Crank–Nicolson

Heya,

so I'm currently trying to implement a simple type of the Crank–Nicolson-method in C++ to numerically solve the time-dependent Schroedinger equation. The code compiles but the program doesn't function as it is supposed to. We're looking at the following equation

$$\mathrm{i}\frac{\partial\psi}{\partial t} = -\frac{\partial^2\psi}{\partial x^2} \tag{1}$$

with the boundary condition:

$$\lim_{x\to 0,L}\psi(t,x) = 0\,. \tag{2}$$

The length of the box L is set to one

$$L \equiv 1 \tag{3}$$

and our discretization steps for t and x are

$$\Delta x = 0.01 \qquad \text{and} \qquad \Delta t = 10^{-5}\,. \tag{4}$$

Now, equation 1 is to be solved up until t = 0.03. We are given an analytical expression of the wave function for t = 0 which reads

$$\psi(0,x) = \frac{1}{\sqrt[4]{2\pi\sigma^2}}\cdot\exp\left(-\frac{(x-x_0)^2}{4\sigma^2} + \mathrm{i}kx\right)\,. \tag{5}$$

All unmentioned parameters can be found as constants at the beginning of the code. Now, picturing the discretization problem on a 2D grid it might look somewhat like this:

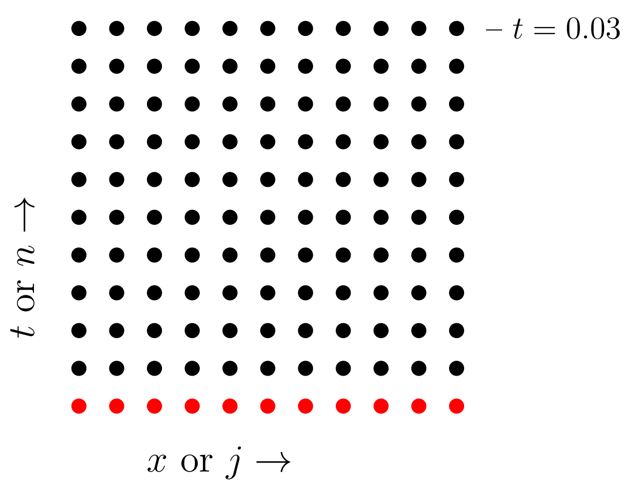

Figure 1: 2D Grid visualization of the discretization problem. The wave function at t_0 for every ($x + j\cdot \mathrm{d}x$, j being an iterator) is known at the beginning (red dots). Starting here, the wave function can now be numerically solved row-wise with equation 6.

Given all ψ of the n-th row, we can implicitly calculate the $(n+1)$-th row following the formula

$$\psi_{n+1,j} - \mathrm{i}\frac{\Delta t}{(\Delta x)^2} \cdot (\psi_{n+1,j-1} - 2 \cdot \psi_{n+1,j} + \psi_{n+1,j+1}) \tag{6}$$

$$= \psi_{n,j} + \mathrm{i}\frac{\Delta t}{(\Delta x)^2} \cdot (\psi_{n,j-1} - 2 \cdot \psi_{n,j} + \psi_{n,j+1}) \tag{7}$$

solving a LSE with its coefficient matrix being tridiagonal:

$$\begin{pmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & & \\ & a_3 & \dots & \dots & \\ & & \dots & \dots & c_{N-1} \\ & & & a_N & b_N \end{pmatrix} \cdot \begin{pmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \\ \dots \\ \psi_N \end{pmatrix} = \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \\ \dots \\ \varphi_N \end{pmatrix} \tag{8}$$

Here, $\vec{a}$, $\vec{b}$ and $\vec{c}$ can be considered column vectors. To determine their elements we go back to equation 6:

$$z := \mathrm{i}\frac{\Delta t}{(\Delta x)^2} \tag{9}$$

$$\psi_{n+1,j} - z \cdot (\psi_{n+1,j-1} - 2 \cdot \psi_{n+1,j} + \psi_{n+1,j+1}) \tag{10}$$

$$= \psi_{n,j} + z \cdot (\psi_{n,j-1} - 2 \cdot \psi_{n,j} + \psi_{n,j+1}) \tag{11}$$

Since everything on the right side of the equation is known as it solely refers to the n-th row, all of it is put into φ_n and the left side is transformed to fit into an LSE:

$$\psi_{n+1,j} - z \cdot \psi_{n+1,j-1} - z \cdot 2 \cdot \psi_{n+1,j} - z \cdot \psi_{n+1,j+1} = \varphi_n \tag{12}$$

$$-z \cdot \psi_{n+1,j-1} + (2z+1) \cdot \psi_{n+1,j} - z \cdot \psi_{n+1,j+1} = \varphi_n \tag{13}$$

$$a_n \cdot \psi_{n+1,j-1} + b_n \cdot \psi_{n+1,j} + c_n \cdot \psi_{n+1,j+1} = \varphi_n \tag{14}$$

As you can see, all elemets of the respective vectors are the same:

$$a_n = -z \tag{15}$$

$$b_n = 2z + 1 \tag{16}$$

$$c_n = -z\ . \tag{17}$$

To solve the LSE I also implemented the Thomas-Algorithm for tridiagonal matrices in C++ (`trisolv`), which I tested multiple times to ensure that it works with both real and complex numbers. Ideally now every single grid point should be associated with a complex number. Printing out its norm in an xyz format into a file following

$$P(x) = |\psi(x)|^2 \tag{18}$$

the amplitude of the wave function for all t and x can be 3D visualized using gnuplot. In an earlier exercise I successfully did that using another method, the code for that will also be added. Now though, the amplitude decreases way too fast, already at $t = 0.0008$ equation 18

is evaluated as 0 for all grid points. Having also tested the `create_d`-function it should yield the correct results as I also did the math by hand multiple times.
I really don't know what's wrong here.

Pastebin of the working Leapfrog-like algorithm: `https://pastebin.com/FFf4RbP5`
Imgur of 3D plot ($P(x,t)$ along z-axis): `https://imgur.com/a/8Czko`
Pastebin of the working Crank-Nicolson algorithm: `https://pastebin.com/4sy11wPC`

I'm thankful for any help.