AEP 4380 HW#7: Time Dependent Schrödinger Equation

Gianfranco Grillo

November 2, 2016

1 Background

1.1 Solving the Time Dependent Schrödinger Equation Using Crank-Nicholson

The central element of the wave formulation of quantum mechanics is the time dependent Schrödinger equation (SE), a partial differential equation that governs the time evolution of the wavefunction ψ associated to an object. The 1D SE for a particle of mass m moving in a potential energy $V(x,\, t)$ is

$$i\hbar\frac{\partial}{\partial t}\psi(x,\, t) = \frac{\hbar^2}{2m}\frac{\partial^2}{\partial t^2}\psi(x,\, t) + V(x,\, t)\psi(x,\, t) \tag{1}$$

where $\hbar = 6.5821 \times 10^{-16}\, eV \cdot sec$ is the reduced Planck constant.

Starting with a given $\psi(x,\, t = 0)$ and $V(x,\, t)$, it is possible to solve this equation numerically via the Crank-Nicholson algorithm. The algorithm works by writing the equation in terms of a finite difference matrix and then solving this matrix in order to give the state of the wavefunction at the next time step. Succesively applying Crank-Nicholson over several time steps allows us to track the evolution of the wave function over a desired period of time. In the case of the SE, its finite difference form (with $V = V(x)$ time independent) is given by

$$\begin{aligned}\psi(x - \Delta x,\, t + \Delta t) + \left[\frac{2m\omega i}{\hbar} - 2 - \frac{2m\Delta x^2}{\hbar^2}V(x)\right]\psi(x,\, t + \Delta t) + \psi(x + \Delta x,\, t + \Delta t)\\ = -\psi(x - \Delta x,\, t) + \left[\frac{2m\omega i}{\hbar} + 2 + \frac{2m\Delta x^2}{\hbar^2}V(x)\right]\psi(x,\, t) - \psi(x + \Delta x,\, t)\end{aligned} \tag{2}$$

This can be written in matrix notation as

$$a_j\psi(x_{j-1},\, t_{n+1}) + b_j\psi(x_j,\, t_{n+1}) + c_j\psi(x_{j+1},\, t_{n+1}) = d_j \tag{3}$$

with $a_j = c_j = 1$, $b_j = \frac{2m\omega i}{\hbar} - 2 - \frac{2m\Delta x^2}{\hbar^2}V(x)$, and $d_j = -\psi(x_{j-1},\, t_n) + \left[\frac{2m\omega i}{\hbar} + 2 + \frac{2m\Delta x^2}{\hbar^2}V(x)\right]\psi(x_j,\, t_n) - \psi(x_{j+1},\, t_n)$. Here, $\omega = \frac{2\Delta x^2}{\Delta t}$, with Δx and Δt corresponding to the sampling size in space and the sampling size in time, respectively. This system can be written as a matrix equation, and can be solved using the Crank-Nicholson method by converting the matrix to an upper diagonal form and then using backsubstitution in order to solve for the unknowns. The size of the matrix (and therefore the maximum value of j) will depend on Δx and the range of x that will be used for the problem.

1.2 Specific Parameters

In this homework, we are interested in solving the SE for an electron, so $m = m_e = 5.6990 \times 10^{-32}\, eV \cdot sec^2 \cdot \dot{A}^{-2}$. The initial wavefunction will be given by

$$\psi(x,\, t = 0) = exp\left[-\left(\frac{x - 0.3L}{s}\right)^2 + ixk_0\right] \tag{4}$$

where $L = 1000\,\dot{A}$, $s = 20\,\dot{A}$, and $k_0 = 1\,\dot{A}^{-1}$. The potential energy $V(x)$ will be given by

$$V(x) = \frac{V_0}{1 + exp\left[\left(0.5L - x\right)/\omega_v\right]}$$

where $V_0 = 4.05\,eV$, and $\omega_v = 7\,\dot{A}$. The region of interest will be the one delimited by $0 < x < L$, and the time period of interest will be $0 < t < 5 \times 10^{-14}\,sec$.

2 Results

2.1 Task 1

The first task consisted in plotting $V(x)$ (Figure 1), the real and imaginary parts of $\psi(x,\, t = 0)$ (Figures 2 and 3), and $|\psi(x,\, t = 0)|^2$ (Figure 4). Given that the average wavenumber k_0 is given as $1\,\dot{A}^{-1}$, and $k_0 = \frac{2\pi}{\lambda_0}$, the average wavelength is $\lambda_0 = 2\pi\,\dot{A}$, so I have used a Δx of $0.1\,\dot{A}$, in order to make sure that Δx is at least 10 times less than the smallest λ. I separated ψ into real and imaginary parts as follows, using Euler's identity

$$\begin{aligned}
\psi(x,\, t = 0) &= exp\left[-\left(\frac{x - 0.3L}{s}\right)^2 + ixk_0\right] \\
&= exp\left[-\left(\frac{x - 0.3L}{s}\right)^2\right] \cdot exp\left[ixk_0\right] \\
&= exp\left[-\left(\frac{x - 0.3L}{s}\right)^2\right] \cdot \left(\cos(xk_0) + i\sin(xk_0)\right) \qquad (5)
\end{aligned}$$

Thus the real and imaginary parts of ψ at $t = 0$ are given by

$$Re(\psi) = exp\left[-\left(\frac{x - 0.3L}{s}\right)^2\right] \cdot \cos(xk_0) \qquad (6)$$

$$Im(\psi) = exp\left[-\left(\frac{x - 0.3L}{s}\right)^2\right] \cdot \sin(xk_0) \qquad (7)$$

The code used for this task is given in section 4.1.

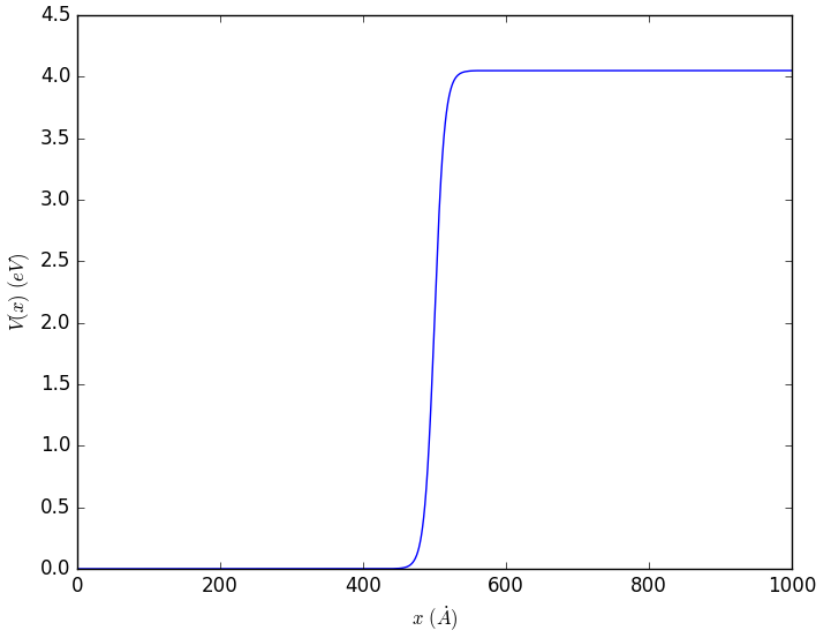

Figure 1: Potential V as a function of position x at $t = 0$

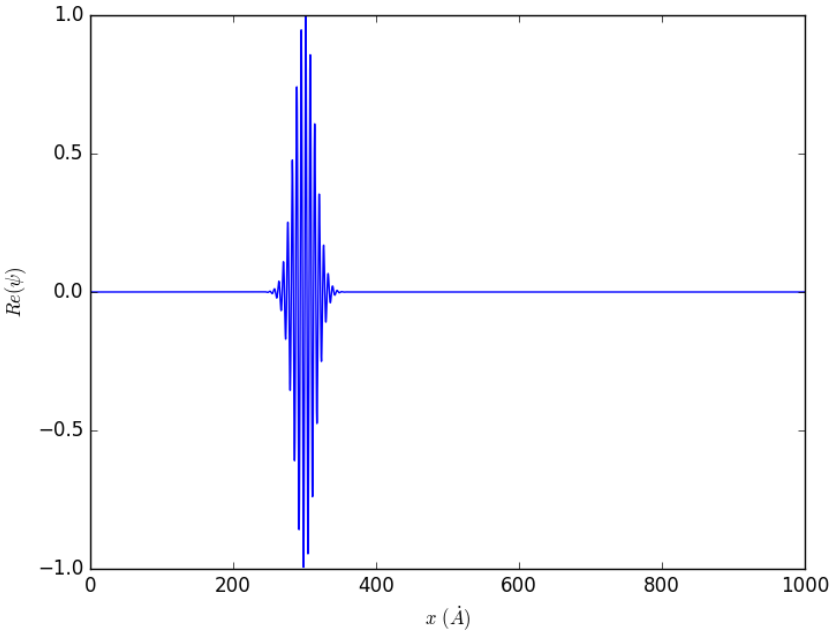

Figure 2: Real part of the wavefunction ψ as a function of position x at $t = 0$

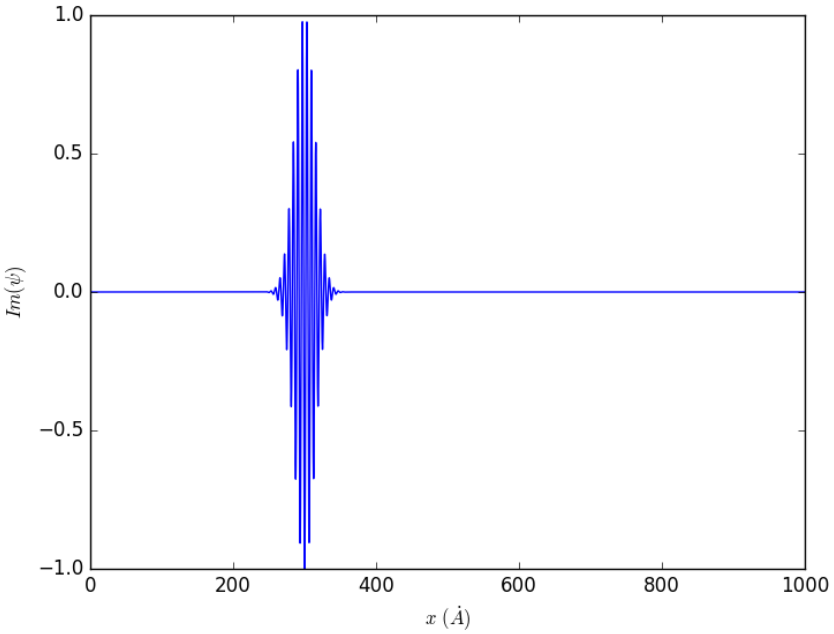

Figure 3: Imaginary part of the wavefunction ψ as a function of position x at $t = 0$

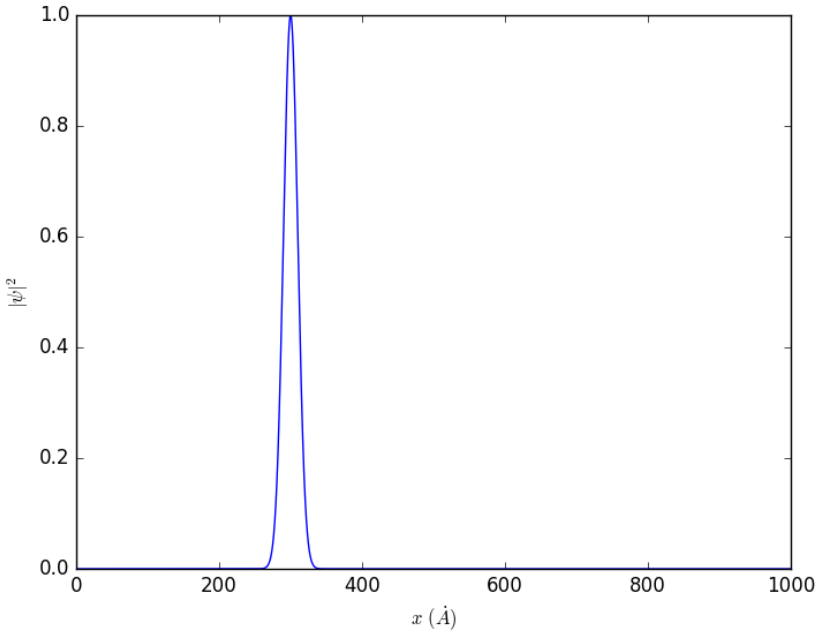

Figure 4: Square magnitude of wavefunction ψ as a function of position x at $t = 0$

2.2 Task 2

Task 2 consisted in using Crank-Nicholson to evolve the wavefunction in time, in order to be able to plot its square magnitude at $t_1 = 1e-14$, $t_2 = 2e-14$, $t_3 = 3e-14$, $t_4 = 4e-14$, and $t_5 = 5e-14$. In order to make sure that the value of Δt was correct, I ran the code using different values of it until the value of Δt did not affect the shape of the plots. Thus, I used $\Delta t = 1e-18\,sec$ since the plots for that Δt look the same as the plots for $\Delta t = 5e-18\,sec$ (and they make sense in the context of the problem). If I increased Δt further than that, the curves start to look different. I also checked the value of the integral for each of the plots (using Python) and found it to be constant to 8 significant figures with a value of 250.66628 for each of the plots. These are shown in Figures 5 through 9. The code used is given in section 4.2.

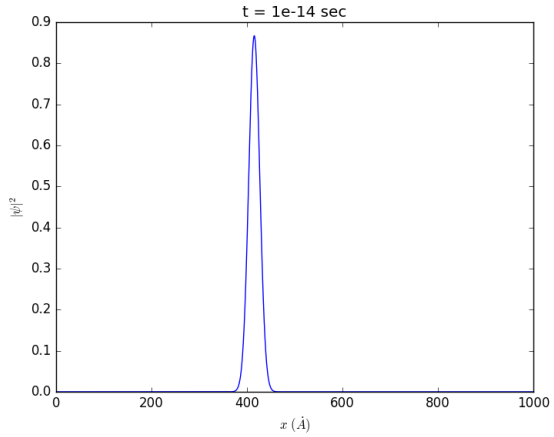

Figure 5: $|\psi|^2$ vs x at $t_1 = 1e-14\,sec$

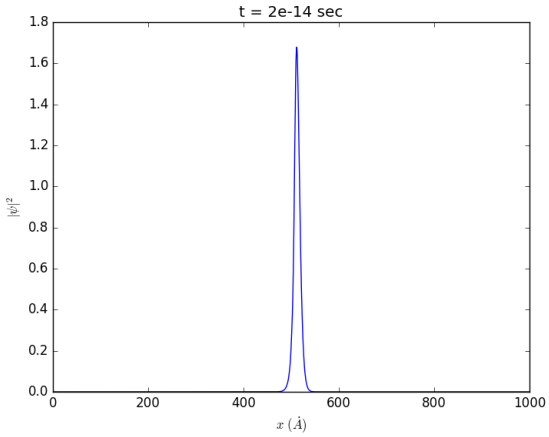

Figure 6: $|\psi|^2$ vs x at $t_2 = 2e-14\,sec$

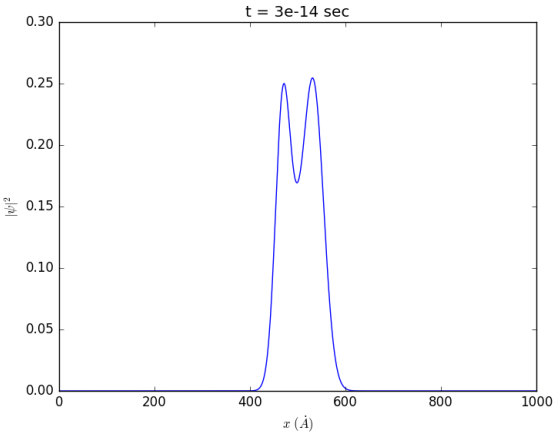

Figure 7: $|\psi|^2$ vs x at $t_3 = 3e-14\,sec$

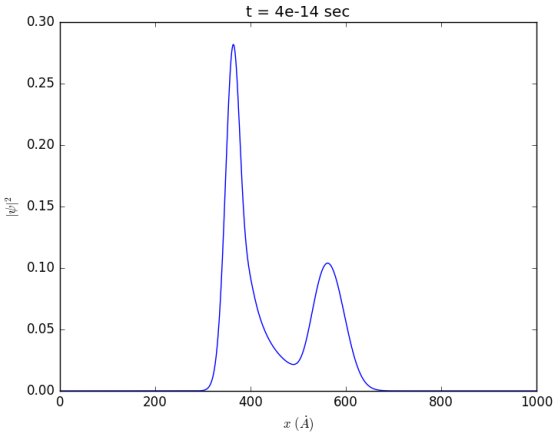

Figure 8: $|\psi|^2$ vs x at $t_4 = 4e-14\,sec$

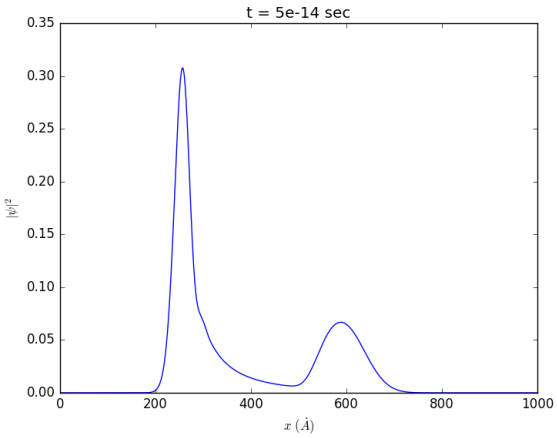

Figure 9: $|\psi|^2$ vs x at $t_5 = 5e-14\,sec$

3 Analysis

The results make sense. The potential $V(x)$ is essentially a step function, and its maximum energy is greater than the particle energy, we expect the particle to be able to propagate from left to right until it reaches the potential barrier, at which point we expect most of its probability amplitude (given by its wavefunction) to be deflected by the potential, and thus start propagating from right to left, while a fraction of the probability amplitude should continue propagating to the right to account for the possibility of tunneling. This is exactly what we see. The particle's wavefunction propagates from left to right with a shape that is unchanged from that with which it started, and then its shape changes once it reaches the potential barrier, dividing up into two pulses, one propagating from left to right, and the other from right to left, with the latter having a greater amplitude than the former.

4 Source code

4.1 hw7a.cpp

```
/* AEP 4380 HW#7a
   Time-dependent Schrodinger equation
   Run on core i7 with g++ 5.4.0 (Ubuntu)
   Gianfranco Grillo 11/01/2016
*/

#include <cstdio>
#include <cstdlib>
#include <cmath>

#include <iostream>
#include <fstream>
#include <iomanip>
#define ARRAYT_BOUNDS_CHECK
#include "arrayt.hpp"
#include <complex>
typedef complex<double> CMPLX;
```

```
using namespace std;

int main() {
    double deltax = 0.1, maxx = 1000.0, x;
    int j, xlength;
    xlength = maxx/deltax;
    arrayt<double> xlist(xlength);
    double pot(double);
    CMPLX initwf(double), wf0;
    ofstream fp;
    fp.open( "task1.dat" );

    for (j = 0; j < xlength; j++) {
        x = j*deltax;
        wf0 = initwf(x);
        fp << x << setw(15) << wf0.real() << setw(15) << wf0.imag() << setw(15) <<
            norm(wf0) << setw(15) << pot(x) << endl;
    }
    fp.close();
    return( EXIT_SUCCESS );
}

CMPLX initwf(double x) {
    double L = 1000.0, s = 20.0, re, im, y, m;
    y = exp(-(x-0.3*L)*(x-0.3*L)/(s*s));
    re = cos(x)*y;
    im = sin(x)*y;
    return CMPLX(re, im);
}

double pot(double x) {
    double L = 1000.0, omv = 7.0, V0 = 4.05;
    return (V0/(1.0+exp((0.5*L-x)/omv)));
}
```

4.2 hw7b.cpp

```
/* AEP 4380 HW#7b
  Time-dependent Schrodinger equation
  Run on core i7 with g++ 5.4.0 (Ubuntu)
  Gianfranco Grillo 11/01/2016
*/

#include <cstdio>
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <fstream>
#include <iomanip>
#include "arrayt.hpp"
#include <complex>
typedef complex<double> CMPLX;
```

```
typedef arrayt<CMPLX> arrayc;

using namespace std;

int main() {
    double deltax = 0.1, deltat, maxt = 5e-14, maxx = 1000.0, x, t, h = 6.5821e-16,
        m = 5.699e-32, w, par1, par2;
    int i, j, N, tlength, counter=0;

    cout << "Please enter time sampling in seconds: "; // ask for time sampling
        input
    cin >> deltat;

    N = maxx/deltax;
    w = 2.0*deltax*deltax/deltat;
    par1 = 2.0*m*w/h;
    par2 = 2.0*m*deltax*deltax/(h*h);
    tlength = maxt/deltat+1;
    arrayt<double> xlist(N), tlist(tlength);
    arrayc alist(N-1), blist(N), clist(N-1), dlist(N), psilist(N), clistor(N-1);
    void dlistcreator(arrayc&, arrayc&, double, double, arrayt<double>&);
    void tridiag(arrayc&, arrayc&, arrayc&, arrayc&, arrayc&);
    double pot(double);
    CMPLX initwf(double), wf0;
    ofstream fp;
    fp.open("task2.dat");

    for (j = 0; j < N; j++) { xlist(j) = j*deltax; }

    for (j = 0; j < tlength; j++) { tlist(j) = j*deltat; }

    for (j = 0; j < N; j++) { // initialize arrays
        x = xlist(j);
        wf0 = initwf(x);
        psilist(j) = wf0;
        blist(j) = CMPLX((-2.0 - par2*pot(x)), par1);
        if (j != N-1) { alist(j) = CMPLX(1.0, 0.0); clist(j) = CMPLX(1.0, 0.0); }
    }
    for (j = 0; j < tlength; j++) {
        t = tlist(j);
        if (counter == (tlength-1)/5 || counter == 2*(tlength-1)/5 || counter ==
            3*(tlength-1)/5 || counter == 4*(tlength-1)/5 || counter == (tlength-1))
             {
             for (i = 0; i < N; i++) {
                 fp << t << setw(15) << xlist(i) << setw(15) << norm(psilist(i)) <<
                     endl; //output to file only in five instances
             }
        }
        dlistcreator(dlist, psilist, par1, par2, xlist);
        tridiag(alist, blist, clist, dlist, psilist);
        counter++;
    }
```

```
    fp.close();
    return( EXIT_SUCCESS );
}

void dlistcreator(arrayc& dlist, arrayc& psilist, double par1, double par2, arrayt<
    double>& xlist) {
    double x;
    int N = psilist.n(), j=0;
    double pot(double);
    for (j = 0; j < N; j++) {
        x = xlist(j);
        if (j == 0) {
            dlist(j) = CMPLX((2.0 + par2*pot(x)), par1)*psilist(0)-psilist(1);
        }
        else if (j == (N-1)) {
            dlist(j) = CMPLX((2.0 + par2*pot(x)), par1)*psilist(j)-psilist(j-1);
        }
        else {
            dlist(j) = CMPLX((2.0 + par2*pot(x)), par1)*psilist(j)-psilist(j-1)-
                psilist(j+1);
        }
    }
    return;
}

CMPLX initwf(double x) {
    double L = 1000.0, s = 20.0, re, im, y, m;
    y = exp(-(x-0.3*L)*(x-0.3*L)/(s*s));
    re = cos(x)*y;
    im = sin(x)*y;
    return CMPLX(re, im);
}

double pot(double x) {
    double L = 1000.0, omv = 7.0, V0 = 4.05;
    return (V0/(1.0+exp((0.5*L-x)/omv)));
}

void tridiag(arrayc &alist, arrayc &blist, arrayc &clist, arrayc &dlist, arrayc &
    psilist) {
    int j, N;
    CMPLX bet;
    N = alist.n();
    arrayc gam(N);
    psilist(0) = dlist(0)/(bet=blist(0));
    for (j=1; j<N; j++){
        gam(j) = clist(j-1)/bet;
        bet = blist(j)-alist(j)*gam(j);
        psilist(j) = (dlist(j)-alist(j)*psilist(j-1))/bet;
    }
    for (j=(N-2); j >= 0; j--)
        psilist(j) -= gam(j+1)*psilist(j+1);
```

```
    return;
}
```

References

[1] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery *Numerical Recipes, The Art of Scientific Computing, 3rd edit.*, Camb. Univ. Press 2007.