Diffusion Equation

Vegard Falmår and Sigurd Sørlie Rustad

University of Oslo
Norway
December 14, 2020

CONTENTS

Abstract

I. INTRODUCTION

For our studies we have used c++ for heavy computation, python for visualization and automation. All the code along with instructions on how to run it, can be cloned from our GitHub repository[1].

II. THEORY

The diffusion equation

The full diffusion equation reads

$$\frac{\partial u(\mathbf{r},t)}{\partial t} = \nabla \cdot \left[D(u,\mathbf{r}) \nabla u(\mathbf{r},t) \right],$$

with $\mathbf{r}$ is a positional vector. If $D(u,\mathbf{r}) = 1$ the equation simplifies to

$$\frac{\partial u}{\partial t} = \nabla^2 u(\mathbf{r},t),$$

or

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \right) u(x,y,z,t) = \frac{\partial u(x,y,z,t)}{\partial t} \quad (1)$$

in cartesian coordinates. In this report we are mainly going to work with the diffusion equation in one and two dimensions, i.e.

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t}$$
$$\wedge$$
$$\frac{\partial^2 u(x,y,t)}{\partial x^2} + \frac{\partial^2 u(x,y,t)}{\partial y^2} = \frac{\partial u(x,y,t)}{\partial t}.$$

Discretization

Equation (1) in one dimension reads

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t} \quad \text{or} \quad u_{xx} = u_t. \quad (2)$$

With $x \in [0, L]$ and boundary conditions

$$u(0,t) = a(t), \quad t \geq 0 \quad \wedge \quad u(L,t) = b(t), \quad t \geq 0, \quad (3)$$

we can approximate the solution by discretization. First introducing $\Delta x = L/(n+1)$ and Δt as small steps in x-direction and time. Then we can define the value domain of t and x,

$$t_j = j\Delta t, \quad j \in \mathbb{N}_0 \quad \wedge \quad x_i = i\Delta x, \quad \{i \in \mathbb{N}_0 | i \leq n+1\}.$$

Where $n+1$ is the number of points at which we evaluate $u(x,t)$.

[1] github.com/sigurdru/FYS3150/tree/master/Project5

Explicit and implicit schemes

It is common divide numerical algorithms for integration into explicit and implicit schemes. When perfoming a numerical integration, we iterate over a discrete set of grid points at which we evaluate the function in question. In explicit schemes, the value at the next grid point is determined entirely by the value at the current grid point. In implicit schemes, the value at a later stage is determined by solving a system of equations containing both the current state and later states.

Using an implicit method instead of an explicit method usually requires more computation in every step, and they are often harder to implement. It can, in turn, save computation by allowing larger step sizes. Implicit methods can be (ARE?) unconditionally stable, meaning they are stable for any choice of step sizes. Explicit schemes are conditionally stable, meaning there is a relation between the step sizes in time and position that must be fulfilled in order to achieve stability. Of course, in order to achieve a desired *accuracy*, the step sizes can not be arbitrarily large for either the explicit or implicit schemes. It is generally the case, though, that implicit schemes allow for larger step sizes than explicit schemes.

REFERENCES, Intro to PDEs book, Wiki

Explicit forward Euler

The algorithm for explicit forward Euler in one dimension (from [1] chapter 10.2.1) reads

$$u_{i,j+1} = \alpha u_{i-1,j} + (1-2\alpha) u_{i,j} + \alpha u_{i+1,j} \quad (4)$$

where

$$\alpha = \frac{\Delta t}{\Delta x^2},$$

and a local approximate error of $O(\Delta t)$ and $O(\Delta x^2)$. The discretization is explained in the appropriate section. Note that the expression on the right hand side, used to calculate the value at a time $t_j + \Delta t$, only contains the state of the system at time t_j. This can be written as a matrix equation (see [1] chapter 10.2.1)

$$\mathbf{u}_{j+1} = (\mathbb{1} - \alpha \mathbf{B}) \mathbf{u}_j$$

where

$$\begin{bmatrix} 2 & -1 & 0 & 0 \dots \\ -1 & 2 & -1 & 0 \dots \\ \dots & \dots & \dots & \dots \\ 0 & \dots & -1 & 2 \end{bmatrix} \quad \wedge \quad \mathbf{u}_j = \begin{bmatrix} u_{1,j} \\ u_{2,j} \\ \dots \\ u_{n,j} \end{bmatrix}$$

The stability requirement for this algorithm (also from [1] chapter 10.2.1) is

$$\rho(\mathbf{B}) \leq 1 \implies \frac{\Delta t}{(\Delta x)^2} \leq 1/2 \quad (5)$$

Where $\rho(\mathbf{B})$ is the spectral radius of $\mathbf{B}$:

$$\rho(\mathbf{B}) = \max\{|\lambda| : \det(\mathbf{B} - \lambda\mathbb{1}) = 0\}.$$

In two dimensions ($u(x, y)$) the method for solving is similar. Assuming the same number of integration points in x- and y-direction ($\Delta x = \Delta y = \Delta l$), the algorithm (from [1] chapter 10.2.5) reads

$$u_{i,j}^{l+1} = u_{i,j}^{l} + \alpha \left[u_{i+1,j}^{l} + u_{i-1,j}^{l} + u_{i,j+1}^{l} + u_{i,j-1}^{l} - 4u_{i,j}^{l}\right]$$

where $u_{i,j}^{l} = u(i\Delta l, j\Delta l, l\Delta t)$ and $\alpha = \Delta t/\Delta l^2$.

Implicit Backward Euler

The Backward Euler algorithm uses the same centered difference in space as Forward Euler to approximate the second derivative

$$u_{xx} \approx \frac{u(x_i + \Delta x, t_j) - 2u(x_i, t_j) + u(x_i - \Delta x, t_j))}{\Delta x^2}, \tag{6}$$

but a backward formula for the time derivative:

$$u_t \approx \frac{u(x_i, t_j) - u(x_i, t_j - \Delta t)}{\Delta t}, \tag{7}$$

which also has a local truncation error of $O(\Delta t)$ and $O(\Delta x^2)$. Again, defining

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

we obtain by inserting (6) and (7) into our differential equation the equation describing Backward Euler:

$$u_{i,j-1} = -\alpha u_{i-1,j} + (1 + 2\alpha)u_{i,j} - \alpha u_{i+1,j} \tag{8}$$

Written out for all i, equation 8 becomes

$$\begin{aligned}
-\alpha u_{0,j} + (1 + 2\alpha)u_{1,j} - \alpha u_{2,j} &= u_{1,j-1} \\
-\alpha u_{1,j} + (1 + 2\alpha)u_{2,j} - \alpha u_{3,j} &= u_{2,j-1} \\
&\dots \\
-\alpha u_{n-3,j} + (1 + 2\alpha)u_{n-2,j} - \alpha u_{n-1,j} &= u_{n-2,j-1} \\
-\alpha u_{n-2,j} + (1 + 2\alpha)u_{n-1,j} - \alpha u_{n,j} &= u_{n-1,j-1}
\end{aligned}$$

In general, this can be rearranged slightly so that

$$\begin{aligned}
(1 + 2\alpha)u_{1,j} - \alpha u_{2,j} &= u_{1,j-1} + \alpha u_{0,j} \\
-\alpha u_{1,j} + (1 + 2\alpha)u_{2,j} - \alpha u_{3,j} &= u_{2,j-1} \\
&\dots \\
-\alpha u_{n-3,j} + (1 + 2\alpha)u_{n-2,j} - \alpha u_{n-1,j} &= u_{n-2,j-1} \\
-\alpha u_{n-2,j} + (1 + 2\alpha)u_{n-1,j} - \alpha u_{n,j} &= u_{n-1,j-1} + \alpha u_{n,j}
\end{aligned}$$

Let $\mathbf{v}_j$ be a vector containing the values of u at $n-1$ points in space at a time t_j

$$\mathbf{v}_j = \begin{bmatrix} u_{1,j} \\ u_{2,j} \\ \vdots \\ u_{n-2,j} \\ u_{n-1,j} \end{bmatrix} \tag{9}$$

and the vector $\mathbf{b}_j$ be defined as follows:

$$\mathbf{b}_j = \begin{bmatrix} u_{1,j-1} + \alpha u_{0,j} \\ u_{2,j-1} \\ \vdots \\ u_{n-2,j-1} \\ u_{n-1,j-1} + \alpha u_{n,j} \end{bmatrix} \tag{10}$$

As the boundary conditions $u_{0,j}$ and $u_{n,j}$ are specified, we already know every component of $\mathbf{b}_j$. We can then rewrite equation (8) as the matrix equation

$$\mathbf{A}\mathbf{v}_j = \mathbf{b}_j \tag{11}$$

where $\mathbf{A}$ is defined as

$$A = \begin{bmatrix} (1+2\alpha) & -\alpha & 0 & 0 & \dots & 0 \\ -\alpha & (1+2\alpha) & -\alpha & 0 & \dots & 0 \\ \vdots & & \ddots & & & \vdots \\ 0 & \dots & 0 & -\alpha & (1+2\alpha) & -\alpha \\ 0 & \dots & 0 & 0 & -\alpha & (1+2\alpha) \end{bmatrix} \tag{12}$$

This is a tridiagonal matrix with $(1+2\alpha)$ on the diagonal and $-\alpha$ directly above and below the diagonal.

MOVE SOMETHING TO METHODS?

Implicit Crank-Nicolson

The Crank-Nicolson algorithm uses a time-centered scheme centered around $t+\Delta t/2$, with a truncation error of $O(\Delta t^2)$ and $O(\Delta x^2)$. The time derivative is given by

$$u_t \approx \frac{u(x_i, t_j + \Delta t) - u(x_i, t_j)}{\Delta t} \tag{13}$$

$$\begin{aligned}
u_{xx} \approx \frac{1}{2\Delta x^2}\Big(&u(x_i + \Delta x, t_j) - 2u(x_i, t_j) + u(x_i - \Delta x, t_j) \\
&+ u(x_i + \Delta x, t_j + \Delta t) - 2u(x_i, t_j + \Delta t) \\
&+ u(x_i - \Delta x, t_j + \Delta t)\Big)
\end{aligned}$$

Inserting these two equations into our differential equation, we obtain (see Appendix, section VII A for derivation) that the equation describing the Crank-Nicolson algorithm can be written as a matrix equation

$$\mathbf{A}\mathbf{v}_j = \mathbf{b}_j \tag{14}$$

The vector $\mathbf{v}_j$ is the same as in the case of Backward Euler, defined in (9). The matrix $\mathbf{A}$ is a tridiagonal matrix with $2(1+\alpha)$ on the diagonal and $-\alpha$ directly above and below the diagonal. The vector $\mathbf{b}_j$ is

$$\mathbf{b}_j = \begin{bmatrix} \alpha u_{0,j} + 2(1-\alpha)u_{1,j} + \alpha u_{2,j} + \alpha u_{0,j+1} \\ \alpha u_{1,j} + 2(1-\alpha)u_{2,j} + \alpha u_{3,j} \\ \vdots \\ \alpha u_{n-3,j} + 2(1-\alpha)u_{n-2,j} + \alpha u_{n-1,j} \\ \alpha u_{n-2,j} + 2(1-\alpha)u_{n-1,j} + \alpha u_{n,j} + \alpha u_{n,j+1} \end{bmatrix} \tag{15}$$

III. METHODS

We will start by solving the one dimensional diffusion equation

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

with initial conditions

$$u(x,0) = 0$$

and boundary conditions

$$u(0,t) = 0, \quad t \geq 0 \quad \text{and}$$
$$u(L,t) = 1, \quad t \geq 0$$

Solving the tridiagonal matrix systems

Solving equations (11) and REF is done by inverting the matrix $\mathbf{A}$. Since this matrix never changes, invert once instead of every time step??

Specialized algorithms can be developed for solving tridiagonal matrix systems. We will be reusing code developed for a previous project[2].

IV. RESULTS

Results

V. DISCUSSION

Discussion

VI. CONCLUSION

Conclusion

[1] Morten Hjorth-Jensen, Computational Physics, Lecture Notes Fall 2015, August 2015, https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf.

VII. APPENDIX

A. Derivation of equation for the Crank-Nicolson scheme

Again we define

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

[2] https://github.com/sigurdru/FYS3150/tree/master/Project1

Inserting the approximations of u_t and u_{xx} stated in the Theory section into our differential equation, we get

$$\begin{aligned}
u_t &= u_{xx} \\
\frac{u_{i,j+1} - u_{i,j}}{\Delta t} &= \frac{1}{2\Delta x^2}\Big(u_{i+1,j} - 2u_{i,j} + u_{i-1,j} + u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}\Big) \\
u_{i,j+1} - u_{i,j} &= \frac{\alpha}{2}\Big(u_{i+1,j} - 2u_{i,j} + u_{i-1,j} + u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}\Big) \\
u_{i,j+1} - \frac{\alpha}{2}\Big(u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}\Big) &= u_{i,j} + \frac{\alpha}{2}\Big(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}\Big) \\
2u_{i,j+1} - \alpha u_{i+1,j+1} + 2\alpha u_{i,j+1} - \alpha u_{i-1,j+1} &= 2u_{i,j} + \alpha u_{i+1,j} - 2\alpha u_{i,j} + \alpha u_{i-1,j} \\
-\alpha u_{i-1,j+1} + 2(1+\alpha)u_{i,j+1} - \alpha u_{i+1,j+1} &= \alpha u_{i-1,j} + 2(1-\alpha)u_{i,j} + \alpha u_{i+1,j}
\end{aligned}$$

Written out explicitly for all i, the equation reads

$$\begin{aligned}
-\alpha u_{0,j+1} + 2(1+\alpha)u_{1,j+1} - \alpha u_{2,j+1} &= \alpha u_{0,j} + 2(1-\alpha)u_{1,j} + \alpha u_{2,j} \\
-\alpha u_{1,j+1} + 2(1+\alpha)u_{2,j+1} - \alpha u_{3,j+1} &= \alpha u_{1,j} + 2(1-\alpha)u_{2,j} + \alpha u_{3,j} \\
&\vdots \\
-\alpha u_{n-3,j+1} + 2(1+\alpha)u_{n-2,j+1} - \alpha u_{n-1,j+1} &= \alpha u_{n-3,j} + 2(1-\alpha)u_{n-2,j} + \alpha u_{n-1,j} \\
-\alpha u_{n-2,j+1} + 2(1+\alpha)u_{n-1,j+1} - \alpha u_{n,j+1} &= \alpha u_{n-2,j} + 2(1-\alpha)u_{n-1,j} + \alpha u_{n,j}
\end{aligned}$$

Similar to the case of Backward Euler, this set of equations can be rearranged as follows

$$\begin{aligned}
2(1+\alpha)u_{1,j+1} - \alpha u_{2,j+1} &= \alpha u_{0,j} + 2(1-\alpha)u_{1,j} + \alpha u_{2,j} + \alpha u_{0,j+1} \\
-\alpha u_{1,j+1} + 2(1+\alpha)u_{2,j+1} - \alpha u_{3,j+1} &= \alpha u_{1,j} + 2(1-\alpha)u_{2,j} + \alpha u_{3,j} \\
&\vdots \\
-\alpha u_{n-3,j+1} + 2(1+\alpha)u_{n-2,j+1} - \alpha u_{n-1,j+1} &= \alpha u_{n-3,j} + 2(1-\alpha)u_{n-2,j} + \alpha u_{n-1,j} \\
-\alpha u_{n-2,j+1} + 2(1+\alpha)u_{n-1,j+1} &= \alpha u_{n-2,j} + 2(1-\alpha)u_{n-1,j} + \alpha u_{n,j} + \alpha u_{n,j+1}
\end{aligned}$$

We can define the vector $\mathbf{b}_j$ to hold the values on the right side of each of these equations:

$$\mathbf{b}_j = \begin{bmatrix} \alpha u_{0,j} + 2(1-\alpha)u_{1,j} + \alpha u_{2,j} + \alpha u_{0,j+1} \\ \alpha u_{1,j} + 2(1-\alpha)u_{2,j} + \alpha u_{3,j} \\ \vdots \\ \alpha u_{n-3,j} + 2(1-\alpha)u_{n-2,j} + \alpha u_{n-1,j} \\ \alpha u_{n-2,j} + 2(1-\alpha)u_{n-1,j} + \alpha u_{n,j} + \alpha u_{n,j+1} \end{bmatrix} \tag{16}$$

Using this and the definition of the vector $\mathbf{v}_j$ from (9), we can write the set of equations as

$$\mathbf{A}\mathbf{v}_j = \mathbf{b}_j$$

where the matrix $\mathbf{A}$ is a tridiagonal matrix with $2(1+\alpha)$ on the diagonal and $-\alpha$ directly above and below the diagonal.