FYS4150: Project 4

Mikael Toresen

November 11, 2013

1 Setting up the differential equation

1.1 Finding a closed form solution

The equation to be solved in this project is the diffusion equation in one dimension:

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t}, \quad x \in [0,1],\, t > 0, \quad u(0,t) = 1,\, u(1,t) = 0,\, u(x,0) = 0$$

This can be solved by assuming separation of variables on a similar equation v with Dirichelet boundary conditions:

$$\begin{aligned}
v(x,t) &= X(x)T(t) \\
\frac{d^2X(x)}{dx^2}T(t) &= X(x)\frac{\partial T(t)}{dt} \\
\frac{X''}{X} &= \frac{T'}{T} = -\lambda^2 \\
X(x) &= A\sin(\lambda x) + B\cos(\lambda x) \\
T(t) &= e^{-\lambda^2 t} \\
X(0) &= B = 0 \\
X(1) &= A\sin(\lambda) = 0 \Rightarrow \\
\lambda &= n\pi \\
v(x,t) &= \sum_n A_n \sin(n\pi) e^{-\lambda^2 t}
\end{aligned}$$

where $n \in \mathcal{N}$ as the negative numbers do not give us any more solutions. To adjust this solution to u we add a steady-state term $u_s = 1 - x$ so that the boundary conditions are fulfilled.

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

The initial condition $u(t=0) = 0$ then gives us:

$$x - 1 = \sum_n A_n \sin(n\pi)$$

By solving this using fourier series we get:

$$A_n = 2\int_0^1 (x-1)\sin(n\pi x)\,dx = -\frac{2}{n\pi}$$

Where n is not to be confused with the index used for time. A_0 is not interesting as this would give a completely static solution: $A_0 = 0 \rightarrow u(x,t) = 0$

So finally we arrive at:

$$u(x,t) = 1 - x - \sum_{n=1}^{\infty} \frac{2}{n\pi}\sin(n\pi x) e^{-n^2\pi^2 t}$$

1.2 Solving the discrete equation

We will in this project solve the diffusion equation using three different schemes: Forward Euler, Backward Euler and Crank-Nicolson. To do this we first discretize the domain into a grid with $N_x + 2$ uniformly distributed points, $x_i = ih_x$, $i \in [0, N_x]$, and $N_t + 2$ uniformly distributed points in time, where $t_n = ih_t$, $n \in [0, N_t]$.

1.2.1 Forward Euler

In Forward Euler(FE), the explicit Euler scheme, we arrive at the new timestep by using the slope from the current timestep:

$$\begin{aligned} u_t &= \frac{u_{i,n+1} - u_{i,n}}{h_t} \\ u_{xx} &= \frac{u_{i+1,n} - 2u_{i,n} + u_{i-1,n}}{h_x^2} \\ u_{xx} &= u_t \\ \frac{u_{i+1,n} - 2u_{i,n} + u_{i-1,n}}{h_x^2} &= \frac{u_{i,n+1} - u_{i,n}}{h_t} \\ u_{n,i+1} &= \alpha \left(u_{i-1,n} - 2u_{i,n} + u_{i+1,n} \right) + u_{i,n} \end{aligned}$$

where $\alpha \equiv \frac{h_t}{h_x^2}$. This corresponds to having a matrix-vector multiplication $\mathbf{u}_{n+1} = A\mathbf{u}_n$ to find the new timestep. Here $\mathbf{u}_n$ are the values at each point at time n, $\mathbf{u}_{n+1}$ are the points at the new timestep and A is a tridiagonal matrix with the upper and lower diagonal equal to α, and the diagonal is $1 - 2\alpha$. This can be solved trivially (explicitly) by performing a set of matrix-vector multiplications.

The algorithm used to solve this would be as follows:

```
Ucurr=0
while time<time_end:
 Unew=A*Ucurr
 Ucurr=Unew
end while
```

Here I note that this can be done more efficiently by diagonalizing A using some optimized gauss elimination on a vector similar to what was done in Project 1. The same goes for the schemes below.

1.2.2 Backward Euler

In the Backward Euler(BE), the implicit Euler scheme, we arrive at the new timestep by using the slope from the previous timestep:

$$\begin{aligned} u_t &= \frac{u_{i,n} - u_{i,n-1}}{h_t} \\ u_{xx} &= \frac{u_{i+1,n} - 2u_{i,n} + u_{i-1,n}}{h_x^2} \\ u_{xx} &= u_t \\ \frac{u_{i+1,n} - 2u_{i,n} + u_{i-1,n}}{h_x^2} &= \frac{u_{i,n} - u_{i,n-1}}{h_t} \\ u_{i,n-1} &= \alpha \left(-u_{i-1,n} + 2u_{i,n} - u_{i+1,n} \right) + u_{i,n} \end{aligned}$$

This corresponds to having a matrix-vector multiplication $\mathbf{u}_{n-1} = A\mathbf{u}_n$ to find the new timestep. Here A is a tridiagonal matrix with the upper and lower diagonal equal to $-\alpha$, and the diagonal

is $1 + 2\alpha$. As α is always positive and non-zero we know that we can invert A to find that $\mathbf{u}_n = A^{-1}\mathbf{u}_{n-1}$, which is then simple to solve.

The algorithm used to solve this would be as follows:

```
Ucurr=0
Ainv=inverse(A)
while time<time_end:
 Unew=Ainv*Ucurr
 Ucurr=Unew
end while
```

1.2.3 Crank-Nicolson

In the Crank-Nicolson(CN) scheme we arrive at the new timestep by using a weighted average of the current and previous timestep:

$$
\begin{aligned}
u_t &= \frac{u_{i,n+1} - u_{i,n}}{h_t} \\
u_{xx} &= \frac{1}{2}\left(\frac{u_{i+1,n} - 2u_{i,n} + u_{i-1,n}}{h_x^2} + \frac{u_{i+1,n+1} - 2u_{i,n+1} + u_{i-1,n+1}}{h_x^2}\right) \\
u_t &= u_{xx} \\
\frac{u_{i,n+1} - u_{i,n}}{h_t} &= \frac{1}{2}\left(\frac{u_{i+1,n} - 2u_{i,n} + u_{i-1,n}}{h_x^2} + \frac{u_{i+1,n+1} - 2u_{i,n+1} + u_{i-1,n+1}}{h_x^2}\right) \\
u_{i,n+1} - \frac{\alpha}{2}\left(u_{i+1,n+1} - 2u_{i,n+1} + u_{i-1,n+1}\right) &= u_{i,n} + \frac{\alpha}{2}\left(u_{i+1,n} - 2u_{i,n} + u_{i-1,n}\right) \\
\Big[2u_i + \alpha(-u_{i+1} + 2u_i - u_{i-1})\Big]_{n+1} &= \Big[2u_i - \alpha(-u_{i+1} + 2u_i - u_{i-1})\Big]_n
\end{aligned}
$$

Here the square brackets show that its content is evaluated at the timestep given in subscript. This equation corresponds to having the matrix-vector equation $(2I + \alpha B)\mathbf{u}_{j+1} = (2I - \alpha B)\mathbf{u}_j$. Here I is the identity matrix and B is a tridiagonal matrix with the upper and lower diagonal equal to -1, and the diagonal is 2. As α is always positive and non-zero, we know that we can invert $(2I + \alpha B)$ to find that $\mathbf{u}_n = (2I + \alpha B)^{-1}(2I - \alpha B)\mathbf{u}_{n-1}$, which can be solved.

The algorithm used to solve this would be as follows:

```
Ucurr=0
Mat1=2I-B
Mat2inv=inverse(Mat1+2B)
Matrix=Mat1*Mat2inv
while time<time_end:
 Unew=Matrix*Ucurr
 Ucurr=Unew
end while
```

2 Results

2.1 Finding the truncation errors

The general truncation errors for the schemes are shown in Table 1. We therefore expect FE and BE to have roughly the same error magnitude. When checking the errors, however I got mixed results as can be seen in Table 2. For few timesteps BE seemed most sucessful, while for large timesteps CN seemed better. In my implementation however the first grid point(s) would go wrong by many orders of magnitude if i choose a too big grid. This is most likely related to some loss

of precision due to α being so large. Also the explicit part of the CN scheme produces values which are both very close to zero and very much greather than zero (ie. for one set of 100 points, I got values like -0.00445392, 0.00702773, 188.304, -194.426) in `vtmp`) and as my method does not involve pivoting, there could easily be some computations which lead to loss of prescision. FE works very well for $\alpha \leq 0.5$, but once one reaches higher values the scheme is not stable as can be seen in Table 2. This is because FE projects its current slope onto the next step. This will lead to it having too large values for non-linear functions. This cannot be corrected for if the timesteps are too large compared to the spatial steps. The BE scheme, on the other hand, will have lower values than the exact for non-linear functions. CN, which is a centered scheme does not have an intrinsic bias to higher or lower values, and will generally, as can be seen from Table 1, give a more correct answer.

Scheme:	Truncation error:
FE	$\mathcal{O}(h_x^2)$ and $\mathcal{O}(h_t)$
BE	$\mathcal{O}(h_x^2)$ and $\mathcal{O}(h_t)$
CN	$\mathcal{O}(h_x^2)$ and $\mathcal{O}(h_t^2)$

Table 1: Truncation errors for the different schemes

h_x	N_t	FE	BE	CN
0.1	10	-1.13	-1.51	-1.36
0.1	100	-1.83	-0.99	-1.72
0.04	10	0.31	-0.49	-1.52
0.04	100	15.5	-0.38	-0.99

Table 2: Relative errors for the different schemes after a given time using $h_t = 0.001$ given in log-scale. This corresponds to having $\alpha = (0.1 \,|\, 0.625)$

2.2 Figures

3 Suggestions/comments

This project was in essence simple, except for the fine details of getting the correct amount of iteration. I would like, however, to see a bit more explicitly that we should pursue the Dirichelet equation v instead of u. This is hinted to in a subtle way in the last sentence in *a)*. There is however nothing computationally difficult about solving u using FE.

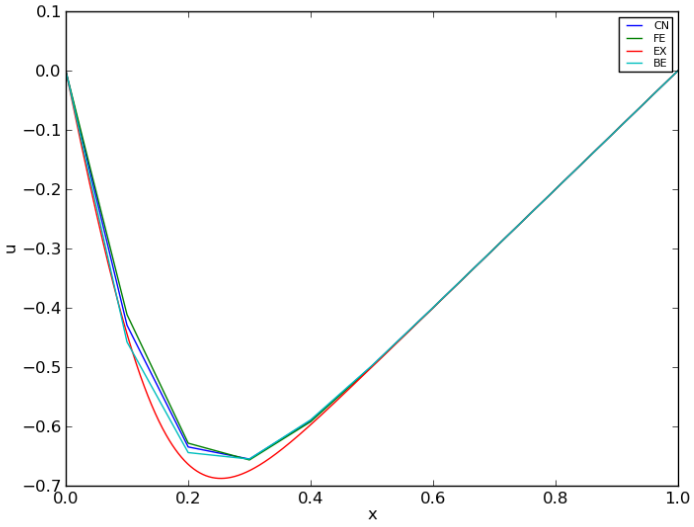

Figure 1: Plot of solutions using different schemes where EX is the closed form expression for the solution where the sum to infinity is approximated by a sum over the first 2000 values. This is given at $t = 10 * 0.01 = 0.1$ using 10 points in the grid.

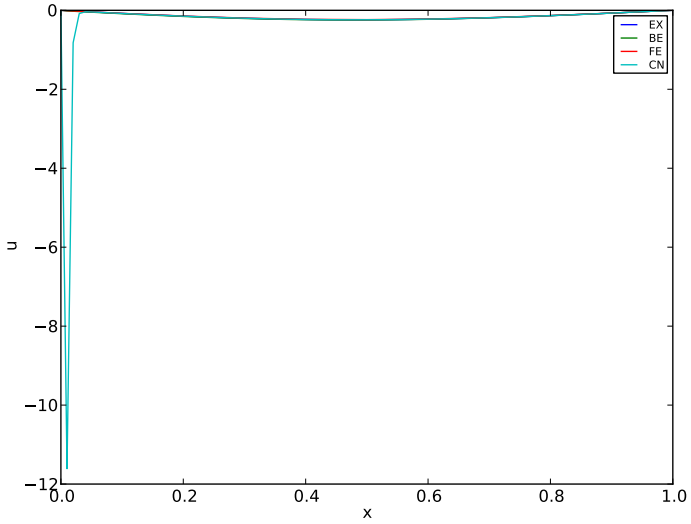

Figure 2: Plot of solutions using different schemes where EX is the closed form expression for the solution where the sum to infinity is approximated by a sum over the first 2000 values. This is given at $t = 1.0$ using 100 points in the grid. We see here that the solution goes towards the equilibrium, but numerical errors cause a severe deviation in the CN graph, as discussed in 2.1.

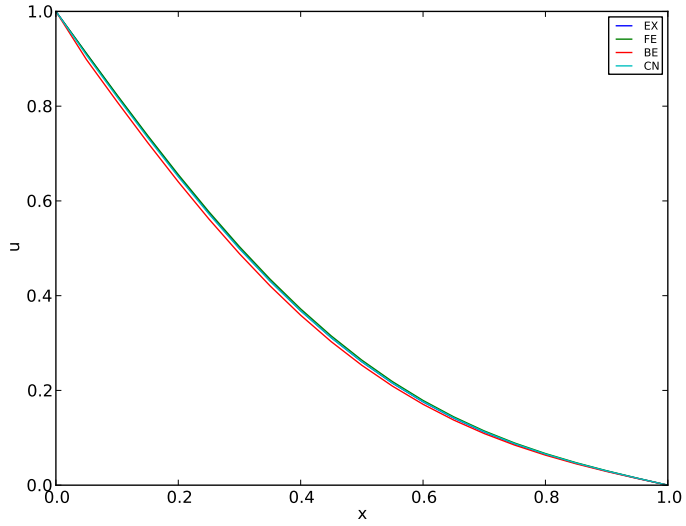

Figure 3: Plot of solutions where we can see that after after adding the static solution to v we get the solution to the original boundary problem. As $t \to \infty \Rightarrow u \to u_s$, the equilibrium state.

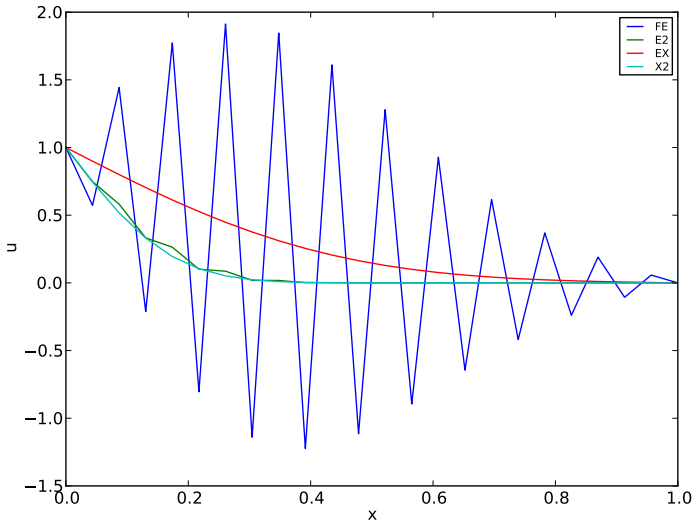

Figure 4: Plot of FE and the "exact solution" at two different timesteps with $\alpha = 0.529$. The error evolves with time, such that the solution is reasonable for some timesteps before it diverges

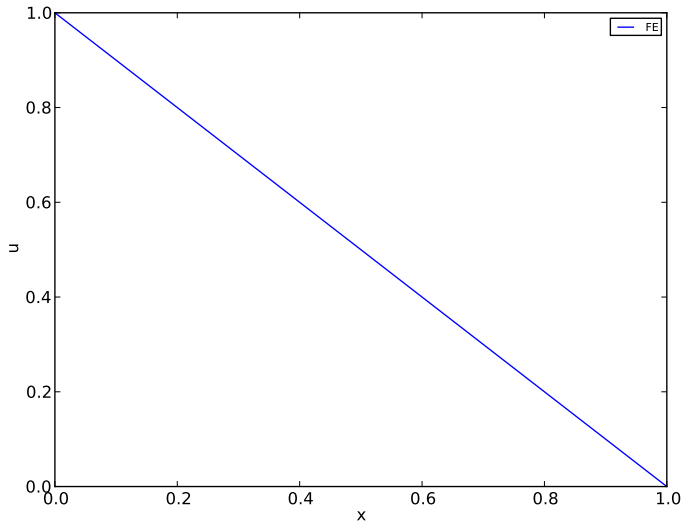

Figure 5: Plot of solution using the FE scheme with $t \gg 1$. We see that the solution becomes the linear expression.