Project 5

Sigrid Videm

January 24, 2018

1 Abstract

I have implemented three schemes for solving the one dimensional diffusion equation numerically, and I have compared the results to the analytical solution of the problem. With a spatial step length of 1/100 there was not much difference to be found by visual inspection. I calculated the relative errors, and I found that making the spatial step length smaller improved the quality of the solution, but reducing the length of the time steps did not. The strange behaviour of the error has bothered me a lot, but I have not found the reason for it. The error was the same for all three schemes, but I have seen that the explicit scheme has a strong stability requirement.

For the two dimensional diffusion equation I used the explicit scheme. I found a two dimensional problem which was easy to solve analytically, and used it to compare the numererical solution to the analytical one. Then I used the solver to solve a more complicated problem. The explicit scheme for the two dimensional problem seemed to have an even stronger stability requirement than the one dimensional one.

Contents

1 Abstract **1**

2 Introduction **2**

3 Methods **2**

3.1 The 1D diffusion equation 2

3.1.1 The analytical solution 3

3.1.2 The explicit scheme 4

3.1.3 The implicit scheme 5

3.1.4 Crank–Nicolson scheme 6

3.2 The 2D diffusion equation 7

3.2.1 An analytic solution 8

3.2.2 The explicit scheme 8

4 Results and Discussion **9**
4.1 The 1D diffusion equation 9
4.1.1 The instability of the explicit scheme 14
4.2 The 2D diffusion equation 14
4.2.1 Homogenous boundaries 14
4.2.2 Error and instability 19
4.2.3 Nonhomogenous boundaries 20

5 Conclusion **22**

2 Introduction

In this project I have used finite differences to approximate the derivatives to develop three different schemes for solving the diffusion equation.

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t} \tag{1}$$

I have used scaled variables - there are no pyhsical constants in my equation. Because of this, the solvers presented here, are suitable to solve any problem which can be modelled by equation 1. Examples are heat conduction problems and particle diffusion, but also the Schrödinger equation of a free particle is on the same form. Boundaries and initial conditions are presented in section 3.1, and in 3.1.1 is the analytical solution to the problem. The three schemes I'm going to use are The explicit scheme (section 3.1.2) which is easy to implement, but has a tedious stability requirement; the Implicit scheme (section 3.1.3) which needs to be solved as a matrix equation; and the Crank–Nicolson (section 3.1.4) scheme which is a combination of the two other schemes. Plots of the solutions and a discussion of the truncation error are in section 4.1.

I have found an analytical solution of the two dimensional diffusion equation, the solution is presented in section 3.2.1, and I have used the explicit scheme to solve it numerically. The explicit scheme is described in section 3.2.2, and the results are presented in section 4.2. My code is available at `https://github.com/sigrivi/Project5`

3 Methods

3.1 The 1D diffusion equation

I want to solve the diffusion equation:

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t} \qquad t > 0 \qquad x \in [0, L]$$

or

$$u_{xx} = u_t,$$

with initial conditions

$$u(x,0) = 0 \quad 0 < x < L$$

with $L = 1$ the length of the x-region of interest. The boundary conditions are

$$u(0,t) = 0 \quad t \geq 0,$$

and

$$u(L,t) = 1 \quad t \geq 0.$$

To do this, I am going to use three different schemes: The explicit scheme, The implicit scheme and the Crank-Nicolson scheme.

3.1.1 The analytical solution

With the boundaries $u(0,t) = 0$ and $u(L,t) = 1$, the system will reach a steady state as $t \to \infty$. The steady state function $v(x)$ must be a linear function, and satisfy the boundary conditions:

$$v(x) = \frac{x}{L}$$

The solution $u(x,t)$ can now be expressed as:

$$u(x,t) = v(x) + w(x,t)$$

$w(x,t)$ is also a solution of $u_t = u_{xx}$, but with different boundaries. The boundaries are:

$$w(0,t) = u(0,t) - v(0) = 0 - 0 = 0$$

$$w(L,t) = u(L,t) - v(0) = 1 - 1 = 0$$

The initial condition is

$$w(x,0) = u(x,0) - v(x) = 0 - v(x) = -\frac{x}{L} \quad 0 < x < L$$

To solve this system, I use the separation of variables technique.

$$w(x,t) = X(x)T(t)$$

$$w_t = T'X$$

$$w_{xx} = TX''$$

Inserted into the equation $w_t = w_{xx}$

$$TX'' = T'X \implies \frac{T'}{T} = \frac{X''}{X} = \text{constant}$$

Let the constant be $-\lambda^2$. We then have two separate equations:

$$T' + \lambda^2 T = 0 \implies T = e^{-\lambda^2 t}$$

$$X'' + \lambda^2 X = 0 \implies X = A \sin \lambda x + B \cos \lambda x$$

The boundary condition $w(0,t) = 0$ requires $B = 0$, while the boundary condition $w(L,t) = 1$ requires that λ is a multiple of π: $\lambda_k = k\pi/L$, where $k = 1, 2, 3, ...$ We now have the solution $w(x,t) = X(x)T(t)$

$$w(x,t) = \sum_k A_k \sin \frac{k\pi x}{L} \cdot e^{-k^2\pi^2 t/L^2}$$

I can find the coefficients A_k by using the initial condition $w(x,0) = -x/L$

$$w(x,0) = \sum_k A_k \sin \frac{k\pi x}{L} = \frac{-x}{L}$$

A_k are the Fourier coefficients (page 19 in the lecture notes [1])

$$A_k = \int_0^L \frac{-x}{L} \sin \frac{k\pi x}{L} dx = -\frac{2}{L}\left[\frac{1}{(k\pi)^2} \sin k\pi x - \frac{x}{k\pi} \cos k\pi x\right]_0^1 = \frac{2}{Lk\pi} \cos k\pi$$

The soulution $u(x,t) = w(x,t) + x/L$ of the original problem is

$$u(x,t) = \sum_k \frac{2}{Lk\pi} \cos k\pi \cdot \sin \frac{k\pi x}{L} \cdot e^{-k^2\pi^2 t/L^2} + \frac{x}{L} \tag{2}$$

In order to calculate this analytical solution, I made the function `exactsolution(x,t)`, which returns the solution `u` for a given time `t`. The vector `x` represents the x values. It was surprisingly hard for me to solve this problem, and I had a lot of help from Boyce and DiPrima [2].

3.1.2 The explicit scheme

The explicit scheme uses the Euler forward approximation for the time derivative:

$$\frac{\partial u}{\partial t} = \frac{u_{j+1} - u_j}{\Delta t} + O(\Delta t) \approx \frac{u_{j+1} - u_j}{\Delta t}$$

This approximation has an error of $O(\Delta t)$. For the second order spatial derivative, I use the central difference approximation:

$$\frac{\partial^2 u}{\partial^2 x} = \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2} + O((\Delta x)^2) \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2}$$

Which has an error of $O((\Delta x)^2)$. The diffusion equation can now be approximated with:

$$\frac{u_i^{l+1} - u_i^l}{\Delta t} \approx \frac{u_{i+1}^l - 2u_i^l + u_{i-1}^l}{(\Delta x)^2}$$

$$u_i^{l+1} = \frac{\Delta t}{(\Delta x)^2}(u_{i+1}^l - 2u_i^l + u_{i-1}^l) + u_i^l = \alpha(u_{i+1}^l - 2u_i^l + u_{i-1}^l) + u_i^l \quad (3)$$

The truncation error of the explicit scheme is of $O((\Delta x)^2)$ and $O(\Delta t)$. Equation 3 can be rewritten as a matrix–vector product.

$$\boldsymbol{u}^{l+1} = \begin{bmatrix} 1-2\alpha & \alpha & 0 & 0 \\ \alpha & 1-2\alpha & \alpha & 0 \\ & & \ddots & \\ 0 & 0 & \alpha & 1-2\alpha \end{bmatrix} \boldsymbol{u}^l$$

There is a danger of using the explicit scheme. In the lecture notes [1], the spectral radius of the matrix is used to show that the explicit scheme is stable only if $\Delta t/(\Delta x)^2 \leq 1/2$. The treacherousness of the explicit method is revealed in section 4.1.1.

The function `explicitsolver(Nt,tfinal,Nx,frequency)` is an implementation of the explicit scheme. It does `Nt` steps, each step is made by calling the function `forwardstep`:

```
def forwardstep(u, alpha):
    unew = u.copy()
    unew[0] = (1-2*alpha)*u[0] + alpha*(u[1])
    unew[1:Nx] = (1-2*alpha)*u[1:Nx] + alpha*(u[2:Nx+1]+u[0:Nx-1])

return(unew)
```

`forwardstep` returns the vector `unew`, which is the solution at the current time step. Every `Nt/frequency` solution is stored in a matrix `U`, which is returned from `explicitsolver`. My implementation of the explicit scheme is in `project5c.py`.

3.1.3 The implicit scheme

For the implcit scheme, I use the same approximation for the spatial derivative, but the Euler backward approximation for the time derivative.

$$\frac{\partial u}{\partial t} = \frac{u^l - u^{l-1}}{\Delta t} + O(\Delta t) \approx \frac{u^l - u^{l-1}}{\Delta t}$$

The approximation to the diffusion equation is

$$\frac{u_i^l - u_i^{l-1}}{\Delta t} \approx \frac{u_{i+1}^l - 2u_i^l + u_{i-1}^l}{(\Delta x)^2}$$

with $\alpha = \Delta t/(\Delta x)^2$:

$$u_i^l(1+2\alpha) - \alpha u_{i+1}^l - \alpha u_{i-1}^l = u_i^{l-1}$$

Now, the new solution is no longer determined explicitely by the previous one. I find the solution by converting the problem into a matrix equation:

$$\begin{bmatrix} 1+2\alpha & -\alpha & 0 & 0 \\ -\alpha & 1+2\alpha & -\alpha & 0 \\ & & \ddots & \\ 0 & 0 & -\alpha & 1+2\alpha \end{bmatrix} \boldsymbol{u}^l = \boldsymbol{u}^{l-1} \tag{4}$$

To solve this equation, I use Gaussian elimination to get an upper triangular matrix, and backsubstitution to find the solution. I have explained the algorithm in more detail in project 1 [3]. The functions `rowreduction` and `backsubstitution` does the job for me.

```
def rowreduction(A,u):
    for i in range(A.shape[0]-1): # Gaussian elimination
        coeff=A[i+1,i]/A[i,i]
        A[i+1,i+1]=A[i+1,i+1]-A[i,i+1]*coeff
        u[i+1]=u[i+1]-u[i]*coeff
```

and

```
def backsubstitution(A, u, alpha):
    Nx = A.shape[0]
    solution = u.copy()
    solution[Nx] = 1                          # boundary condition
    for i in range(1,Nx+1):                   #backward substitution
        k = Nx-i
        solution[k] = (u[k]+alpha*solution[k+1])/A[k,k]
    return(solution)
```

The matrix equation has to be solved for every time step. Therfore, the function `implictsolver(Nt,tfinal,Nx,frequency)` loops over `Nt`, where `Nt` is the number of time steps. `implictsolver` constructs the matrix `A` and calls `rowreduction` and `backsubstitution` for every time step. Every `Nt/frequency` solution is stored in a matrix `U`, which is returned from `implicitsolver`. The implicit scheme has the same truncation errors as those of the explicit, because the approximations to the derivatives are of the same order, but it is shown in the lecture notes [1] that this scheme is stable for all α.

3.1.4 Crank–Nicolson scheme

For the Crank–Nicolson scheme, the approximations of the derivatives are:

$$\frac{\partial u}{\partial t} \approx \frac{u_i^{l+1} - u_i^l}{\Delta t}$$

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{1}{2}\left(\frac{u_{i+1}^{l+1} - 2u_i^{l+1} + u_{i-1}^{l+1}}{\Delta x^2} + \frac{u_{i+1}^{l} - 2u_i^{l} + u_{i-1}^{l}}{\Delta x^2}\right)$$

with $\alpha = \frac{\Delta t}{2(\Delta x)^2}$:

$$u_i^{l+1} - u_i^l = \alpha(u_{i+1}^{l+1} - 2u_i^{l+1} + u_{i-1}^{l+1} + u_{i+1}^l - 2u_i^l + u_{i-1}^l)$$

reorganize so that all u^{l+1} terms are on the left side:

$$u_i^{l+1} - \alpha u_{i+1}^{l+1} + 2\alpha u_i^{l+1} - \alpha u_{i-1}^{l+1} = u_i^l + \alpha u_{i+1}^l - 2\alpha u_i^l + \alpha u_{i-1}^l$$

Rewrite as matrix vector products:

$$\begin{bmatrix} 1+2\alpha & -\alpha & 0 & 0 \\ -\alpha & 1+2\alpha & -\alpha & 0 \\ & & \ddots & \\ 0 & 0 & -\alpha & 1+2\alpha \end{bmatrix} \boldsymbol{u}^{l+1} = \begin{bmatrix} 1-2\alpha & \alpha & 0 & 0 \\ \alpha & 1-2\alpha & \alpha & 0 \\ & & \ddots & \\ 0 & 0 & \alpha & 1-2\alpha \end{bmatrix} \boldsymbol{u}^l$$

The right hand side is identical to the matrix vector product of the explicit method, while the left hand side is identical to the matrix vector product of the implicit method. The only difference is α. That means that I can use the forward step function to calcuate the rigth hand side, and then use use row reduction and backsubstitution as I did in the implicit scheme.

```
def cranknicolson(Nt,tfinal,Nx,frequency):

    alpha,u,U,counter = initialize(Nt,tfinal,Nx,frequency)
    beta = alpha/2
    B = makeA(Nx,beta)

    for t in range(Nt):
        u = forwardstep(u, beta)

        A = B.copy()
        rowreduction(A, u)

        u = backsubstitution(A, u, beta)

        if t%frequency == 0: # to store every frequency solution
            U[1:,counter] = u
            counter += 1
    return(U)
```

In the lecture notes, it is shown that the error of the Crank Nicolson scheme is $O(h^2)$ and $O(\Delta t)^2$, and that the Crank Nicolson scheme is stable for all α

3.2 The 2D diffusion equation

In two dimensions, the diffusion equation look like this:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

3.2.1 An analytic solution

I learned a lot of things from the 1 dimensional case. 1) Separation of variables is a good technique 2) the solution will be easier to find if the boundaries are zero and if 3) the initial condition is a sine function. When I am free to choose the boundaries as I like, I do it the simple way, and I will use $L = 1$ as the length of the system in both x and y direction. The boundaries I choose are:

$$u(0, y, t) = u(1, y, t) = u(x, 0, t) = u(x, 1, t) = 0$$

And the initial condition: $u(x, y, 0) = \sin \pi x \sin \pi y$

I can write $u(x, y, t) = X(x)Y(y)T(t)$, and I find $T = e^{-\lambda^2 t}$ like i did in the 1D case. Nothing stops me from separating the variables again:

$$\frac{X''}{X} = -\frac{Y''}{Y} - \lambda^2 = -\kappa^2$$

This gives an equation for X which is similar to the 1D

$$X(x) = A_x \sin \kappa x + B_x \cos \kappa x$$

The boundary conditions require $B_x = 0$, and κ must be a muliple of π. The initial condition(which indeed was wisely choosen) gives $A_k = 1$ and $\kappa = \pi$. For Y, we have

$$\frac{Y''}{Y} = \kappa^2 - \lambda^2$$

With $\kappa^2 - \lambda^2 = -\eta^2$

$$Y = A_y \sin \eta y + B_y \cos \eta y$$

Again, boundary conditions require $B_y = 0$, and η a multiple of π. Initial conditions give $A_y = 1$ $\eta = \pi$. When κ and η are determined, I can determine $\lambda^2 = \eta^2 + \kappa^2 = 2\pi^2$. The solution is

$$u(x, y, t) = e^{-2\pi^2 t} \sin \pi x \sin \pi y \tag{5}$$

3.2.2 The explicit scheme

The explicit scheme for the 2 dimensional diffusion equation uses the Euler forward approximation for the time derivative, and the central difference approximation for the spatial derivative:

$$\frac{u_{i,j}^{l+1} - u_{i,j}^{l}}{\Delta t} \approx \frac{u_{i+1,j}^{l} - 2u_{i,j}^{l} + u_{i-1,j}^{l}}{(\Delta x)^2} + \frac{u_{i,j+1}^{l} - 2u_{i,j}^{l} + u_{i,j-1}^{l}}{(\Delta y)^2}$$

With $\Delta x = \Delta y = h$, I can rewrite the equation so that i get $u_{i,j}^{l+1}$ alone on the left hand side:

$$u_{i,j}^{l+1} = \alpha(u_{i+1,j}^{l} + u_{i-1,j}^{l} + u_{i,j+1}^{l} + u_{i,j-1}^{l} - 4u_{i,j}^{l}) + u_{i,j}^{l}$$

To solve this, I made the function `explicitsolver2D(Nt,tfinal,Nx,frequency,n)`. The `explicitsolver2D` makes a matrix `U` of size `NX+1` and the vector `axis` which concist of evenly distributed numbers between 0 and 1. The elements of `U` are

```
x0 = np.sin(axis *np.pi)
np.outer(x0, x0, U)
```

For every time step, the elements of `U` are used to update `U`:

```
for l in range(Nt):

    U[1:Nx,1:Nx] = alpha*(U[2:Nx+1,1:Nx] + U[0:Nx-1,1:Nx]
+ U[1:Nx,2:Nx+1] + U[1:Nx,0:Nx-1] - 4*U[1:Nx,1:Nx]) + U[1:Nx,1:Nx]
```

The function returns one matrix, namely solution number `frequency * n`, but it can easily be modified so that it returns other solutions as well. I also included a plot function which plots a cross section for different times. A contour plot at $t = 0.02$ and the cross section plot are in section 4.2.1. Because I cannot come up with a plausible physical system that is described by these boundary and initial conditions, I used the explicit solver to solve the equation for different conditions:

$$u(0, y, t) = u(x, 0, t) = 0 \wedge u(1, y, t) = u(x, 1, t) = 1$$

$$u(x, y, 0) = 0, 0 < x < 1, 0 < y < 1$$

These conditions could describe a block being in contact with a heat reservoir at high temperature at tweo sides, and low temperature at the other two sides. The initial temperature of the block is the same a that of the low temperature reservoir. Plots illustrating the evolving system are in section 4.2.3. The code for the explicit scheme is in `project5f.py`

4 Results and Discussion

4.1 The 1D diffusion equation

The plots of figure 1, 2, 3 and 4 were made with the function `plotsolutions(ue,ui,uc, t`. Figure 1 shows the solutions from the three solvers together with the analytical solution at time $t = 0.02$. u is close to zero for a large part of the system, but the system has evolved from the discontinous initial state. The plot of figure 2 shows the same solution at $t = 0.2$. At this time, the graphs are almost linear, the system is close to the stady state. For both plots, I used only 10 grid points for x. Still I got graphs that look similar to the exact one, but they all lie below the exact, and the difference is largerst in the middle. Then, I increased the number of grid point by a factor 10. The resulting plots are in figure 3 and 4. Here it is difficult to distinguish the graphs by mere inspection. To look closer at the error, I calculated the error for varying Δx (table 1), and varying Δt (table 2).

Table 1 shows how the relative error decreases with Δx. The error of the three schemes are of same magnitude, and the first leading digit is equal. According to the theory, decreasing Δx should have the same effect on all three schemes. However, the error does not decrease as $(\Delta x)^2$. This indicates that

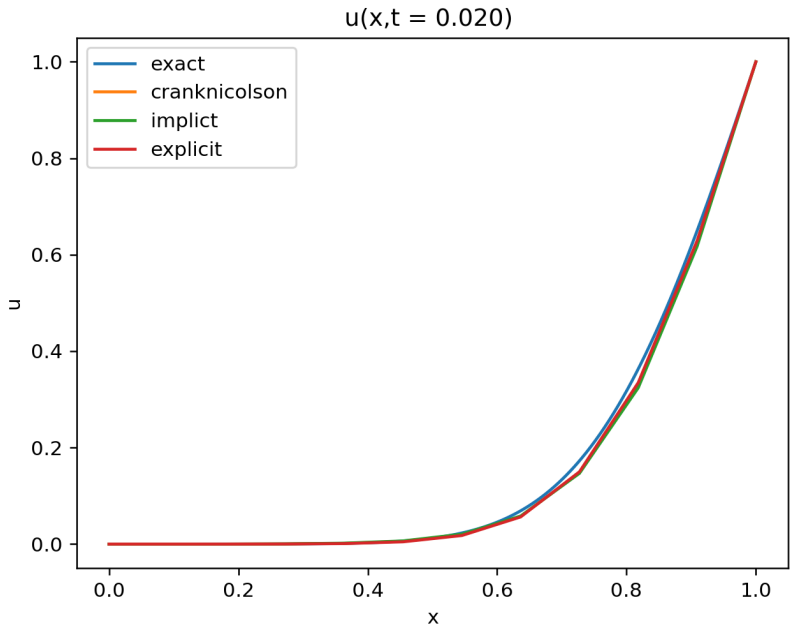

Figure 1: The exact solution plotted together with the numerical ones. Nx = 10, Nt = 1000. The numerical solutions lie close to each other, and below the exact one

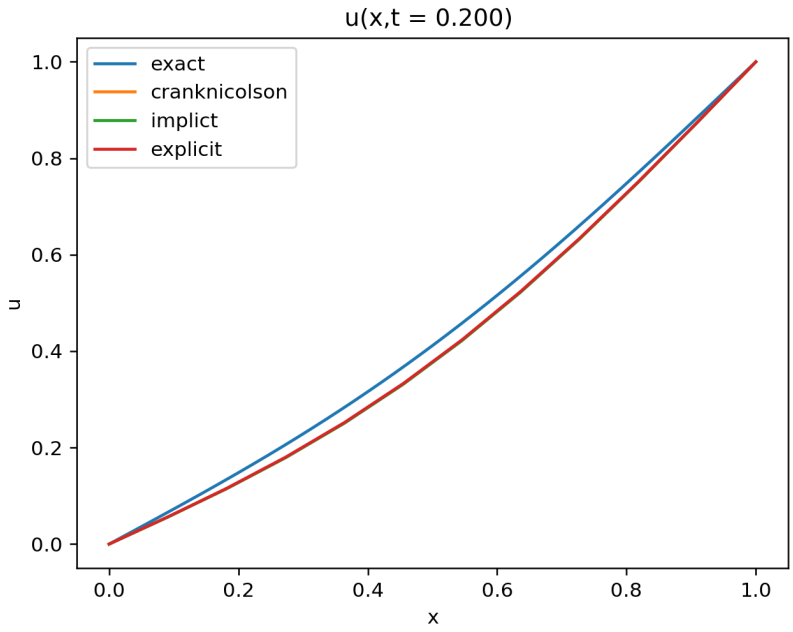

Figure 2: The exact solution plotted together with the nume)rical ones. Nx = 10, Nt = 1000.The numerical solutions are very similar, but differs from the exact one.

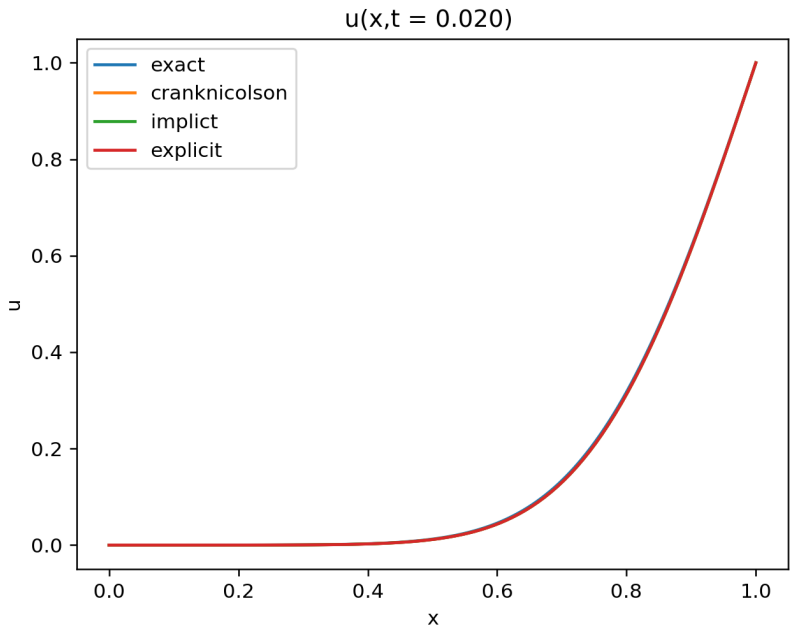

Figure 3: The exact solution plottet together with the numerical ones. Nx = 100, Nt = 100000. The solutions seems equal to the exact.

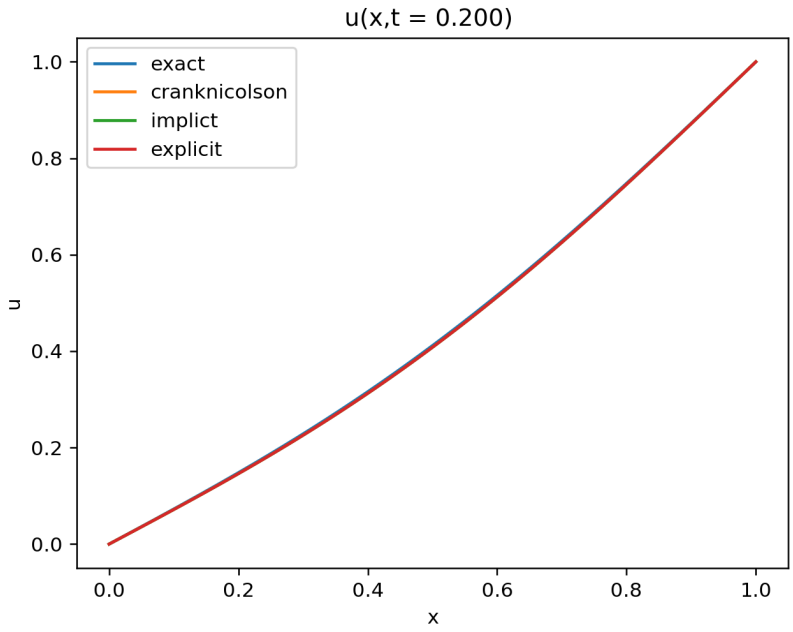

Figure 4: The exact solution plotted together with the numerical ones. Nx = 100, Nt = 100000. The solutions seems equal to the exact, and the graph is almost linear.

Table 1: Relative error of the three schemes with varying Δx. $t = 0.02$ and $\Delta t = 1/2000000$.

	$\Delta x = 1/10$	$\Delta x = 1/100$	$\Delta x = 1/1000$
explicit	0.0547113714079	0.00745993979441	0.000760541674696
implicit	0.0547168789102	0.00746703401772	0.000767790391857
Cranck Nicolson	0.0547141251303	0.00746348636423	0.000764152600736

Table 2: Relative error of the three schemes with varying Δ. $t = 0.02$ and $\Delta x = 1/100$.

	$\Delta t = 1/20000$	$\Delta t = 1/200000$	$\Delta t = 1/2000000$
explicit	0.00620196835303	0.00734337585799	0.00745993979441
implicit	0.0068930615448	0.00741431787145	0.00746703401772
Cranck Nicolson	0.00653247845369	0.00737879226342	0.00746348636423

there is someting wrong with either the way I calulate the error, or with the way I have implemented the schemes. The errors of table 2 are even more disappointing. Increasing the number of time steps don't improve the error at all. What is more, the Crank Nicolson scheme is not doing better than the others. The latter leads me to think that there is something seriously wrong with how I have implemented the schemes, and that the big issue is the loop over `Nt` .

4.1.1 The instability of the explicit scheme

The stability requirement of the explicit scheme is

$$\alpha = \frac{\Delta t}{(\Delta x)^2} = \frac{Nx^2}{Nt} < \frac{1}{2}$$

In figure 5, $\alpha = 1/2$, and the system is stable, also for the explicit solver. In figure 6, $\alpha = 2/3$, we see that the solutions from the Crank–Nicolson and the implicit solver are stable, while the solution of the explicit solver is not stable. I tried with even smaller Nt and kept Nx constant, and the smaller Nt got, the more bouncy was the solution. The explicit scheme was easy to implement, and it runs quickly for small `Nx`, but if you want high spatial resulution, you will soon discover the stability requirement is hard to meet, and that you are better off spending your time implementing the implicit scheme, rather than waiting for the explicit solver.

4.2 The 2D diffusion equation

4.2.1 Homogenous boundaries

Because I found solving PDE's analytically rather tiresome, I chose the simplest initial and boundary conditions I could come up with. I am going to use the solution of section 3.2.1 to make sure that my 2D solver works as I expect. In

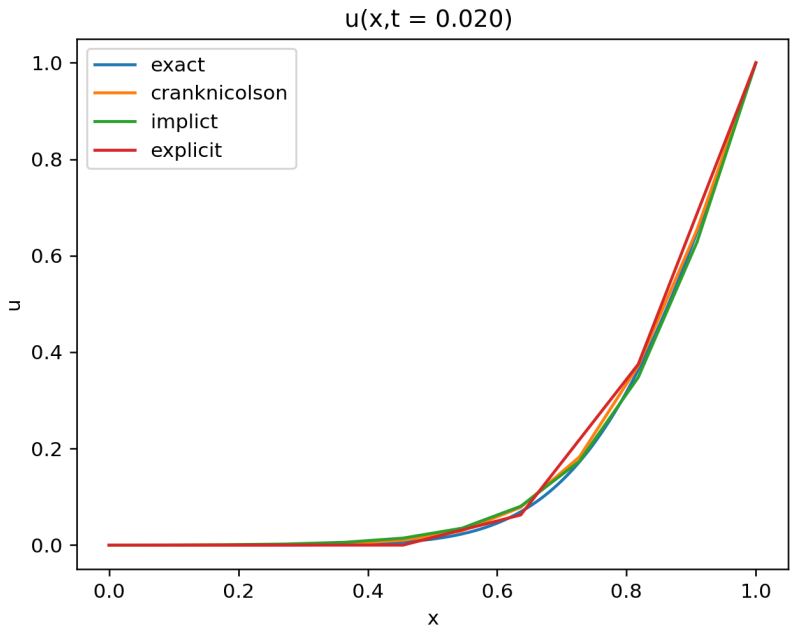

Figure 5: The exact solution plotted together with the numerical ones $\alpha = 1/2$. Nx = 10, Nt = 200. The solution of the explicit solver is still stable.

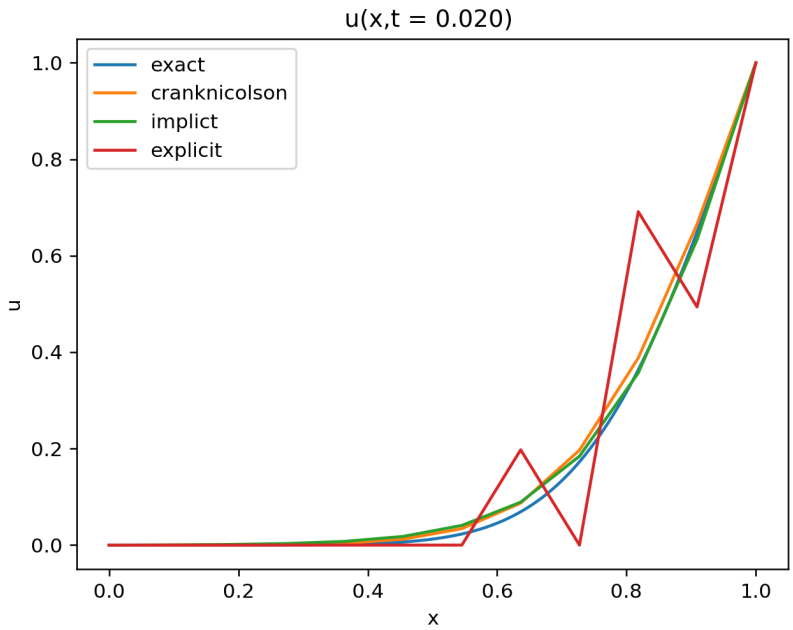

Figure 6: The exact solution plotted together with the numerical ones $\alpha = 2/3$. Nx = 10, Nt = 150. The solution of the explicit solver has startet to go crazy.

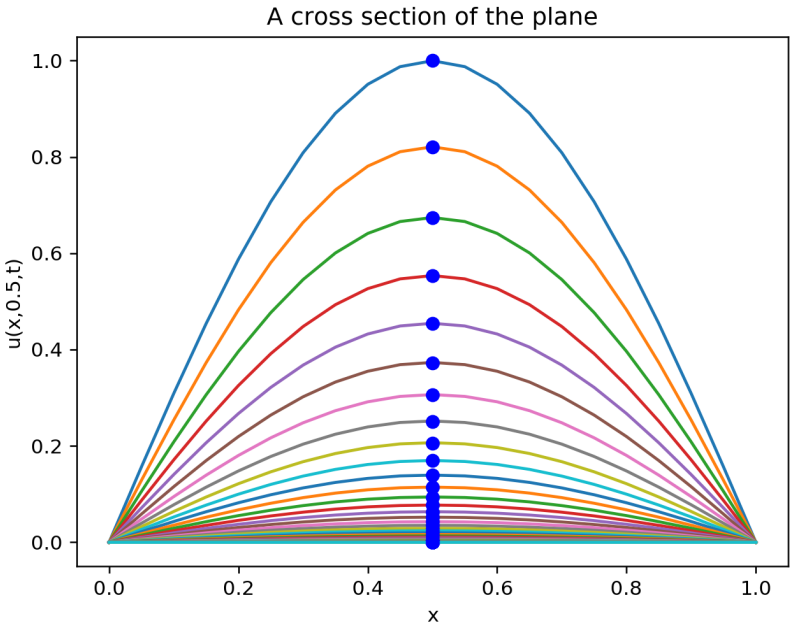

Figure 7: A cross section parallell with the x axis. u is plotted for $t \in [0, 0.01, 0.02, ..., 1]$. Nx = Ny = 20, Nt = 100000. The solution has the shape of a sine curve. Blue dots are the theoretical values of $u(0.5, 0.5, t)$.

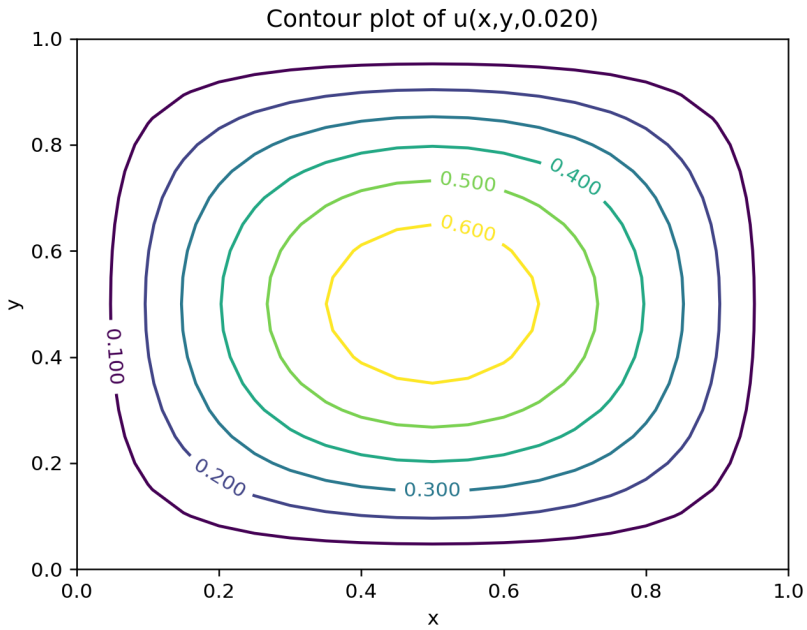

Figure 8: A contour plot made at t=0.02. The curve has two mirror planes, $x = 0.5$ and $y = 0.5$. Nx =Ny = 20, Nt = 100000.

order to see that the time dependency behaves as I expect, I plotted a cross section of the xy plane, parallell to the x axis. Figure 7 shows that $u(x, 0.5, t)$ has the shape of $\sin \pi x$, and that the amplitude gets smaller for larger t. u is plotted for $t \in [0, 0.01, 0.02, ..., 1]$. According to the analytical solution, the amplitude should be $e^{-2\pi^2 t}$. For the first 10 solutions, the amplitudes are

$$[0.820, 0.673, 0.553, 0.454, 0.372, 0.305, 0.251, 0.206, 0.169, 0.138]$$

The blue dots in figure 7 are the analyticl values.

Figure 8 shows $u(x, y, 0.02)$ which corresponds to the green line in 7. The curve has two mirror planes, $x = 0.5$ and $y = 0.5$. It has its highest point in the middle, and approaches zero in the edges. I feel confident that the numerical solution I find is the correct one, because it has the same behaviour as the analytical solution.

4.2.2 Error and instability

I did en attempt on error calculation. I had a small hope that calculating the local error of the first step (instead of the global error after many steps in time) would give errors that were more in agreement with the theoretical errors. I used Nt= 10000 and Nx =10,20, 30, 40, and calculated the error of the very first solution, that is, at time 0.0001. The expression I used for the error is

$$\frac{||e^{-2\pi^2 0.0001} \sin \pi x - u_{numerical}||}{||e^{-2\pi^2 0.0001} \sin \pi x||}$$

The results were discouraging.

Nx = 10 gave error 0.00194350451375

Nx = 20 gave error 0.00196775702548

Nx = 30 gave error 0.00197226135521

Nx = 40 gave error 0.00197383884111

In other words, the error did not change at all, even though the quality of the plots got better. It is clear that I have misunderstood something very important when it comes to error calculation, and because of the disappointment when it comes to the error's dependency on Δx, I did not feel the urge to go on and find the dependeny on Δt .

With Nt= 10000, the system was stable for Nx = 50, but for Nx=60 it was absolutly out of control. Compared to the stability criterion of the 1D explicit scheme

$$\Delta t/(\Delta x)^2 = Nx^2/Nt = 60^2/10000 = 0.36 < 0.5$$

I can say that there must be a stronger stability criterion of the explicit shceme for the 2D problem.

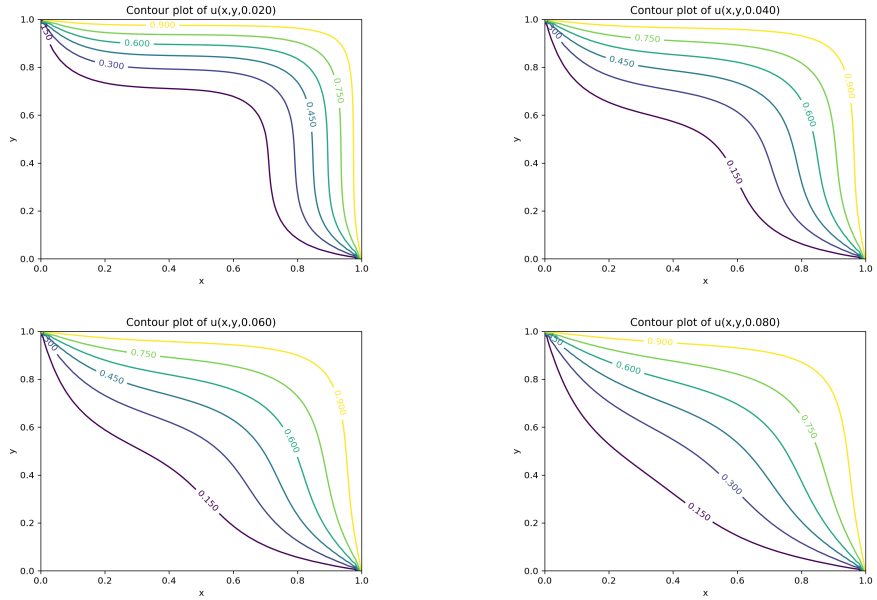

Figure 9: The contour plots shows the development of u. u is plotted for $t = 0.02, 0.04, 0.06, 0.08$.Nx = Ny = 50, Nt = 100000

4.2.3 Nonhomogenous boundaries

Even though I have a hard time evaluating the quality of my results, the solution I have found using my explicit solver resembles the analytical one. I am brave enough to think that the solver does its job, so I tried it on an other system, mentioned in section 3.2.2. The pysical problem in focus could be heat conduction in a block, where two sides of the block is in contact with a reservoir with high temperature, and the other two sides in contact with a reservoir of low temperature .Figure 9 shows the progression of u towards steady state. Figure 10 shows the system in equilibrium. The solution has two mirror planes, one along the $(0, 0)(1, 1)$ diagonal, and one along the $(0, 1)(1, 0)$ diagonal. The first reflects the symmetry of the boundary conditions, the other tells me that that I can switch the x and y values and still get the same result. In the corners, the plot of the soluton is not smooth. This is an artefact of the contour plot function in python. Instead of cutting off the contour lines at the point where no values are calculated, the contoul lines are all forced to continue to the edge of the plot.

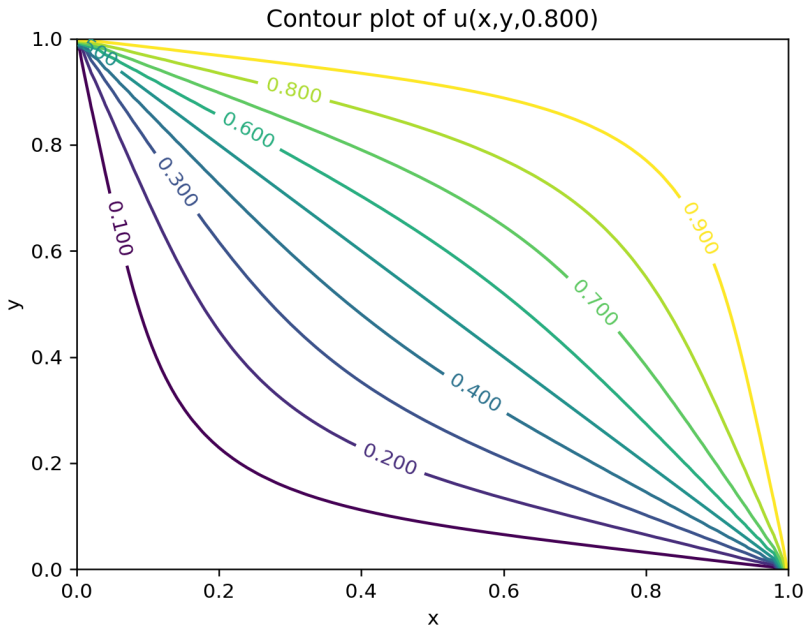

Figure 10: A contour plot made at t=0.8. Nx = Ny = 50, Nt = 100000. The system has reached equilibrium

5 Conclusion

The explicit scheme, the implicit scheme and the Cranck–Nicolson scheme gave the same quality of the solution as long as the stability requirement of the explicit scheme were met. For high spatial resolution, the explicit scheme require incredibly many time steps, which makes it unsuitable for problems where the quality of the solution is important. The implicit scheme does not have stability requirements, but is still quite easy to implement, as is is the solution of a matrix equation with a tridiagonal matrix. My implementation of the Crank–Nicolson scheme did not prove to have smaller error than the others, for ths reason I would prefer the implicit scheme. The fact that the error did not behave as expected – it did not decrease with smaller time steps, is a serous flaw of my project, and it is something I definitely have to investigate further, because I need to be able to make reliable error analysis in the future e.g. in my masters thesis.

For two dimensions, I have only implemented the explicit scheme. It made me capable of finding solutions of the 2D diffusion equation that are boring or difficult to find analytically. Furter work with the diffusion equation would involve the implementation of the 2D implicit scheme, to get high resolution without extremely many time steps.

References

[1] M. Hjort-Jensen: Computational Physics: Partial Differential Equations,
`http://compphysics.github.io/ComputationalPhysics/doc/pub/pde/pdf/pde-print.pdf`

[2] R. C. DiPrima and W. E. Boyce: Elementary Differential Equations and Boundary Value Problems,
`http://www.labma.ufrj.br/ mcabral/cursos/2017-1/Boyce-cap10.5.pdf`

[3] S. Videm: Computational Physics: Project 1,
`https://github.com/sigrivi/Project1`