# Solution of the Time-dependent Schrödinger Equation using the Crank-Nicolson algorithm with MPI on a 2-D regular Cartesian grid

## Seminar on High Performance Computing 2
## Summer term 2017

Peter Leitner & Stefan Hofmeister

Wednesday, May 10, 2017

# Time-dependent Schrödinger Equation

Schrödinger Equation in 2-D for an electron in a given potential

$$\mathrm{i}\hbar\frac{\partial}{\partial t}\left|\psi(t,\boldsymbol{x})\right\rangle = \hat{H}(\hat{\boldsymbol{p}},\hat{\boldsymbol{x}})\left|\psi(t,\boldsymbol{x})\right\rangle, \quad \hat{H} = \frac{\hat{p}_x^2}{2m} + \frac{\hat{p}_y^2}{2m} + \hat{V}(x,y) \tag{1}$$

Attempt to simply discretize the differential operators,

$$\partial_t\psi(t,\boldsymbol{x}) \approx \frac{\psi_{j,l}^{n+1} - \psi_{j,l}^{n}}{\Delta t},$$

$$\frac{\hat{\boldsymbol{p}}^2}{2m}\psi(t,\boldsymbol{x}) = -\frac{\hbar^2}{2m}\nabla^2\psi(t,\boldsymbol{x}) \approx -\frac{\hbar^2}{2m}\frac{\psi_{j+1,l}^{n+1} - 2\psi_{j,l}^{n+1} + \psi_{j-1}^{n+1}}{(\Delta x)^2} + \dots$$

is – as the von Neumann stability analysis shows – unconditionally stable. However, each solution of Schrödinger's equation has to fulfill a continuity equation for the probability $w(t,\boldsymbol{x})\,\mathrm{d}^3x = \psi^*\psi\,\mathrm{d}^3x$ to find the electron within a volume $\mathrm{d}^3x$ around $\boldsymbol{x}$,

$$\partial_t w(t,\boldsymbol{x}) + \nabla\cdot\boldsymbol{j} = 0, \quad \boldsymbol{j} = \frac{\hbar}{2\mathrm{i}m}[\psi^*(\nabla\psi) - (\nabla\psi^*)\psi] \tag{2}$$

From the continuity equation follows the conservation of the norm of the wave function:

$$\frac{\mathrm{d}}{\mathrm{d}t}\int \mathrm{d}^3x\, w(t,\boldsymbol{x}) = \int \mathrm{d}^3x\, \partial_t w(t,\boldsymbol{x}) = -\int \mathrm{d}^3x\, \nabla\cdot \boldsymbol{j}(t,\boldsymbol{x}) = -\int \mathrm{d}\boldsymbol{S}\cdot\boldsymbol{j}(t,\boldsymbol{x})$$

$$\int_{\mathbb{R}^3} \mathrm{d}^3x\, \|w(0,\boldsymbol{x})\|^2 = 1 \xrightarrow{\text{SE}} \int_{\mathbb{R}^3} \mathrm{d}^3x\, \|w(t,\boldsymbol{x})\|^2 = 1 \tag{3}$$

The normalization of the wave function $\psi(t,\boldsymbol{x})$ is however not conserved by the fully implicit discretization as this method is not unitary $\Rightarrow$ a discretization preserving the hermiticity of the Hamiltonian is needed that maintains the normalization of the wavefunction.

The temporal evolution of the wave function can be expressed using the unitary time-evolution operator $\hat{U}(t)$:

$$|\psi(t+\Delta t,\boldsymbol{x})\rangle \simeq \hat{U}(t)\,|\psi(t,\boldsymbol{x})\rangle\,, \qquad \hat{U}(t) = \exp(-\mathrm{i}\hat{H}t/\hbar) \tag{4}$$

With the time-evolution operator one can find the explicit Forward-Time Centered-Space (FTCS) scheme $\psi_{j,l}^{n+1} = (1-\mathrm{i}\hat{H}\Delta t)\psi_{j,l}^{n}$ and the implicit Backward-Time scheme $(1+\mathrm{i}\hat{H}\Delta t)\psi_{j,l}^{n+1} = \psi_{j,l}^{n}$. Neither scheme is unitary and thus violates probability conservation.

A finite-difference approximation of $\exp(-\mathrm{i}\hat{H}\Delta t/\hbar)$ that is unitary was found by Cayley:

$$\exp(-\mathrm{i}\hat{H}\Delta t/\hbar) = \frac{1 - \frac{\mathrm{i}\hat{H}\Delta t}{2\hbar}}{1 + \frac{\mathrm{i}\hat{H}\Delta t}{2\hbar}} + \mathcal{O}((\Delta t)^2)$$

$$(4) \Rightarrow \left(1 + \frac{\mathrm{i}}{2\hbar}\hat{H}\Delta t\right)|\psi(t + \Delta t, \boldsymbol{x})\rangle = \left(1 - \frac{\mathrm{i}}{2\hbar}\hat{H}\Delta t\right)|\psi(t, \boldsymbol{x})\rangle \tag{5}$$

This is exactly the Crank-Nicolson scheme for the Schrödinger equation.

# Discretization

By substituting the 2-D discrete wave function $\psi(t; x, y) \to \psi^n_{j,l}$ into Eq. (5) and considering the Hamiltonian in the form

$$\hat{H} = |\hat{\boldsymbol{p}} = -\mathrm{i}\hbar\nabla| = -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} - \frac{\hbar^2}{2m}\frac{\partial^2}{\partial y^2} + \hat{V}(x, y)$$

one obtains the discretized 2-D Schrödinger Equation

$$\psi^{n+1}_{j,l} - \frac{\mathrm{i}\Delta t}{2\hbar}\left(\frac{\hbar^2}{2m}\frac{\psi_{j+1,l}n + 1 - 2\psi^{n+1}_{j,l} + \psi^{n+1}_{j-1,l}}{(\Delta x)^2} + \frac{\psi^{n+1}_{j,l+1} - 2\psi^{n+1}_{j,l} + \psi^{n+1}_{j,l-1}}{(\Delta y)^2} - V_{j,l}\psi^{n+1}_{j,l}\right)$$
$$= \psi^n_{j,l} + \frac{\mathrm{i}\Delta t}{2\hbar}\left(\frac{\hbar^2}{2m}\frac{\psi^n_{j+1,l} - 2\psi^n_{j,l} + \psi^n_{j-1,l}}{(\Delta x)^2} + \frac{\hbar^2}{2m}\frac{\psi^n_{j,l+1} - 2\psi^n_{j,l} + \psi^n_{j,l-1}}{(\Delta y)^2} - V_{j,l}\psi_{j,l}\right) \quad (6)$$

Rewrite Eq. (6) in matrix-form. Assign a short-hand notation for the prefactors of the wave function at the next time step at various spatial grid points:

$$
\begin{aligned}
\psi_{j,l}^{n+1}: &\qquad b_{j,l} = 1 + \frac{\mathrm{i}\Delta t}{\hbar}\frac{\hbar^2}{2m}\left(\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2}\right) + \frac{\mathrm{i}\Delta t}{2\hbar}V_{j,l} \\
\psi_{j-1,l}^{n+1}: &\qquad c = -\frac{\mathrm{i}\Delta t}{2\hbar}\frac{\hbar^2}{2m}\frac{1}{(\Delta x)^2} \\
\psi_{j+1,l}^{n+1}: &\qquad a = c \\
\psi_{j,l+1}^{n+1}: &\qquad d = -\frac{\mathrm{i}\Delta t}{2\hbar}\frac{\hbar^2}{2m}\frac{1}{(\Delta x)^2} \\
\psi_{j,l-1}^{n+1}: &\qquad e = d
\end{aligned}
$$

and for the RHS:

$$
\begin{aligned}
\psi_{j,l}^{n}: &\qquad f_{j,l} = 1 - \frac{\mathrm{i}\Delta t}{\hbar}\frac{\hbar^2}{2m}\left(\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2}\right) - \frac{\mathrm{i}\Delta t}{2\hbar}V_{j,l} \\
\psi_{j+1,l}^{n}: &\qquad g = \frac{\mathrm{i}\Delta t}{2\hbar}\frac{\hbar^2}{2m}\frac{1}{(\Delta x)^2} \\
\psi_{j-1,l}^{n}: &\qquad h = g \\
\psi_{j,l+1}^{n}: &\qquad k = \frac{\mathrm{i}\Delta t}{2\hbar}\frac{\hbar^2}{2m}\frac{1}{(\Delta y)^2} \\
\psi_{j,l-1}^{n}: &\qquad p = k
\end{aligned}
$$

# LHS of the discretized Schrödinger Equation

Map the two indices $j$ and $l$ of the wave function $\psi^n_{j,l}$ on a single one $(j,l) \mapsto i$:

$$i = 1 + (j-1)(L-1) + l - 1 \quad \text{for} \quad j = 0, 1, \ldots, J; \;\; l = 0, 1, \ldots, L$$

Thereby $\psi^n_{1,1} \to \psi^n_1$, $\psi^n_{1,2} \to \psi^n_2$, …, $\psi^n_{1,J} \to \psi^n_J$, $\psi^n_{2,1} \to \psi^n_{J+1}$, … For simplicity (and lack of space) shown here only for $J = L = 4$, though later $J, L \sim 10^3$ will be used. It exhibits sub-blocks of shape $J \times L$:

$$\begin{pmatrix} b_{1,1} & d & 0 & a & 0 & 0 & 0 & 0 & 0 \\ e & b_{1,2} & d & 0 & a & 0 & 0 & 0 & 0 \\ 0 & e & b_{1,3} & 0 & 0 & a & 0 & 0 & 0 \\ c & 0 & 0 & b_{2,1} & d & 0 & a & 0 & 0 \\ 0 & c & 0 & e & b_{2,2} & d & 0 & a & 0 \\ 0 & 0 & c & 0 & e & b_{2,3} & 0 & 0 & a \\ 0 & 0 & 0 & c & 0 & 0 & b_{3,1} & d & 0 \\ 0 & 0 & 0 & 0 & c & 0 & e & b_{3,2} & d \\ 0 & 0 & 0 & 0 & 0 & c & 0 & e & b_{3,3} \end{pmatrix} \begin{pmatrix} \psi_1^{n+1} \\ \psi_2^{n+1} \\ \psi_3^{n+1} \\ \psi_4^{n+1} \\ \psi_5^{n+1} \\ \psi_6^{n+1} \\ \psi_7^{n+1} \\ \psi_8^{n+1} \\ \psi_9^{n+1} \end{pmatrix} = \boldsymbol{r}$$

# RHS of the discretized Schrödinger Equation

Vectorization of the RHS is not necessary when working with Fortran exclusively but essential when using the `numpy` module with Python:

$$\boldsymbol{r} = \begin{pmatrix} f_{1,1} & k & 0 & g & 0 & 0 & 0 & 0 & 0 \\ p & f_{1,2} & k & 0 & g & 0 & 0 & 0 & 0 \\ 0 & p & f_{1,3} & 0 & 0 & g & 0 & 0 & 0 \\ h & 0 & 0 & f_{2,1} & k & 0 & g & 0 & 0 \\ 0 & h & 0 & p & f_{2,2} & k & 0 & g & 0 \\ 0 & 0 & h & 0 & p & f_{2,3} & 0 & 0 & g \\ 0 & 0 & 0 & h & 0 & 0 & f_{3,1} & k & 0 \\ 0 & 0 & 0 & 0 & h & 0 & p & f_{3,2} & k \\ 0 & 0 & 0 & 0 & 0 & h & 0 & p & f_{33} \end{pmatrix} \begin{pmatrix} \psi_1^n \\ \psi_2^n \\ \psi_3^n \\ \psi_4^n \\ \psi_5^n \\ \psi_6^n \\ \psi_7^n \\ \psi_8^n \\ \psi_9^n \end{pmatrix}$$

After vectorization, the discretized Schrödinger Equation simplifies to

$$\hat{A}\boldsymbol{\psi}^{n+1} = \boldsymbol{r} = \hat{B}\boldsymbol{\psi}^{n}$$

Pseudo-code:

```
for n in range(N):
```

- Compute $\hat{B}\boldsymbol{\psi}^{n} = \boldsymbol{r}$ as a matrix-vector product
- Solve $\hat{A}\boldsymbol{\psi}^{n+1} = \boldsymbol{r}$ with LU decomposition
- This yields $\boldsymbol{\psi}^{n+1}$
- Map the rolling index $i$ back to indices $j$, $l$: $\psi_i^{n+1} \to \psi_{j,l}^{n+1}$
- Check that the wave packet remains normalized:

$$\int_{\Delta^2} \mathrm{d}^2 x \, \|\psi(t, \boldsymbol{x})\|^2 \approx \sum_{j,l=1}^{J,L} \Delta x \Delta y \, \left\|\psi_{j,l}^{n}\right\|^2 \stackrel{!}{=} 1$$

# Setting up the matrices in Python

```
# Build the main diagonal
b_main_diag = []   # coefficient matrix of LHS
f_main_diag = []   # coefficient matrix of RHS
for j in range(J-1):
    for l in range(J-1):
        b_main_diag.append(b[j,l]); f_main_diag.append(f[j,l])

e_minor_diag = np.ones((len(b_main_diag)-1,), dtype=complex)*e
d_minor_diag = np.ones((len(b_main_diag)-1,), dtype=complex)*d
a_minor_diag = np.ones((len(b_main_diag)-3,), dtype=complex)*a
c_minor_diag = np.ones((len(b_main_diag)-3,), dtype=complex)*c
k_minor_diag = np.ones((len(f_main_diag)-1,), dtype=complex)*k
p_minor_diag = np.ones((len(f_main_diag)-1,), dtype=complex)*p
g_minor_diag = np.ones((len(f_main_diag)-3,), dtype=complex)*g
h_minor_diag = np.ones((len(f_main_diag)-3,), dtype=complex)*h

# Build the coefficient matrices for A*psi^(n+1) = B*psi^n
A = np.diag(b_main_diag) + np.diag(e_minor_diag,-1) + np.diag(d_minor_diag,1) \
    + np.diag(c_minor_diag,-3) + np.diag(a_minor_diag,3)
B = np.diag(f_main_diag) + np.diag(p_minor_diag,-1) + np.diag(k_minor_diag,1) \
    + np.diag(h_minor_diag,-3) + np.diag(g_minor_diag,3)
```

# Setting-up the RHS in Python

```
for n in range(Nt):
    if n == 0:
        psi_n_mat = psi_0
        # Rewrite matrix psi_jl as a vector psi_i
        psi_n_vec = np.empty(((J-1)**2,), dtype=complex)
        for j in range(J-1):
            for l in range(J-1):
                i = ind(j,l)
                psi_n_vec[i] = psi_n_mat[j,l]
    else:
        psi_n_vec = psi_np1_vec

    RHS = np.dot(B,psi_n_vec)
    r = RHS
```

# LU decomposition; finally to be parallelized as Fortran code

```
# LU decomposition:
P, L, U = scipy.linalg.lu(A)
LU = L + U
nA = np.size(A, 0)
y = np.zeros((nA,), dtype=complex)
x = np.zeros((nA,), dtype=complex)

r = np.dot(np.linalg.inv(P),r)

for i in range(1, nA+1):
   s = r[i-1]
   for j in range(1, i):
       s = s - LU[i-1,j-1]*y[j-1]
   y[i-1] = s

for i in range(nA, 0, -1):
    s = y[i-1]
    for j in range(i+1, nA+1):
        s = s - LU[i-1,j-1]*x[j-1]
    x[i-1] = s/LU[i-1,i-1]

psi_np1_vec = x
```

# Parallelizing the code

The following strategies for code parallelization systems are envisaged:

- Application of Distributed SuperLU (`SuperLU_DIST`) of the SuperLU library
- Crank-Nicolson predictor-corrector (CNPC) method
  - Decomposition of spatial domain $\Omega$ in subdomains $\Omega_i$ which can be assigned to their own processor
  - Forward Euler to predict unknown values on the interface $\Gamma$ between subdomains $\Omega_i \subset \Omega$
  - Solving the problem independently on the subdomains $\Omega_i$ with Backward Euler using the predicted values $\psi_\Gamma^*$ at $t_{n+1}$
  - Correction of interface values $\psi_\Gamma^{n+1}$ with Backward Euler using interior values