Users Guide: Quantum Well Solver

Charlie Duke, Emeritus Professor of Physics, Grinnell College*

November 10, 2020

Abstract

This quantum-well solver, written with python3, numerically solves the time-independent, one-dimensional Schrodinger equation for potential wells defined by the user. Well types include finite and infinite square wells, square wells separated by potential barriers, with sloping bottoms, and with harmonic-oscillator shapes. The GUI windows show the potential well and the wavefunction for a specific energy set by the user. Or, the solver can determine stationary-state energies and wavefunctions within a specified energy range. The python data class containing all well parameters and the numerical values of a selected stationary state wavefunction and its derivative may be sent to an output file for later use. Little prior knowledge of quantum mechanics is necessary, and no prior experience with python is required. The QuickStart section in the README file and in this document provides sufficient information for installation and usage.

Many of the QuantumWell python modules can be used in Jupyter notebooks or in python scripts for more advanced work such as following the time development of composite wavefunctions in multibarrier wells, investigating the development of energy bands multibarrier wells, and looking for bound states in simple one-electron molecules. Such projects, requiring a basic knowledge of python, are supported by the jupyter tutorial notebooks in this package and are appropriate for students at both the introductory and advanced level.

This project, initially completed during the summer and fall of 2014, was written in python3 with PyQt based GUIs. The Anaconda3 package from Continuum Analytics contains all code dependencies. The solver runs on MacOS, Windows7/10, and Linux operating systems.

The package is fully licensed under gnu_gpl_v3 as detailed in the included LICENSE file.

Contents

*Physics Department, Grinnell College. Grinnell, IA 50112 USA

List of Figures

1 Introduction

The solution to the one-dimensional, time-dependent Schrodinger equation,

$$-\frac{\hbar^2}{2m}\frac{\partial^2\Psi(x,t)}{\partial x^2}+V(x)\Psi(x,t)=i\hbar\frac{\partial\Psi(x,t)}{\partial t}$$

is the probability amplitude, $\Psi(x,t)$, for a particle of mass,m, as a function of position and time. After variable separation with $\Psi=\psi(x)\theta(t)$ and with the energy, E, as the separation constant, the solutions to the one-dimensional, time-independent Schrodinger equation

$$-\frac{\hbar^2}{2m}\frac{d^2\psi(x)}{dx^2}+V(x)\psi(x)=E\psi(x)$$

are the spatial wavefunctions and stationary-state energies. For bound systems, $\psi(x)$, must approach zero as $x\rightarrow\pm\infty$. These boundary conditions place restrictions on physically acceptable stationary-state energies and ensure that $\psi(x)$ is square integrable over the full range of x.

This solver produces numerical solutions of the one-dimensional, time-independent Schrodinger equation. Currently available potential energy functions are finite and infinite square-well potentials that may include internal barriers, sloped well bottoms, and finite harmonic-oscillator potentials, The user defines the well and may request a solution at a specified energy. The solver then determines the boundary conditions on $\psi(0)$ and $\frac{d\psi(0)}{dx}$ so that $\lim_{x\rightarrow-\infty}\psi(x)=0$. The boundary conditions provide a starting point for the solver which

then proceeds to calculate the wavefunction and its slope from left to right across the well. If the specified energy deviates from a stationary-state energy, the wavefuncion will eventually diverge past the right edge of the well.

The solver can also determine the stationary-state energies and wavefunctions within a given energy range. It first determines the wavefunction for closely spaced energies and, for each energy, stores the value of the wavefunction and its first derivative at the right edge of the well. The solver then uses an interpolative process to find the energies where these boundary values result in an exponentially decreasing function outside of the well.

The solver is written in the python3 with PyQt-based GUIs. The scipy odeint differential equation solver provides the Schrodinger equation solution. The Continuum Analytics Anaconda3 distribution contains all dependencies for the solver. No knowledge of python is required to use the solver. Neither is a detailed knowledge of quantum mechanics required.

2 QuickStart

2.1 Prerequisites

- macOS, Windows 7/10, or Linux operating system
- python $>$ 3.5, DO NOT USE PYTHON 2.x
- python modules numpy, scipy, pyqt (version 5), matplotlib, and their prerequisites

We use the Continuum Analytics Anaconda3 distribution system which includes all prerequisites, is free, and is widely used. The installation instructions are clear and complete. For most computers, you should use the 64-bit graphical installer. Unless you use additional python platforms, I recommend that users accept all of the anaconda install defaults while paying particular attention to these two options:

- Select *Install for me only* rather than selecting *Install for all users* to avoid needing admin privileges.
- For installs on Windows, select *Add Anaconda3 to my path environmental variable* even though the installer recommends not using this option. Otherwise, you will have path issues when using the QuantumWell startup scripts.

2.2 Startup

Download the *QuantumWell-main.zip* compressed file by clicking this link, QuantumWell Download. If you are familar with git and github, you may wish to clone the QuantumWell repository using standard git/github techniques.

- Move the *QuantumWell-main.zip* file to any directory; I recommend your Desktop. Open the compressed file (double-clicking usually works. But you may have to give permission to open this downloaded file.). You should see the *QuantumWell-main* root directory. From now on, I'll refer to this directory as the *QuantumWell* directory. If you wish to rename it, do that now before proceeding further.
- Open a file browser in the *QuantumWell* directory. Execute the *config.py* script by double-clicking it: (ditto per above: you may have to give permission to execute this downloaded script)
  - *config.bat* for Windows
  - *config.command* for macOS
  - *config.sh* for Linux
  - or from a terminal or cmd window in the QuantumWell directory, execute *python config.py*
- The config.py file that you just executed had three tasks:
  - Confirm that all the above prerequisites are installed and available.

  - Create a qwconfig/config file in your home directory that contains the path to the QuantumWell directory (you can ignore this file unless you wish to use the QuantumWell/src python modules e.g. in a jupyter notebook. See the tutorial notebooks in the Tutorials directory)
  - Create the startQWell.xx startup script both in your QuantumWell directory and on your Desktop (you can copy this file for use in any directory). You should see one of the following scripts in the QuantumWell directory and on your Desktop:
    * *startQWell.bat* for Windows machines
    * *startQWell.command* for macOS machine
    * *startQWell.sh* for Linux systems

Open the QuantumWell GUI (Figure 1)using one of the following:

- Double-clicking the *startQWell* script from a file browser.
- Executing this script from a terminal from any directory containing the startup script
- If you wish to bypass the startQWell script:
  - for Windows execute: *python <path to QuantumWell dir>\quantumWell.py*
  - for macOS and Linux execute: *python <path to QuantumWell dir>/quantumWell.py*
- Any files that you create from GUI options will be in the directory from which you opened the GUI.
- To exit the GUI, click the exit button on the lower left of the main GUI window.

If this is your first time with the GUI-based solver, I recommend the following short tutorial which uses only default parameters:

- **Build a Well**: Click the *BuildWell* button to open its dialog box. Note the default values, especially the *inf* notation for an infinite well height. After you accept the well parameters, a plot of your new well will appear in the main window plot region. After you select your next option, the well parameters will appear in the lower message box (ditto for all options. The results of the previous option is added to the list in the message box)
- **Add an Internal Barrier**: Click the *AddBarrierPE* button to open its dialog box. Take the defaults and add the barrier. The internal barrier will appear in the main window plot and in the lower message box after you select your next option.
- **Remove the Internal Barrier**: Click the *RemoveAddedPE* button to open its dialog box and remove your internal barrier. You can also add and remove the other potential types. Watch the message box to see if there's a problem with overlap.
- **Solve the Schrodinger Equation**: Click the *SolveSchrodingerEq* button to open the solver window. Note the default energy of 10 eV. If this energy corresponds to a stationary-state energy (unlikely), the wavefunction will approach zero on the right-side of the well. Click the button and you'll see that 10 eV is not a stationary state.
- **Find the Stationary States**: Click the *FindStationaryStates* button to open the stationary-state window. The stationary-state algorithm will find the stationary states between the minimum and maximum energies listed in the window. Click *Click to Find States*. You should see the stationary-state energy lines on the main well plot with values listed in the upper message box. You can identify each state by its ID number. Choose a state ID and make a plot of the stationary-state time-independent wavefunchtion with the *Plot Psi* button. Then, continue to make Psi plots with other stationary state choices. When you select another option from the main GUI window, the stationary-state energies will appear in the lower message box.

  If you select the *show graph* option and once again find the states, a graph with a red line will also appear. The zero-crossings of the red line occur at the stationary-state energies. The points on the graph are written to the *energy_ slope.txt* file in your current directory.

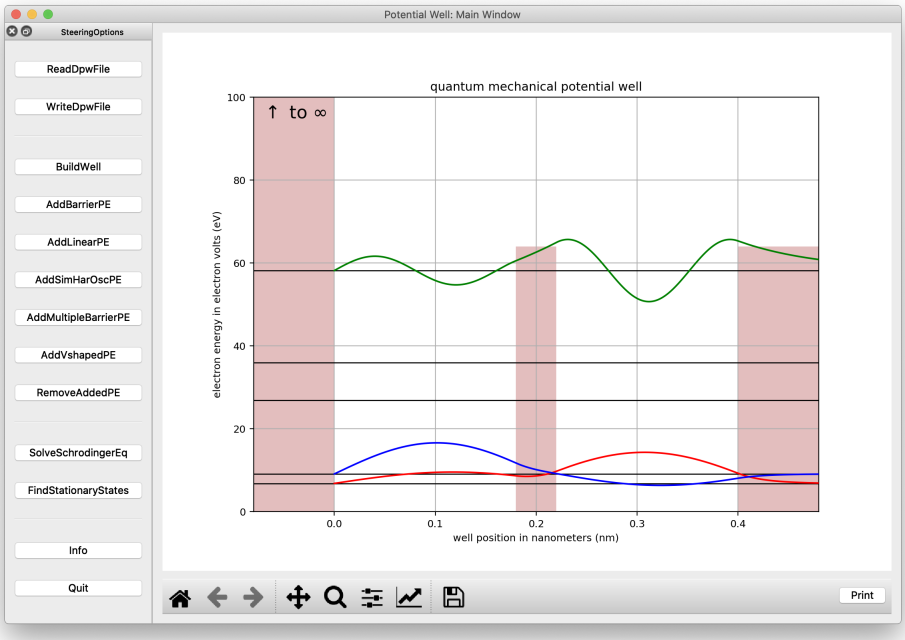

Figure 1: Main GUI Window

- **Input/Output**: You can print the main window using the button in the lower right. You can save or print your log file using the buttons in the lower message box. And, you can save all well parameters plus your last plotted stationary-state wavefunction and its derivative using the *WriteDpwFile* button on the main window. You can also read the well parameters into the solver using the *ReadDpwFile* button. I'll describe the dpw file in more detail later in the section on reading and writing dpw files. (dpw is the file extension and stands for **D**ata **P**otential **W**ell)

3 GUI Windows

Depending on your machine type, double click on the appropriate Start script to open the main GUI window and the message box. Or, you can open a terminal window and execute *python quantumWell.py*.

3.1 MainWindow

The main window (see Figure 1) shows the potential-well plot with the usual matplotlib icons, see Appendix 6.5 for more details. The various buttons in the main window create dialog windows for changing the well dimensions, adding internal potentials, removing internal potentials, solving the Schrodinger equation for a specific energy, and finding the stationary states over a specified energy range. The read/write buttons provide access to the python class instance describing the well and the stationary-state wavefunction and its derivative.

3.2 Building a Potential Well

Well dimensions are in nanometers (nm) and energies are in electron volts (eV). The particle mass is the mass of the electron.

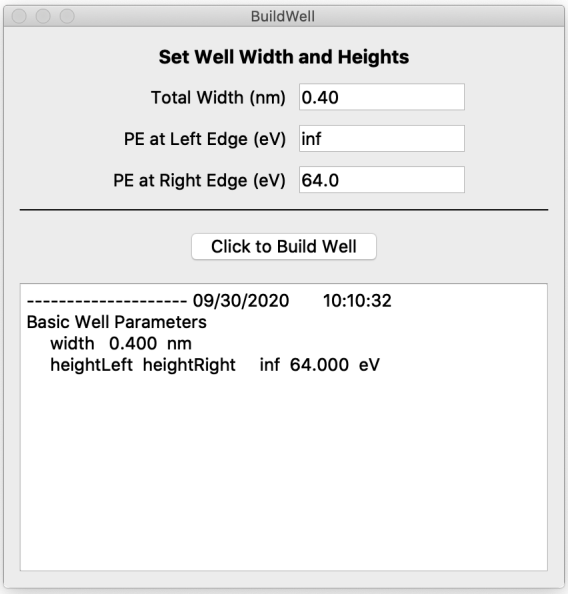

Figure 2: Build Well Window

3.2.1 BuildWell Dialog Window

See Figure 2

TotalWidth: Well width in nanometers edge to edge

$PE\,at\,Left\,Edge$: Well height, left edge, in electron volts; use inf for infinite height

$PE\,at\,Right\,Edge$: Well height, right edge, in electron volts; use inf for infinite height

$Click\,To\,Build\,Well$: Creates the new well; changes appear in the plot window

3.2.2 AddBarrierPE Dialog Window

See Figure 3

There is no limit on the number of added barriers as long there is no overlap between barriers. The message box will inform you of overlapping barriers.

$Left\,Edge\,of\,Barrier$: Left edge in nm of rectangular barrier, must lie within the basic potential well.

$Right\,Edge\,of\,Barrier$: Right edge in nm of rectangular barrier, must be greater than the left edge of the barrier and lie within the well

$Barrier\,Potential\,Energy$: Height in electron volts of the rectangular barrier

$Click\,to\,Add\,Barrier$: Create the barrier, barrier appears in the plot window

Note the barrier description in the message box; in addition to the lower and upper barrier limits, there are three polynomial coefficients, a constant, linear, and quadratic coefficient. These coefficients describe all of our added potentials.

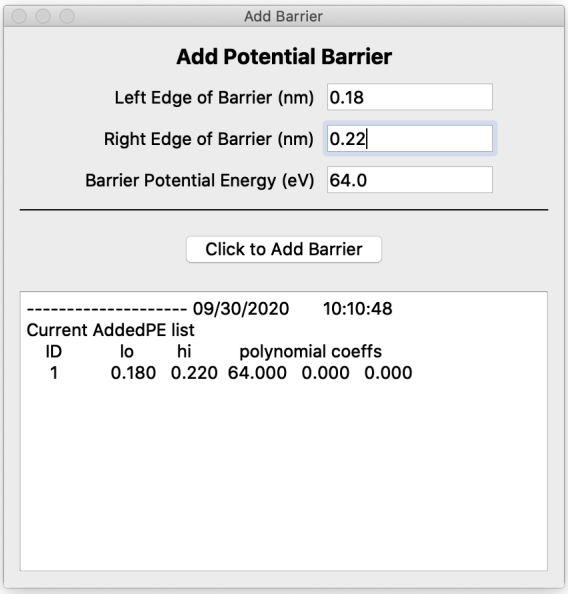

Figure 3: Add Barrier PE Window

3.2.3 AddMultipleBarrierPE

See Figure 4

Add any number of barriers where the distance between adjacent barriers is fixed. Fixing the distance between barriers facilitates creating a simple one-dimensional crystal and investigating stationary-state energies and wavefunctions with increasing barrier number. Note the distance scaling in the well graph as you change the number of barriers; the total well width must increase as you increase the number of barriers to maintain the fixed distance between adjacent barriers.

$Number\,of\,Barriers$: Number of Interior Barriers to add (can be zero)

$SubWell\,Width$: Distance in nm between each barrier, edge-to-edge, not center-to-center.

$Barrier\,Width\,and\,Height$: Width in nm and height in eV of each barrier (excluding well edges)

Note the barrier description in the message box; in addition to the lower and upper barrier limits, there are three polynomial coefficients, a constant, linear, and quadratic coefficient. These coefficients describe all of our added potentials.

3.2.4 AddLinearPE Dialog Window

See Figure 5

$Xmin$: Left starting position in nm for the linear potential, must lie within the well

$Xmax$: Right ending position in nm for the linear potential, must be greater than $xmin$ and lie within the basic well

$PE\,at\,Xmin$: Potential energy in electron volts at xmin

$PE\,at\,Xmax$: Potential energy in electron volts at xmax

$Click\,to\,Add\,Potential$: Create the linear potential, the potential line appears in the plot window.

The message box shows the left starting and right ending positions. The first two polynomial parameters give the equation of the potential line relative to $Xmin$ and $Xmax$. The first parameter is the height at $Xmin$; the second parameter is the slope relative to $Xmin$ and $Xmax$, i.e. $slope=\frac{\nabla V}{Xmax-Xmin}$.

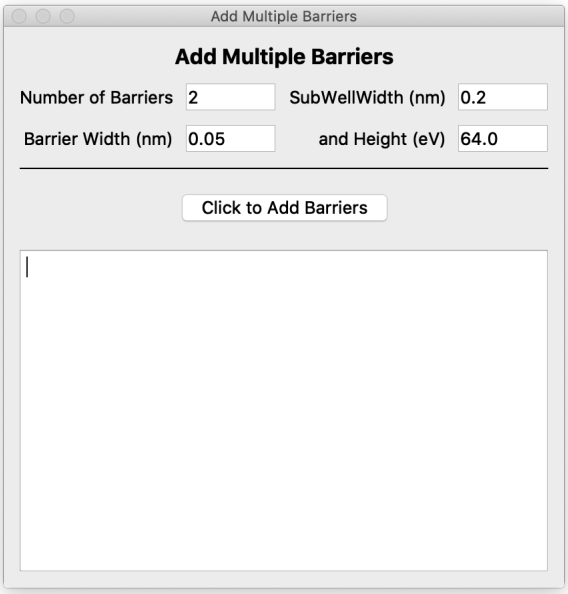

Figure 4: Add Multiple Barrier PE Window

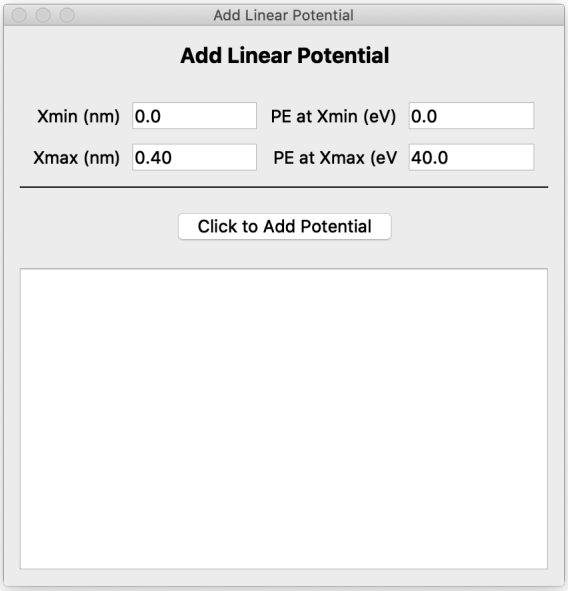

Figure 5: Add Linear PE Window

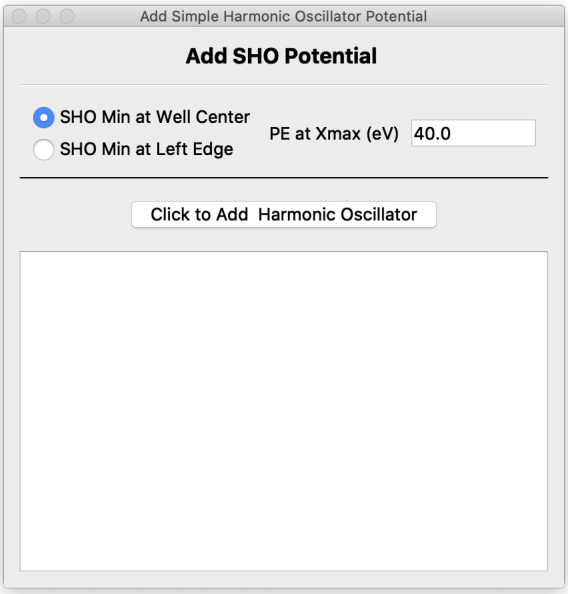

Figure 6: Add SHO Potential Window

3.2.5 AddSimHarOscPE Dialog Window

See Figure 6

SHO Min. at Well Center : Create simple harmonic-oscillator internal well with minimum at the center of the basic well

SHO Min. at Left Edge : Create simple harmonic-oscillator internal well with minimum at the left edge of the basic well

PE at Xmax : Potential at the right edge of the well.

Click to Add Harmonic Oscillator : Create the SHO potential; potential curve appears in the plot window

Since the SHO covers the entire interior of the well, the message box shows the positions of the left and right edges of the basic well as the starting and ending positions of the SHO potential. The message box also shows the parameters a, b, c of the second-order polynomial that describes the SHO potential, i.e. $V_{SHO} = a$+b*x+c*x^2.

3.2.6 AddVshapedPE Dialog Window

See Figure 7

PE Zero in Well : Well location in nm of the tip of the V-shaped potential. Potential energy of this point will always be zero.

PE Left Edge : Potential energy in eV of V potential at the left edge of the well.

PE Right Edge : Potential energy in eV of V potential at the right edge of the well.

The message box shows the usual polynomial coefficients for each side of the V potential.

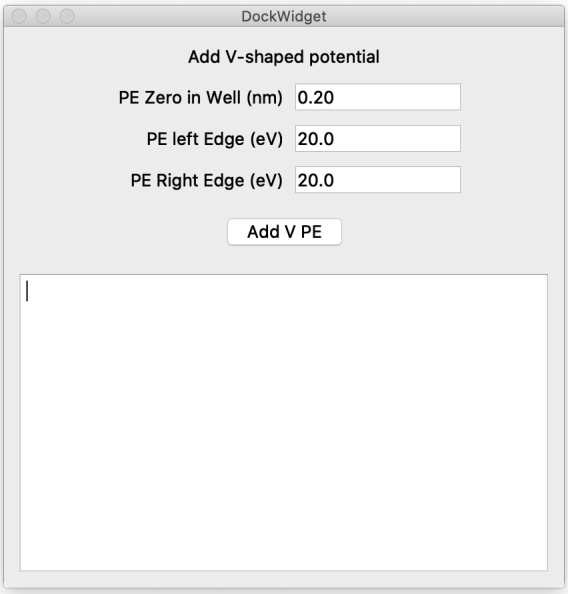

Figure 7: Add V Shaped Potential Window

3.2.7 RemoveAddedPE Dialog Window

See Figure 8

ID Number of Added Potential : The ID number of the added potential from the AddedPE potential list that you wish to remove. The list is in the box in this window.

Click to Remove Added PE : Remove the selected internal potential. The potential curve will disappear in the plot window

3.3 Solving the Schrodinger Equation

3.3.1 SolveSchrodingerEq Dialog Window

See Figure 9

Energy : The energy in eV of the electron in the potential well (which may or may not lead to a physically acceptable solution)

Click to Find Psi : Solve the Schrodinger equation and plot the solution in the plot window.

This solver uses the scipy *odeint* function to find the wavefunction and its derivative associated with the potential well and a specified energy. The *odeint* solver moves from left to right, starting at $x = 0.0$ with initial values of the wavefunction and its derivative, and determines the solution at increasing values of x. The solver chooses the initial values so that the wavefunction for negative values of x is a decreasing exponential for a finite potential well edge or is zero for an infinite potential-well edge. Thus, the $x < 0.0$ section of the plot is not part of the *odeint* solution, rather these points are calculated from the known decreasing exponential function. The *odeint* code calculates the wavefunction for $x > 0.0$. You see these points plotted on the graph in the main window. The solver imposes no boundary conditions at the right-edge of the well and the solution will be physically meaningless if the wavefunction diverges. (See the following FindStationaryStates section for further details.)

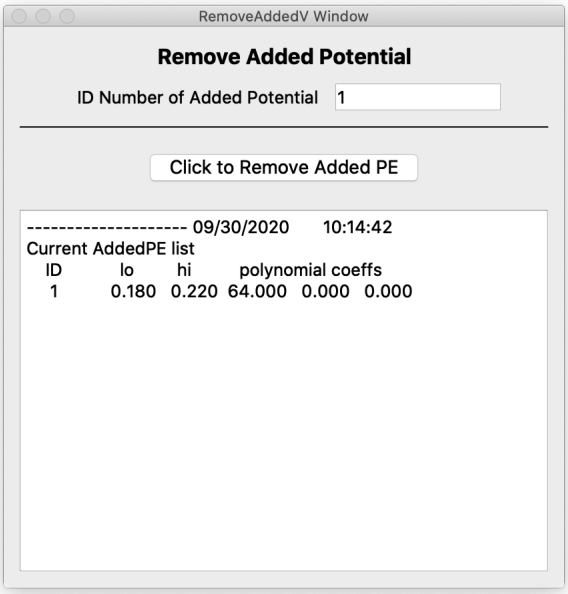

Figure 8: Remove Added PE Window

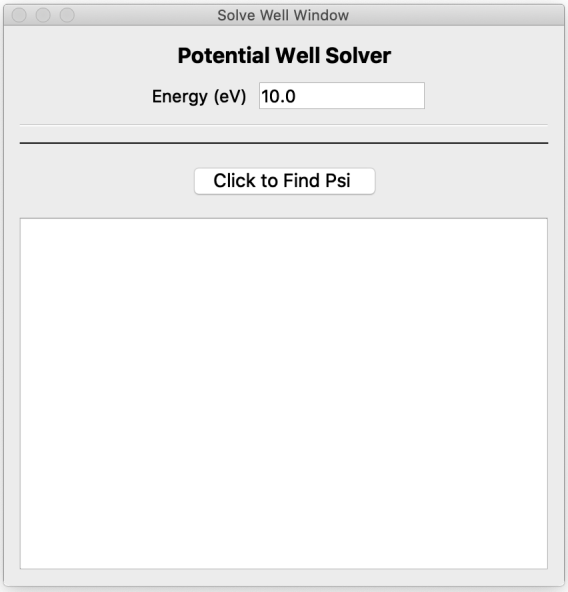

Figure 9: Solve Schrodinger Equation Window

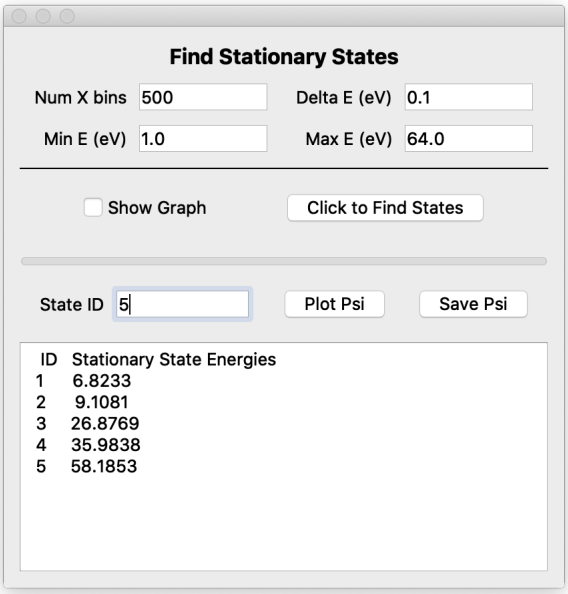

Figure 10: Find Stationary States Window

3.3.2 FindStationaryStates Dialog Window

See Figure 10

Click to Find States : Find the stationary states within the stated energy range and show these states in the main window plot.

$Min\,E$: The stationary state search range is between $Min\,E$ and $Max\,E$

$Max\,E$: The stationary state search range is between $Min\,E$ and $Max\,E$

$Num\,X\,bins$: The number of x values between the left and right edges of the potential well. When in doubt, take the default value of 500.

$Delta\,E$: The energy difference between trial energy values.

Show Graph : When checked, the solver displays a graph plotting the difference at the right-hand well edge between the wavefunction slope from the *odeint* solution and the calculated slope required for the wavefunction to become a decreasing exponential function for $x > right - well\ edge$. If the right-hand well edge is *inf,* the plot is the value of *psi* at the right edge (*psi* must be zero at the *inf* edge for stationary states). The zero-crossings for this plot are the stationary-state energies and are found by a linear interpolation between consecutive E points, one above and one below the energy axis. The matplotlib icons (See Section 6.5)which enable you to pan and zoom are very useful for determining optimum parameter values, especially when the difference between the stationary energies are only slightly larger than $Delta\,E$.

State ID : Selects a specific stationary state for plotting. The ID numbering starts at 1 not 0.

Plot Psi : Plots the time-independent wavefunction for the selected stationary state on the main window graph. The plots are cumulative and color coded.

Save Psi : Writes the time-independent wavefunction to a specified text file. The four columns in the file are: the wavefunction, its derivative, x, and V(x). The wavefunction is not normalized.

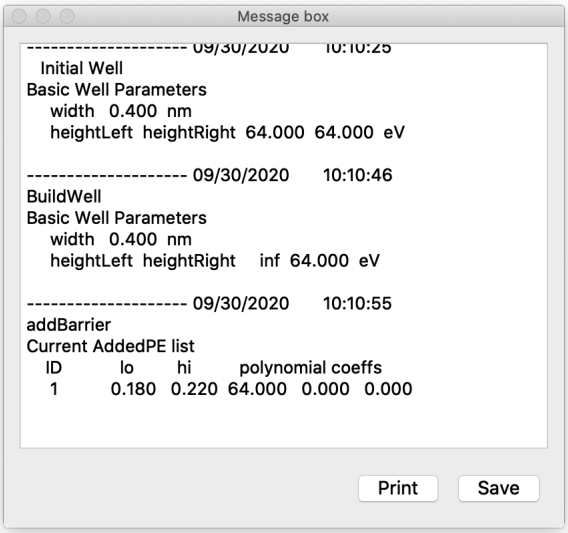

Figure 11: Message Box Window

3.4 Message Box

See Figure 11

The message box is always open and contains the history of your work with the solver including the date and time of each action.

Print : Print the contents of the message box to a specified printer.

Save : Save the contents of the message box in a specified text file.

4 Example GUI-Based Studies

- Using the default well, plot psi for energies both above and below the energy of a stationary state while watching the divergence of psi on the right side of the well. Plot psi at the energy of the stationary state.
- Using the default well, determine the stationary states. Discuss the relationship between the stationary-state energy, the curvature (second derivative) of the wavefunction, and the sign of the wavefunction. Discuss the relationship between the number of nodes of the wavefunction and the stationary-state energy.
- Add a linear internal potential and discuss the shape of the wavefunction as a function of position in the well. Do the same for a V-shaped internal potential and a simple harmonic oscillator internal potential.
- For the above potentials, discuss the symmetry of the stationary-state wavefunctions with respect to the center of the well. What can be said about the probability density associated with each energy state? What value do you expect for the position expection value?
- Add a single barrier potential and consider the first two energy states and their wavefunctions. For example, is there a relationship between the shape of the wavefunctions and the energy separation between the two states?

- Create a simple diatomic ion with two potential wells separated by a barrier and having a single electron and no Coulomb forces. (use the addMultipleBarrier option with well edges and barrier heights set to the same value. Set the distance between barriers to some reasonable value such as 0.2 nm. A barrier width of 0.05 nm is also a reasonable value.) Discuss whether or not this ion will be a stable (or a bound) system. What are the conditions for a bound system? If the ion is a stable system, what is its binding energy in eV as determined from the QuantumWell solver options.
- How do you expect the energy levels for the above diatomic molecule to change as the barrier width increases? What about the stability of the ion? Does it increase or decrease?
- Investigate the energy levels and their wavefunctions of a well with multiple barriers. What do you expect as the number of barriers increases? You may see the stationary-state wavefunctions begin to diverge on the right-side of the well. If so, go to the SolveSchrodingerEq option and calculate wavefunctions with energies slightly above and below the calculated stationary-state energy.
- For stationary-state wavefunctions that appear to diverge, recalculate the stationary states with the graph option enabled. The zero crossings determine the stationary-state energies. Use the Matplotlib toolbar icon to select and expand a small area around the crossing. How can you improve the calculation so that the wavefunctions do not diverge? To what extent do these improvements change the value of the stationary-state energy.
- Save one or more of your wells by writing it to a dpw file. Later, recreate the well by reading the dpw file back into the solver.

5 Tutorials for Jupyter Notebooks

These tutorials emphasize using QuantumWell python modules in jupyter notebooks. The Tutorials folder contains the following notebooks:

tutorial0_ addPath.ipynb: jupyter notebook for setting path to src folder

tutorial1_ readDpwFile.ipynb: jupyter notebook tutorial for reading a dpw file

tutorial2_ writeDpwFile.ipynb: jupyter notebook to create a potential well with any number of equally spaced, fixed height barriers. The notebook writes a dpw file of this well for entering into the solver for further analysis. Of course, you can create the same well using the multipleBarrier option in the solver. But, for example, if you wanted to create equally spaced simple harmonic oscillator subwells or a series of V-shaped potentials, you could make use of the techniques in this tutorial.

tutorial3_ analysis.ipynb: jupyter notebook with analysis examples, including calculating expectation values and making comparisons with the uncertainty principle.

tutorial4_ composite.ipynb: jupyter notebook for studying a composite wave function. The notebook illustrates calculating probability densities, expectation values, etc.

tutorial5_ animation.ipynb: Jupyter notebook for following time evolution of composite wave function.

getPath.py: python script used in jupyter notebooks to add path to src folder

psi1.

dpw/psi2.dpw: two dpw files used in the tutorial notebooks

6 Appendices

6.1 Importing QuantumWell Modules

You can easily create your own python scripts or jupyter notebooks using python modules in the *QuantumWell/src* directory. For example, you may wish to calculate expectation values or watch an animation of the time development of the probability density determined from composite wavefunctions. However, your enviromental path shown in *sys.path* must include the path to the *QuantumWell/src* directory.

The helper script, *getPath.py* in the *Tutorials* directory, obtains the path to your QuantumWell directory from the qwconfig/config file in your home directory that was created when you initially executed the *config* script. This is illustrated in the *tutorial0_ addPath.ipynb* notebook in the Tutorial directory which shows two methods using functions in *getPath.py*.

Don't forget to move the *getPath.py* file from the Tutorials folder to the your current directory if you are not working in the Tutorials directory.

6.2 Reading and Writing Dpw Files

The solver python class, $DataPotentialWell$, contains the well construction details. It also contains the stationary-state energies (if available) determined within the $FindStationaryStates$ window. In addition, the values of the last wavefunction (if available) plotted in the $PlotPsi$ option in the $FindStationaryStates$ window, its derivative, and associated parameters are contained in the class instance. At any time, you may write this class to a file and later read this file back into the solver. Or you may use the file to create a $DataPotentialWell$ class instance in a jupyter notebook or python script and have direct access to all the well data for further analysis such as calculating expectation values of position and momentum. (See the jupyter notebook tutorials in the Tutorials directory and described in later sections of this guide)

$WriteDpwFile$: Clicking this main window button will write the $DataPotentialWell$ class to a specified file with a .dpw file extension. A text editor cannot read this binary file.

$ReadDpwFile$: Clicking this main window button will fill the $DataPotentialWell$ class instance in the solver with the data from the .dpw file.

6.3 The DataPotentialWell Class

Use the DataPotentialWell class methods to access the well data. For example

```
dpw1 = DataPotentialWell()
dpw1.readDpwFile('myfilename.dpw')
x = dpw1.getXArray()
```

returns the x values used in the solver as a numpy float array.

To see the doc string of the class, place your cursor on the dpw1 instance and do a shift-tab. To see the list of class methods, place your curser immediately after dpw1. and hit the tab key. Select the method you wish to use with your cursor to add it to dpw1. , then immediately do a shift-tab to see the method's doc string.

6.3.1 Useful DataPotentialWell Methods for Accessing Well Data: Partial list

$getBasicWellProperties()$: Returns (wellWidth, wellHeightLeft, wellHeightRight).

$getXArray()$: Returns array of x values.

$getVArray()$: Returns array of potential energy values.

$getPsiArray()$: Returns wavefunction array.

getPsiArrayNormalized(): Returns normalized wavefunction array

Figure 12: Matplotllib Toolbar

getPsiPrimeArray(): Returns wavefunction derivative array.

getPsiEnergy(): Returns the stationary state energy associated with the wavefunction.

printData(): prints well data (but not the large arrays) to the screen

All of the methods are described in the doc string of the DataPotentialWell class and individually in each method doc string.

6.4 The BuildPotentialWell Class

The BuildPotentialWell class can be used within a python script or a Jupyter notebook to build potential wells with specific width and barrier criteria. Have a look at the *tutorial* notebooks in the Tutorials directory for further details and checkout the doc strings in the BuildPotentialWell class.

More details about this class will be in the next version of this User's Guide.

6.5 Matplotlib Toolbar

All python matplotlib graphs come with a navigation toolbar (see Figure 12) with very useful tools for expanding selected sections of the graph.

You'll find documentation for the toolbar here. The most useful toolbar buttons are the Pan/Zoom, the Zoom-to-Rectangle, and the Save buttons.

6.5.1 The Pan/Zoom Button

This Pan/Zoom button (see Figure 12) has two modes: pan and zoom. Click the toolbar button to activate panning and zooming, then put your mouse somewhere over an axis. Press the left mouse button and hold it to pan the figure, dragging it to a new position. When you release it, the data under the point where you pressed will be moved to the point where you released. If you press 'x' or 'y' while panning the motion will be constrained to the x or y axis, respectively. Press the right mouse button to zoom, dragging it to a new position. The x axis will be zoomed in proportionate to the rightward movement and zoomed out proportionate to the leftward movement. Ditto for the y axis and up/down motions. The point under your mouse when you begin the zoom remains stationary, allowing you to zoom to an arbitrary point in the figure. You can use the modifier keys 'x', 'y' or 'CONTROL' to constrain the zoom to the x axis, the y axis, or aspect ratio preserve, respectively.

6.5.2 The Zoom-to-Rectangle Button

Click the Zoom-to-Rectangle button (see Figure 12) to activate this mode. Put your mouse somewhere over an axis and press the left mouse button. Drag the mouse while holding the button to a new location and release. The axes view limits will be zoomed to the rectangle you have defined. There is also an experimental 'zoom out to rectangle' in this mode with the right button, which will place your axes in the region defined by the zoom out rectangle.

6.6 Code Development

6.6.1 Overview

The *quantumWell.py* file contains the QuantumWell class. Python code following the *if __name__ == __main__:* statement instantiates this class and involkes the pyqt event loop to service button presses on the GUI windows.

The QuantumWell class assembles the main window, instantiates all action classes in src modules, and handles all window buttons as decribed below.

The code that constructs each window QWidget was built using Qt *designer* with the *.ui* files included in the *src* directory. The classes that serve action buttons in the subwindows are in their respective modules in the *src* directory.

Since the action classes have their own module, they can be instantiated in user-produced code; the Tutorials directory includes example tutorials for such use.

All well data, calculated or specified energy levels, and the last calculated (if available) wave function and its derivative are stored in the important *DataPotentialWell* class contained in the *DataPotentialWell.py* module in the src directory.

I recommend using the *doxygen* documentation software to scan the *quantumWell.py* file with its QuantumWell class and the classes in the modules in the *src* directory. After installing doxygen, copy the Doxygen configuration file from the Documentation directory to the QuantumWell directory and execute the command, *doxygen,* from a terminal. The software will use the configuration file and create an *html* directory with associated html documentation files. Go to that directory and open a browser on the *index.html* file. You should see the README file in the main window. Click on the menus at the top of the main window to move through the files or the classes. Most useful is to select the QuantumWell class; clicking on one of its methods shows this method in a listing of all methods that includes the method doc strings. Ditto for all other classes.

6.6.2 Qt Windows

I used QT *designer* software (included in the anaconda system) to construct the GUI window widgets (with the exception of the Matplotlib widget). The *designer* software creates .ui files which must be converted to python .py files using *pyuic5* from the anaconda python package. Each .ui file must be processed from a terminal e.g. (all on one line):

```
pyuic5 -x AddBarrierDock.ui -o ui_AddBarrierDock.py
```

To make this conversion from Mac or Linux machines, you can use the *make* utility by simply executing *make* in the src directory. This code uses its default file, $Makefile$, to produce all .py files. (Have a look at the $Makefile$ file and you'll see the .ui files produced in QT designer.). If you have not installed the *make* utility, you can execute the above line from a terminal for each of the .ui files. Or, you can create a bash script to automate the commands.

From windows machines, you can convert all .ui files by executing from a cmd terminal (have a look at this file):

```
run_pyuic_Windows.bat
```

You can observe each window (for testing) by executing each file in python to run the code following the usual *if __name__ == __main__ :* statement.

Note that I did not use *designer* to create the *MatplotlibWidget.py* file; you can execute this file to see an example plot in the matplotlib QWidget window.

The QuantumWell class instance imports each of these files as *QWidgets* from the *utilQt.py* file after a simple inheritance from the ui_...py file. The QuantumWell code creates and assembles the main window from the various *QWidgets* described above. The class also instantiates the subwindows and sets, with the exception of the *message* subwindow, their *visibility* parameter to *False*. Thus, you only observe the *message* subwindow when you initially run the *quantumWell.py* script.

6.6.3 Button Management

When you depress a button on the main window, the visibility of the current subwindow is set to *False* and that of the selected subwindow is set to *True*. Thus, all subwindows are open and overlapping, but only one is visible at any given time. Each button on the main window connects to a corresponding QuantumWell method whose name begins with *open*, e.g. *openBuildWellDock(self)*. A corresponding *close* method sets the visibility to *False*.

When you depress a button on a subwindow (window to the left of the main window), the connect method in the QuantumWell class reads the subwindow parameters, checks for formatting accuracy, and calls the appropropriate action class in its module in the *src* directory.

6.6.4 Action Classes

The action classes, located in modules of the same name in the *src* directory, are those that perform a specific task, such as finding the stationary states. They use the potential well information in the DataPotentialWell class instance and have no knowledge of the windows or subwindows from the QuantumWell class. This separation permits their use in user produced python scripts and jupyter notebooks (see the Tutorials directory notebooks). The most useful action classes have python doc strings that describe their use.