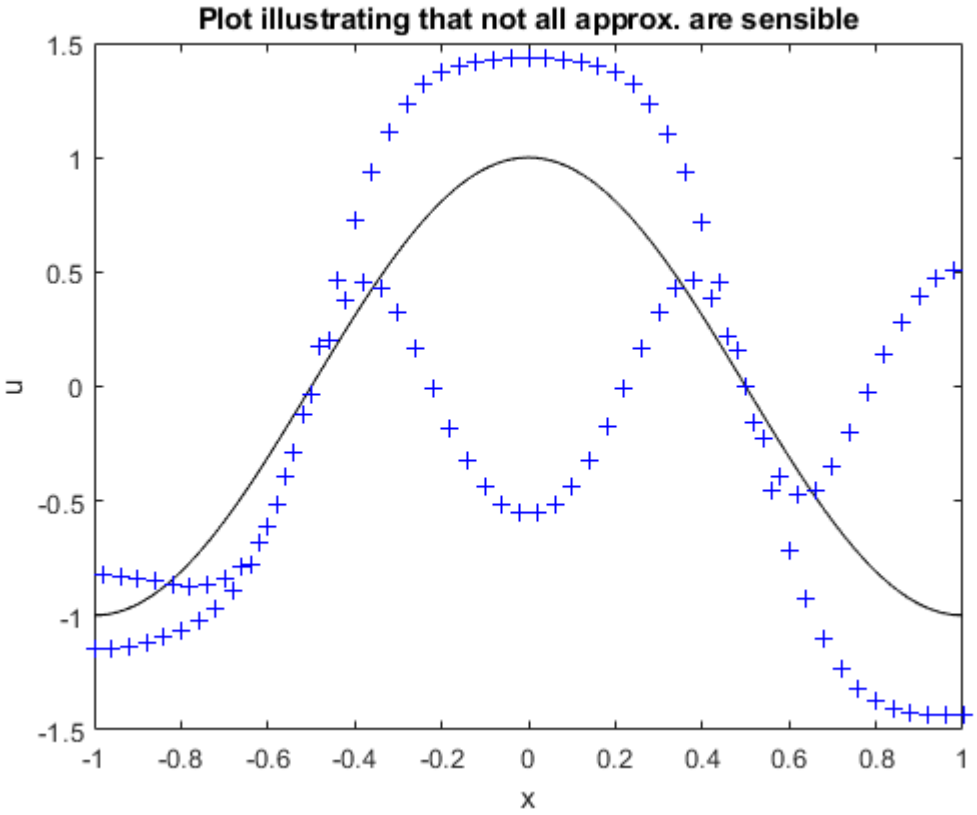
```
h=zeros(3,1);
MinStableNt=zeros(3,1);
Nx=200;
for i=1:3
    [Nt1,~]=FindMaxDt(0,0.01,Nx,200);
    MinStableNt(i)=Nt1;
    h(i)=2/Nx;
    Nx=0.5*Nx;
end
NOTSensible=MinStableNt(2);
```

```
[~]=reaction_diffusion1d(0,0.01,20,100,NOTSensible,1,1);
title('Plot illustrating that not all approx. are sensible');
kappa=0.01*ones(3,1);
SensibleNt=[4000,1000,253]';
dt=20./SensibleNt;
T0=table(kappa,h,MinStableNt,SensibleNt,dt);
T0.Properties.VariableNames={'Kappa','h','MinimumNtForApprox','SensibleNt','dt4Sensible'};
disp(T0);

h2=zeros(3,1);
MinStableNt2=zeros(3,1);
Nx2=800;
for i=1:3
    [Nt2,~]=FindMaxDt(0,0.0001,Nx2,200);
    MinStableNt2(i)=Nt2;
    h2(i)=2/Nx2;
    Nx2=0.5*Nx2;
end
kappa2=0.0001*ones(3,1);
SensibleNt2=[642,161,54]';
dt2=20./SensibleNt2;
T1=table(kappa2,h2,MinStableNt2,SensibleNt2,dt2);
T1.Properties.VariableNames={'Kappa','h','MinimumNtForApprox','SensibleNt','dt4Sensible'};
disp(T1);
```

Kappa	h	MinimumNtForApprox	SensibleNt	dt4Sensible
0.01	0.01	3997.7	4000	0.005
0.01	0.02	997.66	1000	0.02
0.01	0.04	247.66	253	0.079051

Kappa	h	MinimumNtForApprox	SensibleNt	dt4Sensible
0.0001	0.0025	639.06	642	0.031153
0.0001	0.005	157.03	161	0.12422
0.0001	0.01	42.188	54	0.37037



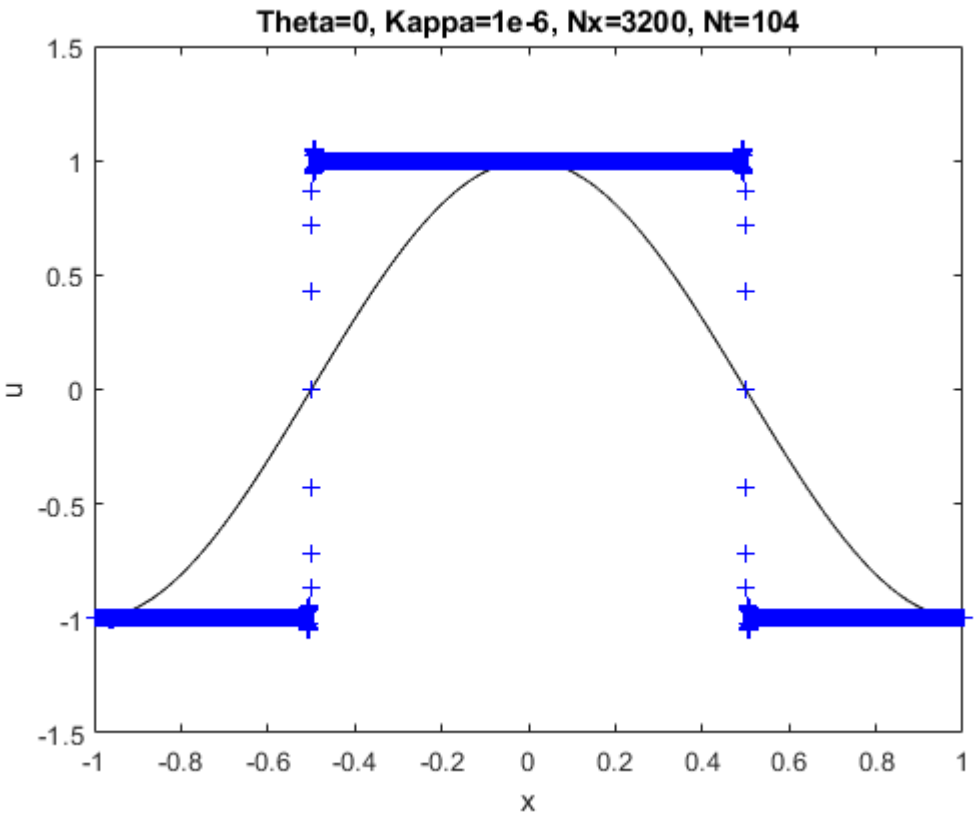
Question 2b) Theta=0,Kappa=1e-6,1e-8

```
h3=[2/3200,2/1600,2/800,2/3200,2/1600]';
SensibleNt3=[104,40,22,22,22]';
dt3=20./SensibleNt3;
Kap=0.000001*ones(5,1);
Kap(4:5)=0.00000001;

T2=table(Kap,h3,SensibleNt3,dt3);
T2.Properties.VariableNames={'Kappa','h','SensibleNt','dt4Sensible'};
disp(T2);

[~]=reaction_diffusion1d(0,0.000001,20,3200,104,1,1);
title('Theta=0, Kappa=1e-6, Nx=3200, Nt=104');
```

Kappa	h	SensibleNt	dt4Sensible
1e-06	0.000625	104	0.19231
1e-06	0.00125	40	0.5
1e-06	0.0025	22	0.90909
1e-08	0.000625	22	0.90909
1e-08	0.00125	22	0.90909



Question 2b) How does the maximum stable time-step depend on kappa and h? (Theta=0)

Fixing h=0.0025 for example and considering kappa=1e-4,1e-6, we see from the tables that the lowering of kappa allows for a greater time-step length to be chosen. For fixed kappa, say kappa=1e-4, decreasing h forces a smaller dt to be chosen. Why is this? The absolute stability constraints for the forward Euler method (theta=0) gives us the inequality mentioned in the lecture notes and Q1, which can be rearranged as (kappa*dt)/h^2<1/2. From this, we see that decreasing kappa allows dt to be chosen larger and decreasing h forces dt to be chosen smaller in order for the inequality to be satisfied. This matches and explains the observations we made regarding the tables produced by the above two sections.

Question 2b) Theta=1

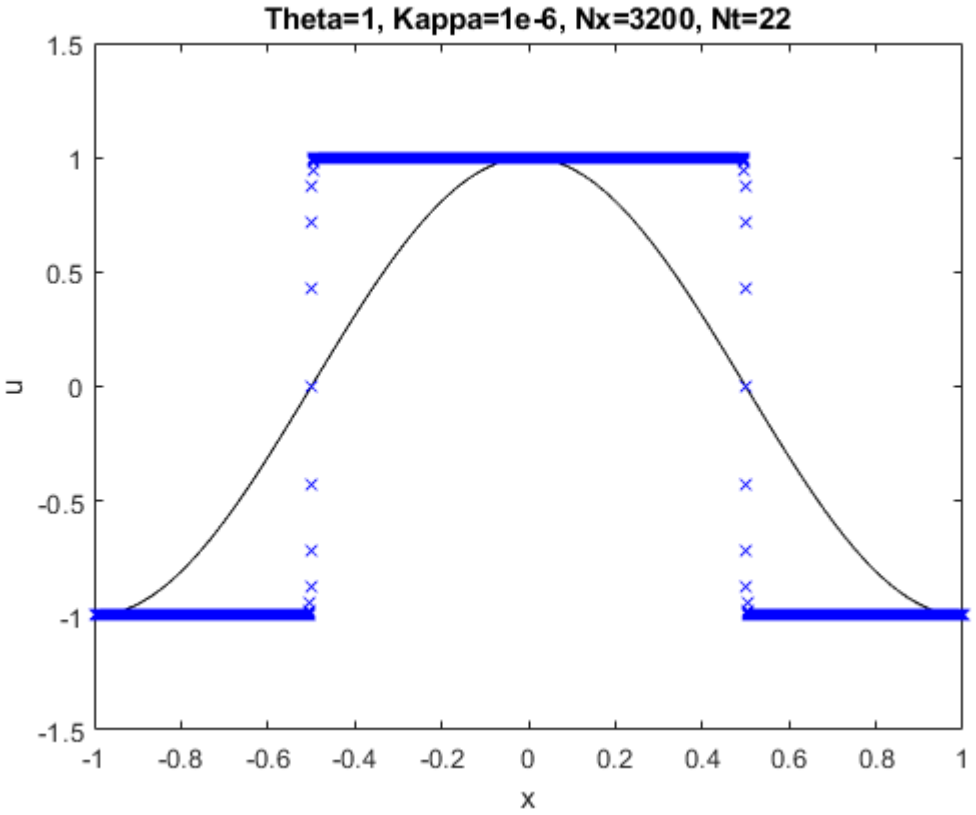
Using the results of von-Neumann analysis for the theta method as studied in exercise 3 of the lecture notes, we expect that there will be no restriction on the time-step for theta=1. Knowing this, I first tested lower values of the number of time-steps Nt (first checking if dt is allowed to be greater than 1). If the plot produced did not resemble what I saw in the previous section, then I increased Nt and checked the plot for this new Nt. I did this until the minimum Nt was found, the results of which can be seen for each value of kappa in the table below. Comparing the results for theta=1 with theta=0, we see that for theta=1 a far greater dt can be chosen for a fixed number of space intervals Nx. There is a plot below for Nx=3200, which uses almost 5x fewer time-steps to produce a sensible plot than what we had in the section above for theta=0. We can also see from the table below that for theta=1, as kappa or h gets smaller the maximum time-step does not change. This is interesting as when theta=0, we see the minimum Nt also reaches 22 for the smaller values of kappa. The only situation where dt>1 gives a sensible plot is when kappa=0.01 and theta=1 (see table, Nt=18). This

minimum at 22 matches the note made in the lecture notes that in practice there are still limits to the size of dt, in our case this limit is dt=0.909.

```
Kap=0.01*ones(12,1);
Kap(4:6)=0.0001;
Kap(7:9)=0.000001;
Kap(10:12)=0.00000001;
StepForKappa=[0.02;0.01;0.005];
Step=repmat(StepForKappa,4,1);
NtKappa=[18;22;22;22];
SensibleNtKappa=repelem(NtKappa,3);
Ttheta1=table(Kap,Step,SensibleNtKappa);
Ttheta1.Properties.VariableNames={'Kappa','h','SensibleNtTheta1'};
disp(Ttheta1);

[~]=reaction_diffusion1d(1,1e-6,20,3200,22,1,4);
title('Theta=1, Kappa=1e-6, Nx=3200, Nt=22');
```

Kappa	h	SensibleNtTheta1
0.01	0.02	18
0.01	0.01	18
0.01	0.005	18
0.0001	0.02	22
0.0001	0.01	22
0.0001	0.005	22
1e-06	0.02	22
1e-06	0.01	22
1e-06	0.005	22
1e-08	0.02	22
1e-08	0.01	22
1e-08	0.005	22



Question 2b) Discussions

As kappa is reduced, the solution of the PDE reaches a point where it appears somewhat discontinuous. For x>0.5 the points lie on the line u=-1, whilst for x<0.5 the points lie on the line u=1. At x very near to -0.5 and 0.5, the points have u values between -1 and 1. If u=-1 and 1 describe phases, then the points near x=-0.5,0.5 describe the interface between the phases. The change in u for these x happens over an incredibly small interval so it is difficult to observe in the plots. As a result, non-uniform discretisations where many more points are considered near to x=-0.5 and 0.5 may be used in order to see how u changes at the interface, since points further away from the interface are known to be at either u=1 or u=-1 from the plots we've produced.

Question 3 File Checklist

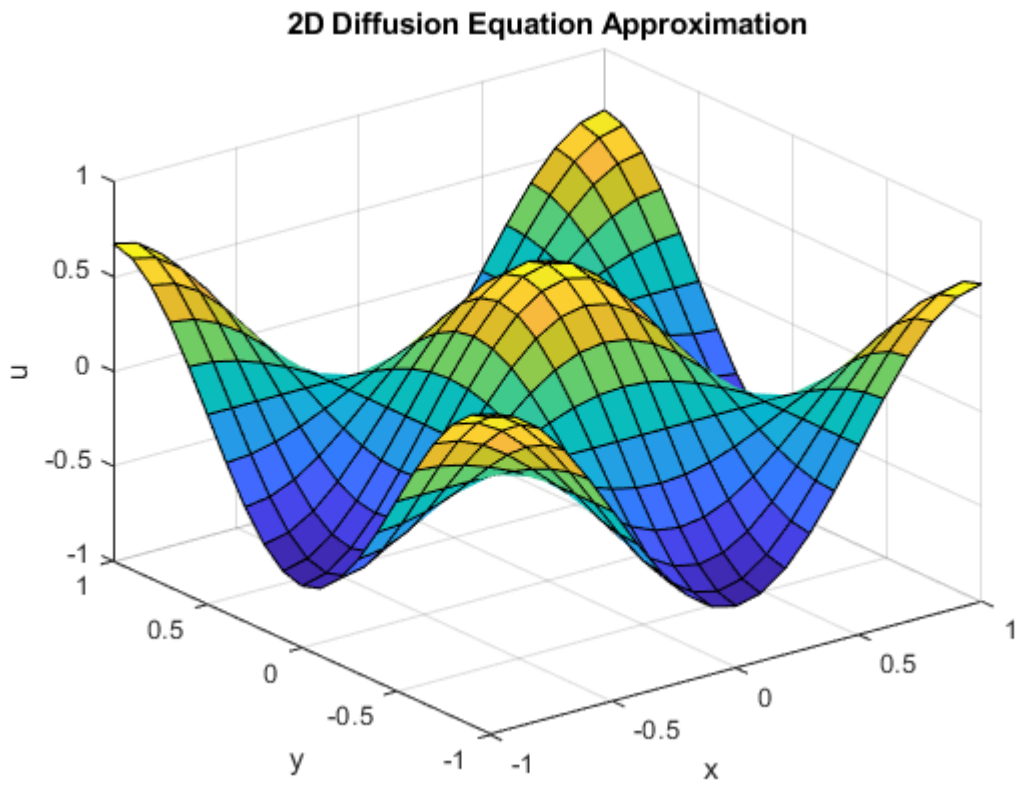
The files diffusion2d.m and Coursework2Script.m are used in this question.

Question 3a)

This section of code calls diffusion2d.m to approximate the 2D diffusion equation when kappa=1 and T=0.02. 20 time-steps are used as well as 20 space-steps in the x and y direction. The surface is plotted and the error is displayed below.

```
[~,~]=diffusion2d(1,0.02,20,20,20,1);
```

```
The 2-norm error at t=0.02 is 0.00047956
```



Question 3b) Nx=Ny Varied With h^2 Refinement Path

We see that for this refinement path, when h is halved (or equivalently Nx and Ny are doubled), the error reduces by a factor of 4 as h approaches zero. This is expected as we used second-order centred difference methods for the approximation in space dimensions x and y, so the method inherits second order of accuracy from this when Nx=Ny (and hx=hy=h) as shown in the table produced by this code section.

```
Nx=20; %Initialise the number of space/time steps
Ny=20;
Nt=20;
TwoDError=zeros(4,1); %Initialise vectors for the table
h=zeros(4,1);
dt=zeros(4,1);
for i=1:4 %This for loop assigns to the associated dt and h the error in the approximation e
    dt(i)=0.02/Nt;
    h(i)=2/Nx;
    [~,e]=diffusion2d(1,0.02,Nx,Ny,Nt,0);
    TwoDError(i)=e;
    Nx=2*Nx; %The number of space-steps are doubled
    Ny=2*Ny;
    Nt=4*Nt; %Under this refinement path, when Nx/Ny double the number of time steps Nt must
quadruple
end
ratio2D=zeros(4,1); %This will describe the ratio between the errors (helpful to analyse)
ratio2D(1)=1; %This is simply so the vectors will have the same length (required for the
table)
for j=1:3
    ratio2D(j+1)=TwoDError(j)/TwoDError(j+1);
```

```
end
T2D=table(h,dt,TwoDError,ratio2D); %Produce the table
T2D.Properties.VariableNames={'h','dt','Error','ErrorRatio'};
disp(T2D);
```

h	dt	Error	ErrorRatio
0.1	0.001	0.00047956	1
0.05	0.00025	0.00011476	4.1788
0.025	6.25e-05	2.8026e-05	4.0948
0.0125	1.5625e-05	6.9222e-06	4.0487

Question 3b) Fixed Ny=20, Nx Varied With hx^2 Refinement Path

We see that when Ny is fixed and a hx^2 refinement path is used to vary Nx and Nt, the error increases a small amount with Nx, until a point is reached where the error levels off (ratio goes to 1 as hx approaches 0). This is due to the fact that the truncation error is constrained by the fixed hy (there is an O(hy^2) term present in the truncation error) and therefore will not go to zero even as Nx gets very large.

```
Nx2=20;%The x-step and t-step are initialised here
Nt2=20;
TwoDError2=zeros(6,1); %Initialise vectors for the table
hx=zeros(6,1);
dt2=zeros(6,1);
for i=1:6 %This for loop assigns to the associated dt and h the error in the approximation e
    dt2(i)=0.02/Nt2;
    hx(i)=2/Nx2;
    [~,e2]=diffusion2d(1,0.02,Nx2,20,Nt2,0); %Ny is fixed at 20
    TwoDError2(i)=e2;
    Nx2=2*Nx2; %The number of space-steps in x are doubled
    Nt2=4*Nt2; %Under this refinement path, when Nx doubles the number of time steps Nt must
quadruple
end
ratio2D2=zeros(6,1); %This will describe the ratio between the errors (helpful to analyse)
ratio2D2(1)=1; %This is simply so the vectors will have the same length (required for the
table)
for j=1:5
    ratio2D2(j+1)=TwoDError2(j)/TwoDError2(j+1);
end
T2D2=table(hx,dt2,TwoDError2,ratio2D2); %Produce the table
T2D2.Properties.VariableNames={'hx','dt','Error','ErrorRatio'};
disp(T2D2);
```

hx	dt	Error	ErrorRatio
0.1	0.001	0.00047956	1
0.05	0.00025	0.00076541	0.62654
0.025	6.25e-05	0.0010583	0.72326
0.0125	1.5625e-05	0.0011267	0.93929
0.00625	3.9063e-06	0.0011419	0.98672
0.003125	9.7656e-07	0.0011447	0.99748

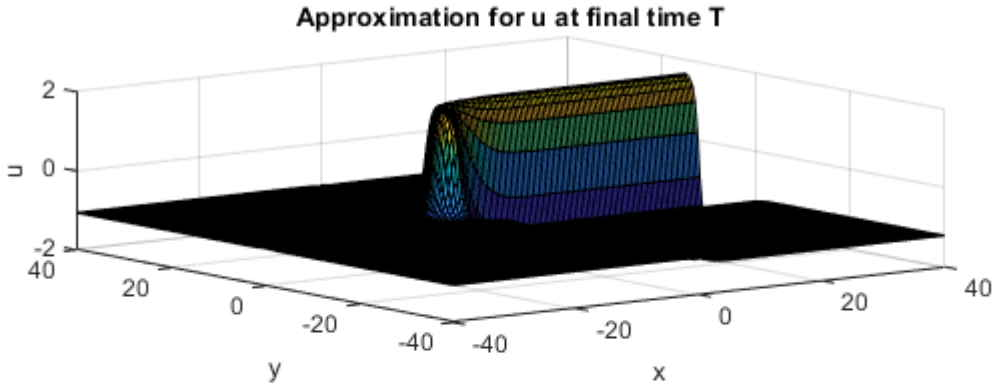
Question 4 File Checklist

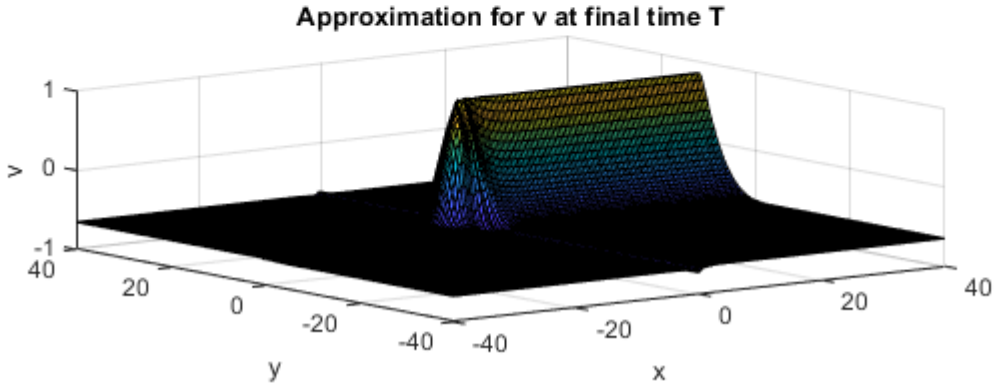
The files Coursework2Script.m and reaction_diffusion2d.m are used for this question.

Question 4a) T=10

This code section first defines the given initial conditions to be inputted into the reaction_diffusion2d.m function, then calls the function with final time T=10, hx=hy=0.5 and dt=0.05 to produce the surfaces for u and v. The next two code sections repeat this for final times T=50 and T=100.

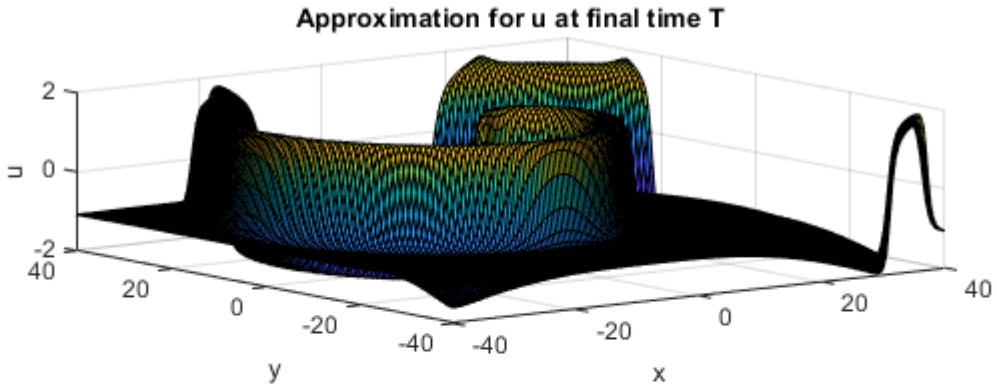
```
Nx=160; %for this problem we need to define the given initial condition outside of the
function
Ny=160;
x=linspace(-40,40,Nx+1);
y=linspace(-40,40,Ny+1);
[xgrid,ygrid]=meshgrid(x,y); %create the mesh
u0=zeros(Ny+1,Nx+1); %initialise u0 and v0 for the for loop below
v0=zeros(Ny+1,Nx+1);
for j=1:Nx+1 %this nested for loop assigns the correct initial conditions to u and v
    for k=1:Ny+1
        if ygrid(k,j)<0 %when y<0, assign -u*=1.08 to element
            u0(k,j)=1.08;
        else %when y>=0, assign u*=-1.08 to element
            u0(k,j)=-1.08;
        end
        if xgrid(k,j)<0 %do the same thing for the v0 mesh
            v0(k,j)=0.6601;
        else
            v0(k,j)=-0.6601;
        end
    end
end
[~,~]=reaction_diffusion2d(0.3,10,0.5,0.75,u0,v0,160,160,200,2,2,1);
```

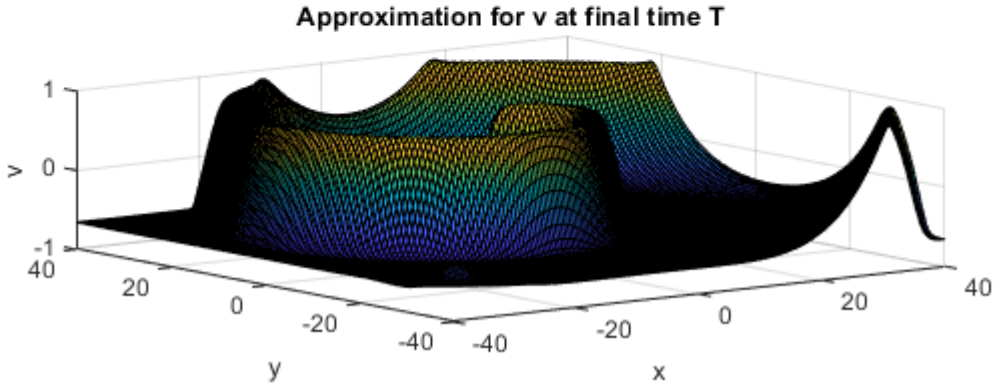




Question 4a) T=50

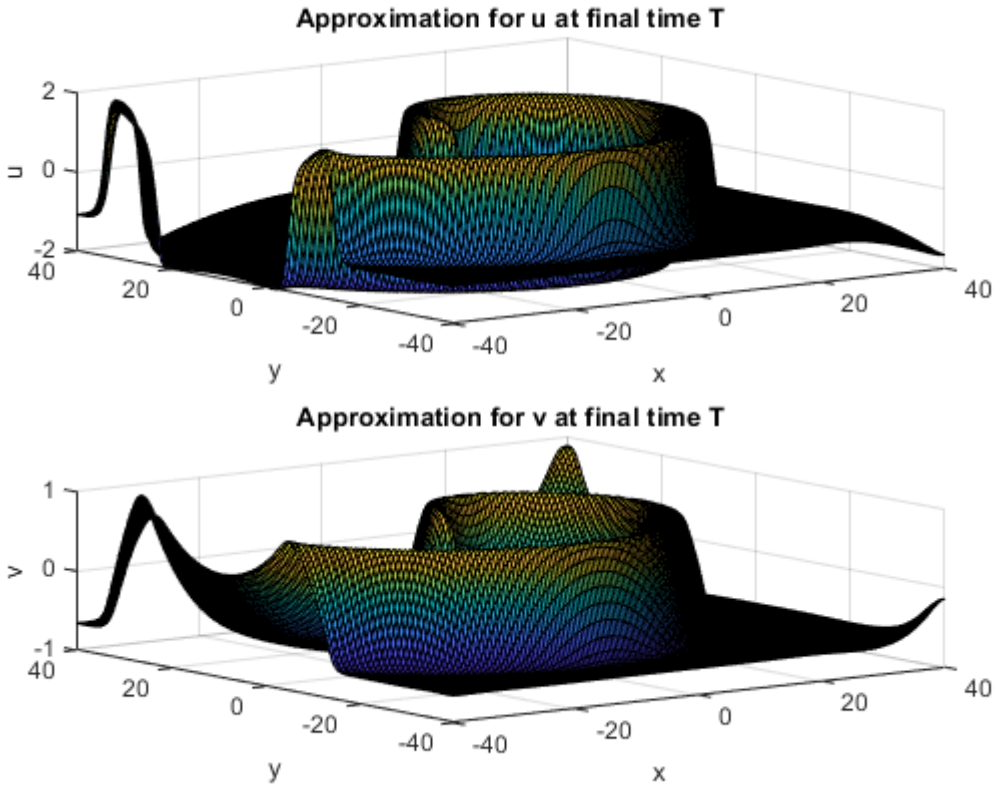
```
[~,~]=reaction_diffusion2d(0.3,50,0.5,0.75,u0,v0,160,160,1000,2,2,1);
```





Question 4a) T=100

```
[~,~]=reaction_diffusion2d(0.3,100,0.5,0.75,u0,v0,160,160,2000,2,2,1);
```

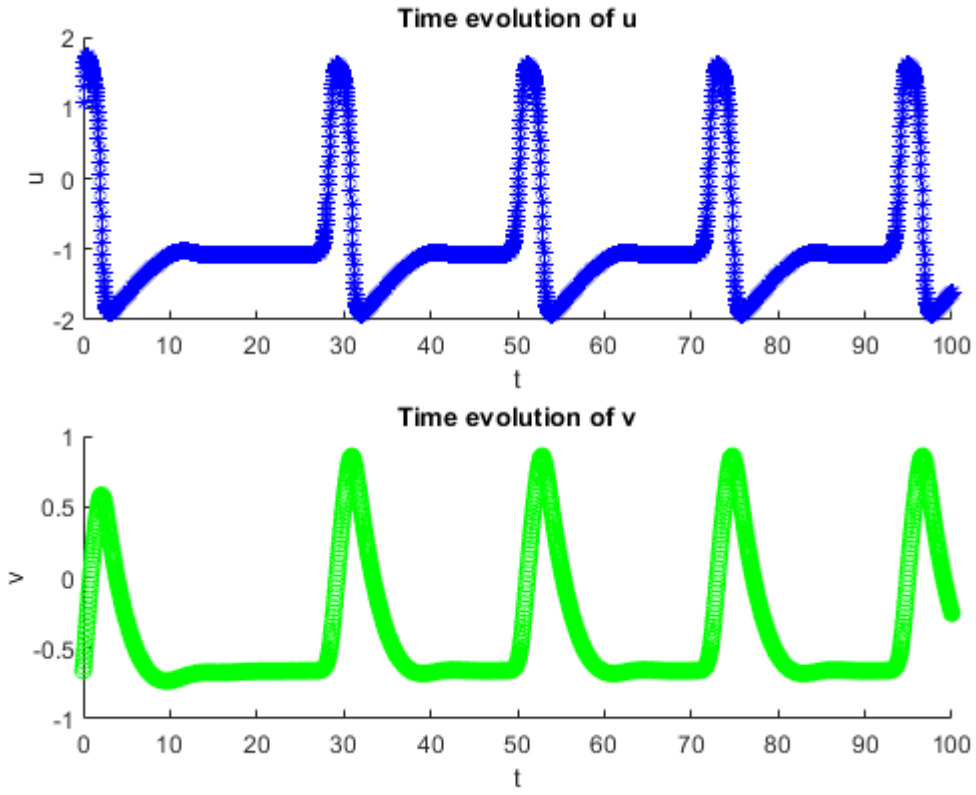


Question 4a) Time variation of (u,v) when (x,y) near (0,-20)

This code section chooses a spatial mesh point and produces a plot for u and v as time evolves from 0 to 100.

```
clear all %had to use this as plots from previous section were overlapping
clf
Nx=160; %for this problem we need to define the given initial condition outside of the
function
Ny=160;
x=linspace(-40,40,Nx+1);
y=linspace(-40,40,Ny+1);
[xgrid,ygrid]=meshgrid(x,y); %create the mesh
u0=zeros(Ny+1,Nx+1); %initialise u0 and v0 for the for loop below
v0=zeros(Ny+1,Nx+1);
for j=1:Nx+1 %this nested for loop assigns the correct initial conditions to u and v
    for k=1:Ny+1
        if ygrid(k,j)<0 %when y<0, assign -u*=1.08 to element
            u0(k,j)=1.08;
        else %when y>=0, assign u*=-1.08 to element
            u0(k,j)=-1.08;
        end
        if xgrid(k,j)<0 %do the same thing for the v0 mesh
            v0(k,j)=0.6601;
        else
```

```
            v0(k,j)=-0.6601;
        end
    end
end
jx=round((Nx+1)/2); %we look for x near to the middle of the interval
ky=round((Ny+1)/4); %we look for y near to -20 (a quarter of the way into the interval)
[~,~]=reaction_diffusion2d(0.3,100,0.5,0.75,u0,v0,160,160,2000,jx,ky,2); %t is required in
interval [0,100]
```



Question 4a) Description of time evolution

When T=10, the surfaces u and v have a single global maximum and minimum, where the maximum has a much larger magnitude. The amplitude of this peak is larger for the u-surface. Both peaks evolve from x approx. -7.5 and do so for x increasing beyond this point in the interval 0<y<20. For T=50,100 the surfaces spiral and there is an interesting peak at (40,-35) for the u-surface/(40,-28.5) for v-surface at T=50. At T=100, the surface appears to have been rotated/mirrored as the x/y-values at these maximums switch. With respect to the time evolution of (u,v) for (x,y) near (0,-20) in [0,100] for t, both u and v have repeated waves. For u, the amplitude of the waves remains constant throughout the interval and one full wave repetition takes about 30 time units. On the other hand, v has a lower peak at t=2.45 before repetitions of larger amplitude waves begin (this amplitude is smaller than that of u, but occur more often, around every 20 time units).

Question 4b) Change in computation time for larger scale problem

The x and y intervals have been scaled up by a factor 2.5, therefore to obtain the same value of h and thus the same accuracy as achieved in 4a) the number of x/y intervals must also be scaled up by this factor 2.5, giving Nx=Ny=400. The truncation error has two terms, one that is first order in dt and another that is second order in h. Therefore, to reduce the error by a factor of 100 we need 100x more time-steps and 10x more space-steps in each direction. The bandwidth M of the matrix

used at each step is approximately Nx*Ny (we are working in 3 dimensions including time) which after scaling back to the original pre-scaled Nx and Ny has increased by factor 2.5^2*100=625. For the NxN system solved, N is scaled up by considering the product of the transformed number of steps Nx*Ny*Nt, so we find that this has increased by factor 2.5^2*100*100=62500. If Gaussian elimination is used, the system requires O(NM^2) operations to solve, therefore an estimate for the factor NM^2 increases by is 625^2*62500=2.441x10^(10). As a result, it would take approximately 2.441x10^(10) times as long to compute the approximation. For each second the original problem took (in my case 11.3108 seconds), it takes 774.2 years to compute this new approximation, thats 8756 years altogether!

Question 4b) Consideration of theta=0.5 method to speed up computation

The theta=0.5 method was considered for the 1D diffusion equation in question 1, where we found that this method is second-order accurate in h. Analysis of the truncation error (done in lecture exercise) gives a second term of O(dt^2). For this part we also have a second-order explicit method for the reaction terms, adding another O(dt^2) term to the truncation error. To reduce the length of the computation time whilst maintaining the desired accuracy we could reduce the number of space or time steps as we now have second-order terms for dt and h as opposed to first-order terms. In this way, the N and the M involved in the computation time are reduced and so the overall computation time is reduced, and since dt<<1, we can keep the error that we wanted in part a). In terms of stability, we are given that the method for the reaction term has similar properties to that of the forward Euler method, so this adds another restriction on how large we can allow h and dt to be, along with the condition that the error remains about the same. Again, the term in dt^2 should make this possible to achieve and since Nt is smaller the N in the NM^2 term will also be smaller. Lastly, in the lecture notes it is noted that the matrix A in the Au^(n+1)=b system (which we are solving) is tridiagonal (bandwidth M=1), so using Gaussian Elimination only O(N) operations are required. This is a massive advantage if one requires the accuracy mentioned for this expanded problem.

Published with MATLAB® R2019a