# AEP4380 HW7
# Time Dependent Schrödinger Equation

Dan Girshovich

April 4, 2013

## 1 Overview

This document details the results of the included C++ program which solves the time dependent Schrödinger equation for an electron wave packet propagating through a potential barrier in one dimension. This is a second-order partial differential equation with one spatial variable and one time variable. To solve this numerically, a finite difference form of the equation is used with the Crank-Nicolson method (section 20.2, Numerical Recipes [1]). The core of the method is formulating the discrete values of $\psi$ at the next time step as solutions to a tridiagonal matrix equation. Note that all wave functions mentioned are unnormalized.

## 2 The Quantum System

### 2.1 The Potential Barrier

The electron interacts with the potential barrier described by

$$V(x) = \frac{V_0}{1 + e^{\frac{(0.5L - x)}{\omega_v}}},$$

where the length of the region $L = 1000$Å, the height of the barrier $V_0 = 4.05$eV, and the softness parameter $\omega_v = 7$Å. The barrier is shown in Figure 1.

### 2.2 Initial Conditions

The electron wave packet starts out with a Gaussian wave function

$$\psi(x, t = 0) = e^{-\left(\frac{x - 0.3L}{s}\right)^2 + ixk_0},$$

where $s = 20$Å and the wavenumber $k_0 = 1\text{Å}^{-1}$. The real and imaginary parts of $\psi(x, t = 0)$ are shown in Figure 2, along with the probability distribution $|\psi(x, t = 0)|^2$.

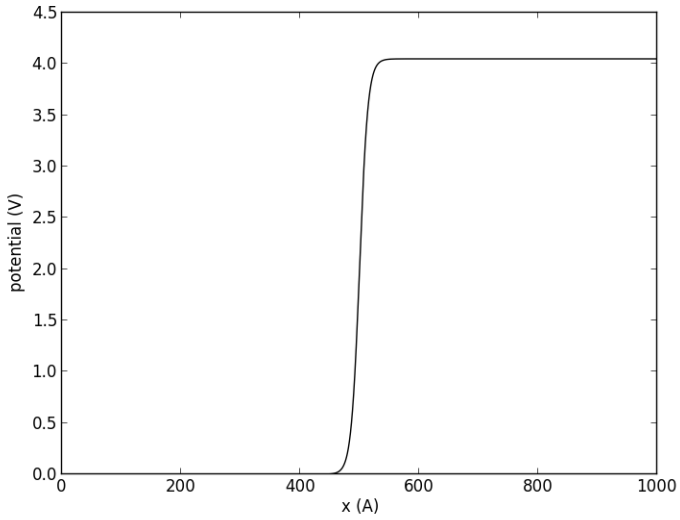



Figure 1: The potential barrier

### 2.3 Time Evolution

The wave packet dynamics are described by Schrödinger's equation:

$$i\hbar\frac{\partial}{\partial t}\psi(x,t) = -\frac{\hbar^2}{2m_e}\frac{\partial^2}{\partial x^2}\psi(x,t) + V(x,t)\psi(x,t).$$

Given this relation, $\psi(x, t = 0)$, and $V(x)$, $\psi(x,t)$ can be found for all $t > 0$ by numerical propagation.

## 3 Crank-Nicolson Method

### 3.1 Discretization

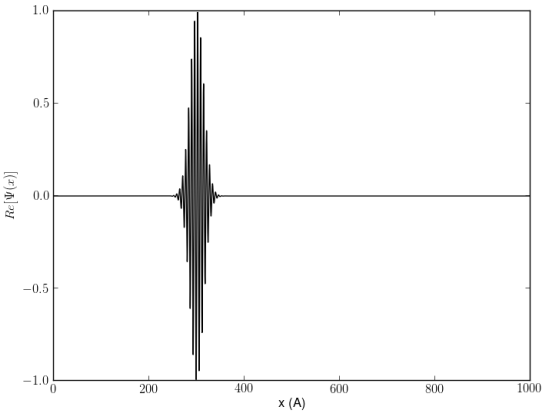
To solve $\psi(x,t)$ for all $t > 0$, the first step is to discretize space and time as follows:

$$x = j\Delta x,\ j = 0, 1, 2, ...j_{max}$$
$$t = n\Delta t,\ n = 0, 1, 2, ...n_{max}$$

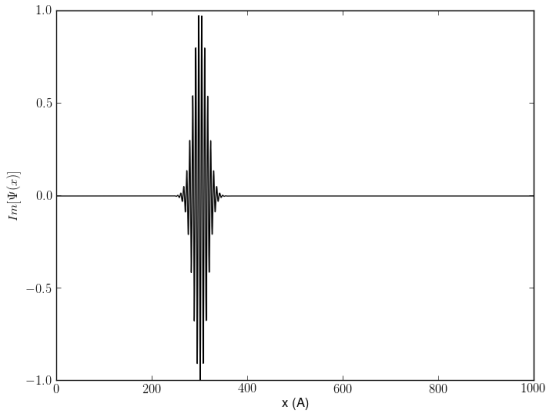
Then, Schrödinger's equation is also discretized into a finite difference form:

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

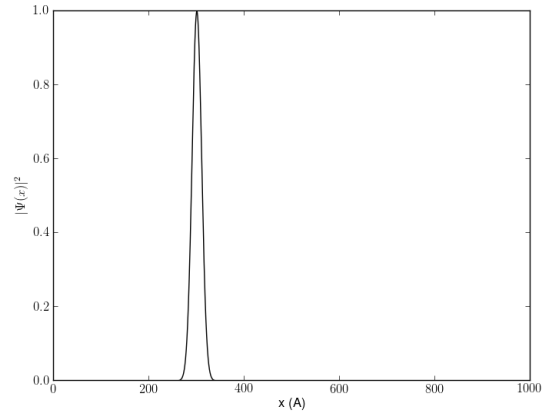
where $\omega = \frac{2\Delta x^2}{\Delta t}$. At time $t$, the right hand side of this equation is a known value for each $x$, while the three $\psi$ terms on the left hand side are unknown. This (Crank-Nicolson) setup is called fully implicit because of this quality.




(a) Real part




(b) Imaginary part




(c) Probability

Figure 2: Components of $\psi(x, t = 0)$ and the probability distribution. The electron is initially well localized at $x = 300$Å

## 3.2 Propagating a Solution

To solve for the values of $\psi$ and the next time step, finite difference equation is first rewritten as

$$\mathbf{M}\vec{\psi} = \vec{d}$$

where $\mathbf{M}$ is a tridiagonal matrix containing the (time independent) coefficients of the $\psi$ terms on the left hand side for each $x$. $\vec{d}$ contains the values of the right hand side of the equation for each $x$ at $t$. Finally, $\vec{\psi}$ has the values for $\psi$ at $t + \Delta t$ for each $x$. So, moving the solution forward in time is done by solving the matrix equation for $\vec{\psi}$.

A special technique can be used to do this which relies on the fact that the matrix is tridiagonal. This is discussed in section 2.4 of Numerical Recipes [1] and the code is given in the function `tridiag`. Instead of storing the entire matrix, the three non-zero diagonals are stored separately in arrays. These arrays, along with those for $\vec{d}$ and $\vec{\psi}$ at $t$ are passed as arguments. Then, $\vec{\psi}$ is modified in place to become $\vec{\psi}$ at $t + \Delta t$.

To find $\vec{\psi}$ at $t = t'$ given $\vec{\psi}$ at $t = 0$, this process is simply repeated $\frac{t'}{\Delta t}$ times. Note, this method also relies on fixed values of $\psi$ at the spatial boundaries, which are set as $\psi(0, t) = \psi(L, t) = 0$ in this case.

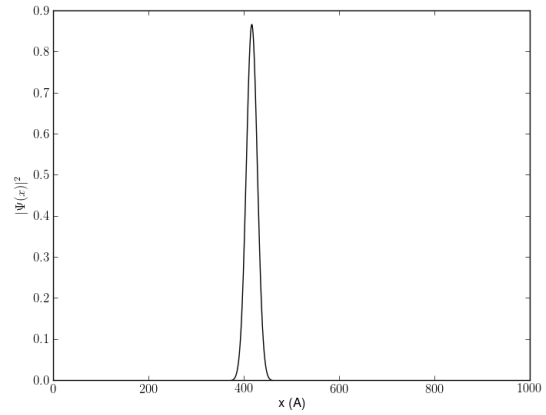
## 3.3 Finding $\Delta x$ and $\Delta t$

To avoid discretization error, $\psi$ should not change drastically over $\Delta x$ ($\Delta x << \frac{1}{k_0}$). For the system described, $\Delta x = 0.1$Å is small enough to meet this condition. To find $\Delta t$, $\psi(x, t)$ is computed for decreasing values of $\Delta t$. The largest value of $\Delta t$ that produces consistent output is assumed to be accurate enough. $\Delta t$ on the order of 0.01fs or lower gives consistent results for propagations on the order of 10fs in duration.

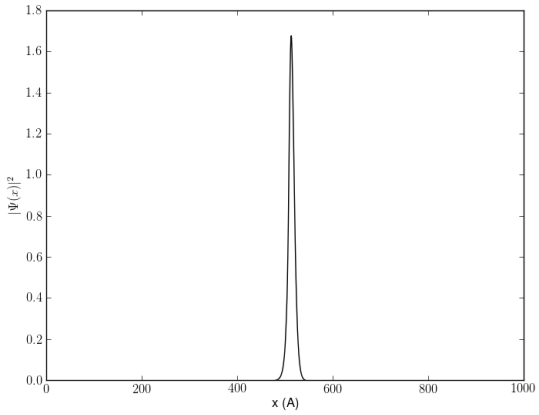
## 3.4 Results

$|\psi(x, t)|^2$ is plotted for $10 < t < 50$fs with steps of 10fs in Figure 3 using $\Delta x = 0.1$Å and $\Delta t = 0.01$fs. As expected, there is a reflected wave and a transmitted evanescent wave.

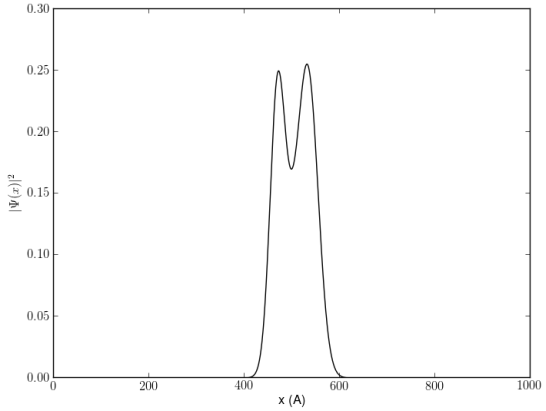
As an extra check, the rough numerical integration of $|\psi(x, t)|^2$ for each $t$ and $0 < x < L$ is performed. This physically corresponds to the (unnormalized) probability of finding the electron anywhere in the region, so it is expected to be constant. As expected, the integral has the constant value of 25.0663 for each $t$.



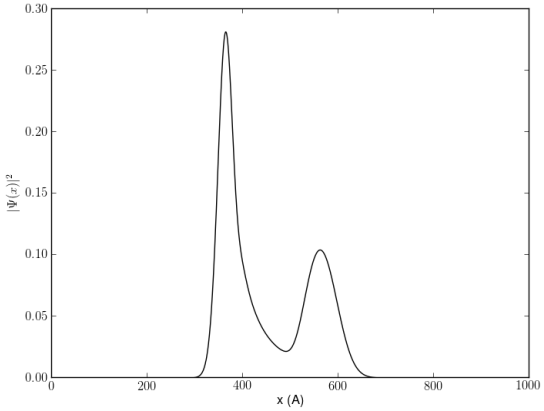

(a) 10 fs



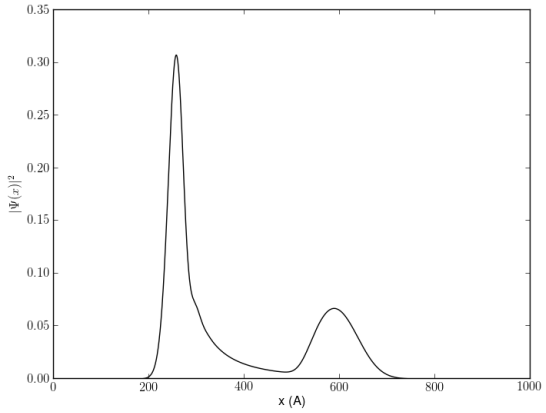

(b) 20 fs




(c) 30 fs




(d) 40 fs




(e) 50 fs

Figure 3: Propagation of $|\psi(x,t)|^2$ near the barrier in intervals 10fs apart.

# 4 Implementation

## 4.1 Overview

The code uses a modified version of `tridiag` that allows the use of complex data types and the bounds checking array class. This was achieved by replacing the `VecDoub` types with `array<T>` and passing in the STL `complex<double>` type for `T`.

The rest of the code is very straighforward, since the `a`, `b`, `c`, and `V` arrays are constant during the propagation. `phi` is updated in place by `tridiag` and `d` is refreshed in step with `phi` in `set_d`.

## 4.2 Source Code

```
/*
AEP 4380 HW #4
Dan Girshovich
4/4/13
Solves the time dependent Schrodinger Eq for the propagation of an electron wave
Compile with: g++ -std=c++0x -O2 -o gen_data hw7.cpp
Tested on Mac OSX 10.8.2 with a Intel Core 2 Duo
*/

#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <fstream>
#include <cmath>
#include <complex>
#include <functional>
// #define ARRAYT_BOUNDS_CHECK
#include "array.hpp"

typedef std::complex<double> cx;

// general constants
const cx i = cx(0, 1); // imaginary constant i
const double h_bar = 6.5821e-16; // eV-sec (Planck's constant / 2 pi)
const double k = 3.801; // eV-A^2 (h_bar^2 / 2m for the electron)

// for the potential
const double L = 1000; // A (length of region)
const double V_0 = 4.05; // eV (height of potential barrier)
const double omega_v = 7; // A (softness of potential barrier)

// for the wavefunction
const double s = 20; // A (width of Gaussian wave function)
const double k_0 = 1; // A^-1 (average wavenumber)

// for the propagation
const double dx = 0.1; // A sample size in space
```

```
const double dt = 0.01; // fs sample size in time
const double omega = 2 * dx * dx / (dt * 1e-15);
const int j_max = L / dx;
array<double> V = array<double>(j_max); // V(x) potential
array<cx> psi = array<cx>(j_max); // psi(x) wavefunction at some time

// elements in matrix equation for propagation
array<cx> a = array<cx>(j_max);
array<cx> b = array<cx>(j_max);
array<cx> c = array<cx>(j_max);
array<cx> d = array<cx>(j_max);

// psi is held at 0 at the ends
cx get_psi(int j) {
    return (j == -1 || j == j_max) ? cx(0, 0) : psi(j);
}

// sets the values for d based on psi's current state (in time)
void set_d() {
    for (int j = 0; j < j_max; j++) {
        cx z = (omega * i * h_bar + 2 * k + dx * dx * V(j)) / k;
        d(j) = -1.0 * get_psi(j - 1) + z * get_psi(j) - get_psi(j + 1);
    }
}

// gets a, b, c, d, V, and psi ready for propagation
void init_arrays() {
    for (int j = 0; j < j_max; j++) {
        double x = j * dx;
        V(j) = V_0 / (1 + exp((0.5 * L - x) / omega_v));
        double z = (x - 0.3 * L) / s;
        psi(j) = exp(-1 * z * z + i * x * k_0); // exp is overloaded for complex
        a(j) = cx(1, 0);
        b(j) = (omega * i * h_bar - 2 * k - dx * dx * V(j)) / k;
        c(j) = cx(1, 0);
        set_d();
    }
}

// From section 2.4 of Numerical Recipes. Modified to use template for matrix
// types and bounds checking array class.
template<class T>
void tridag(array<T> &a, array<T> &b, array<T> &c, array<T> &d, array<T> &u) {
    int j, n = a.n();
    T bet;
    array<T> gam(n);
    if (b(0) == 0.0) throw("Error 1 in tridag");
    u(0) = d(0) / (bet = b(0));
    for (j = 1; j < n; j++) {
        gam(j) = c(j - 1) / bet;
        bet = b(j) - a(j) * gam(j);
        if (bet == 0.0) throw("Error 2 in tridag");
        u(j) = (d(j) - a(j) * u(j - 1)) / bet;
    }
```

```
    for (j = (n - 2); j >= 0; j--)
        u(j) -= gam(j + 1) * u(j + 1);
}

void output_potential() {
    std::ofstream f;
    f.open("potential.dat");
    for (int j = 0; j < j_max; j++)
        f << j *dx << " " << V(j) << "\n";
    f.close();
}

void output_psi(std::string fname) {
    std::ofstream f;
    f.open(fname.c_str());
    for (int j = 0; j < j_max; j++) {
        cx p = psi(j);
        double x = j * dx;
        f << x << " " << p.real() << " " << p.imag() << " " << norm(p) << "\n";
    }
    f.close();
}

// simple Reimann sum to find the total probability
double integral_norm_psi() {
    double integral = 0;
    for (int j = 0; j < j_max; j++)
        integral += norm(psi(j)) * dx;
    return integral;
}

int main() {
    init_arrays();
    output_potential();
    double t = 0;
    for (int n = 0; n <= 5; n++) {
        std::stringstream fname;
        fname << "psi" << n << ".dat";
        output_psi(fname.str());
        std::cout << integral_norm_psi() << "\n";
        // propagation
        for (double t_next = t + 10; t < t_next; t += dt) {
            tridag(a, b, c, d, psi);
            set_d();
        }
    }
}
```

# References


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