PHYS 340 Final Project:
Quantum Localization in a 1-D Random Potential

Prof. Francis Starr

December 2014

1 Introduction

In the final project, you will examine the behavior of the Schrödinger equation for a particle in a random potential, which results in an unexpected behavior for the time evolution of an initial wave packet. Specifically, we will show how a simple model of a 1-D crystal with random values of the potential at the crystal sites results in a localization of the wave function. This phenomenon is known as "Anderson localization", as it was first discussed in Nobel laureate Philip Anderson's famous work ["Absence of Diffusion in Certain Random Lattices", Physical Review **109**, 1492-1505 (1958)].

Practically speaking, the final project combines the determination of stationary states of the Schrödinger equation (a boundary value problem) with earlier work on initial value problems to track the evolution of a wave function. Previously, we determined the eigenfunctions (stationary states) and energy levels of the 1-D square-well potential via the shooting method, and you will build on that here. Similarly, we will also utilize the evolution methods we developed for the harmonic oscillator and gravitational motion.

Your task for the final project consists of writing and reporting the results for two separate, but closely related codes:

Eigenfunctions and Energy Levels: You will numerically determine these for our simple model crystal. This is a boundary value problem.

Evolution: Starting with a Gaussian wave packet (which you can think of as adding a particle to the lattice), you will track the evolution of the wave function in the crystal lattice. This is an initial value problem.

We will examine two cases for our simple 1-D crystal potential: (i) uniformly spaced, identical potential wells, and (ii) uniformly spaced, but random depth of the potential wells. For the random case, we will use the same mean well depth as the case of identical wells, to facilitate comparison. When the wells are identical, you should find that the wave function spreads, but when you introduce randomness, there is a non-intuitive localization of the wave function.

In this document, I explain the mathematical system, discuss the practical elements of implementing the problem numerically, and finally give a detailed explanation of what you must turn in.

The project is due at 11:59PM on Friday December 12. This is a strict deadline! This allows around 2.5 weeks to complete the project, including two regular class periods. Thus, barring truly exceptional circumstances, no project will be accepted past this deadline. This is more time than I anticipate you will need. However, if you wait until the last minute, this will be more work than you could possibly hope to complete. Hence, I encourage you to start early. In particular, note that you will need to do some numerical experiments with your code and write a brief report. Accordingly, you need to leave some time for this, in addition to writing and **debugging** the code.

2 Formal Description of the Problem

2.1 Eigenfunctions and Energy Levels

This project focuses attention on the Schrödinger equation, both time independent and time dependent. The time-independent Schrödinger equation (TISE)

$$\mathcal{H}\psi(x) = E\psi(x) \tag{1}$$

determines the eigenfunctions $\psi(x)$ and allowable energies $\{E\}$ of a system, which are defined by the Hamiltonian $\mathcal{H}$. For quantum systems, $\mathcal{H}$ is an operator that has the form (in 1-D)

$$\mathcal{H} = -\frac{p^2}{2m} + V(x) \tag{2}$$

$$= -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + V(x). \tag{3}$$

Thus, the problem is completely determined by the potential $V(x)$. From this point forward, we use reduced units where $\hbar = m = 1$, and so $\hbar^2/2m = 1/2$.

For the final project, we study the behavior of a particle in a 1-D crystal. To keep things simple, we limit the spatial range of the crystal to $0 \le x \le L$ – in other words there will be infinite barriers at 0 and L, like the square-well problem.

The sites of crystal lattice are evenly spaced at $x = 1, 2, 3, ...L-1$ (assuming $L \in \mathcal{Z}$), and the particle will be attracted to the lattice sites. The simplest representation for the attraction to the site is a Dirac-δ function $\delta(x)$. Hence the overall form for the potential is

$$V(x) = \begin{cases} \infty & ; \quad x < 0 \\ -\sum_{n=1}^{L-1} V_n\,\delta(x-n) & ; \quad 0 \le x \le L \\ \infty & ; \quad x > L \end{cases}, \tag{4}$$

where the minus sign is because we take the strength of the δ function $V_n > 0$.

While we cannot technically draw a delta function, it is useful to consider the finite discretized version of the potential shown in figure 1. To solve the TISE, we proceed using any method designed for boundary value problems, like the shooting method. We will use this same potential to track the time evolution of an initial arbitrary wave packet.

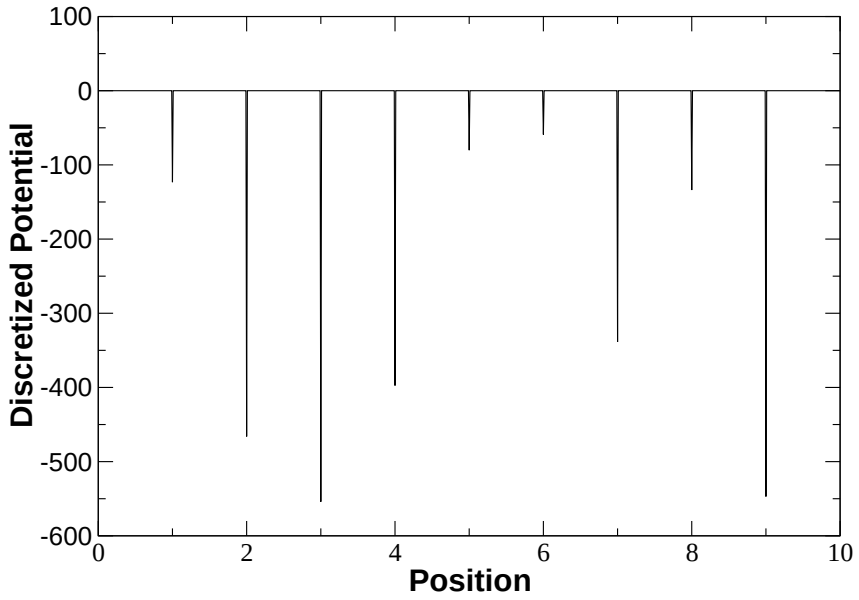

Figure 1: Example of a potential consisting of random δ functions. For illustration purposes, the δ functions have a finite width and depth.

2.2 Evolution

To track the evolution of some initial state $\phi(x, t = 0)$ – which we distinguish from $\psi(x)$, the eigenfunctions – we must examine (in reduced units) the time-dependent Schrödinger equation (TDSE)

$$\mathcal{H}\phi(x,t) = i\frac{\partial\phi(x,t)}{\partial t}. \tag{5}$$

Recognizing that ϕ is a complex function, we can define

$$\phi(x,t) = \mathcal{R}(x,t) + i\,\mathcal{I}(x.t) \tag{6}$$

where $\mathcal{R}$ is the real part of ϕ, and $\mathcal{I}$ is the imaginary part. Using the definition of $\mathcal{H}$, a simple rearrangement of the TDSE yields the following pair of coupled, first-order partial-differential equations:

$$\frac{\partial\mathcal{R}(x,t)}{\partial t} = \left[-\frac{1}{2}\frac{\partial^2}{\partial x^2} + V(x)\right]\mathcal{I}(x,t) \tag{7}$$

$$\frac{\partial\mathcal{I}(x,t)}{\partial t} = \left[\frac{1}{2}\frac{\partial^2}{\partial x^2} - V(x)\right]\mathcal{R}(x,t) \tag{8}$$

These equations represent an initial value problem (like the gravitation assignment), which we can solve using one of the integration schemes we have discussed. Your task will be to evolve these equations for many different realizations of the random potential to obtain the ensemble "mean" evolution.

Note that for the TDSE, there is actually a better approach than direct integration that is based on a matrix formulation, and can be solved very rapidly by fast-Fourier transform. However, for pedagogical purposes, we will do direct numerical integration.

3 Computational Implementation

Since the process of finding the eigenfunctions ψ is completely separate from the evolution of the initial state ϕ, I suggest writing two separate codes. Feel free to draw upon the earlier exercises, including solutions that I have provided. This should particularly help to expedite the calculation of the eigenfunctions and energies.

3.1 Eigenfunctions and Energy Levels

I suggest using as a starting point the recent exercise where we determined the eigenfunctions and energies for the square-well potential. The computational procedure will be the same here, except that we use the more complicated potential given in the previous section. Since we must discretize space into intervals of size dx, we need a discrete representation of the Dirac-δ function.

3.1.1 Discrete Representation of $\delta(x)$

The first problem that arises is how to implement the δ function potential used in equation 4. Loosely, we think of the δ function as having an infinite spike at a single location. Mathematically, the Dirac-δ function is only defined by its integral

$$\int_{-\infty}^{\infty} \delta(x)\, dx = 1. \tag{9}$$

Since we discretize space, we will need to use a Kronecker-δ function

$$\delta_{ij} = \begin{cases} 1 & ; \quad i = j \\ 0 & ; \quad i \neq j \end{cases} . \tag{10}$$

A direct substitution of δ_{ij} for $\delta(x)$ leads to problems, since $\delta(x)$ is defined by the integral. To illustrate this, suppose we convert the integral of equation 9 to a numerical integral using the trapezoidal rule, *i.e.*

$$\int_{-\infty}^{\infty} \delta(x)\, dx \to \sum_{n=-\infty}^{\infty} A\, \delta_{0n}\, dx = A\, dx = 1. \tag{11}$$

We must choose $A = 1/dx$ in order to get the correct answer for the integral, where dx is the size of our discretization. Naïvely choosing $A = 1$ would not yield the correct value for the numerical integral. So if we use the trapezoidal rule to discretize, we have

$$\delta(x) \to \frac{1}{dx}\delta_{0n}. \tag{12}$$

I reiterate that the value of prefactor when you discretize depends on the numerical method you use to integrate! Thus, if you wish to use the above discretization, you must use the trapezoidal rule for integration. You are free to use another integration technique if you wish, but be sure to work out the correct multiplicative factor to preserve the integral.

3.1.2 Parameter Choices

In your solution, I require that you use some specific choices of parameters. This will make your life easier, since, for example, you will not have to determine what discretization provides a good approximation.

Uniform wells: First, choose $L = 10$ and $dx = 0.01$. This means your wells will be located at $x = 1, 2, ...9$. For the first case where all δ functions have the same amplitude, take $V_n = 3$. Since have hard walls, there will be normalizable bound states. Since there are also δ function wells, it is possible that $E < 0$. For the tolerance on the convergence of the energy, use a value of 10^{-8}.

If you wish to check if your energy values are reasonable, consider the limit of a single δ function when $L \to \infty$. For this case, we know analytically that the ground state solution has $E = -V_0^2/2 = -4.5$. If you use a single delta function in the middle with $L = 10$, the finite system size raises the minimum slightly to ≈ -4.46. For the case with nine δ function wells, the lowest energy is lower; the lowest lying state I found has energy ≈ -5.187. A sample bound state wave function (for a different energy) is shown in figure 2. If you find lower energy states, be careful to check that they have a normalizable wave function (so that they are physically meaningful).

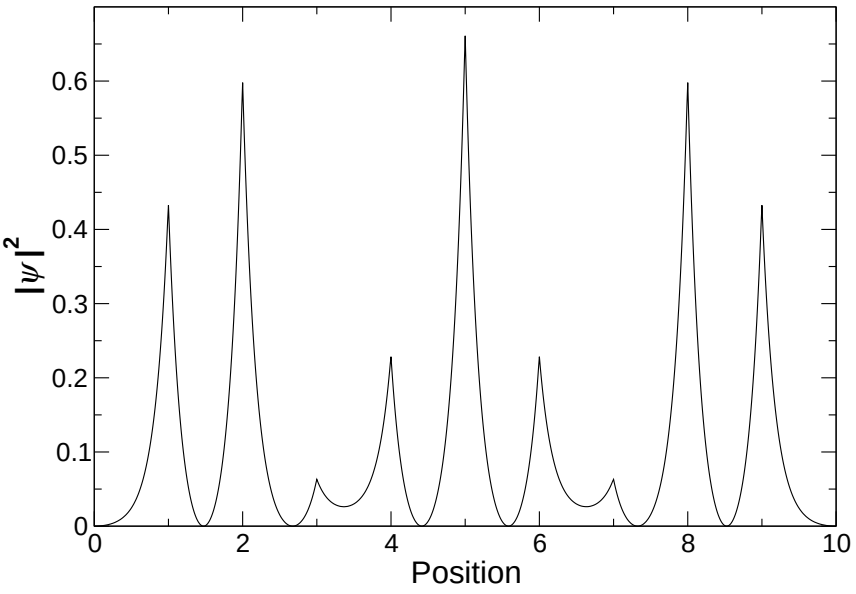

Figure 2: A sample bound state wave function for the uniform potential. This state has energy ≈ -3.8.

Random wells: For the case of wells with random depth, we pick V_n using a random number generator. I suggest using the random number generator `erand48` which requires and array of 3 seeds of type `unsigned short`. `erand48` generates random values in the range $[0, 1)$. Therefore, you will need to scale the values from `erand48` so that $\langle V_n \rangle = 3$ – the same mean as for the uniform potential. It should be apparent that you will not know by how much to scale the values until they are picked. If you want to see an example of the use of the random number generator, see the Monte Carlo code you worked on. Like the case of uniform wells, there will be bound states with

negative energy, but the lowest energy bound state will likely differ.

For both the uniform and random case, I found it convenient to define an array that stores the potential at each lattice site at the start of the code. If you do so, you can use the same section of code to define the potential for the evolution code.

3.1.3 Code requirements

Your code must prompt the user whether to study wells of uniform or random depth, and the starting energy to begin looking for solutions. The code must generate a file named `psi.dat` that contains 3 columns. The first column should be the position x, followed by the wave function $\psi(x)$, and the probability $|\psi(x)|^2$. Of course, your wave function must be normalized. In other words, be sure you enforce the condition

$$\int_0^{10} |\psi(x)|^2 dx = 1. \tag{13}$$

In addition, generate a file named `potential.dat` that has two columns, the position x and the potential $V(x)$. This is just a check to be sure you correctly generate the potential.

3.2 Evolution

Just like the determination of the eigenfunctions and energy levels, your evolution code must be able to solve for the case of wells with either uniform or random depth. You should be able to cut and paste your definition of the potential from the previous code (which will also help avoid making new mistakes!). Unlike the previous code, equations 7 and 8 show that we need to explicitly follow the behavior of the real and imaginary parts of the wave function, as they will feed off each other.

3.2.1 Parameter Choices

Just like the eigenfunction code, choose $L = 10$ and $dx = 0.01$. Choose your potential also exactly as the previous code. Again, be sure you can do either wells of constant or random depth. The only additional parameter you need is the discretization of the time; choose $dt = 10^{-4}$.

3.2.2 Initial Wave Packet

For initial conditions, we will start with a Gaussian wave packet of the form

$$|\phi(x,0)|^2 = A \exp\left(-\frac{(x - L/2)^2}{2\sigma^2}\right) \tag{14}$$

In order that $|\phi(x,0)|^2$ has this form, take the real part $\mathcal{R}(x,0) = \sqrt{|\phi|^2}$ and the imaginary part $\mathcal{I}(x,0) = 0$. Choose $\sigma = 0.2$. Imposing the normalization condition requires that

$$A = [\sqrt{2\pi}\ \sigma\ \mathrm{erf}(L/(2\sqrt{2}\sigma))]^{-1}, \tag{15}$$

where $\mathrm{erf}(x)$ is the error function. The C math library includes the error function. A plot of this initial state is shown in fig. 3.

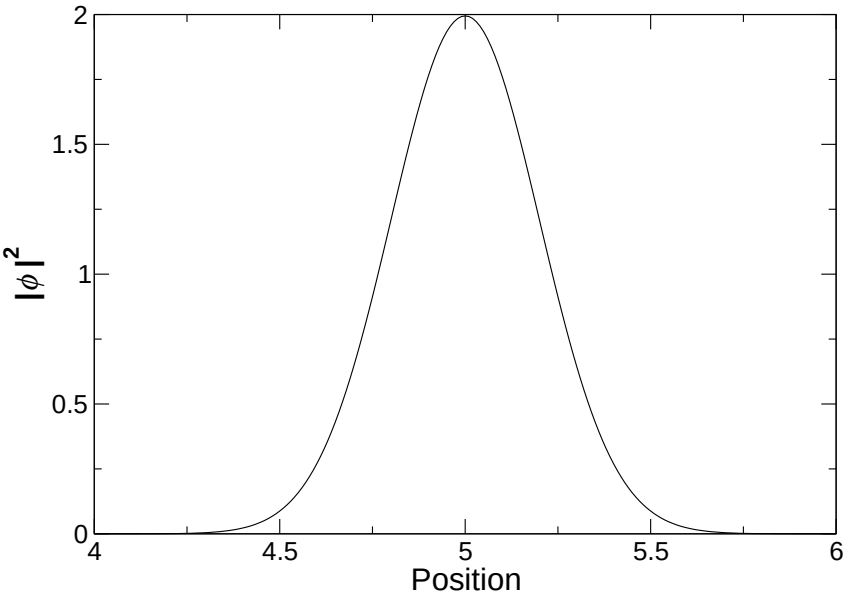

Figure 3: The initial Gaussian wave packet.

3.2.3 Numerical Integration

The main task is the numerical integration of equations 7 and 8. Since you have ϕ defined as a discrete function, evaluating the spatial derivative required by these equations can be immediately done using the numerical approximation to the second derivative.

Previously, we have worked with three different evolution algorithms: (i) the Euler method, (ii) the Runge-Kutta method, and (iii) the velocity-Verlet method. Unfortunately, method (i) and (ii) do not preserve the normalization (*i.e.* integral $|\phi|^2$ is not conserved), and method (iii) is formulated specifically for differential equations in the form of Newton's equations. You could potentially reformulate the velocity-Verlet method, but I recommend using another simple scheme known as the "leapfrog" method that will preserve the normalization. Formally, when applied to equations 7 and 8, the leapfrog method becomes

$$\mathcal{I}(x, t + dt/2) = \mathcal{I}(x, t - dt/2) + dt \frac{\partial \mathcal{I}(x,t)}{\partial t} \tag{16}$$

$$\mathcal{R}(x, t + dt) = \mathcal{R}(x, t) + dt \frac{\partial \mathcal{R}(x, t + dt/2)}{\partial t} \tag{17}$$

By paying attention to the times, you can see the name leapfrog comes from the fact that $\mathcal{R}$ and $\mathcal{I}$ are always a half step apart, and keep passing each other. This scheme works well for this problem, because we need $\partial\mathcal{R}/\partial t$ at a half step ahead of $\mathcal{R}$, and equation 7 shows that $\partial\mathcal{R}/\partial t$ is a function of $\mathcal{I}$, which is already half a step ahead! The same is true about the integration of $\mathcal{I}$.

The only problem with this scheme is, "how do we start?", since we need $\mathcal{R}$ and $\mathcal{I}$ separated by a half step? This simple answer that at the start of the simulation, we can take $\mathcal{I}$ either a half

step backward or forward using the Euler method once. Then we use the leapfrog approach for all subsequent steps. After you have finished all the time integration steps, before your program finishes, do one more Euler step to get $\mathcal{R}$ and $\mathcal{I}$ at the same time again. Having just two Euler steps out of the many steps via the leapfrog method will not result in significant error.

Note that if you ignore my advice and use the Runge-Kutta method, I have found that the time step dt needs to be smaller than that used for leapfrog for Runge-Kutta to be stable.

Finally, when you do the evolution, be careful about the boundaries. For hard walls, the values of ϕ at $x = 0$ and $x = L$ will never change from 0. If you are feeling adventurous, you might try using periodic boundary conditions instead, *i.e.* use boundary conditions $\phi(x = 0) = \phi(x = L)$. This means that lattice wraps around on itself. Such boundary conditions mean you are eliminating the wall at the boundaries, and you should put another well at $x = 0$ to keep the spacing between wells fixed. The use of periodic boundary conditions reduces any finite size effects that may be present. Periodic boundary conditions are *not* required for a full score on the project, but offer a chance to dig deeper, if you wish.

3.2.4 Code requirements

Your code must prompt the user whether to study wells of uniform or random depth, the number of time steps to evolve the system, and the value of at least one of the three seeds. It is important that the user be able to supply changing seeds; otherwise the random number generator will always supply the same random numbers. The seeds for the random number generator are of type `unsigned short`; the correct formatting for scanning this data type is something like

```
scanf("%hu", &seed[0]);
```

The code must generate a file named `psi.dat` that contains 4 columns. The first column should be the position x, followed by the real and imaginary parts of wave function $\phi(x)$, and the probability $|\phi(x)|^2$. The file should contain the values obtained at the end of the evolution.

3.2.5 What should happen?

Hopefully you recall that a free Gaussian wave packet will spread. The same is true when you have wells of uniform depth, but you will notice that as it spreads there is an enhancement of the wave function near the wells. The situation changes when the wells have a random depth: the wave function becomes localized around a particular position.

Another question that comes up when you run your simulation: how long does the simulation need to be? Answer: long enough for the localization to occur! In practice, I found that around 10^5 time steps (for our choice of dt) is enough. As with all parameters, I encourage you to play and see what affect your choices have.

For a single specific choice of random well depths, the localization may not be that pronounced. However, you should still be able to tell a difference from the case where the wave function spreads. By chance of playing with the seeds for the random number generator, I discovered that the seeds 749, 8734, 270 give a very obvious localization (recall that `erand48` takes an array of 3 seeds), if you pick your wells in same order I do – that is, using the first 9 random numbers to determine the 9 well depths. Trying this seed combination may help you decide if things are going well.

4 Reporting your Results

By the **deadline of 11:59PM on Friday December 12** you must turn in your code and a final report. This report should be concise, but should be clear and well presented and should consist of three sections. Except for compiling and running the code, the report must be **completely self contained as a pdf file**. In other words, all text and figures should appear in this pdf file.

In the first section you should describe the problem and the method you have used to solve it. Any detail about computational techniques and algorithms should also be presented there. For example, you should specify what integration scheme you used, what you did for the boundary conditions, *etc.* However, you need not re-write all the material I have presented. I wrote it, so I do not need to read it again!

In the second section you should explain how one compiles and runs your code. In particular, the information on how to use your code, should be clear and exhaustive. The user should be informed on how to compile and link, about any parameters that may have to be specified on input, a description of any files (beyond those required) that will be produced, what information they contain, *etc.* In other words, nothing should be left for the user/grader to guess or to decipher from your code.

The third section should provide a summary of your results. Specifically, I expect to see at least the following:

Eigenfunctions and Energy Levels:

1. For both the uniform and random wells, make a plot of the energy levels as a function of "level"; you can label your levels $1, 2, ..., n$. Go up to at least $n = 10$. Since you do not know what the states are ahead of time, be careful that you do not to skip any states. Note that for the random potential, the states you find depend on the seed chosen for the random number generator. Therefore, be sure you use the same seed as you search for energies. You will notice that the energy of some states for the uniform and random potentials are the same. Explain why. (Hint: look at the wave function for these states and think about the potential).
2. Plot at least 2 wave functions for each system. Try to explain your observed wave functions qualitatively, and relate them the the wave functions of the square-well potential, if possible.

Evolution: I wish to see a demonstration of the localization. As previously mentioned, the localization for a single run may be weak. However, if you average the final behavior of $|\phi|^2$ over many different choices for the random wells, you will find that $|\phi|^2$ is exponentially localized around its initial position. For each run, be sure you run for at least 10^5 time steps — or an equivalent longer time if you use a smaller dt than I suggested. You should evolve the wave function for at least 10 different choices of random well depths. For each attempt, save the resulting wave function, and then average them together. Figure 4 shows my results when I averaged over many attempts. Your figure will be different for different seeds, but you should see the exponential behavior near the center indicated in the figure. Since you are attempting to show exponential localization, be sure to use a logarithmic scale for the $|\phi|^2$ axis.

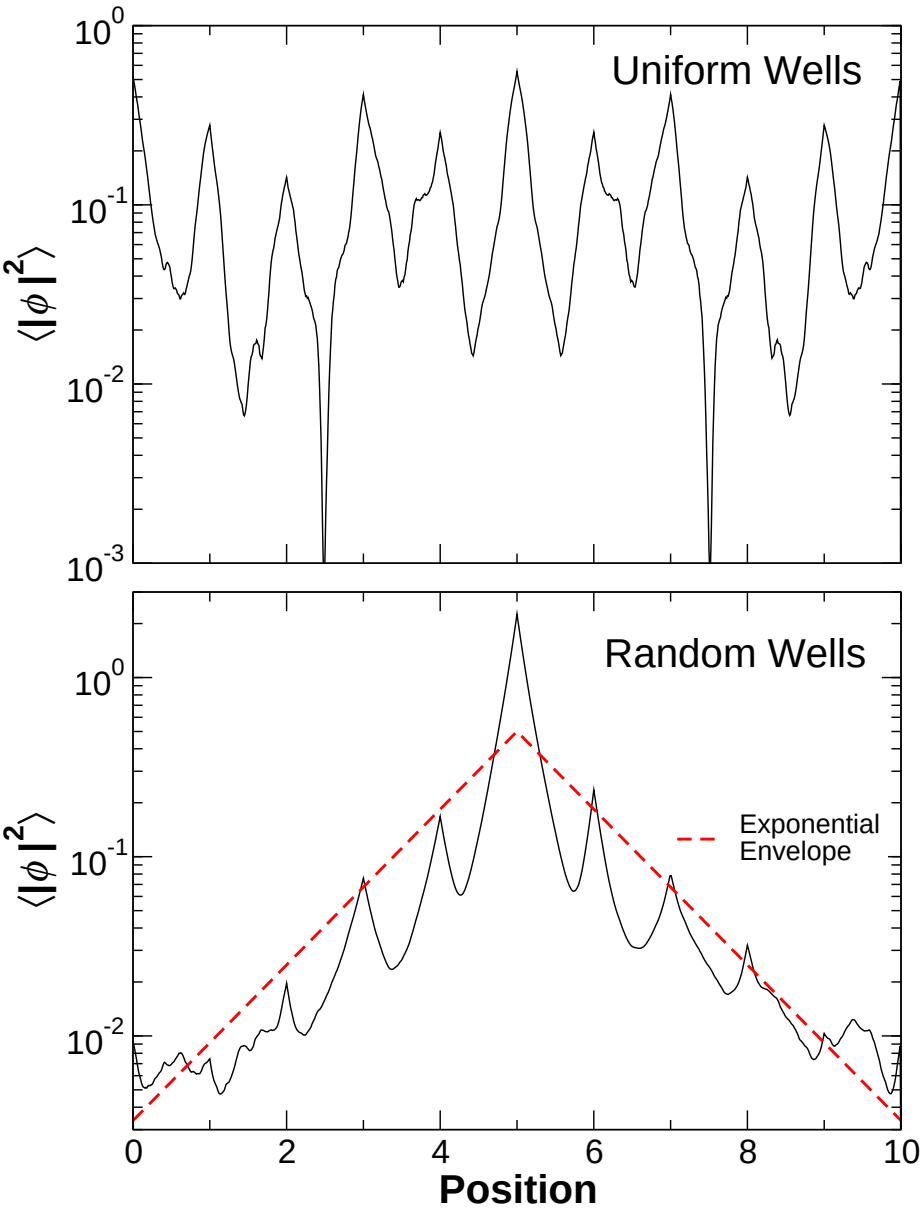

Figure 4: Mean behavior of the probability $|\phi(x)|^2$ for the uniform wells (top) and for wells with random depth (bottom). The results are averaged over 100 realizations of the random potential. The dotted red line indicates the approximate exponential envelope of the localization. Note that is localization is *not* the small peaks of $|\phi(x)|^2$, which appear for both potentials.

This report and code should be turned in electronically to the directory

`~fstarr/phys340/`*username*`/final`

where you should obviously replace *username* with your username. The date on the files will guarantee that they were turned in on time.

Barring truly exceptional circumstances, no project will be accepted past the deadline. I encourage you to start working on the final immediately. If you are unable to complete your work

by the deadline, send me whatever you have done.

Again, the report should be concise, but clear. A good presentation, a clearly written and efficient code and correct results are sufficient for an A; a mark of A+ will be reserved for exceptional projects. Turning in a massive tome of extraneous information will result in a small decrease of your score! Choose your words and figures wisely to make a clear and compelling presentation.

If you would like to try to use LaTeX to prepare your report, the LaTeX source for this document is available as an example. Of course, you may use whatever tool you like, so long as the final report is a completely self-contained pdf document.

Good luck. Please feel free to contact me with questions. I hope you enjoyed this course and that you learned a few things.

HAPPY COMPUTING!

5 Appendix: Optional Visualization Code

For the evolution of the wave function, it can be helpful to be able to watch the evolution of $\phi(x,t)$ in real time. Hence, I am making available a simple visualization code similar to what we used for the gravitation assignment. The code is available in ~fstarr/final/plot2d.c. This is *not* a requirement!

There are two subroutines:

```
void init_window(int size, int *argc, char **argv);
void plot2d(int nsteps, int size, double *phi);
```

The `init_window` subroutine opens a new window. Hence it only needs to be called once at the beginning of your code. The arguments are the `size` of the window in pixels (around 500 is good), `argc` a pointer to an integer, and `argv` a double pointer of character type. A sample call of this subroutine is

```
init_window(500, &argc, argv);
```

assuming that `main` takes as arguments `argc` and `argv`, as we have done for all our codes.

The `plot2d` subroutine plots an array `phi`. The arguments are the size of the array `nsteps`, the size of the window `size` you specified in `init_window`, and the array `phi` you want to plot. This array could be the real part, imaginary part, or square of the wave function $\phi(x)$. A sample call of this subroutine is

```
plot2d(nsteps, 500, phi);
```

Call `plot2d` when you want to update the rendering of `phi`. You should avoid calling `plot2d` every time step, or you code will run extremely slowly.

To compile with this code, use a compile statement something like
`gcc -I/usr/include/GL final.c plot2d.c -o final -lglut -lGL -lX11 -lm`