Computational Physics
Numericat

Timo Borner, Tom Herrmann

March 14, 2021

1 Theory

1.1 Crank-Nicolson

The Crank-Nicolson method combines the forward- and backward *Euler method*. Therefore, if

$$\frac{\partial u}{\partial t} = F\left(u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2}\right),$$

the iteration is defined as follows:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{1}{2}\left(F_i^{n+1}\left(u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2}\right) + F_i^n\left(u, x, t, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2}\right)\right)$$

The Crank-Nicolson method is numerically stable (i.e. its total variation is bounded as $\Delta t \to 0$, i.e. it doesn't go wild oscillating). Furthermore, its error is in $\mathcal{O}(\Delta t^2)$ (as its essentially the trapezoidal rule for approximating integrals).

It is an implicit method, as the value u_i^{n+1} is used to calculate the value itself.

1.2 Schrödinger-equation

The Schrödinger-equation is:

$$i\hbar\frac{\partial}{\partial t}\left|\psi(t)\right\rangle = \hat{H}\left|\psi(t)\right\rangle$$

With the Hamiltonian:

$$\hat{H} = \frac{\hat{\vec{p}}^2}{2m} + U(\vec{r})$$

When using the 3-points space-derivatives one finally gets for the iteration with the Crank-Nicolson method:

$$\left(\mathbb{1} + \frac{i}{\hbar}\hat{H}\frac{\Delta t}{2}\right)|\psi(t+\Delta t)\rangle = \left(\mathbb{1} - \frac{i}{\hbar}\hat{H}\frac{\Delta t}{2}\right)|\psi(t)\rangle$$

2 Implementation

We divided the project into two parts, Frontend and Backend. The Frontend is written in **Java** and the Backend in **C**.

All calculations (including the parsing of the input, complex matrix operations and solving of tridiagonal linear equations) are done in the Backend. After each new implemenation we also wrote tests for the respective part to make sure that everything works as it should. We have used no third-party tools / libraries etc. for those operations, but rather implemented them ourselves.

The animated visualisation is done in the Frontend, using Swing and a custom plotting implementation.

To make both parts work together we used **Make**, so each part has its own *Makefile*.

3 Conclusion

3.1 Results

An example using the Theta function would look like this, but it should be noted that this is only one frame and not the entire visualization.

Figure 1: Created by using the parameters: "$20 * h(0 - (x - 0.15)) + 10 * h(x - 0.25)$", "$(1/(2 * 3.14159 * 0.005\hat{}2))\hat{}(1/4) * e(0 - (x - 0.1)\hat{}2/(4 * 0.005\hat{}2)) * e(200000i * x)$"

3.2 Problems we had

- Efficiency in the inversion of 1000×1000 matrices $\rightarrow$ Thomas-Algorithm
- Visual Performance $\rightarrow$ runtime optimazition and in memory processing

3.3 Improvements

- Bordercase behavoir (absorbing walls)
- Multidimensionality

References

[1] Wikipedia, Crank-Nicolson method
https://en.wikipedia.org/wiki/Crank-Nicolson_method

[2] Wikipedia, Thomas Algorithm
https://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm

[3] Prof. Dr. Alexander Lichtenstein, *Computational Physics script*