# Computing the Discretised Schrödinger Equation

xd

April 21, 2017

## 1 Math

### 1.1 Discretising

Start off with the Schrödinger Equation in one-dimension:

$$i\hbar\frac{\partial}{\partial t}\psi = \Big[ -\frac{\hbar}{2m}\frac{\partial^2}{\partial x^2} + V(x)\Big]\psi$$

Define the complex wave function $\psi$ in terms of its real and imaginary parts:

$$\psi(x,t) = a(x,t) + ib(x,t)$$

$$\psi = \begin{pmatrix} a(x,t) \\ b(x,t) \end{pmatrix}, i\psi = \begin{pmatrix} -b(x,t) \\ a(x,t) \end{pmatrix}$$

Substituting this back into the Schrödinger Equation, we see:

$$\frac{\partial}{\partial t}\begin{pmatrix} -b(x,t) \\ a(x,t) \end{pmatrix} = -\frac{\hbar}{2m}\frac{\partial^2}{\partial x^2}\begin{pmatrix} a(x,t) \\ b(x,t) \end{pmatrix} + \begin{pmatrix} \frac{V(x)}{\hbar}a(x,t) \\ \frac{V(x)}{\hbar}b(x,t) \end{pmatrix}$$

Define the following variables:

$$\begin{aligned} i &= 0, \ldots, n \\ x_i &= i\delta x \\ \delta x &= \text{lattice spacing} \\ \delta x &= \frac{L}{n} \\ a(x_i, t) &= a_i(t) \\ b(x_i, t) &= b_i(t) \\ V(x_i) &= V_i \end{aligned}$$

Recall the second derivative approximation discussed in class:

$$\begin{aligned} \Big(\frac{\partial^2 a}{\partial x^2}\Big) x_i &\approx \frac{a_{i+1} + a_{i-1} - 2a_i}{(\delta x)^2} \\ \Big(\frac{\partial^2 b}{\partial x^2}\Big) x_i &\approx \frac{b_{i+1} + b_{i-1} - 2b_i}{(\delta x)^2} \end{aligned}$$

We can see the following:

$$\begin{pmatrix} \partial_t(-b_i) \\ \partial_t(a_i) \end{pmatrix} = -\frac{\hbar}{2m\delta x^2} \begin{pmatrix} a_{i+1} + a_{i-1} - 2a_i \\ b_{i+1} + b_{i-1} - 2b_i \end{pmatrix} + \begin{pmatrix} \frac{V_i}{\hbar} a_i(t) \\ \frac{V_i}{\hbar} b_i(t) \end{pmatrix}$$

## 1.2 Removing dimensions

Define the following dimensionless quantities:

$$\begin{aligned} t &= \Big(\frac{2m\delta x^2}{\hbar}\Big)\tau \\ &= (\delta t)\tau \leftarrow \text{dimensionless} \\ V_i &= V_0 \widetilde{V}_i \leftarrow \text{dimensionless} \end{aligned}$$

Therefore:

$$\begin{pmatrix} -\frac{\partial}{\partial\tau}b_i(\tau) \\ \frac{\partial}{\partial\tau}a_i(\tau) \end{pmatrix} = \begin{pmatrix} 2a_i - (a_{i+1} + a_{i-1}) \\ 2b_i - (b_{i+1} + b_{i-1}) \end{pmatrix} + \begin{pmatrix} \frac{\delta t}{\hbar}V_i a_i(t) \\ \frac{\delta t}{\hbar}V_i b_i(t) \end{pmatrix}$$

$$\begin{pmatrix} \frac{\partial}{\partial\tau}b_i(\tau) \\ \frac{\partial}{\partial\tau}a_i(\tau) \end{pmatrix} = \begin{pmatrix} a_{i+1} + a_{i-1} - 2a_i \\ 2b_i - (b_{i+1} + b_{i-1}) \end{pmatrix} + \left(\frac{\delta t}{\hbar}V_0\right) \begin{pmatrix} \widetilde{V}_i a_i(\tau) \\ \widetilde{V}_i b_i(\tau) \end{pmatrix}$$

Let us define $\beta$ such that:

$$\begin{aligned} \beta &:= \frac{\delta t V_0}{\hbar} \\ &= \frac{2mL^2}{\hbar^3 n^2} V_0 \end{aligned}$$

This is a dimensionless **coupling constant** on the **lattice**.

# 2 Computation

## 2.1 Necessary resources/steps

Next: Build an evolution scheme (algorithm) through **small** ($\delta\tau \ll 1$) dimensionless time steps.
Tradeoff: We want to do evolutions through enough time steps to see disturbances in the wave function propogate across the entire length of the lattice. This means that if we step forward in time by $\delta\tau \approx 1$, we will need total time $N\delta t$, $N \approx \frac{L\delta t}{\delta x/\delta t} = \frac{L}{\delta x} = n$

If we choose $\Delta\tau \approx \frac{1}{m}$ ($m$ some integer $> 0$) we will need between $n \times m$ and $2nm$ **total time steps at a minimum** to see instructions that propogate across the total lattice.
This means we prepare for 20,000 time steps on a 1000 point lattice (at the minimum) and probably more like 50,000 to 100,000.

This means that if we want to see real results in a real time $T$, a single evolution time step must run in a time less than $10^{-5}T$. (e.g. If $T = 10$s then a single evolution time step must run in $10^{-3}$s)

This gives you a rule of thumb about the time your code can take to do an evolution time step for data on the lattice. Clearly, to be fast you need to get single time step evolution execution time under a millisecond. Since this scales with the lattice size, you will probably want to debug a lot of code on lattices with smaller number of points (like 200 - 500).

## 2.2 Boundary conditions

Boundary conditions: $a_0 = b_0 = a_n = b_n = 0$

Hold these values fixed for "box" boundary conditions. These values are not **evolved** during the program run.

SO FAR: We have translated a **partial differential equation** into a very large number of **ordinary differential equations** that are coupled together.
NEXT: We turn to the problem of approximating the time evolution in continuous time into an approximate series of time step evolutions. You will want to encapsulate the time evolution step in a single function or routine. I will suggest a baseline approach; however, there is room to upgrade or try out different evolution time step algorithms. Encapsulating this part of the program will facilitate experimentation.