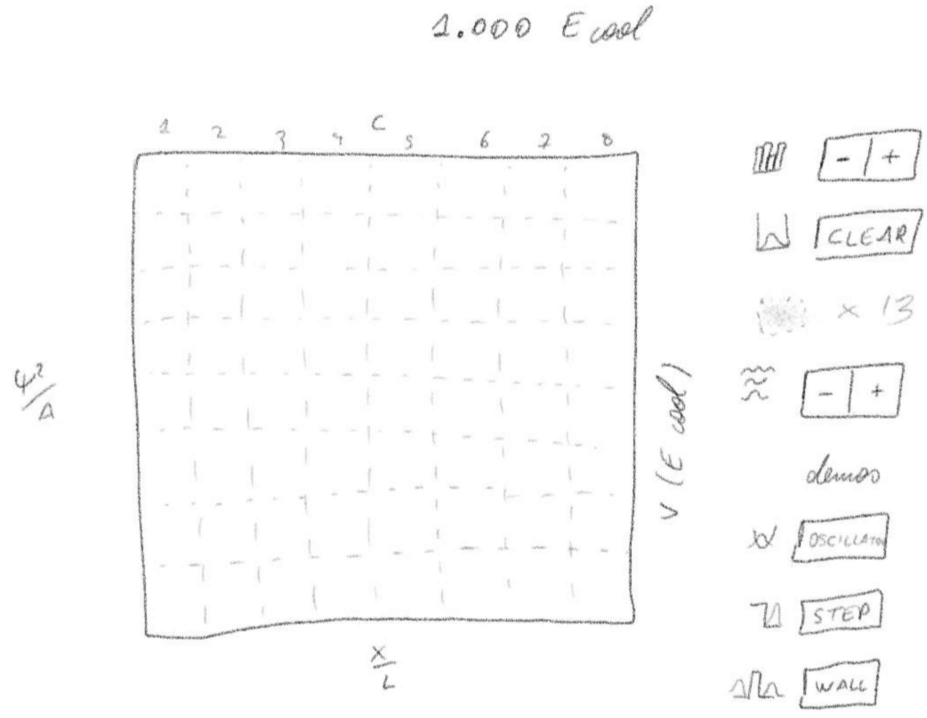
INTERACTIVE PIECEWISE QUANTUM POTENTIAL WELL



Rafa da Silva

QuantumUBlab 2018-19

Pràctiques en empresa

Phase 0: Initial Idea. The step potential

The idea of QuantumUBlab is to divulge Quantum Mechanics through visual software. It started with ultracoldUB looking into solitons behaviour: time evolution, collisions, etc. The computation used to simulate them solves the Gross–Pitaevskii equation (GPE) numerically with the Cranck-Nicholson method, a stable numerical method used for diffusion equations.

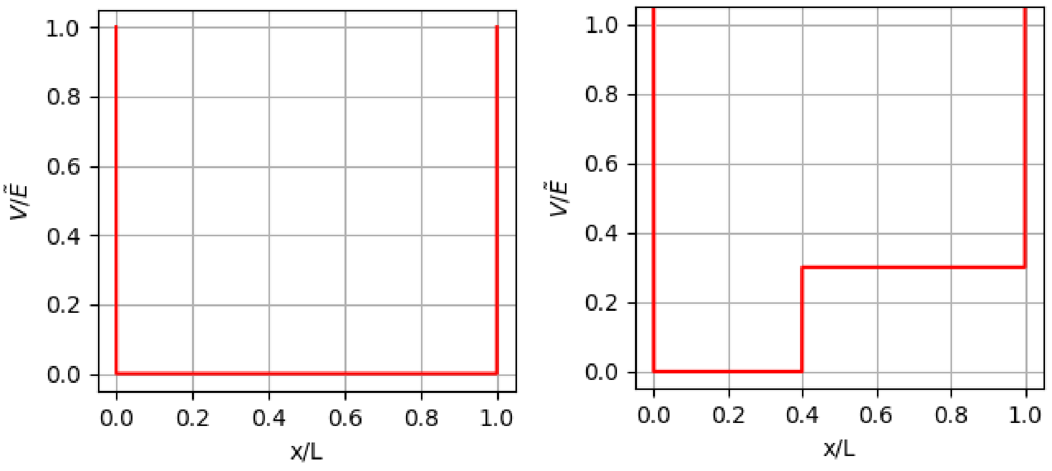


Fig. 1 Square well potential with infinite walls on the left. And the same potential with a finite step on the right.

In a quantum mechanics introduction lecture, a square potential well with infinite walls is the first example used to introduce wave functions, because of its simplicity. Usually, the very next example given is the same potential but with a little step. One gains the intuition that a particle on the groundstate is more likely to be found on lower potential. This intuition will later be useful on many other areas of physics.

But it is not only for its usefulness that it is introduced so early. The reason the step potential is the traditional second example is because its boundary conditions give an easy analytical result to the Schrödinger equation. The same happens with three, four or N steps. An exact solution of the wave-function can be found regardless of the number of steps chosen. These multi-step potentials are called piecewise potentials.

Moreover, a program intended to find and analytical solution is much faster than one with an iteration method, like Crank-Nicholson. The idea is then to understand how to obtain the analytical solution of a piecewise potential (phase 1), and what interesting simulations can come out of it (phase 2). One could simulate for example, a piecewise approximation of the harmonic oscillator, as we will see later.

Phase 1: One step potential.

Numerical approach

As an initial stage, to warm things up, we started with the second example. An infinite square well with one step only, like in figure #. We consider the time-independent one-dimensional Schrödinger equation:

$$-\frac{2m}{\hbar^2}(E-V)\Psi(x) = \Psi''^{(x)} \qquad \textbf{(1)}$$

Given an infinite square potential of size L, with a discontinuity on x_{steep} dividing it into two regions, we have a wave-function for each region:

$$\begin{cases} \infty & x = 0 \\ 0 & 0 < x < x_s \\ V_s & x_s < x < L \\ \infty & x = L \end{cases} \quad \rightarrow \quad \begin{cases} \Psi_L(x) = Ae^{i\alpha x} + Be^{-i\alpha x} \\ \Psi_R(x) = Ce^{i\beta x} + De^{-i\beta x} \end{cases} \qquad \textbf{(2)}$$

with $\alpha = \frac{\sqrt{2mE}}{\hbar}$ and $\beta = \frac{\sqrt{2m(E-V_s)}}{\hbar}$.

The determination of the parameters A, B, C, D and the energy E is given by the boundary conditions of the wave function plus the normalization condition.

The first condition is that where $V \rightarrow \infty$, $\Psi \rightarrow 0$. In this example, the values of x where the potential is infinite, $V(x') = \infty$, have no wave function at all, $\Psi(x') = 0$. This gives us two of the five necessary equations:

$$\begin{cases} \Psi_L(0) = 0 \\ \Psi_R(L) = 0 \end{cases} \qquad (3)$$

From (3) we reduce the parameters from A, B, C, D to A and C only:

$$\begin{cases} A + B = 0 \\ Ce^{i\beta L} + De^{-i\beta L} = 0 \end{cases} \rightarrow \begin{cases} \Psi_L(x) = A\left(e^{i\alpha x} - e^{-i\alpha x}\right) \\ \Psi_R(x) = Ce^{i\beta L}\left(e^{i\beta(x-L)} - e^{-i\beta(x-L)}\right) \end{cases} \qquad \textbf{(4)}$$

The second boundary appeals for the continuity of both the wave function and its derivative at the point where V changes abruptly, x_s:

$$\begin{cases} \Psi_L(x_s) = \Psi_R(x_s) \\ \Psi_L{}'(x_s) = \Psi_R{}'(x_s) \end{cases} \qquad (5)$$

We can transform (5) into

$$\begin{cases} \Psi_L(x_s)\big/\Psi_L{}'(x_s) = \Psi_R(x_s)\big/\Psi_R{}'(x_s), \\ \Psi_L(x_s)\,\Psi_L{}'(x_s) = \Psi_R(x_s)\,\Psi_R{}'(x_s) \end{cases} \qquad (6)$$

$$\frac{\left(e^{i\alpha x_s} - e^{-i\alpha x_s}\right)}{i\alpha\left(e^{i\alpha x_s} + e^{-i\alpha x_s}\right)} = \frac{\left(e^{i\beta(x_s-L)} - e^{-i\beta(x_s-L)}\right)}{i\beta\left(e^{i\beta(x_s-L)} + e^{-i\beta(x_s-L)}\right)} \qquad (6)$$

$$\alpha A^2\left(e^{2i\alpha x_s} - e^{-2i\alpha x_s}\right) = \beta C^2\left(e^{2i\beta(x_s-L)} - e^{-2i\beta(x_s-L)}\right) \qquad (7)$$

The equation above (7) gives the relation between α and β, that translates into the relation between E and V_s. Therefore, given x_s and V_s, the eigenenergies E_n of our system will be the ones which satisfy (7). We can write (7) as a root-finding function, a function of E that becomes 0 if $E \in E_n$.

$$f(E, V_s, x_s) = \frac{\left(e^{i\alpha x_s} - e^{-i\alpha x_s}\right)}{i\alpha\left(e^{i\alpha x_s} + e^{-i\alpha x_s}\right)} - \frac{\left(e^{i\beta(x_s-L)} - e^{-i\beta(x_s-L)}\right)}{i\beta\left(e^{i\beta(x_s-L)} + e^{-i\beta(x_s-L)}\right)} \qquad (9)$$

$$f(E, V_s, x_s) = 0 \quad \rightarrow E \in E_n \qquad (10)$$

The equation below (8) reduces A and C to only A, the normalization constant. We write C^2 as a function of A^2:

$$C^2 = \frac{\alpha\left(e^{2i\alpha x_s} - e^{-2i\alpha x_s}\right)}{\beta\left(e^{2i\beta(x_s-L)} - e^{-2i\beta(x_s-L)}\right)} A^2 \qquad (11)$$

And finally find the last constant A imposing Ψ to be normalized. In the code, we will integrate (12) using the Simpson's rule, a numerical integration method which is simple but precise.

$$\int_0^L \Psi^*(x)\Psi(x)dx = 1 \qquad (8)$$

Checking with $V_S = 0$ and $V_s = \infty$, with $x_s = \frac{1}{2}$

We want to always test if the algorithm used to find the roots of (9) is right. One way is to reduce our potential to the first example, the infinite square potential well.

Here we only have one region and one wave-function,

$$\Psi(x) = Ae^{ikx} + Be^{-ikx}, \qquad (9)$$

with $k = \sqrt{2mE}/\hbar$.

Applying the boundary condition

$$\begin{cases} \Psi(0) = 0 \\ \Psi(L) = 0 \end{cases} \rightarrow \begin{cases} A + B = 0 \\ Ae^{ikL} + Be^{-ikL} = 0 \end{cases}, \qquad (14)$$

we get that $B = -A$ and that the energy is quantized as

$$e^{ikL} - e^{-ikL} = 0 \quad \rightarrow \quad k = \frac{\pi n}{L} \text{ , where } n \in \mathbb{N} \text{ .} \qquad (15)$$

As a bonus, we also find the proper units of this system

$$k = \frac{\sqrt{2mE}}{\hbar} = \frac{\pi n}{L} \quad \rightarrow \quad E = \frac{\pi^2 n^2}{2}\frac{\hbar^2}{mL^2} = \frac{\pi^2 n^2}{2}\tilde{E} \quad . \qquad (10)$$

The first energy level, the ground state, is $E_{n=1} = \frac{\pi^2}{2}\tilde{E} \approx 4.9348\,\tilde{E}$. We are going to start looking for this value.

One way to ensure our program is finding the right ground state is to make a step on $x_S = \frac{1}{2}$ and grow to $V_S \to \infty$. The ground state for $V_S \to \infty$ should be equivalent to the ground state of a square well, $V_S = 0$, but with half the width, $L = L/2$. Since the energy is proportional to the width as seen in (ye), $E \sim 1/L^2$, as the potential grows to infinity the energy should grow by a factor of 4.

As seen in fig #. our root finding algorithm (9) does just that.

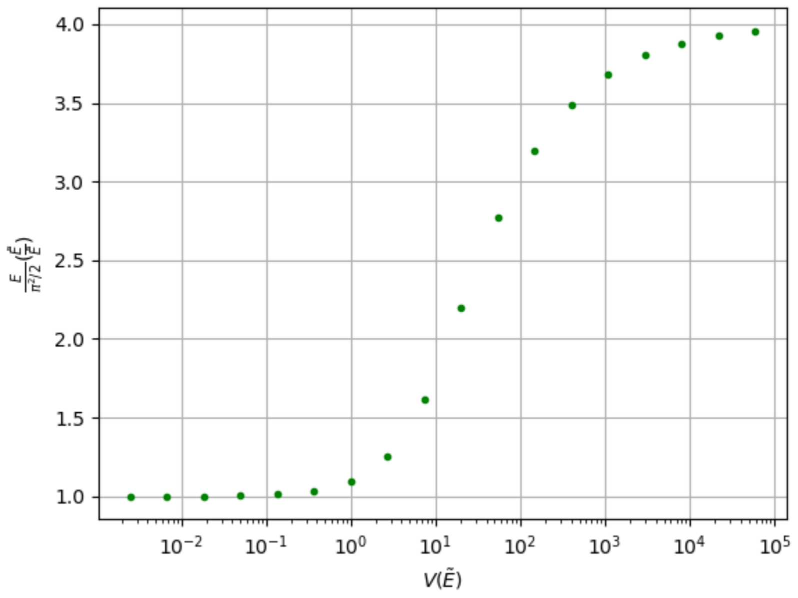


Fig. 2 Energies for $n = 1$ and $V \to \infty$. Which at the end is asymptotically equivalent to cut in half the width, $L \to L/2$.

Checking with $V_S = 0$ and $\forall x_s$

Another way to check if the algorithm is right is to use a "numerical zero". That is, for a very small value of V_S, $V_S \sim 0$, we should get the same eigenenergies regardless of the step position x_S.

For $V_S \sim 0$ and $x_S = 0.5\,L$, we find the correct first energy $E_{n=1}$. Moving this very thin step towards $x_S = 0.9\,L$ shouldn't change this value, but it does. This means our root-finding function is wrong.

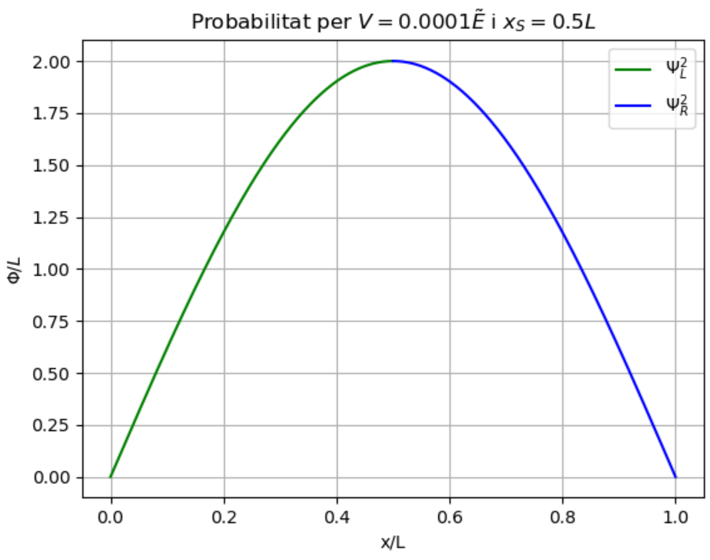


Fig. 3 Wave function for $V{\sim}0$ and $x_s = 0.5L$. The right and left wave-functions are continuous. It resembles the function it should be, a sinus. The energy is found ri

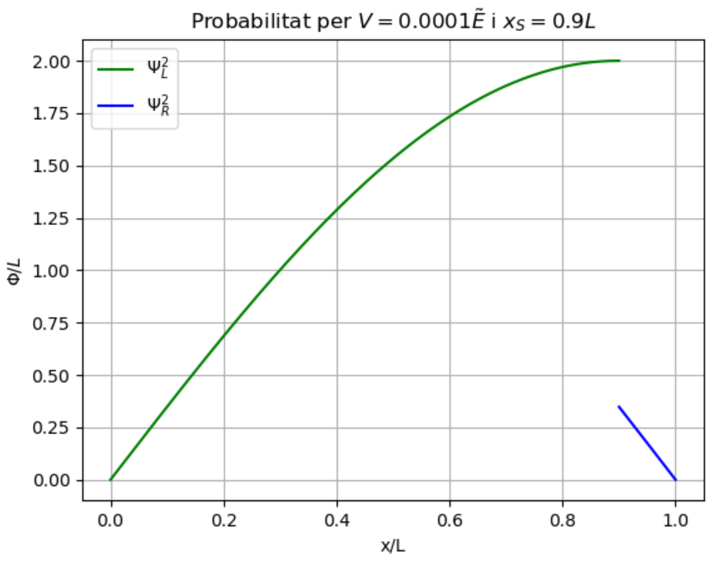


Fig. 4 Wave function for $V{\sim}0$ and $x_s = 0.9L$. There is a discontinuity. Both waves seem to be part of a much wider sinus. The energy is found wrong.

Refining the root-finding function

The problem with the function # is that it is a difference between two trigonometric functions. When iterating it, values near the roots diverge. This overflows the machine which starts giving complex values to the real function #. Sometimes this means that extra roots are founded, like in fig. #.

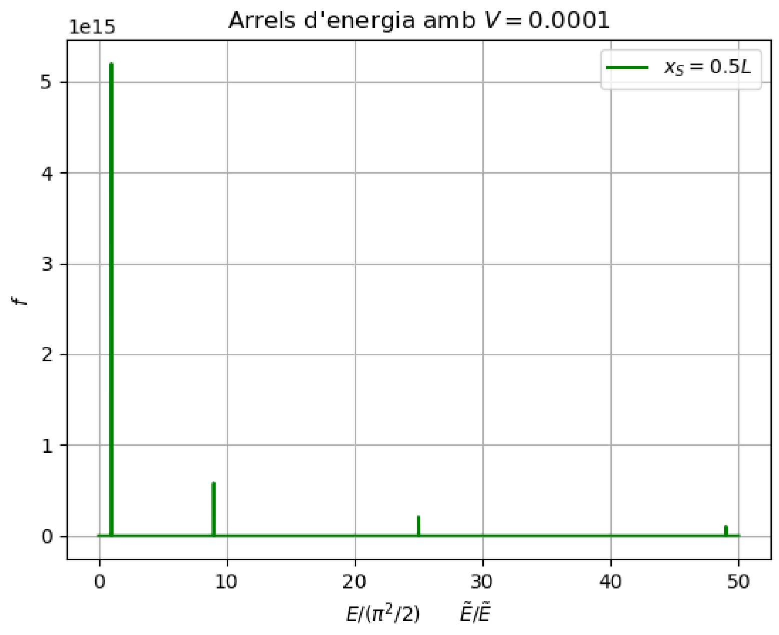


Fig. 5 The root-finding function iterated for different values of the energy E. For $V{\sim}0$ and $x_s = 0.5L$. It should have roots on the eigenenergies. But there is an overflow near them. We cannot actually see where the function corsses the axis.

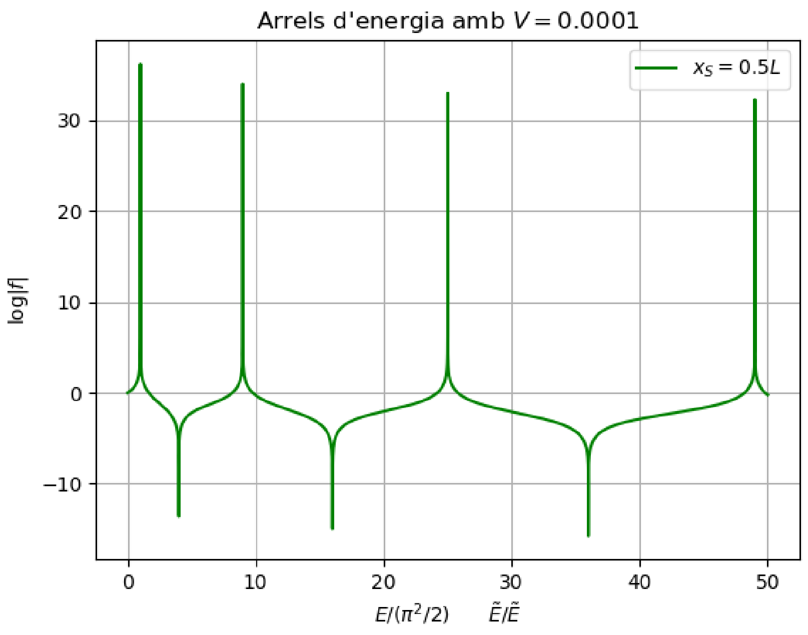


Fig. 6 The logarithm of the root-finding function for different values of the energy E. For $V{\sim}0$ and $x_s = 0.5L$. It has the eigenenergies on peaks instead of roots. We identify the peaks as $E_n = n^2 E_1$.

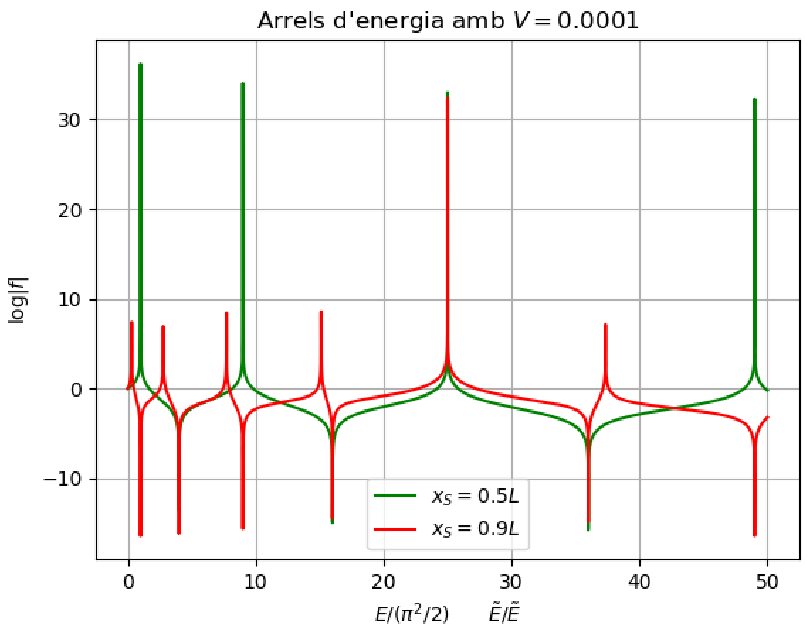


Fig. 7 The logarithm of the root-finding function for different values of the energy E. For $V\sim 0$ and two different step positions $x_s = 0.5L$ and $x_s = 0.9L$. One can see that moving the step produces extra roots, and it should not.

To avoid it, what finally worked was to reformulate # as the relation (fraction) between these two trigonometric functions. The problem was that around the root both of them grew enormously but with opposite signs because we were subtracting them. Since on the root they have the same value, so the ratio is $1:1$ and we can rewrite the root-finding function as

$$f(E, V_s, x_s) = \frac{(e^{i\alpha x_s} - e^{-i\alpha x_s})}{\alpha(e^{i\alpha x_s} + e^{-i\alpha x_s})} \Big/ \frac{(e^{i\beta(x_s-L)} - e^{-i\beta(x_s-L)})}{\beta(e^{i\beta(x_s-L)} + e^{-i\beta(x_s-L)})} - 1\,. \quad (17)$$

Figures 8-9 show that this function worked perfectly. The final code used to generate this figures is explained in annex 1.

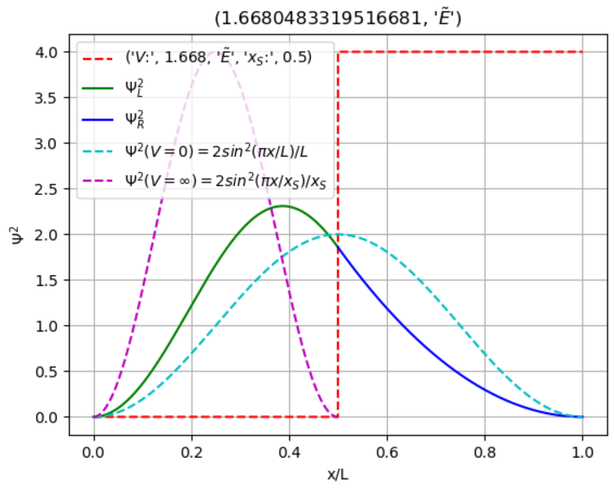


Fig. 8 Probability density of a square well potential with one step. The two dashed sinus are the cases where the step is too big or too small. This isan intermediate case that proofs the function (17) works.

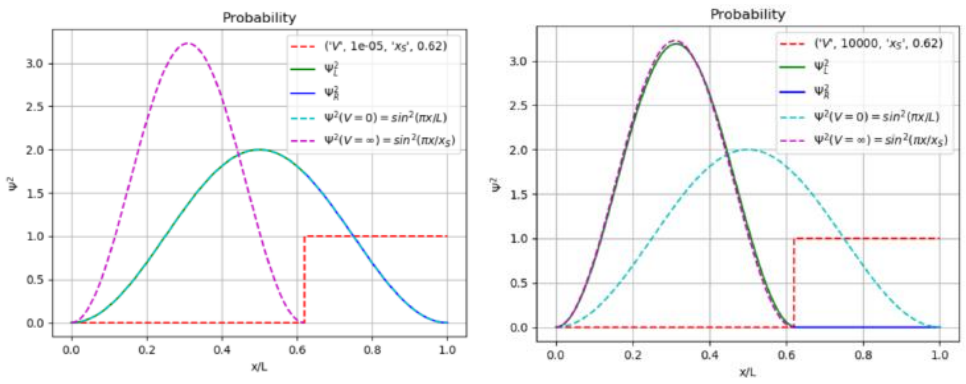


Fig. 9 Probability density of the extremes. The left step is too small and the energy level is the ground state of a square well of width L. The right step is too big, equivalent to a square well of $L = x_S$. This means the root-finding algorithm on (17) passes the test we made, and works.

Phase 2: Multistep

After the successful computation of the probability density for one step, we could go ahead and do a full potential profile with many different pieces. This is called a piecewise potential, and my tutor Bruno Julià Diaz, from now on BJD, helped with the analytical resolution. His derivation of a root-finding function is annex 5. In this paper a summary of it is given. as well as the changes made on the root-finding function.

Summary of the solution for piecewise potentials

Following BJD notes, for every piece of piecewise potential

$$\begin{cases} \infty & x = 0 \\ V_1 & 0 < x < x_1 \\ \cdots & \cdots \\ V_k & x_{k-1} < x < x_k \\ \cdots & \cdots \\ V_N & x_{N-1} < x < 1 \\ \infty & x = 1 \end{cases} \tag{11}$$

there will be a different wave-function $\Psi_k(x)$ acting as the solution of the Schrödinger equation for each piece $x_{k-1} < x < x_k$. The interesting part of it is that each one of them will be a different plane wave, thanks to the horizontal potential. The interesting part of it is that each one of them will be a different plane wave, thanks to the horizontal potential:

$$\Psi_k(x) = A_k e^{ip_k x} + B_k e^{-ip_k x} \tag{12}$$

This piecewise wave will satisfy both a boundary condition at the two infinite walls, $x = 0$ and $x = 1$ and a continuity condition at each step x_k.

$$\begin{cases} \Psi_1(x=0) = 0 \\ \Psi_N(x=1) = 0 \end{cases} \text{ and } \begin{cases} \Psi_k\,(x_k) = \Psi_{k+1}\,(x_k) \\ \Psi_k{}'(x_k) = \Psi_{k+1}{}'(x_k) \end{cases}$$

As suggested by BJD, it is easier to deal with a simplified wave function written as

$$\phi_k(x) = \begin{pmatrix} A_k \\ B_k \end{pmatrix} \tag{13}$$

since the exponentials are already determined by the V_k and E of the problem.

We start by imposing $\Psi_1(x=0) = 0$ to the first piece of wave function. What matters is not the value of A_k and B_k, but the relation between them. We can work with an unnormalized $\tilde{\phi}_k$ and later find their value with the normalization.

$$\Psi_1(x=0) = 0 \;\rightarrow B_1 = -A_1 \tag{14}$$

$$\phi_1(x) = A_1 \begin{pmatrix} 1 \\ -1 \end{pmatrix} \rightarrow \tilde{\phi}_1(x) = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \tag{15}$$

Having $\tilde{\phi}_1$, we can find its neighbours $\tilde{\phi}_k$ using the continuity condition. BJD writes it as a $2x2$ matrix that transforms $\tilde{\phi}_k$ into $\tilde{\phi}_{k+1}$:

$$\left.\begin{array}{l}\Psi_k\,(x_k) = \Psi_{k+1}\,(x_k)\\ \Psi_k{}'(x_k) = \Psi_{k+1}{}'(x_k)\end{array}\right\} \rightarrow \mathcal{M}(p_k, x_k)\tilde{\phi}_k = \mathcal{M}(p_{k+1}, x_k)\tilde{\phi}_{k+1} \tag{16}$$

$$\mathcal{M}(p,x) = \begin{pmatrix} e^{ipx} & e^{-ipx} \\ ipe^{ipx} & -ipe^{-ipx} \end{pmatrix} \tag{17}$$

We then go from $\tilde{\phi}_1$all the way to $\tilde{\phi}_N$ iterating with these matrices, and then ask $\tilde{\phi}_N$ if it fulfils the condition $\Psi_N(x=1)=0$. Since we will be only interested on $\tilde{\phi}_N$ while iterating for different values of E, we will find it directly with an effective matrix $\mathcal{M}_{eff}$:

$$\tilde{\phi}_2 = \mathcal{M}^{-1}(p_2, x_1)\mathcal{M}(p_1, x_1)\tilde{\phi}_1 \tag{18}$$

$$\dots$$

$$\tilde{\phi}_N = \underbrace{\prod_{k=1}^{N-1} \mathcal{M}^{-1}(p_{k+1}, x_k)\mathcal{M}(p_k, x_k)}_{\mathcal{M}_{eff}}\,\tilde{\phi}_1 \tag{19}$$

$$\begin{pmatrix} \tilde{A}_N \\ \tilde{B}_N \end{pmatrix} = \mathcal{M}_{eff}\begin{pmatrix} 1 \\ -1 \end{pmatrix} \tag{20}$$

We ask for the last condition $\Psi_N(x=1)=0$, the boundary condition of the right wall. Having imposed that the wave-function is null on the left wall, only eigenenergies will null it on the right wall. We know the energy used is an eigenenergy if $\tilde{\phi}_N$satisfies this last condition.

$$\Psi_N(x=1) = 0 \;\rightarrow \tilde{B}_N = -\tilde{A}_N e^{2ip_N x} \tag{21}$$

$$\tilde{\phi}_N(x) = \tilde{A}_N \begin{pmatrix} 1 \\ -e^{2ip_N x} \end{pmatrix} \tag{22}$$

Following the same idea I used for the one-step case, BJD proposed a candidate function for the root-finding algorithm. This function is zero when the relation between $\tilde{A}_N$ and $\tilde{B}_N$ is $-e^{2ip_N x}$, as found in (29).

$$f(E) = \frac{\tilde{A}_N}{\tilde{B}_N} - \frac{1}{-e^{2ip_N x}} \tag{23}$$

Recovery of the square well solution with $V_k = 0, \forall k$

Repeating the test made on the previous root-finding function, we choose $N=2$ pieces and both $V_1 = V_2 = 0$. We should find the well-known square-potential well roots $E_N = n^2\tilde{E}$ regardless the value of x_1.

The root-finding function # needs to be reformulated again as something that crosses the axis instead of diverging from it. An alternative way to impose the right wall boundary is to directly ask the wave function to be null there. This way, you do not eat up some angles on the fraction between the two unnormalized constants $\tilde{A}_N/\tilde{B}_N$. The function is the value read on $x = 1$:

$$f(E) = \widetilde{\Psi}_N(x = 1) = \tilde{A}_N e^{ip_N x} + \tilde{B}_N e^{-ip_N x}. \tag{24}$$

This has the advantage that we understand its behaviour already. If the last potential piece V_N is smaller than the energy chosen, $\widetilde{\Psi}_N$ should be a pure real, the tail of a sinusoidal wave. And if V_N is bigger than the energy, $\widetilde{\Psi}_N$ should be a pure imaginary, the exponentially decreasing end of an evanescent wave.

Therefore, for increasing E, $\widetilde{\Psi}_N$ makes a transition from "pure" real to "pure" imaginary- The "" are because the other part is never really 0 on the machine. It remains floating between $\pm 10^{-16}$, constantly crossing 0.

At $E \sim V_N$ the real and imaginary parts of $\widetilde{\Psi}_N$ have different signs. If added together, $\mathbb{R}e\widetilde{\Psi}_N + \mathbb{I}m\widetilde{\Psi}_N$ like in #, at $E = V_N$ you get an extra root. That's why the final root-finding function was chosen to be

$$f(E) = \mathbb{R}e\widetilde{\Psi}_N(x = 1) - \mathbb{I}m\widetilde{\Psi}_N(x = 1). \tag{25}$$

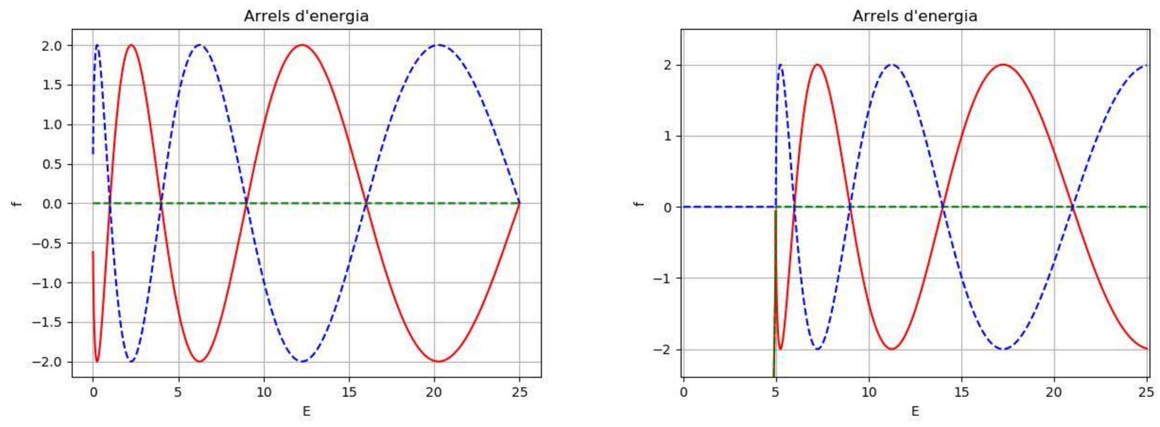


Fig. 10 Computation of the final root-finding function (32) for different values of E. On the left for a square infinite potential, $V_k = 0 \quad \forall k$. It cuts the horizontal axis on 1, 4, 9, etc. On the right, the same but with $V_k = 5E_1 \quad \forall k$. It cuts the horizontal axis on 1+5, 4+5, 9+5, etc. One can see that there is only the imagaginary contibution to the function. It makes sense since all energies are above V_N. This checks that the analytical resolution works fine for the square-well potential regardless of the value of its bottom.

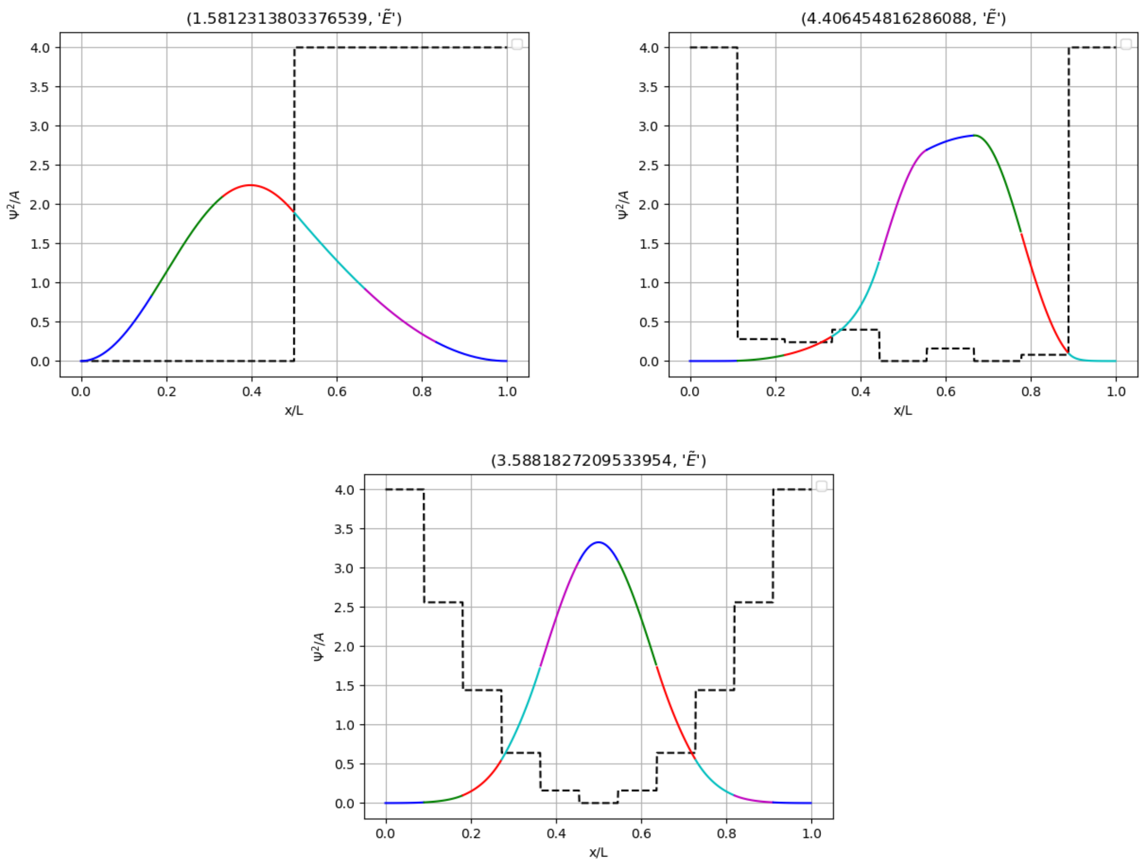


Fig. 11 The ground state of different piecewise potentials. The first, a one-step like potential composed by 6 pieces. The second, a random potential. And the last one, an approximation of an harmonic oscillator, with a potential array of:

$$V_k = [25, 16, 9, 4, 1, 0, 1, 4, 9, 16, 25]\tilde{E} \ . \qquad (26)$$

Phase 3: Make it interactive

In agreement with the other simulations from QuantumUBlab, now that we have solved the system we ought to make it interactive. We found fig. 12 to be the perfect interface. It took a while to learn how to add some buttons and make the array to be drawn with a mouse. Fig 13 shows the interactive version of the simulations on fig 11. You can draw the potential desired clicking with the mouse directly on the grid, instead of typing it on the code.

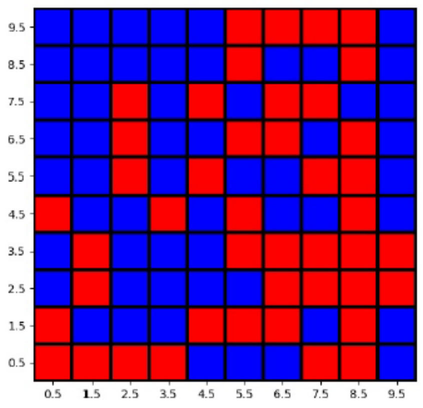


Fig. 12 Mapping of a 2D array to a grid on matplotlib. Found in a forum (https://stackoverflow.com/questions/52566969/python-mapping-a-2d-array-to-a-grid-with-pyplot).

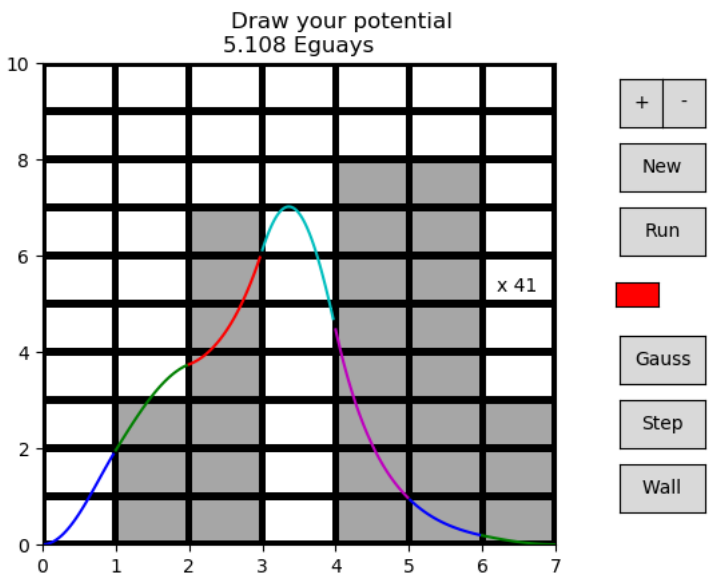


Fig. 13 Interactive piecewise potential inside an infinite square-well. Buttons "+" and "-" add and subtract columns respectively. "New" clears the potential drawn to nothing (a square-well only). "Run" draws the probability density with the code from fig11. The number of "bricks" (amount of potential used, in $E_{guay} \equiv E_1$) is registered. The rest are demos: "Gauss" draws the harmonic oscillator, "Step" a one-step potential and "Wall" a vertical column on the centre. The eigenenergy of the groundstate is displayed in the title.

Make it a game: Draw my potential

The original idea was to leave it as fig 13, but since the program is analytical, it was fast enough to run at each click. That is, while drawing a potential with 3 clicks, the program is fast enough to compute the ground state each one of these times. This made the simulation far more interactive and engaging, as well as fun. When BJD tested it, he asked us to get some kind of game out of this.

Fig. 14 Photos from the YoMo 2019 exposition. "La UB divulga" participating at the event.

We had just seen at YoMo that it is not enough for a simulation to work, it needs some kind of points. So BJD suggested a game where you have to make the probability density to "touch" one of the squares in the array. This challenge was called $survival$ mode, and it was different from the zen mode, which was the previous simulation-only interface.

On the zen mode, you have access to the same buttons as in fig 13, except for the no longer required "Run" button (since now it ran at each click).

On the $survival$ mode, columns/pieces are fixed to $N = 7$, and you have 3 lives "<3<3<3" and a score count. A green square, the "ball", appears on the grid, with a number inside of it. This number is the number of clicks, "moves", you have until this square disappears. If you touch the ball you score the number of moves remaining on it. If you don't, at zero moves the ball disappears and you lose one life. Every time one of these two happens, another ball appears in a new square of the grid.

The lower the ball the easier it is to touch it, so you will have less moves/points for it. The objective is to score the maximum punctuation before losing the three lifes.

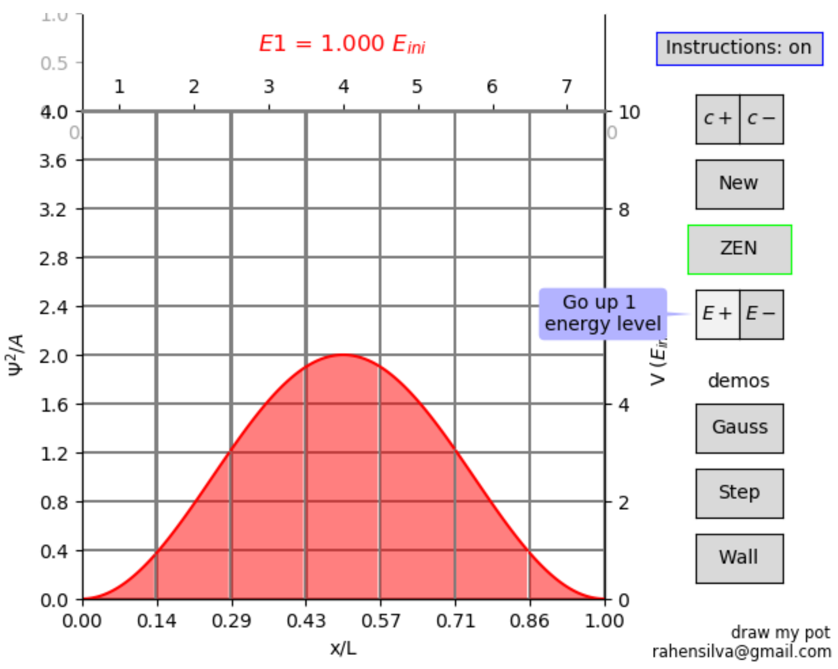


Fig. 15 Zen mode. Same interactive piecewise potential as the one in fig. 13 but without the "Run" button because it runs automatically at each click. "E_{guay}" has been changed to "$E_{initial}$" on the title since it is easier to explain the energy unit as the eigenenergy of the ground state when you have nothing, when you have the infinite square well. The extra buttons are:

- "Zen" Indicates the mode you are on. Click now would switch to $survival$ mode.
- "$E+/E-$" Go from the ground state $E_{n=1}$ to the next one E_2 and back. Up to $n = 5$. The energy you are on is shown at the title "$E1 = \ldots$".

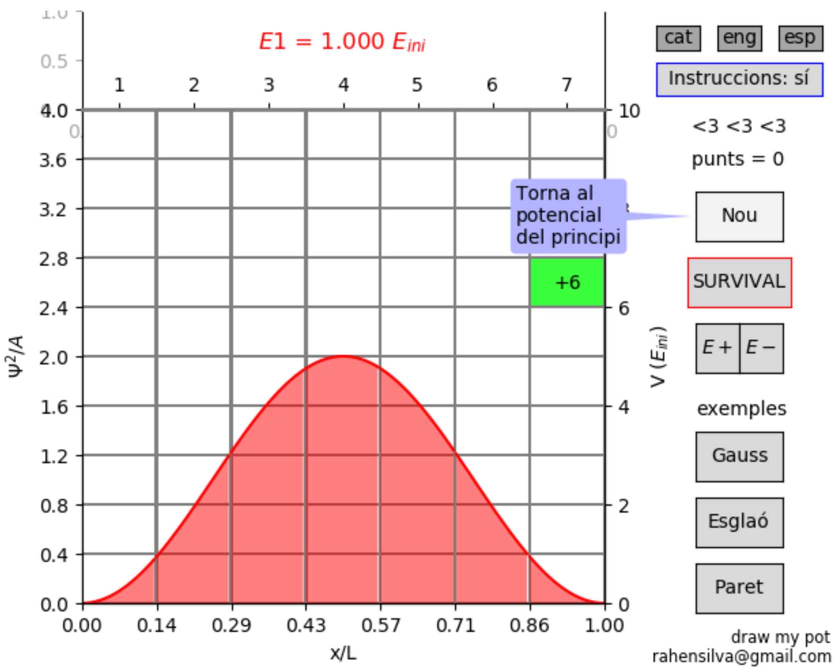


Fig. 16 Survival mode. Start screen of the game. The column buttons have been substituted by the number of lifes and the score. The initial number on the square is exactly its height, and decreases the number of moves you make until disappearing. This is also a latter version with lnguage options on it at the top. This version was presented on May 17th for the "Fira de la ciència UB".

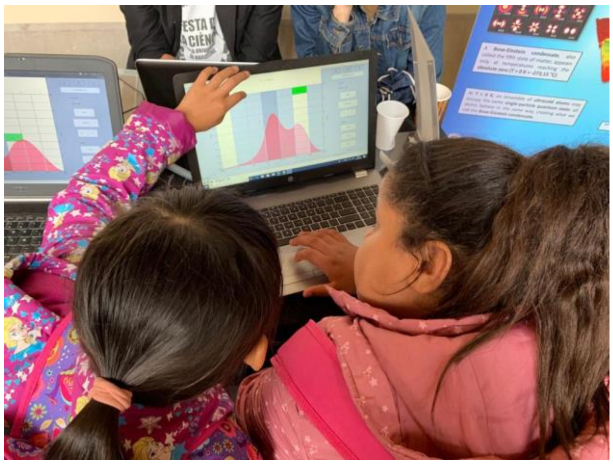


Fig . "Festa de la Ciència UB" 2019. A photo taken when I was not there, and a photo I took of the youngest students to try the game.

Adding different difficulties

When the game was presented at the "Festa de la Ciència" on May 7th, we saw it was too easy. The students who understood how the probability density worked, played all the same strategy shown in fig. 17: filling up to the top all columns except one. You will then have a hole, and the density will reach the top of it touching all the squares inside. If you have made the hole on the 3rd column, e.g., and a square appears on the 6th, you only have to undraw the 6th piece of potential and draw the 3rd one.

This allowed them to score up to 2000 points, compared to the rest of students who usually scored 50-200 points. For this kind of players, three difficulties levels were latter added: $easy$, $fair$ and $hard$.

- $Easy$ is the normal $survival$ mode from fig. 16.
- $Fair$ was an idea from the previous version on fig. 13, the bricks counting. It limits the number of bricks to a reasonable amount of half the grid, or 5 bricks per column.
- $Hard$ only gives a minimum amount of 40%, 4 bricks per column. The ball is forbidden to appear in a couple of impossible positions. The ones that are possible to touch with 40% of the bricks and an infinite amount of moves are permitted. But you still play with the limited amount of moves.

Each 100 points the game increases the difficulty one level. So if you start on $easy(fair)$, at $score = 100$ you move on to $fair(hard)$, and at $score = 200$ to $hard$. You can however start directly on $fair$ or $hard$. the score is then shadowed with the colour of the initial difficulty for the player to show it at the game over screen.

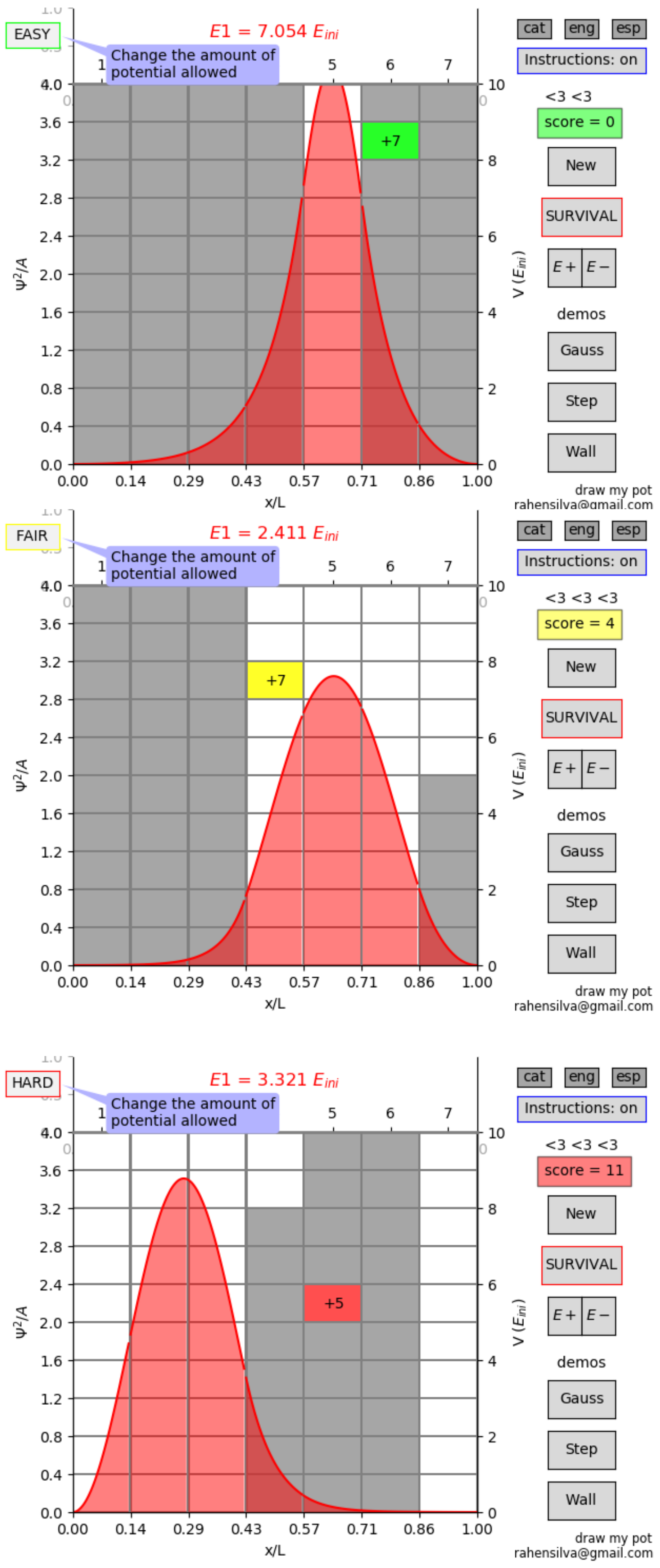


Fig. 17 Last version of "Draw my Potential". New button whit 3 different difficulties added. When clicked, it changes the difficulty and starts a new game. The first figure actually shows the recurrent strategy when with unlimited bricks. Now it is only possible to score with this strategy up to 100 points, then it will jump alone to $fair$ level.

Conclusions

We ended up with a funny game about the first couple systems you are introduced at an undergraduate Quantum Mechanics lecture. This interactive potential turned out to be quite interesting because of its analyticity.

Usually you cannot solve the Schrödinger equation "exactly" on a machine, not even with simple systems. The numerical approximation takes a while to compute and if you want a better result you need more resolution and eventually more computation time or power.

In this case, the resolution doesn't matter. And the only computation needed is a root-finding algorithm. Furthermore, we know how the roots are distributed (near the ground state, with a distribution similar to $E_n = n^2 E_1$) and we only need the first one most of the time.

There is no extended program used as an interface because there is no animation in the whole game, only plots and buttons from the built-in plot software, matplotlib.

This piecewise problem is a recurrent example in Quantum Mechanics because it gives an introduction of how the probability density works. If the game can give this intuiton to a 7 years old students, it sure should ease the early days of the subject for an undergraduate.

Acknowledgements

I want to thank the teachers that game me the opportunity to develop this game with them, Bruno Julià Diaz and Muntsa Guilleumas. They have taught me a lot about how to start a project and the importance of consistency when coming up with ideas. They have also guided me to attack a physical system using the computer. Simplifying it at the beginning and complicating it little by little, correcting it constantly at each step.

I also want to thank the other students/developers of the group, and all my friends who betatested this. Half the ideas in phase 3 are theirs.

Annex 1: Code of phase 1

```
# -*- coding: utf-8 -*-
"""
Rafa da Silva
Created Oct 18 2018

Last check Oct 19 2018

Given the Schrödinger equation and a simple squared potential side L=1,
compute the energy levels and draw its wavefunction. """

import matplotlib.pyplot as plt import
numpy as np


#--DEFINITIONS------------------------------------------------------------

#Ground state of an infinite square potential with side L=1
Eguay = np.pi*np.pi*0.5

#Choose your potential
# V <= 10^4 to avoid overflow

V = 1.668*Eguay
xs = 0.5

#The analitical finder of the energy levels from root.py

def fun(E,V,x):

    alpha = np.sqrt(2*E+0j)
    beta = np.sqrt(2*(E-V)+0j)

    eA = np.exp(2j*alpha*x)
    eB = np.exp(2j*beta*(x-1))

    f = ((eA-1)/(alpha*(eA+1)))/((eB-1)/(beta*(eB+1)))

    return np.absolute(f)-1

#--ITERATIONS-------------------------------------------------------------
#draw potential with the wave function of the first energy level

#Making a list of energy values to try the function with


a = 0
b = 100*Eguay
N = 1000001
deltaE = (b-a)/np.float(N)
EE=np.arange(a,b+deltaE,deltaE)

f0 = fun(EE[0],V,xs)
f1 = fun(EE[1],V,xs)

#see between which two values of EE the function changes
#sign (and thus passes through 0)
```

```
for j in range(len(EE)-1):
        if (f1*f0) < 0 : E=0.5*(EE[j-
                1]+EE[j]) break
        f0 = fun(EE[j],V,xs)
        f1 = fun(EE[j+1],V,xs)

#E=EE[0]
print("simplepot")
print("V=",V/Eguay,"Eguays")
print("E=",E/Eguay,"Eguays")

def phi_fun_L(E,x):

        alpha = np.sqrt(2*E+0j)

        return np.exp(1j*alpha*x)-np.exp(-1j*alpha*x)

def phi_fun_R(E,V,x):

        beta = np.sqrt(2*(E-V)+0j)

        return np.exp(1j*beta)*(np.exp(1j*beta*(x-1))-np.exp(-1j*beta*(x-1)))

def simpson(a,b,h,vect): #Simpson for discrete vectors

        #add the extrems
        add=(vect[a]+vect[b])

        #add each parity its factor
        for i in np.arange(a+2,b,2):
                  add+=2*vect[i]

        for i in np.arange(a+1,b,2):
                  add+=4*vect[i]

        #add the global factor
        add*=(h/np.float(3))

        return add def

c2a2(E,V,x):

        alpha = np.sqrt(2*E+0j)
        beta = np.sqrt(2*(E-V)+0j)

        eA = np.exp(2j*alpha*x)
        eB = np.exp(2j*beta*(x-1))

        f = (alpha*(eA-eA**-1))/(beta*np.exp(2j*beta)*(eB-eB**-1))

        return np.absolute(f)

a=0
b=1
N=100007
deltax=(b-a)/np.float(N)
xx=np.arange(a+deltax,b,deltax)
```

```
phi_val_L=1j*np.zeros([len(xx)])
phi_val_R=1j*np.zeros([len(xx)])

i_L=int((len(xx)-len(xx)%(1/xs))*xs)
i_R=i_L+2

for i in range(0,i_L+1):
        phi_val_L[i]=phi_fun_L(E,xx[i])
for i in range(i_R,len(xx)):
        phi_val_R[i]=phi_fun_R(E,V,xx[i])

phi2_L=np.absolute(phi_val_L)**2
phi2_R=np.absolute(phi_val_R)**2

A2=1/(simpson(0,i_L,deltax,phi2_L)
+np.absolute(c2a2(E,V,xs))*simpson(i_R,len(xx)-1,deltax,phi2_R))

C2=np.absolute(c2a2(E,V,xs))*A2
C2_fake=(phi2_L[i_L]/phi2_R[i_R])*A2

phi2_L=np.multiply(phi2_L,A2)
phi2_R=np.multiply(phi2_R,C2)

suma=0
for i in range(0,i_L+1):
        suma+=phi2_L[i]*deltax for i
in range(i_R,len(xx)):
        suma+=phi2_R[i]*deltax

print('norma:',suma)

#------------------------------------------------------------------------
#What the whole thing looks like

def pot(V,x,xs):
        if x<=xs and x>0:
                return 0
        elif x<1 and x>xs:
                return V
        else:
                return 1000000000

pot_vect =[]
sin2 = [] sin22
= [] for x in
xx:
        sin2.append(2*np.sin(x*np.pi)**2)
        sin22.append((1/xs)*2*np.sin(x*np.pi*(1/xs))**2)
        pot_vect.append(pot(V,x,xs))

plt.plot(xx,np.multiply(pot_vect,2*(1/xs)*(1/V)),
                'r--',label=("$V$:",V/Eguay,"$\tilde{E}$","$x_S$:",xs))
plt.plot(xx[:i_L],phi2_L[:i_L],'g-',label='$\Psi^2_L$')
plt.plot(xx[i_R:],phi2_R[i_R:],'b-',label='$\Psi^2_R$') plt.plot(xx,sin2,'c--',
                label='$\Psi^2(V=0) = 2 sin^2(\pi x / L)/ L$')
plt.plot(xx[:i_L],sin22[:i_L],'m--',
                label='$\Psi^2(V=\infty) = 2 sin^2(\pi x / x_S)/ x_S$')
```

```
plt.grid(True)
plt.legend()
plt.title((E/Eguay, "$\tilde{E}$"))
plt.xlabel("x/L") plt.ylabel("$\Psi^2$")
plt.show()
```

Annex 2: Code of phase 2

```
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 14 07:49:48 2018

@author: rafa

The program follows the nomenclature from:
	Notes on the solution to 1D Schrodinger equation for
	piecewise potentials - bjd
And finds its eigenenergies. """

import matplotlib.pyplot as plt import
numpy as np

#------------------------------------------------------------
#units
Eguay = np.pi*np.pi*0.5

#------------------------------------------------------------
#Generate the easy potential

#walls only
#Vk=[0,2]

#step
#Vk=[0,2]

#this is symetrical from 0 to 25
#Vk=[0,30,0]

#harmonic oscillator with max value = 25
Vk=[25,16,9,4,1,0,1,4,9,16,25]

Vk=np.dot(Vk,Eguay)

N=len(Vk) #number of potential columns

xk=np.zeros(N)
for k in range(N):
	xk[k]=(k+1)/N
```

```
#-----------------------------------------------------------
#find the eigen-energies

phi=1j*np.zeros(shape=(N,2))
phi_bis=1j*np.zeros(shape=(N,2))

phi[0][0]=1
phi[0][1]=-1

phi_bis[0][0]=1
phi_bis[0][1]=-1

invM=1j*np.zeros(shape=(2,2))
M=1j*np.zeros(shape=(2,2))

dE=Eguay/np.float(1007)
EE=np.arange(0.9*Eguay,10*Eguay,dE)

elf=[]
relf=[]
ielf=[]
melf=[]

elf_min=np.infty
E_min=-123

kk=1j*np.zeros(N)

for E in EE:

    for k in range(N):
        kk[k]=np.sqrt((2+0j)*(E-Vk[k]))

    M_eff=np.eye(2)+1j*np.zeros(shape=(2,2))

    for k in range(N-1):
        #k_python = k_notes -1
        #k_notes = 1 : N-1
        #k_python = 0 : N-2

        #make invM21 and M11 phi_2

        ex1=np.exp(1j*kk[k]*xk[k])
        ex_1=ex1**-1
        ex2=np.exp(1j*kk[k+1]*xk[k])
        ex_2=ex2**-1

        M[0][0]=ex1
        M[0][1]=ex_1
        M[1][0]=1j*kk[k]*ex1
        M[1][1]=-1j*kk[k]*ex_1

        invM[0][0]=0.5*ex_2
        invM[0][1]=-0.5j*(1/kk[k+1])*ex_2
        invM[1][0]=0.5*ex2
        invM[1][1]=0.5j*(1/kk[k+1])*ex2
```

```
            M_eff=np.dot(np.dot(invM,M),M_eff)

        phi[-1]=np.dot(M_eff,phi[0])

    elfE=np.log(phi[-1][1]+0j)-np.log(phi[-1][0]+0j)-2j*kk[-1]+1j*np.pi

    elfE=np.exp(elfE)-1

#       elf.append(np.absolute(elfE))

    relf.append(np.real(elfE))

    ielf.append(np.imag(elfE))

    melf.append(np.real(elfE)+np.imag(elfE))

    if len(melf)>3:
        if melf[-1]*melf[-3]<0:
            if relf[-1]*relf[-3]<0 or ielf[-1]*ielf[-3]<0: E_min=E
                break

#    if elf_min>np.absolute(elf[-1]):
#        elf_min=np.absolute(elf[-1])
#        E_min=E

print("V =",Vk[-1]/Eguay,"Eguays")
print("E =",E_min/Eguay,"Eguays")

#------------------------------------------------------------

E = E_min

dx=1/np.float(1000)
x_vect=np.arange(0,1+dx,dx)

for k in range(N):
    kk[k]=np.sqrt((2+0j)*(E-Vk[k]))

for k in range(N-1):
    #k_python = k_notes -1
    #k_notes = 1 : N-1
    #k_python = 0 : N-2

    #make invM21 and M11 phi_2

    ex1=np.exp(1j*kk[k]*xk[k])
    ex_1=ex1**-1
    ex2=np.exp(1j*kk[k+1]*xk[k])
    ex_2=ex2**-1

    M[0][0]=ex1
    M[0][1]=ex_1
    M[1][0]=1j*kk[k]*ex1
    M[1][1]=-1j*kk[k]*ex_1

    invM[0][0]=0.5*ex_2
```

```
        invM[0][1]=-0.5j*(1/kk[k+1])*ex_2
        invM[1][0]=0.5*ex2
        invM[1][1]=0.5j*(1/kk[k+1])*ex2

        phi[k+1]=np.dot(np.dot(invM,M),phi[k])

psi_vect=np.zeros(shape=(len(x_vect)))
V_vect=np.zeros(shape=(len(x_vect)))

ixk=[0]
ik=0


for ix in range(len(x_vect)):
        if x_vect[ix]>xk[ik]:
                ik+=1 ixk.append(ix-
                1) ixk.append(ix)

        V_vect[ix]=Vk[ik]*(4/np.max(Vk))

        ex=np.exp(1j*kk[ik]*x_vect[ix])

        psi_vect[ix]=np.absolute(
                        phi[ik][0]*ex+phi[ik][1]*ex**-1)**2



def simpson(h,vect): #Simpson for discrete vectors

        #add the extrems
        add=(vect[0]+vect[len(vect)-1])

        #add each parity its factor
        for i in np.arange(2,len(vect)-1,2):
                        add+=2*vect[i]

        for i in np.arange(1,len(vect)-1,2):
                        add+=4*vect[i]

        #add the global factor
        add*=(h/np.float(3))

        return add psi_vect=np.dot(simpson(dx,psi_vect)**-1,psi_vect)


ixk.append(ix-1) plt.plot(x_vect,V_vect,'k--') color=['b-',

'g-', 'r-', 'c-', 'm-'] for k in range(N):
        plt.plot(x_vect[ixk[2*k]:ixk[2*k+1]],
                        psi_vect[ixk[2*k]:ixk[2*k+1]], color[k%5])

plt.grid(True)
plt.legend()
plt.title((E/Eguay, '$\tilde{E}$')) plt.xlabel("x/L")
plt.ylabel("$\Psi^2 /A$")
plt.show()
```

Annex 3: Code of phase 3

```
# -*- coding: utf-8 -*-
""""
Created on Mon Feb 11 15:35:06 2019
@author: rafa
The program follows the nomenclature from:
        Notes on the solution to 1D Schrodinger equation for
        piecewise potentials - bjd
And draws its eigenenergies. It has
4 sections:
        -WAVE-FUNCTION: computes eigen energies for a given piecewise potential
        -CANVAS: draws both the input potential and the output wave-function
        (gives WAVE-FUNCTION the potential and gets its energy to plot)
        -TOOLS: additional featurings to interact easier with CANVAS,
        or change its properties (such like asking it to ask WAVE-FUNCTION
        to give different energies apart from the ground state)
        -SCORE: for the survival mode
The grid idea is taken from:
https://stackoverflow.com/questions/52566969/python-mapping-a-2d-array-to-a-grid-with-pyplot
The annotations are taken from:
https://stackoverflow.com/questions/7908636/possible-to-make-labels-appear-when-hovering-over- The detrans[late]
inversion is from:
https://stackoverflow.com/questions/483666/python-reverse-invert-a-mapping
""""

from matplotlib import pyplot as plt from
matplotlib import colors
from matplotlib.widgets import Button
from matplotlib.ticker import FormatStrFormatter from
matplotlib.patches import Ellipse
import numpy as np
import random

# Default number of potential pieces
pieces = 7

data_new = np.zeros((pieces,10),dtype=int) Vk =
np.zeros(pieces,dtype=int)
data = np.copy(data_new)

# Maximum number of potential pieces (buttons won't work for more than 7)
max_pieces = 11

# Default energy level
level = 1

# Maximum number of energy levels to compute
N_root = 5

# Survival initial stuff
max_value = 9
score=0
ball_value=9
lives=3
x_ball=0
y_ball=9
X_hovered=30
Y_hovered=30
difficulty=10
no_bricks=False
score_color=(0,1,0)
ball_color=(0,1,0)
```

```
#-----------------------------------------------------------------------
# WAVE FUNCTION
# Generated following the nomenclature of BJD derivation.

#k_code = k_notes -1
#k_notes = 1 : N-1
#k_code = 0 : N-2
#-----------------------------------------------------------------------


# The energy at the ground state os a square infinite potential
Eguay = np.pi*np.pi*0.5

# Vectors to plot dx=1/np.float(420)
x_vect=np.arange(0,1+dx,dx)
psi_vect=np.zeros(shape=(len(x_vect)))

# Boundaries position for L=1
xk=np.zeros(pieces)
for k in range(pieces):
        xk[k]=(k+1)/pieces

# We index every piece with the index k, because every
# piece has its own wave vector k: kk=sqrt(2(E-Vk))
kk=1j*np.zeros(pieces)

# phi(k)=Ak*exp(ikkx)+Bk*exp(-ikkx)=[Ak Bk]
phi=1j*np.zeros(shape=(pieces,2))

# Boundary condition for the left wall:
# At x=0 the wave has to be null
# We impose that A0=-B0, and give A0 the arbitrary value 1
# (Arbitrary because the wave will be normalized)
phi[0][0]=1
phi[0][1]=-1

# Continuity condition from piece to piece.
invM=1j*np.zeros(shape=(2,2))
M=1j*np.zeros(shape=(2,2))

# Simpson integration for discrete vectors.
# We use it to normalize the squared wave
def simpson(h,vect):

        #add the extrems
        add=(vect[0]+vect[len(vect)-1])

        #add each parity its factor
        for i in np.arange(2,len(vect)-1,2):
                add+=2*vect[i]

        for i in np.arange(1,len(vect)-1,2):
                add+=4*vect[i]

        #add the global factor
        add*=(h/np.float(3))
```

```
    return add

# Bisection root-finding algorithm
#
# Improved to recursively:
#    - check both sides of c if needed
#   - check an extra c before discarting an interval
#   - divide the given function by the roots already founded
#     (given the polinomical shape of our function)
#
# To see how it works, uncomment the prints and try fun(x)=(x-1)(x-2)(x-3).
#
# It doesn't find all the roots. If an interval has 2 roots, but the extra c
# doesn't fall between them, you will miss them.

def bis(a, b, eps, fun, roots):

    def fun_eff(x, fun, roots):
        fx=1
        for xr in roots:
            fx/=x-xr
        fx*=fun(x)
        return fx

    fa=fun_eff(a, fun, roots)
    fb=fun_eff(b, fun, roots)

    c=0.5*(a+b)
    fc=fun_eff(c, fun, roots)

#   print(a,b,c)

    if fa==0:
        roots.append(a)
#       print('OK fa==0')
#       print(roots)
#       print('')
#       print('R3? fa==0')
        bis(a+eps, b, eps, fun, roots)
    if fb==0:
        roots.append(b)
#       print('OK fb==0')
#       print(roots)
#       print('')
#       print('L3? fb==0')
        bis(a+eps, b, eps, fun, roots)
    if fc==0:
        roots.append(c)
#       print('OK fc==0')
#       print(roots)
#       print('')
#       print('R3? fc==0')
        bis(c+eps, b, eps, fun, roots)
#       print('L3? fc==0')
        bis(a, c-eps, eps, fun, roots)

    if fa*fb<0:
        if abs(a-b)<eps:
                 roots.append(c)
#           print('OK fa*fb<0 & abs(a-b)<eps')
#           print(roots)
#           print('')
```

```
        else:
            if fa*fc<0:
#          print('L! fa*fb<0 & fc*fa<0')
                bis(a, c, eps, fun, roots)
#          print('')
#          print('R2? fa*fb<0 & fc*fa<0')
                bis(b, c, eps, fun, roots)
                elif fc*fb<0:
#           print('R! fa*fb<0 & fc*fb<0')
                    bis(c, b, eps, fun, roots)
#           print('')
#           print('L2? fa*fb<0 & fc*fa<0')
                    bis(a, c, eps, fun, roots)

    if fa*fb>0:
        if fa*fc<0:
#         print('L1? fa*fb>0 & fc*fa<0')
            bis(a, c, eps, fun, roots)
        if fc*fb<0:
#         print('R1? fa*fb<0 & fc*fb<0')
            bis(c, b, eps, fun, roots)
    return roots

# Boundary condition for the right wall: At x=1 the wave has to be null
# We compute the left value as a function of the energy.
# Having imposed that the wave is null on the left wall, only eigen-energies
# will also null the wave on the right wall.

def wf_right_wall(E):
    global Vk, pieces, phi, invM, M, kk, xk, Eguay

    for k in range(pieces):
        kk[k]=np.sqrt((2+0j)*(E-Vk[k]*Eguay))

    # Accumulate the continuity condition from piece to piece
    # in one effective matrix:

    M_eff=np.eye(2)+1j*np.zeros(shape=(2,2))

    for k in range(pieces-1):
        ex1=np.exp(1j*kk[k]*xk[k])
        ex_1=ex1**-1
        ex2=np.exp(1j*kk[k+1]*xk[k])
        ex_2=ex2**-1

        M[0][0]=ex1
        M[0][1]=ex_1
        M[1][0]=1j*kk[k]*ex1
        M[1][1]=-1j*kk[k]*ex_1

        invM[0][0]=0.5*ex_2
        invM[0][1]=-0.5j*(1/kk[k+1])*ex_2
        invM[1][0]=0.5*ex2
        invM[1][1]=0.5j*(1/kk[k+1])*ex2

        M_eff=np.dot(np.dot(invM,M),M_eff)

    # Multiply the last piece of wave by the effective matrix

    phi[-1]=np.dot(M_eff,phi[0])

    wf_value = phi[-1][0]*np.exp(1j*kk[-1])+phi[-1][1]*np.exp(-1j*kk[-1])

    # If the last piece of potential is smaller/bigger than the energy,
    # the last piece of wave will be real/imaginary. Increasing E, when
    # E=Vk[-1], the wave makes a transition from 'pure' real to 'pure'
    # imaginary values. '' because the other part is never 0, as it remains
    # floating between +- e-16 and not 0. I give one part minus the other
    # to only cross the axis when one of the parts 'purely' does it.
    # I don't give one part PLUS the other, because at E<Vk[-1] the real part
    # is -ve/+ve and at E>Vk[-1] the imaginary part is +ve/-ve.
```

```
    return np.real(wf_value)-np.imag(wf_value)

# Computes only the first energy level to inmediate plot
def eigen_energies(N_root_computed):
    global pieces, Vk, E, ixk, psi_vect, Eguay, phi, invM, M, kk, xk, DE, dE

    # Step to locate an interval for each root
    DE=Eguay/17
    # Root value precision
    dE=Eguay/1000

    # Will return this vector
    E_root=[]

    # Start from E=0.99*Eguay
    wave1=wf_right_wall(E)

    # Look for the first value
    while len(E_root)<N_root_computed:
        E+=DE
        wave0=wave1
        wave1=wf_right_wall(E)

        # If you find an interval, polish it to dE
        if wave0*wave1<0:
                bis(E-DE,E,dE,wf_right_wall,E_root)

    # In case you found more than one root, give the first one
    E_root.sort()
    return E_root

# Computes the squared wave function for a given energy E
def psivect():
    global pieces, Vk, E, ixk, psi_vect, Eguay, phi, invM, M, kk, xk, DE, dE

    for k in range(pieces):
        kk[k]=np.sqrt((2+0j)*(E-Vk[k]*Eguay))

    # Apply in each region the continuity condition to find each piece
    # of the wave function.

    for k in range(pieces-1):
        #k_python = k_notes -1
        #k_notes = 1 : N-1
        #k_python = 0 : N-2

        ex1=np.exp(1j*kk[k]*xk[k])
        ex_1=ex1**-1
        ex2=np.exp(1j*kk[k+1]*xk[k])
        ex_2=ex2**-1

        M[0][0]=ex1
        M[0][1]=ex_1
        M[1][0]=1j*kk[k]*ex1
        M[1][1]=-1j*kk[k]*ex_1
```

```
invM[0][0]=0.5*ex_2
invM[0][1]=-0.5j*(1/kk[k+1])*ex_2
invM[1][0]=0.5*ex2
invM[1][1]=0.5j*(1/kk[k+1])*ex2

phi[k+1]=np.dot(np.dot(invM,M),phi[k])
```

```
        ixk=[0]
        ik=0

        # Build the squared wave-function vector from the wave function computed
        # above and x_vect defined at the begining of the code.

        for ix in range(len(x_vect)):
                if x_vect[ix]>xk[ik]:
                        ik+=1 ixk.append(ix-
                        1) ixk.append(ix)

                ex=np.exp(1j*kk[ik]*x_vect[ix])

                psi_vect[ix]=np.absolute(phi[ik][0]*ex+phi[ik][1]*ex**-1)**2

        ixk.append(len(x_vect)-1)

        # Normalize it
        psi_vect = np.dot(simpson(dx,psi_vect)**-1,psi_vect)

        return psi_vect


#-------------------------------------------------------------------------
# CANVAS
# what is being constantly drawn
#-------------------------------------------------------------------------

# Function that draws it all before plotting
def draw_wave():

        global trans, late, data, cmap, ax, txt, bricks, \ left_extrem,
        right_extrem, top_extrem, bottom_extrem, \ data_new, ax, x_vect,
        V_vect, Vk, ixk, psi_vect, E, \ N_root, eigen_energies_list, level, \
        x_ball, y_ball, \
        score, lives, ball_value, value_txt, score_txt, lives_txt,\
        bdiff, axdiff, ldiff, difficulty, score_color, ball_color


        if score>=100:
                if bdiff.label.get_text()=='EASY':
                        difficulty=5 ball_color=(1,1,0)
                        bdiff.label.set_text('FAIR.')
                        axdiff.spines['top'].set_color((1,1,0))
                        axdiff.spines['bottom'].set_color((1,1,0))
                        axdiff.spines['right'].set_color((1,1,0))
                        axdiff.spines['left'].set_color((1,1,0))
                if bdiff.label.get_text()=='FAIR':
                        difficulty=4 ball_color=(1,0,0)
                        bdiff.label.set_text('HARD')
                        axdiff.spines['top'].set_color((1,0,0))
                        axdiff.spines['bottom'].set_color((1,0,0))
                        axdiff.spines['right'].set_color((1,0,0))
                        axdiff.spines['left'].set_color((1,0,0))

        if score>=200:
```

```
if bdiff.label.get_text()=='FAIR.':
        difficulty=4
        ball_color=(1,0,0)
```

```
                bdiff.label.set_text('HARD')
                axdiff.spines['top'].set_color((1,0,0))
                axdiff.spines['bottom'].set_color((1,0,0))
                axdiff.spines['right'].set_color((1,0,0))
                axdiff.spines['left'].set_color((1,0,0))

        ax.cla() Vk=data_to_Vk(data)

        E=eigen_energies(level)[level-1]

#    if draw_level==1:
#        E=eigen_energies(1)[0]
#        level=1
##          print(level)
#    elif draw_level>1 and draw_level<=N_root:
#        E=eigen_energies_list[draw_level-1]
#    else:
#        E=0
        psi_vect=psivect()

#    color=['b-', 'g-', 'r-', 'c-', 'm-']
        color=['r']

        touched=np.zeros(pieces,dtype=int)

        for k in range(pieces):
            x    =    np.dot(pieces,x_vect[ixk[2*k]:ixk[2*k+1]])    y    =
            np.dot(10/4,psi_vect[ixk[2*k]:ixk[2*k+1]])
            ax.fill(np.append(x, [x[-1], x[0]]),
                    np.append(y, [0,0]),
                    color[k%len(color)], alpha=(0.5+0.12*(level-1)))
            ax.plot(x, y, color[k%len(color)])

            touched[k]=int(max(y)-0.5)
#        print(touched[-1][0]+1,touched[-1][1]+1)
#    print('')

        play_ball(touched)
        ball_the_data()

        fig.suptitle(t = '$E %.i$ = %.3f $E_{ini}$'%(level,(E/Eguay)), x = 0.42, y = 0.97, color='
        ax.set_xticks([piece+0.5 for piece in range(pieces)])
        ax.set_xticklabels([piece+1 for piece in range(pieces)])
        ax.yaxis.set_label_text('V ($E_{ini}$)')
        ax.spines['top'].set_color(gridc)
        ax.spines['top'].set_linewidth(2*gridw)
        ax.pcolormesh(np.transpose(data[::-1]), cmap=cmap(data), edgecolors=gridc, linewidths=grid
        ax.axis([0,pieces,0,10])

        unball_the_data()

        if lives==0: fig.canvas.mpl_disconnect(cid_click)
            lives_txt.remove()
            lives_txt = fig.text(0.815, 0.815, trans[late]['game over'])#, transform=ax.transAxes)
            score_txt.remove()
            score_txt = fig.text(0.82, 0.765, trans[late]['score = %d']%(score),bbox={'facecolor':
        else:
```

```
lives_txt.remove()
lives_txt = fig.text(0.82, 0.815, '<3 '*lives)#, transform=ax.transAxes)
score_txt.remove()
score_txt     =     fig.text(0.82,     0.765,     trans[late]['score     =     %d']%(score)*(int(lives/abs(liv
        value_txt.remove()
        value_txt = fig.text(axl+(axr-axl)*(1-(x_ball+0.675)/pieces),\
                             axb+(axt-axb)*((y_ball+0.35)/10), \
                             '+%d'%(ball_value)*(int(lives/abs(lives))))#, transform=ax.transA

    return True

# in case of resizing the window, take all this into account
axl=0.12
axr=0.72
axt=0.85
axb=0.1

# First draw, when initialized fig =
plt.figure()
fig.patch.set_facecolor('0.65')

# Upper white rectangle
upper_ax = plt.Axes(fig, [axl, axt, axr-axl, 1-axt], )
fig.add_axes(upper_ax) upper_ax.tick_params(color='0.65',
labelcolor='0.65') upper_ax.spines['top'].set_color('w')

# Fake plot below the real one, to put the wave-funtion axis fake_ax =
fig.add_subplot(111) fake_ax.set_xticks([(1/pieces)*i for i in
range(pieces+1)])
fake_ax.xaxis.set_major_formatter(FormatStrFormatter('%.2f'))
fake_ax.set_xlabel('x/L')
fake_ax.set_yticks([0.1*i for i in range(10+1)])
fake_ax.set_yticklabels([(4*i)/10 for i in range(10+1)])
fake_ax.set_ylabel('$\Psi^2 /A$')

# Real plot where both the potential and the wave are represented
# but only with the potential axis. (The rest is drawn on draw())
ax = fake_ax.twiny().twinx()
ax.xaxis.tick_top()
ax.xaxis.set_label_position('top')
ax.yaxis.tick_right()
ax.yaxis.set_label_position('right')

# colors to plot the matrix 'data' as a relief map:
#                   background, hover, potential
def cmap(data):
    global ball_value, ball_color try:
        ball_value

    except NameError: #The game just started
        ball_value = max_value

    if bmode.label.get_text()=='SURVIVAL':
        value_color=(max(ball_color[0],0.7*(1-ball_value/(max_value))),
                     max(ball_color[1],0.7*(1-ball_value/(max_value))),
                     max(ball_color[2],0.7*(1-ball_value/(max_value))))

        cmap = colors.ListedColormap(['white','0.5','0.65',value_color])
        if data.min()==1:
            cmap = colors.ListedColormap(['0.5','0.65', value_color])
        if data.min()==2:
            cmap = colors.ListedColormap(['0.65', value_color])
else:
      cmap = colors.ListedColormap(['white','0.5','0.65'])
      if data.min()==1:
                   cmap = colors.ListedColormap(['0.5','0.65'])
```

```
        if data.min()==2:
                cmap = colors.ListedColormap(['0.65'])
    return cmap
# color of the grid
gridc='0.5'
# width of the grid
gridw=1

# recurrent function to connect with the wave-function
def data_to_Vk(data):

    global Vk

    translated_Vk = np.copy(Vk)

    for X in range(len(Vk)):
        translated_Vk[X]=0 for
        Y in range(10):
            translated_Vk[X]+=data[len(Vk)-1-X][Y]/2 return

    translated_Vk

# Changes the color of the grid and calls draw_wave
def ClickColor(event):
    global data, cmap, X_hovered, Y_hovered, ax, txt, bricks, \ left_extrem,
    right_extrem, top_extrem, bottom_extrem, \ data_new, ax, x_vect,
    V_vect, Vk, ixk, psi_vect, E, \
    lives, score, score_txt, lives_txt,\
    level, no_bricks

    # If the mouse is inside the grid
    if event.x < right_extrem and event.x > left_extrem and event.y < top_extrem and event.y >

        #locate the cell X=int(event.xdata-
        event.xdata%1) Y=int(event.ydata-
        event.ydata%1) X=(len(data)-1)-X

        # hover the cell
        X_hovered=X
        Y_hovered=Y

        # save the grid before changing it
        data0=np.copy(data)

        # if I don't have enough bricks, draw the remaining only
#       if (Y+1)>(bricks(data)+data_to_Vk(data)[pieces-1-X]):
#           print('you try',(Y+1),'when',data_to_Vk(data)[pieces-1-X])
#           print('can give',(bricks(data)+data_to_Vk(data)[pieces-1-X]))
#           Y=bricks(data)+data_to_Vk(data)[pieces-1-X]-1
#           if Y==-1:
#               no_bricks=True
#               print(X,Y,'no bricks',bricks(data))

#       print('-----------------------------')
#       print(Y+1,'-',data_to_Vk(data)[pieces-1-X],'=<',bricks(data),
#           (Y+1-data_to_Vk(data)[pieces-1-X])<=bricks(data))
#       #if I don't have enough bricks, erase and draw the remaining only
#        if (Y+1)>(bricks(data)+data_to_Vk(data)[pieces-1-X]):
#            print('you try',(Y+1),'when',data_to_Vk(data)[pieces-1-X])
#            if (Y+1)<data_to_Vk(data)[pieces-1-X]:
#                print('okey u erasing')
#                no_bricks=False
#            else:
```

```
#               print('you have no bricks')

            print(bricks(data))
            #only give the remaining bricks
            if bricks(data)>0 and \
                (Y+1)>(bricks(data)+data_to_Vk(data)[pieces-1-X]):
                Y=bricks(data)+data_to_Vk(data)[pieces-1-X]-1 print('te doy
                menos')
            #or if you are erasing
            elif bricks(data)<=0:
                if (Y+1)<=data_to_Vk(data)[pieces-1-X]:
                    Y=Y_hovered
                    no_bricks=False print('vale,
                    borrando')
                else: no_bricks=True print('click
                    invalido')

            # if V=click you can delete the entire column
            if Y+1==data_to_Vk(data)[pieces-1-X]:
                data[X][:Y+1]=0
            else:
                data[X][:Y+1]=2
            data[X][Y+1:]=0

#       # if V=1 you can delete the brick
#       if Y==0 and data_to_Vk(data)[pieces-1-X]==1:
#           data[X][:Y+1]=0
#       else:
#           data[X][:Y+1]=2
#       data[X][Y+1:]=0

            # do I have enough bricks?
            if no_bricks:
                print('no bricks')
                data=np.copy(data0)
                return True
            else: draw_wave()
                event.canvas.draw()

# only changes the color of the grid
def HoverColor(event):
        global trans, late, data, cmap, X_hovered, Y_hovered, X_hovered, Y_hovered, \
        ax, left_extrem, right_extrem, top_extrem, bottom_extrem,  \
        gridc, gridw, \
        binfo, axinfo, linfo, axmore, lmore, axless, lless, \ axnew, lnew,
        axmode, lmode, axup, lup, axdown, ldown, \ axgauss, lgauss, axstep,
        lstep, axwall, lwall

        # If the mouse is outside the grid
        if event.x > right_extrem or event.x < left_extrem or event.y > top_extrem or event.y < bo

            # dishover
            X_hovered=30
            Y_hovered=30


            data[data==1]=0
```

```
# draw dishovered grid
ball_the_data()
ax.pcolormesh(np.transpose(data[::-1]), cmap=cmap(data), edgecolors=gridc, linewidths=
event.canvas.draw()
```

```
unball_the_data()

#Also label all buttons the mouse passes by

if event.inaxes == axinfo:
        if not linfo.get_visible():
                linfo.set_visible(True)
else:
        linfo.set_visible(False)

if binfo.label.get_text()==trans[late]['Instructions: on']:

        if axdiff.get_visible():
                if event.inaxes == axdiff:
                        if not  ldiff.get_visible():
                                ldiff.set_visible(True)
                else:
                        ldiff.set_visible(False)

        if axmore.get_visible():
                if event.inaxes == axmore:
                        if not lmore.get_visible():
                                lmore.set_visible(True)
                else:
                        lmore.set_visible(False)

        if axless.get_visible():
                if event.inaxes == axless:
                        if not lless.get_visible():
                                lless.set_visible(True)
                else:
                        lless.set_visible(False)

        if axnew.get_visible():
                if event.inaxes == axnew:
                        if not lnew.get_visible():
                                lnew.set_visible(True)
                else:
                        lnew.set_visible(False)

        if axmode.get_visible():
                if event.inaxes == axmode:
                        if not lmode.get_visible():
                                lmode.set_visible(True)
                else:
                        lmode.set_visible(False)

        if axup.get_visible():
                if event.inaxes == axup:
                        if not lup.get_visible():
                                lup.set_visible(True)
                else:
                        lup.set_visible(False)

        if axdown.get_visible():
                if event.inaxes == axdown:
                        if not ldown.get_visible():
```

```
                ldown.set_visible(True)
        else:
            ldown.set_visible(False)

    if axgauss.get_visible():
        if event.inaxes == axgauss:
            if not lgauss.get_visible():
                    lgauss.set_visible(True)
            else:
                lgauss.set_visible(False)

        if axstep.get_visible():
            if event.inaxes == axstep:
                if not lstep.get_visible():
                    lstep.set_visible(True)
            else:
                lstep.set_visible(False)

        if axwall.get_visible():
            if event.inaxes == axwall:
                if not lwall.get_visible():
                    lwall.set_visible(True)
            else:
                lwall.set_visible(False)


    return True

# If the mouse is inside the grid
elif isinstance(event.xdata, float) and isinstance(event.ydata, float):

    linfo.set_visible(False)
    ldiff.set_visible(False)
    lmore.set_visible(False)
    lless.set_visible(False)
    lnew.set_visible(False)
    lmode.set_visible(False)
    lup.set_visible(False)
    ldown.set_visible(False)
    lgauss.set_visible(False)
    lstep.set_visible(False)
    lwall.set_visible(False)

    X=int(event.xdata-event.xdata%1)
    Y=int(event.ydata-event.ydata%1)
    X=(len(data)-1)-X

    # don't hover twice
    if X==X_hovered and Y==Y_hovered:
        return True

    else:

        #hover    this
        X_hovered=
        X
        Y_hovered=
        Y

        data0=np.copy(data)

        #only hover the remaining bricks
```

```
if bricks(data)>0 and \
    (Y+1)>(bricks(data)+data_to_Vk(data)[pieces-1-X]):
    Y=bricks(data)+data_to_Vk(data)[pieces-1-X]-1
#or if you are erasing
elif bricks(data)<=0:
        if (Y+1)<=data_to_Vk(data)[pieces-1-X]:
                Y=Y_hovered
        else:
                Y=-1

data[X][:Y+1]=1
```

```
                ball_the_data()
                ax.pcolormesh(np.transpose(data[::-1]), cmap=cmap(data), edgecolors=gridc, linewid
                event.canvas.draw()
                unball_the_data()

                data=data0


                return True



plt.subplots_adjust(left=axl,right=axr,top=axt, bottom=axb)
left_extrem=(fig.get_size_inches()*fig.dpi)[0]*axl+gridw
right_extrem=(fig.get_size_inches()*fig.dpi)[0]*axr-gridw
top_extrem=(fig.get_size_inches()*fig.dpi)[1]*axt-gridw
bottom_extrem=(fig.get_size_inches()*fig.dpi)[1]*axb+gridw

def ResizeExtrems(event):
    global left_extrem, right_extrem, top_extrem, bottom_extrem, axr, axl, axt, axb, gridw
    left_extrem=(fig.get_size_inches()*fig.dpi)[0]*axl+gridw
    right_extrem=(fig.get_size_inches()*fig.dpi)[0]*axr-gridw
    top_extrem=(fig.get_size_inches()*fig.dpi)[1]*axt-gridw
    bottom_extrem=(fig.get_size_inches()*fig.dpi)[1]*axb+gridw

# okey, connect the mouse
cid_click = fig.canvas.mpl_connect('button_press_event', ClickColor) cid_hover =
fig.canvas.mpl_connect('motion_notify_event', HoverColor) cid_resize =
fig.canvas.mpl_connect('resize_event', ResizeExtrems)

#-----------------------------------------------------------------------
# TOOLS
# Buttons and miscelaneous at the left part of the screen.
# From top to bottom
#-----------------------------------------------------------------------

#language buttons
#changes the language of the entire plot

def CAT(event):
    global trans, late, lated, language_reboot lated=late
    late=1
    language_reboot(event)
#    print(trans[late]['turtle'])

def ENG(event):
    global trans, late, lated, language_reboot lated=late
    late=0
    language_reboot(event)
#    print(trans[late]['turtle'])

def ESP(event):
    global trans, late, lated, language_reboot lated=late
    late=2
    language_reboot(event)
```

```
#    print(trans[late]['turtle'])

def language_reboot(event):
        global trans, late, detrans, lated,\
        lives, lives_txt, score_txt,\
        binfo, linfo, lmore, lless, bnew, lnew, bmode, lmode, lup, ldown,\
        demos_txt, lgauss, bstep, lstep, bwall, lwall, ldiff

        if lives==0:
                lives_txt.remove()
                lives_txt = fig.text(0.815, 0.815, trans[late]['game over'])#, transform=ax.transAxes)
                score_txt.remove()
                score_txt = fig.text(0.82, 0.765, trans[late]['score = %d']%(score),bbox={'facecolor':
        else:
                score_txt.remove()
                score_txt = fig.text(0.82, 0.765, trans[late]['score = %d']%(score)*(int(lives/abs(liv


        binfo.label.set_text(trans[late][detrans[lated][binfo.label.get_text()]])
        linfo.set_text(trans[late][detrans[lated][linfo.get_text()]])

        ldiff.set_text(trans[late]['Change the amount of\npotential allowed'])

        lmore.set_text(trans[late]['Add a column'])
        lless.set_text(trans[late]['Remove a column'])

        bnew.label.set_text(trans[late]['New'])
        lnew.set_text(trans[late]['Go back to the\ninitial potential'])

        lmode.set_text(trans[late][detrans[lated][lmode.get_text()]])

        lup.set_text(trans[late]['   Go up 1\nenergy level'])
        ldown.set_text(trans[late][' Go down 1\nenergy level'])

        demos_txt.remove()
        demos_txt = fig.text(0.825, 0.385, trans[late]['demos'])#, transform=ax.transAxes)

        lgauss.set_text(trans[late]['  Draw our old friend\nthe Harmonic Oscillator'])

        bstep.label.set_text(trans[late]['Step'])
        lstep.set_text(trans[late]['Draw a high step'])

        bwall.label.set_text(trans[late]['Wall'])
        lwall.set_text(trans[late][' Draw a wall\non the middle'])

#axbrick = plt.axes([0.820, 0.62, 0.05, 0.0375])
#bbrick = Button(axbrick, '', color='0.65', hovercolor='0.5')

axCAT = plt.axes([0.78, 0.94, 0.05, 0.0375])
bCAT = Button(axCAT, 'cat', color='0.65', hovercolor='0.5')
bCAT.on_clicked(CAT)

axENG = plt.axes([0.85, 0.94, 0.05, 0.0375])
bENG = Button(axENG, 'eng', color='0.65', hovercolor='0.5')
bENG.on_clicked(ENG)

axESP = plt.axes([0.92, 0.94, 0.05, 0.0375])
bESP = Button(axESP, 'esp', color='0.65', hovercolor='0.5')
bESP.on_clicked(ESP)

trans=[{'turtle':'turtle',     'game over':'game over',       'score = %d':'score = %d',  'I
```

```
                              {'turtle':'tortuga',    'game over':'has perdut',      'score = %d':'punts = %d',  'I
                            {'turtle':'sapoconcha',  'game over':'fin de partida',   'score = %d':'puntos  %d',  'I

detrans=[{v: k for k, v in language.items()} for language in trans]

late=0
#print(trans[late]['turtle'])
```

```
#----------------------------------------------------------------------------

# info button
# labels everything with an explanation

def info(event):

        global trans, late, binfo, axinfo, linfo,\
        ball_value

        linfo.set_visible(False)

        ball_value+=1

        if binfo.label.get_text()==trans[late]['Instructions: on']:

                binfo.label.set_text(trans[late]['Instructions: off'])
                axinfo.spines['top'].set_color((0,0,0.5))
                axinfo.spines['bottom'].set_color((0,0,0.5))
                axinfo.spines['right'].set_color((0,0,0.5))
                axinfo.spines['left'].set_color((0,0,0.5))

                linfo.set_text(trans[late]['Click to see\ninstructions\non balloons'])
                linfo.set_visible(False)


        elif binfo.label.get_text()==trans[late]['Instructions: off']:

                binfo.label.set_text(trans[late]['Instructions: on'])
                axinfo.spines['top'].set_color((0,0,1))
                axinfo.spines['bottom'].set_color((0,0,1))
                axinfo.spines['right'].set_color((0,0,1))
                axinfo.spines['left'].set_color((0,0,1))

                linfo.set_text(trans[late]['Click to stop\nseeing balloons'])
                linfo.set_visible(False)

        draw_wave()
        event.canvas.draw()

axinfo = plt.axes([0.78, 0.87, 0.19, 0.05])
binfo = Button(axinfo, trans[late]['Instructions: on'])
binfo.on_clicked(info) axinfo.spines['top'].set_color((0,0,1))
axinfo.spines['bottom'].set_color((0,0,1))
axinfo.spines['right'].set_color((0,0,1))
axinfo.spines['left'].set_color((0,0,1))

el = Ellipse((2, -1), 0.5, 0.5)
axinfo.add_patch(el)

linfo = axinfo.annotate(trans[late]['Click to stop\nseeing balloons'], xy=(0,0.5),
                        xytext=(-95,-10), textcoords='offset points',
                        size=10, va='center',
                        bbox=dict(boxstyle='round', fc=(0.7, 0.7, 1), ec='none'),
                        arrowprops=dict(arrowstyle='wedge,tail_width=1.',
fc=(0.7, 0.7, 1), ec='none',
patchA=None,
patchB=el, relpos=(0.2,
0.5)))
```

```
linfo.set_visible(False)

#------------------------------------------------------------------------


# [ + ] button
# adds an piece up to max_pieces

def more(event):
        global level, data, data_new, ax, Vk, pieces, txt, bricks, xk, kk, phi, max_pieces


        if pieces<max_pieces:

                pieces+=1

                data_new = np.zeros((pieces,10),dtype=int) Vk =
                np.zeros(pieces,dtype=int)
                data = np.copy(data_new)

                xk=np.zeros(pieces)
                for k in range(pieces):
                        xk[k]=(k+1)/pieces

                kk=1j*np.zeros(pieces)
                phi=1j*np.zeros(shape=(pieces,2))
                phi[0][0]=1
                phi[0][1]=-1

                level=1 draw_wave()
                event.canvas.draw()

                return True else:

        return True

axmore = plt.axes([0.825, 0.75, 0.05, 0.075]) bmore =
Button(axmore, '$c+$') bmore.on_clicked(more)
axmore.set_visible(False)

lmore = axmore.annotate(trans[late]['Add a column'], xy=(0,0.5),
                                                xytext=(-90,0), textcoords='offset
                                                points', size=10, va='center',
                                                bbox=dict(boxstyle='round', fc=(0.7, 0.7, 1), ec='none'),
                                                arrowprops=dict(arrowstyle='wedge,tail_width=1.',
                                                                fc=(0.7, 0.7, 1), ec='none',
                                                                patchA=None,
                                                                patchB=el, relpos=(0.2,
                                                                0.5)))
lmore.set_visible(False)


#------------------------------------------------------------------------

# [ - ] button
# substract a piece down to 1

def less(event):
```

```
global data, data_new, ax, Vk, pieces, txt, bricks, xk, kk, phi
```

```
        if pieces>1:

            pieces+=-1

            data_new = np.zeros((pieces,10),dtype=int) Vk =
            np.zeros(pieces,dtype=int)
            data = np.copy(data_new)

            xk=np.zeros(pieces)
            for k in range(pieces):
                xk[k]=(k+1)/pieces

            kk=1j*np.zeros(pieces)
            phi=1j*np.zeros(shape=(pieces,2))
            phi[0][0]=1
            phi[0][1]=-1

            draw_wave()

            event.canvas.draw()

            return True

        else: return True

    axless = plt.axes([0.875, 0.75, 0.05, 0.075]) bless =
    Button(axless, '$c-$') bless.on_clicked(less)
    axless.set_visible(False)

    lless = axless.annotate(trans[late]['Remove a column'], xy=(0,0.5),
                            xytext=(-100,0), textcoords='offset
                            points', size=10, va='center',
                            bbox=dict(boxstyle='round', fc=(0.7, 0.7, 1), ec='none'),
                            arrowprops=dict(arrowstyle='wedge,tail_width=1.',
                                            fc=(0.7, 0.7, 1), ec='none',
                                            patchA=None,
                                            patchB=el, relpos=(0.2,
                                            0.5)))
    lless.set_visible(False)

    #---------------------------------------------------------------------------

    # total lives and score

    lives_txt = fig.text(0.935, 0.875, '<3 '*lives)#, transform=ax.transAxes)
    score_txt = fig.text(0.885, 0.875, 'x %d'%(-max_value))#, transform=ax.transAxes)

    #---------------------------------------------------------------------------

    # new botton
    # draws the squared infinite potential (nothing)

    def new(event):

        global trans, late, level, data, data_new, score, cid_click, fig, score, lives, \
ball_value, bmode, x_ball, y_ball, demos_txt data =

np.copy(data_new)

cid_click = fig.canvas.mpl_connect('button_press_event', ClickColor)

axup.set_visible(True)
```

```
        bup.set_active(True)
        axdown.set_visible(True)
        bdown.set_active(True)
        axgauss.set_visible(True)
        bgauss.set_active(True)
        axstep.set_visible(True)
        bstep.set_active(True)
        axwall.set_visible(True)
        bwall.set_active(True)
        demos_txt.remove()
        demos_txt = fig.text(0.825, 0.385, trans[late]['demos'])#, transform=ax.transAxes)
        if bmode.label.get_text()=='SURVIVAL':
                score=0
                lives=3
                x_ball=0
                y_ball=9

        level=1 draw_wave()
        event.canvas.draw()

axnew = plt.axes([0.825, 0.65, 0.1, 0.075]) bnew =
Button(axnew, trans[late]['New'])
bnew.on_clicked(new)

lnew = axnew.annotate(trans[late]['Go back to the\ninitial potential'], xy=(0,0.5),
                                                xytext=(-95,0),
                                                textcoords='offset points',
                                                size=10, va='center',
                                                bbox=dict(boxstyle='round', fc=(0.7, 0.7, 1), ec='none'),
                                                arrowprops=dict(arrowstyle='wedge,tail_width=1.',
                                                                fc=(0.7, 0.7, 1), ec='none',
                                                                patchA=None,
                                                                patchB=el, relpos=(0.2,
                                                                0.5)))
lnew.set_visible(False)

#------------------------------------------------------------------------

# mode botton
# changes from zen mode to survival mode and vs

def mode(event):

        global trans, late, level, data, data_new, score, \
        cid_click, fig, score, lives, ball_value, \
        bmode, axmode, lmode, axmore, axless, pieces, demos_txt

        lmode.set_visible(False)

        data = np.copy(data_new)

        if bmode.label.get_text()=='SURVIVAL':
                bmode.label.set_text('ZEN')
                axmode.spines['top'].set_color((0,1,0))
                axmode.spines['bottom'].set_color((0,1,0))
                axmode.spines['right'].set_color((0,1,0))
                axmode.spines['left'].set_color((0,1,0))
#       print(bmode.label.get_text())
                cid_click = fig.canvas.mpl_connect('button_press_event', ClickColor)
                lives=-1

                axmore.set_visible(True)
                bmore.set_active(True)
                axless.set_visible(True)
                bless.set_active(True)
                axup.set_visible(True)
                bup.set_active(True)
                axdown.set_visible(True)
                bdown.set_active(True)
                axgauss.set_visible(True)
                bgauss.set_active(True)
                axstep.set_visible(True)
                bstep.set_active(True)
                axwall.set_visible(True)
                bwall.set_active(True)


                demos_txt.remove()
                demos_txt = fig.text(0.825, 0.385, trans[late]['demos'])#, transform=ax.transAxes)


                lmode.set_text(trans[late]['     Change to\nSURVIVAL mode'])
                lmode.set_visible(False)

            elif bmode.label.get_text()=='ZEN':
                if pieces!=7:
                    pieces=6
                    more(event)
                bmode.label.set_text('SURVIVAL')
                axmode.spines['top'].set_color((1,0,0))
                axmode.spines['bottom'].set_color((1,0,0))
                axmode.spines['right'].set_color((1,0,0))
                axmode.spines['left'].set_color((1,0,0))
#       print(bmode.label.get_text())
                cid_click = fig.canvas.mpl_connect('button_press_event', ClickColor)
                lives=3 score=0
                axmore.set_visible(False)
                bmore.set_active(False)
                axless.set_visible(False)
                bless.set_active(False)

                lmode.set_text(trans[late]['Change to\nZEN mode'])
                lmode.set_visible(False)

            level=1 draw_wave()
            event.canvas.draw()

axmode = plt.axes([0.815, 0.55, 0.12, 0.075]) bmode =
Button(axmode, 'SURVIVAL')
bmode.on_clicked(mode)
axmode.spines['top'].set_color((1,0,0))
axmode.spines['bottom'].set_color((1,0,0))
axmode.spines['right'].set_color((1,0,0))
axmode.spines['left'].set_color((1,0,0))

lmode = axmode.annotate(trans[late]['Change to\nZEN mode'],
                        xy=(0,0.5), xytext=(-95,0),
                        textcoords='offset points',
                        size=10, va='center',
                        bbox=dict(boxstyle='round', fc=(0.7, 0.7, 1), ec='none'),
                        arrowprops=dict(arrowstyle='wedge,tail_width=1.',
                                                fc=(0.7, 0.7, 1), ec='none',
                                                patchA=None,
                                                patchB=el, relpos=(0.2,
                                                0.5)))
```

```
lmode.set_visible(False)

#-------------------------------------------------------------------------

# [ ↑] button
# changes from the first energy level to the others, up to N_root

def up(event):
        global level, N_root, eigen_energies_list


        if level==1:
                eigen_energies_list=eigen_energies(N_root)
#         print(eigen_energies_list)

        if level>=1 and level<N_root:

                level+=1
#                 print(level)
                draw_wave()
                event.canvas.draw()

                return True else:

        return True

axup = plt.axes([0.825, 0.45, 0.05, 0.075]) bup =
Button(axup, '$E+$') bup.on_clicked(up)

lup = axup.annotate(trans[late]['   Go up 1\nenergy level'], xy=(0,0.5),
                                                xytext=(-80,0), textcoords='offset points',
                                                size=10, va='center',
                                                bbox=dict(boxstyle='round', fc=(0.7, 0.7, 1), ec='none'),
                                                arrowprops=dict(arrowstyle='wedge,tail_width=1.',
                                                                                fc=(0.7, 0.7, 1), ec='none',
                                                                                patchA=None,
                                                                                patchB=el, relpos=(0.2,
                                                                                0.5)))
lup.set_visible(False)

#-------------------------------------------------------------------------

# [ ↓] button
# changes from any level down to the first one

def down(event):
        global level, N_root

        if level>1 and level<=N_root:
                level-=1
#                 print(level)
                draw_wave()
                event.canvas.draw()

                return True else:

        return True
```

```
axdown = plt.axes([0.875, 0.45, 0.05, 0.075]) bdown =
Button(axdown, '$E-$') bdown.on_clicked(down)
```

```
ldown = axdown.annotate(trans[late][' Go down 1\nenergy level'], xy=(0,0.5),
                        xytext=(-80,0), textcoords='offset points',
                        size=10, va='center',
                        bbox=dict(boxstyle='round', fc=(0.7, 0.7, 1), ec='none'),
                        arrowprops=dict(arrowstyle='wedge,tail_width=1.',
                                        fc=(0.7, 0.7, 1), ec='none',
                                        patchA=None,
                                        patchB=el, relpos=(0.2,
                                        0.5)))
ldown.set_visible(False)

#---------------------------------------------------------------------------

# demo buttons
# demos are drawn by hand for pieces=1..7 only

demos_txt = fig.text(0.825, 0.385, trans[late]['demos'])#, transform=ax.transAxes)

def Vk_to_data(Vk):

    global data_new

    translated_data = np.copy(data_new)

    for X in range(len(Vk)): Y=Vk[X]-1
        translated_data[X][:Y+1]=2
        translated_data[X][Y+1:]=0

    return translated_data


def gauss(event):
    global level, data, pieces


    Vk=[[0], [0,0],[5,1,5], [5,1,1,5],[8,2,0,2,8],
        [6,1,0,0,1,6],[9,4,1,0,1,4,9]]

    if pieces<=7:
        data = Vk_to_data(Vk[pieces-1])
        level=1 draw_wave()
        event.canvas.draw()
    else:
        return True

axgauss = plt.axes([0.825, 0.285, 0.1, 0.075]) bgauss =
Button(axgauss, 'Gauss') bgauss.on_clicked(gauss)

lgauss = axgauss.annotate(trans[late]['  Draw our old friend\nthe Harmonic Oscillator'], xy=(0,0.5), xytext=(-
                        130,0),
                        textcoords='offset points',
                        size=10, va='center',
                        bbox=dict(boxstyle='round', fc=(0.7, 0.7, 1), ec='none'),
                        arrowprops=dict(arrowstyle='wedge,tail_width=1.',
                                        fc=(0.7, 0.7, 1), ec='none',
                                        patchA=None,
patchB=el,
                                        relpos=(0.2, 0.5)))
lgauss.set_visible(False)


def step(event):
        global level, data, pieces

        Vk=[[0], [0,9],[0,9,9], [0,0,9,9],[0,0,9,9,9],
                [0,0,0,9,9,9],[0,0,0,9,9,9,9]]

        if pieces<=7:
                data = Vk_to_data(Vk[pieces-1])
                level=1 draw_wave()
                event.canvas.draw()
        else:
                return True

axstep = plt.axes([0.825, 0.185, 0.1, 0.075]) bstep =
Button(axstep, trans[late]['Step']) bstep.on_clicked(step)

lstep = axstep.annotate(trans[late]['Draw a high step'], xy=(0,0.5),
                        xytext=(-100,0), textcoords='offset
                        points', size=10, va='center',
                        bbox=dict(boxstyle='round', fc=(0.7, 0.7, 1), ec='none'),
                        arrowprops=dict(arrowstyle='wedge,tail_width=1.',
                                        fc=(0.7, 0.7, 1), ec='none',
                                        patchA=None,
                                        patchB=el, relpos=(0.2,
                                        0.5)))
lstep.set_visible(False)


def wall(event):
        global data, pieces, level

        Vk=[[0],[0,0], [0,10,0],
                [0,10,10,0],
                [0,0,10,0,0],
                [0,0,10,10,0,0],
                [0,0,0,10,0,0,0]]

        if pieces<=7:
                data = Vk_to_data(Vk[pieces-1])
                level=1 draw_wave()
                event.canvas.draw()
        else:
                return True

axwall = plt.axes([0.825, 0.085, 0.1, 0.075]) bwall =
Button(axwall, trans[late]['Wall'])
bwall.on_clicked(wall)

lwall = axwall.annotate(trans[late][' Draw a wall\non the middle'], xy=(0,0.5),
                        xytext=(-80,0), textcoords='offset points',
                        size=10, va='center',
                        bbox=dict(boxstyle='round', fc=(0.7, 0.7, 1), ec='none'),
                                    arrowprops=dict(arrowstyle='wedge,tail_width=1.', fc=(0.7,
                                                    0.7, 1), ec='none', patchA=None,
                                                    patchB=el, relpos=(0.2,
                                                    0.5)))
lwall.set_visible(False)

#---------------------------------------------------------------------------

# difficulty button

# remaining bricks
# Counts how many bricks you have.
# at 6*pieces people start to get creative.

def bricks(data):
    global pieces,difficulty,bmode

    if bmode.label.get_text()=='SURVIVAL':
        bricks0 = difficulty*pieces else:
        bricks0 = 10*pieces

    return bricks0-np.sum(data_to_Vk(data))

#axbrick = plt.axes([0.820, 0.62, 0.05, 0.0375])
#bbrick = Button(axbrick, '', color='0.65', hovercolor='0.5')
#
#txt = fig.text(0.885, 0.625, 'x %d'%(int(bricks(data))), transform=ax.transAxes)

def diff(event):

    global trans, late, new, \
    cid_click, fig, \
    bdiff, axdiff, ldiff, difficulty, score_color, ball_color

    ldiff.set_visible(False)

    if bdiff.label.get_text()=='EASY':
        difficulty=5
        bdiff.label.set_text('FAIR')
        score_color=(1,1,0)
        ball_color=(1,1,0)
        axdiff.spines['top'].set_color((1,1,0))
        axdiff.spines['bottom'].set_color((1,1,0))
        axdiff.spines['right'].set_color((1,1,0))
        axdiff.spines['left'].set_color((1,1,0))
        cid_click = fig.canvas.mpl_connect('button_press_event', ClickColor)


    elif bdiff.label.get_text()=='FAIR' or bdiff.label.get_text()=='FAIR.':
        difficulty=4
        bdiff.label.set_text('HARD')
        score_color=(1,0,0)
        ball_color=(1,0,0)
        axdiff.spines['top'].set_color((1,0,0))
        axdiff.spines['bottom'].set_color((1,0,0))
        axdiff.spines['right'].set_color((1,0,0))
        axdiff.spines['left'].set_color((1,0,0))
        cid_click = fig.canvas.mpl_connect('button_press_event', ClickColor)


    elif bdiff.label.get_text()=='HARD':
        difficulty=10
```

```
                score_color=(0,1,0) ball_color=(0,1,0)
                bdiff.label.set_text('EASY')
                axdiff.spines['top'].set_color((0,1,0))
                axdiff.spines['bottom'].set_color((0,1,0))
                axdiff.spines['right'].set_color((0,1,0))
                axdiff.spines['left'].set_color((0,1,0))
                cid_click = fig.canvas.mpl_connect('button_press_event', ClickColor)

        draw_wave()
        event.canvas.draw()
        new(event)

axdiff = plt.axes([0.02, 0.92, 0.08, 0.05]) bdiff =
Button(axdiff, 'EASY') bdiff.on_clicked(diff)
axdiff.spines['top'].set_color((0,1,0))
axdiff.spines['bottom'].set_color((0,1,0))
axdiff.spines['right'].set_color((0,1,0))
axdiff.spines['left'].set_color((0,1,0))

ldiff = axdiff.annotate(trans[late]['Change the amount of\npotential allowed'], xy=(1,0.5),
                        xytext=(+35,-20),
                        textcoords='offset points',
                        size=10, va='center',
                        bbox=dict(boxstyle='round', fc=(0.7, 0.7, 1), ec='none'),
                        arrowprops=dict(arrowstyle='wedge,tail_width=1.',
                                        fc=(0.7, 0.7, 1), ec='none',
                                        patchA=None,
                                        patchB=el, relpos=(0.2,
                                        0.5)))
ldiff.set_visible(False)

#-------------------------------------------------------------------------
# SCORE
# How survival mode works
# The green square is calles ball
#-------------------------------------------------------------------------

def play_ball(touched):
        global x_ball, y_ball, ball_value, max_value, score, lives,\ bmode,
                axup,axdown,axgauss,axstep,axwall,demos_txt,\ no_bricks,bdiff

             if bmode.label.get_text()=='SURVIVAL':
                if x_ball==0 and y_ball==9:
                        new_ball(touched)
                        ball_value+=1
                if touched[len(touched)-1-x_ball]>=y_ball:
#               print('%d + %d = %d'%(score,ball_value+1,score+ball_value+1))
                        score+=ball_value
                        new_ball(touched)
                else:
                        try:
                                ball_value

                        except NameError: #The game just started
                                new_ball(touched)
```

```
else:
        ball_value-=1 if
        no_bricks:
                print('stupid')
                ball_value+=1
```

```
                        no_bricks=False if
                    ball_value==0:
                         lives-=1
                        if lives==0: axup.set_visible(False)
                                bup.set_active(False)
                                axdown.set_visible(False)
                                bdown.set_active(False)
                                axgauss.set_visible(False)
                                bgauss.set_active(False)
                                axstep.set_visible(False)
                                bstep.set_active(False)
                                axwall.set_visible(False)
                                bwall.set_active(False)
                                demos_txt.remove()
                                demos_txt = fig.text(0.838, 0.425, '')#, transform=ax.transAxes)
#                   print('game over')
                        else:
#                   print('bum')
                                new_ball(touched)



        elif bmode.label.get_text()=='ZEN':
                x_ball=0
                y_ball=9
                ball_value=0



def new_ball(touched):
        global x_ball, y_ball, ball_value, bdiff

        x_ball  = random.randint(0,len(touched)-1)
        while touched[len(touched)-1-x_ball]>7:
                x_ball  = random.randint(0,len(touched)-1)
        y_ball  = random.randint(touched[len(touched)-1-x_ball]+1,9)
        ball_value = y_ball

        if x_ball==0 or x_ball==len(touched)-1:
                if y_ball==9:
                        new_ball(touched)

        if y_ball==1 or y_ball==0:
                new_ball(touched)

           if bdiff.label.get_text()=='FAIR': if
                x_ball==0 and y_ball==8:
                        new_ball(touched)
                if y_ball>8:
                        new_ball(touched)

          if bdiff.label.get_text()=='HARD': if
                x_ball==0 and y_ball==8:
                        new_ball(touched)
                if y_ball>7:
                        new_ball(touched)
```

```
value_txt = fig.text(0.885, 0.925, '%d'%(max_value))#, transform=ax.transAxes)

def ball_the_data():
	global x_ball,y_ball,data,pre_ball
	pre_ball=data[x_ball,y_ball]
```

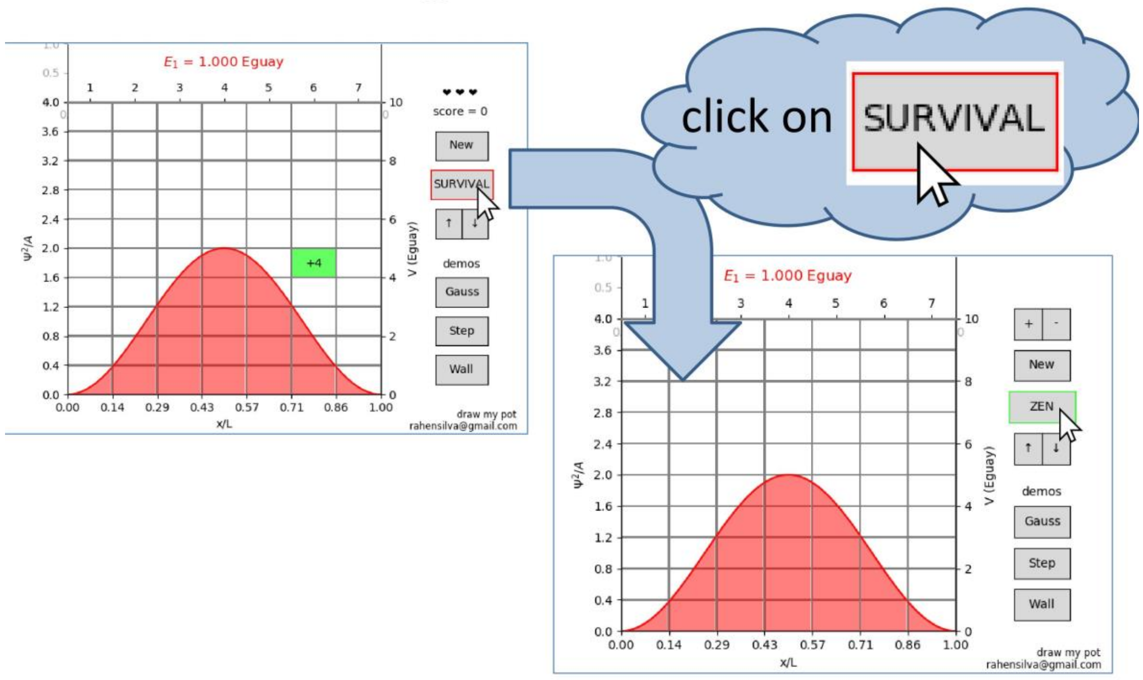
```
        if bmode.label.get_text()=='SURVIVAL':
                data[x_ball,y_b
        all]=3 return data

def unball_the_data():
        global
        x_ball,y_ball,data,pre_ball
        data[x_ball,y_ball]=pre_ball
        return data

#---------------------------------------------------------------------

# end
fig.text(0.865, 0.04, 'draw my pot', fontsize='smaller')
fig.text(0.775, 0.01, 'rahensilva@gmail.com', fontsize='smaller')

draw_wave()

plt.show()
```

Annex 4: Manual of the game

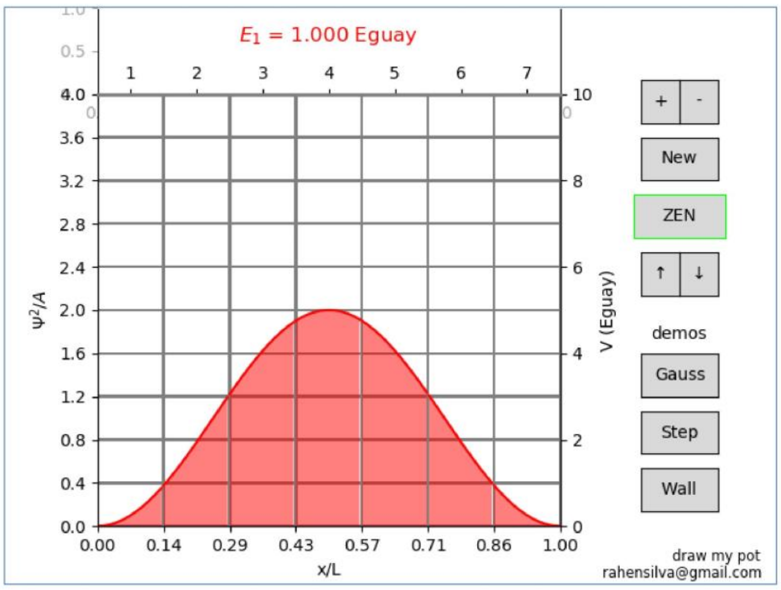
Draw My Pot

Interactive Piecewise Potential
-Rafa da Silva-

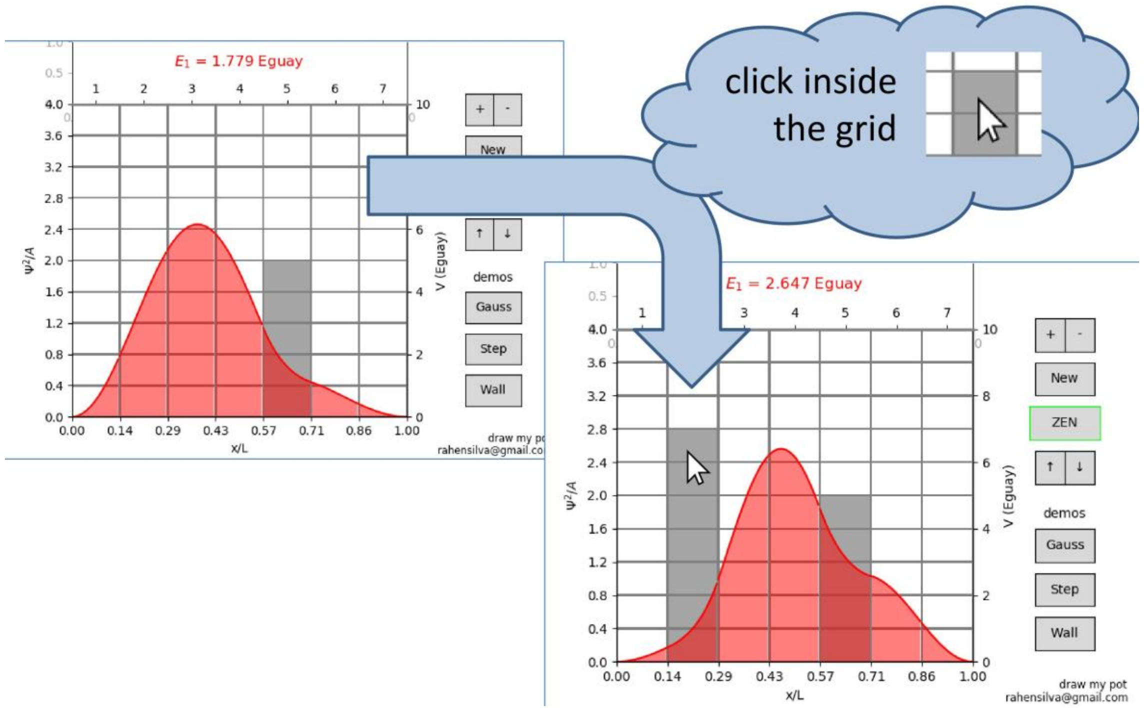
Change to ZEN mode



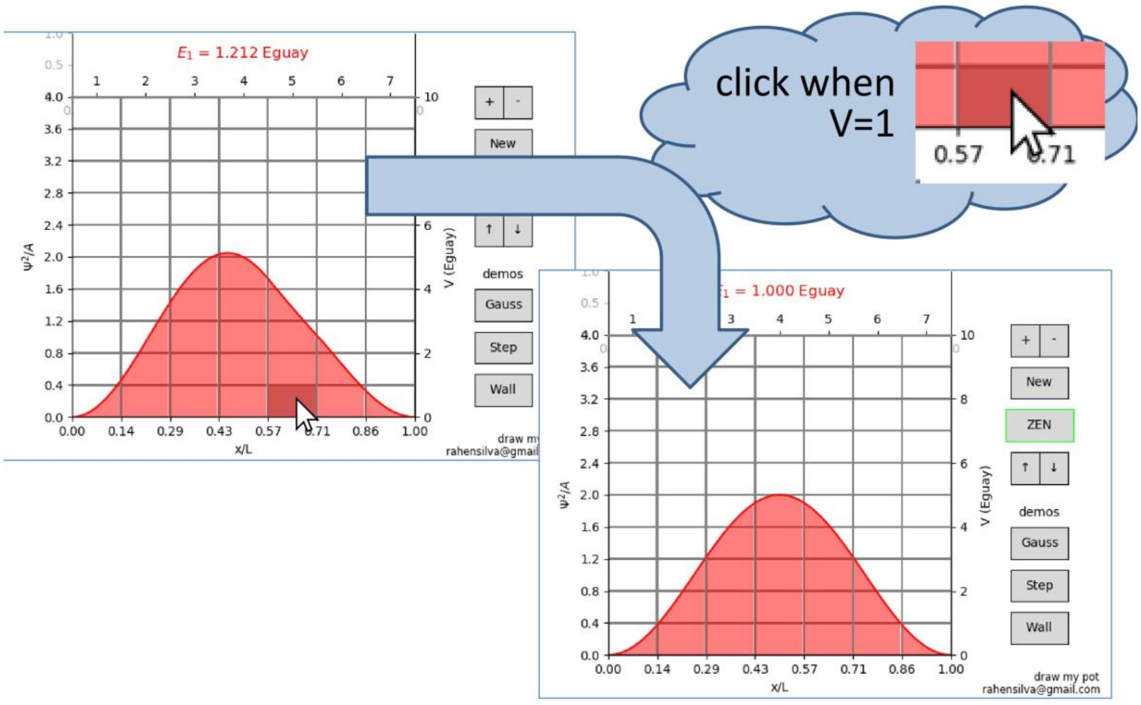
Eigenfunctions are instantly plotted



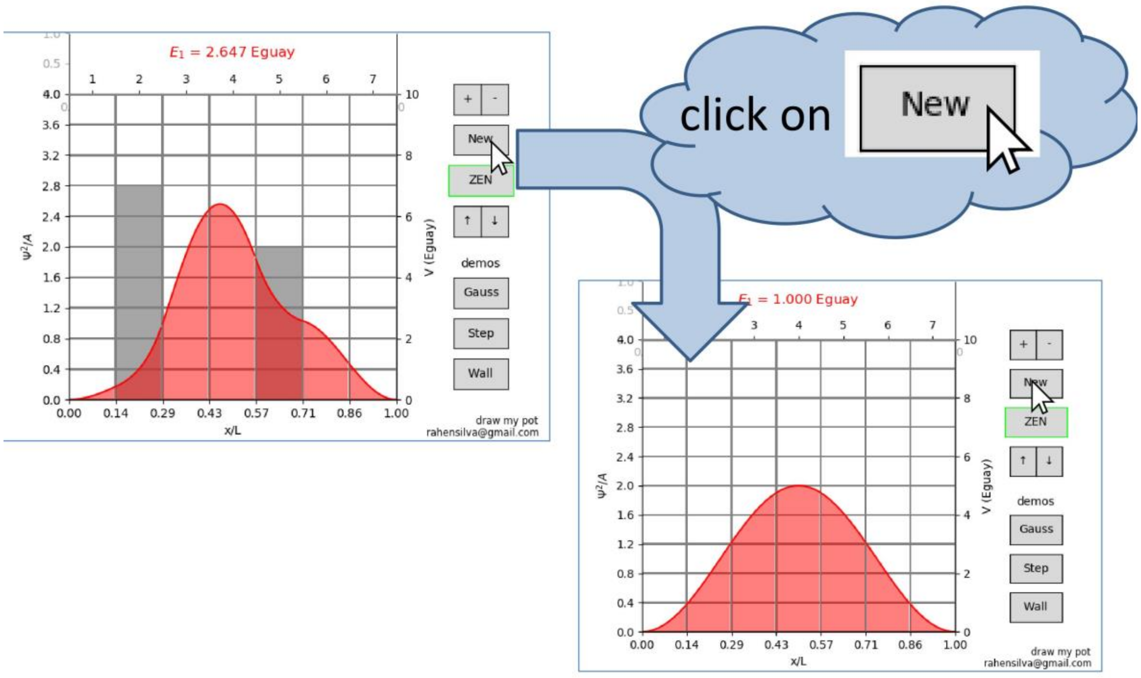
Draw the potential with the mouse



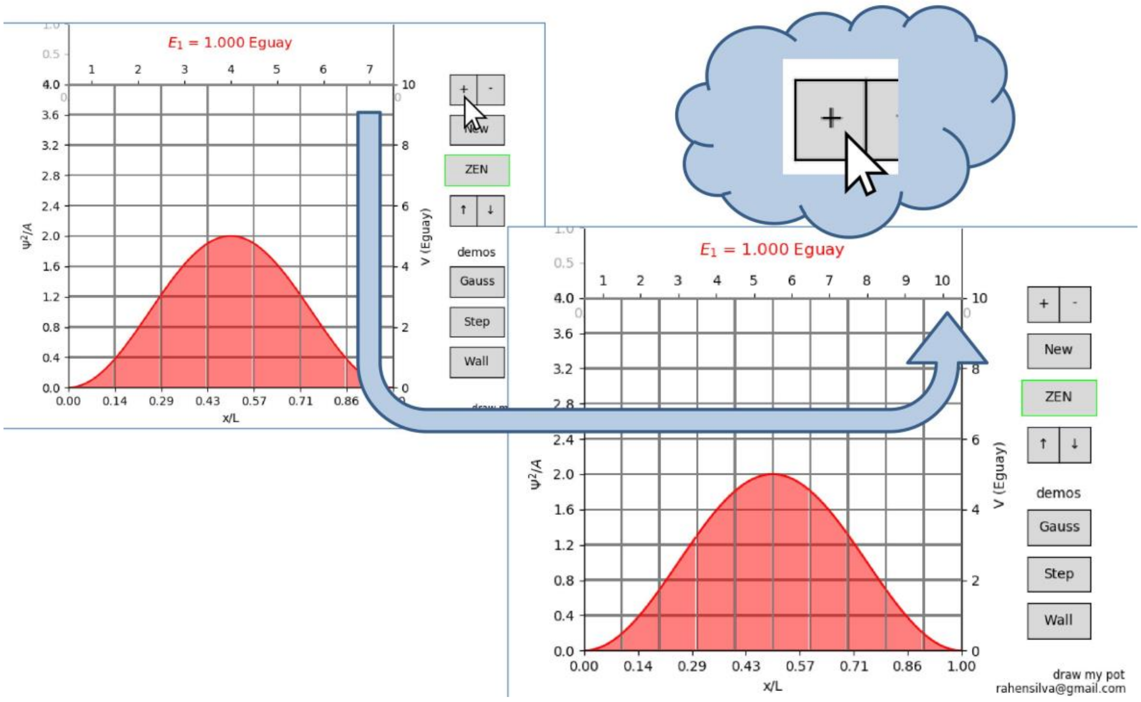
Undraw too



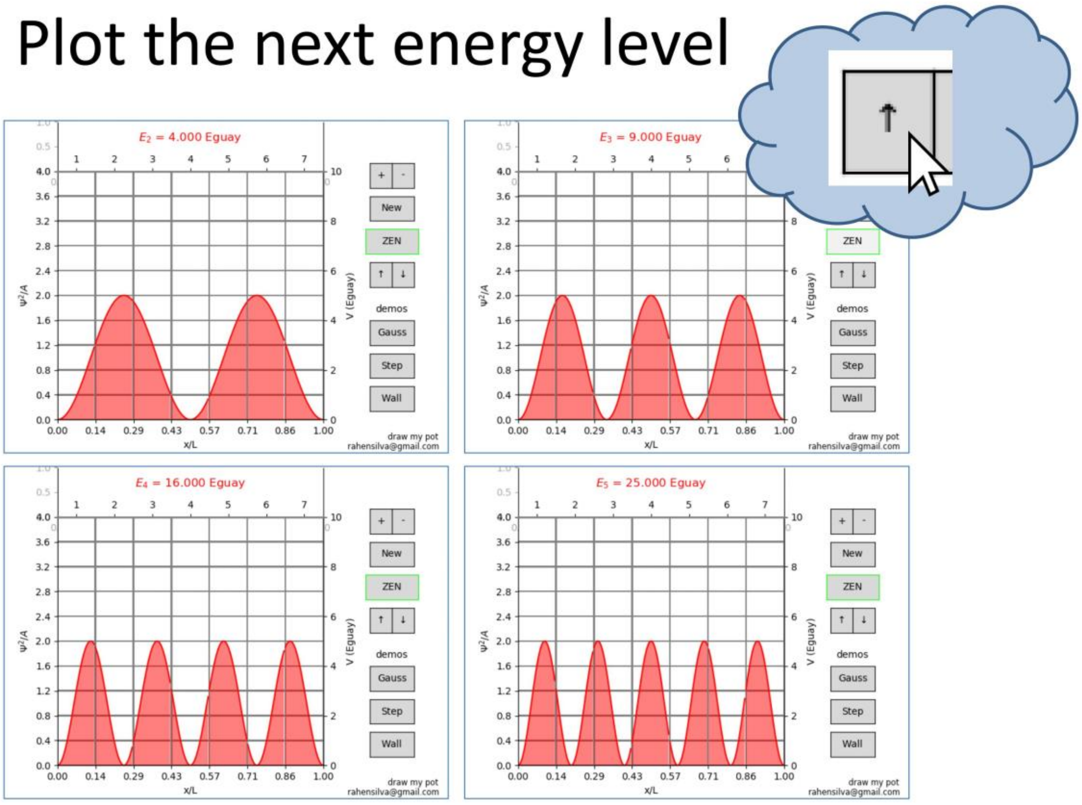
Return to the infinite square well



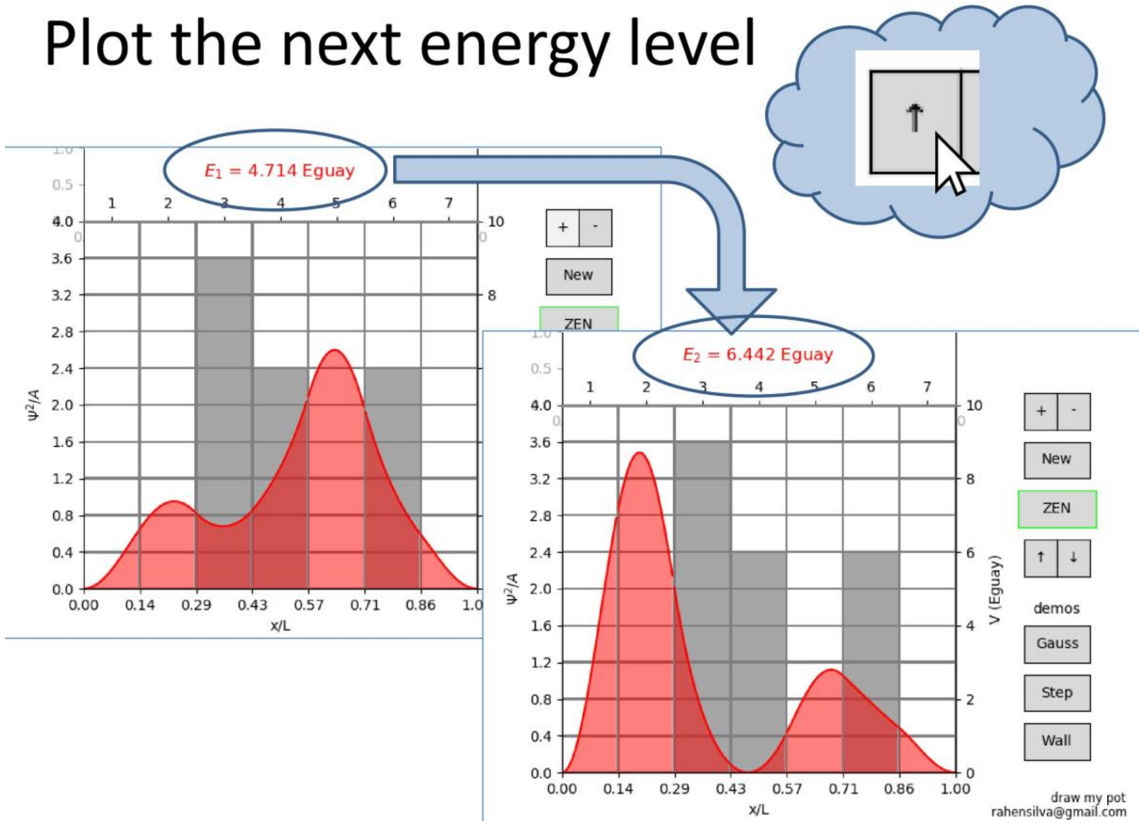
Change the number of pieces



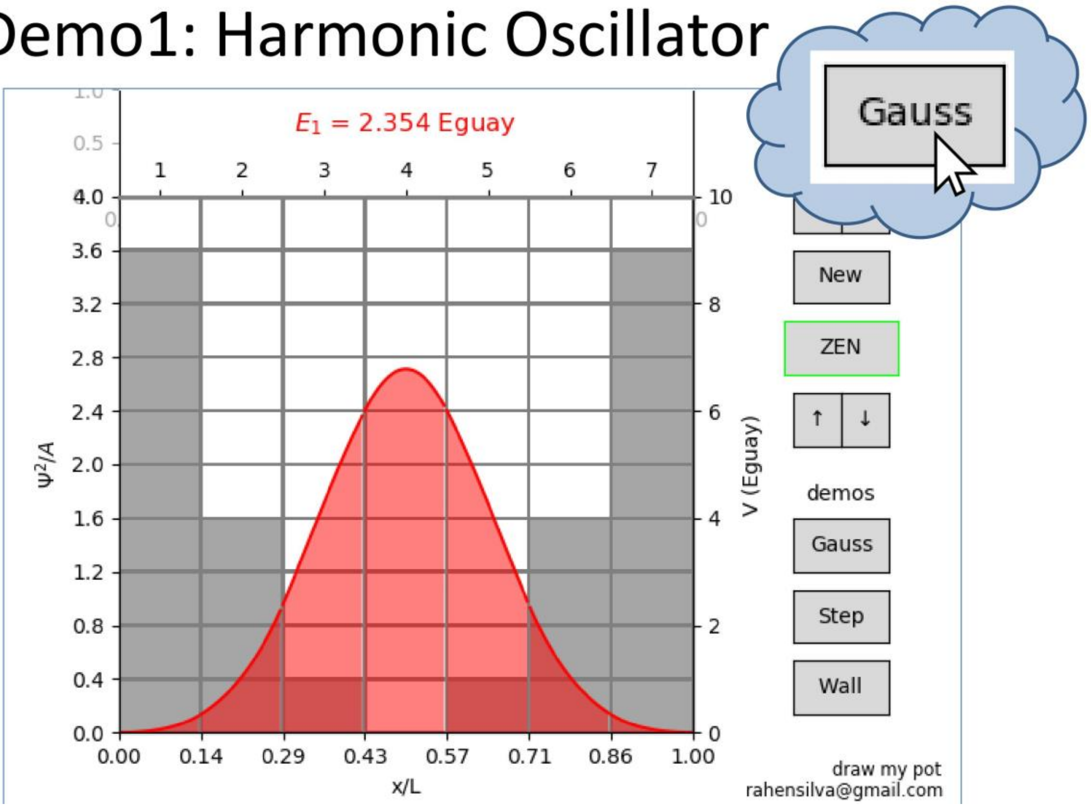
Plot the next energy level



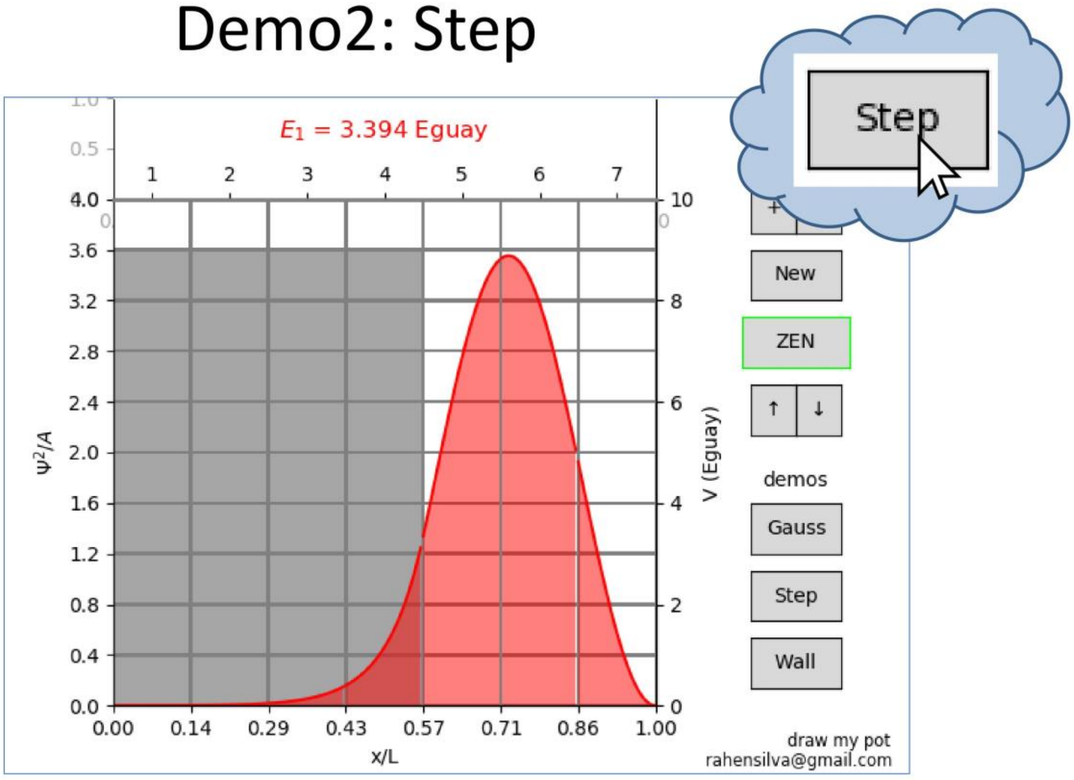
Plot the next energy level



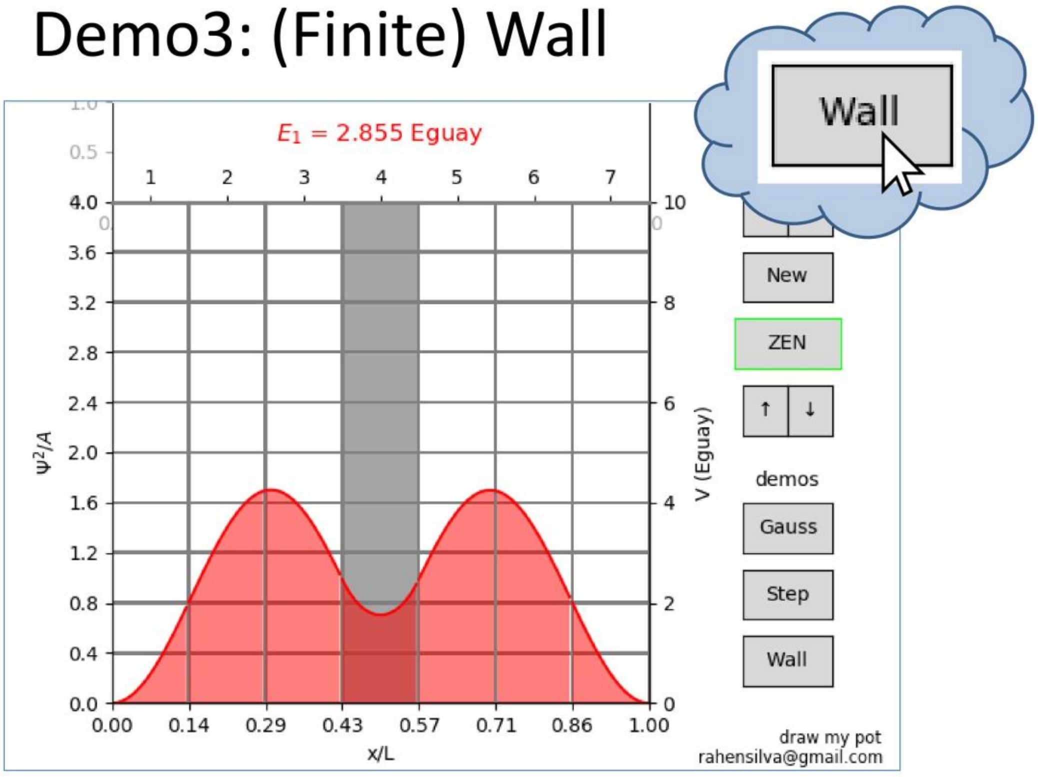
Demo1: Harmonic Oscillator



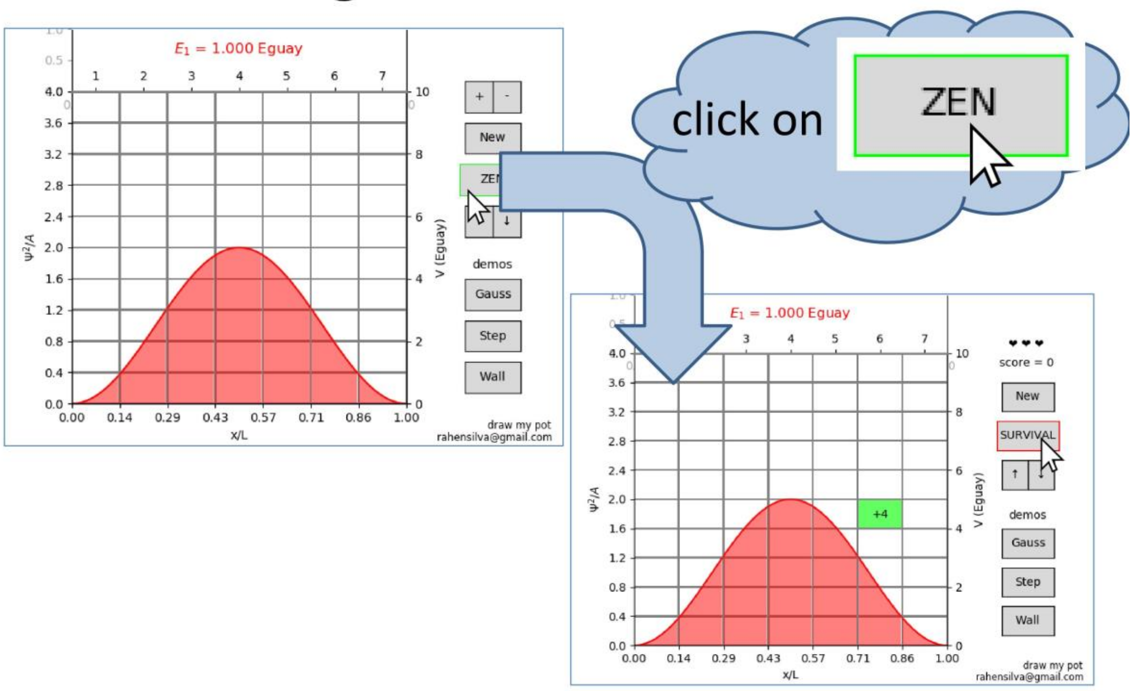
Demo2: Step



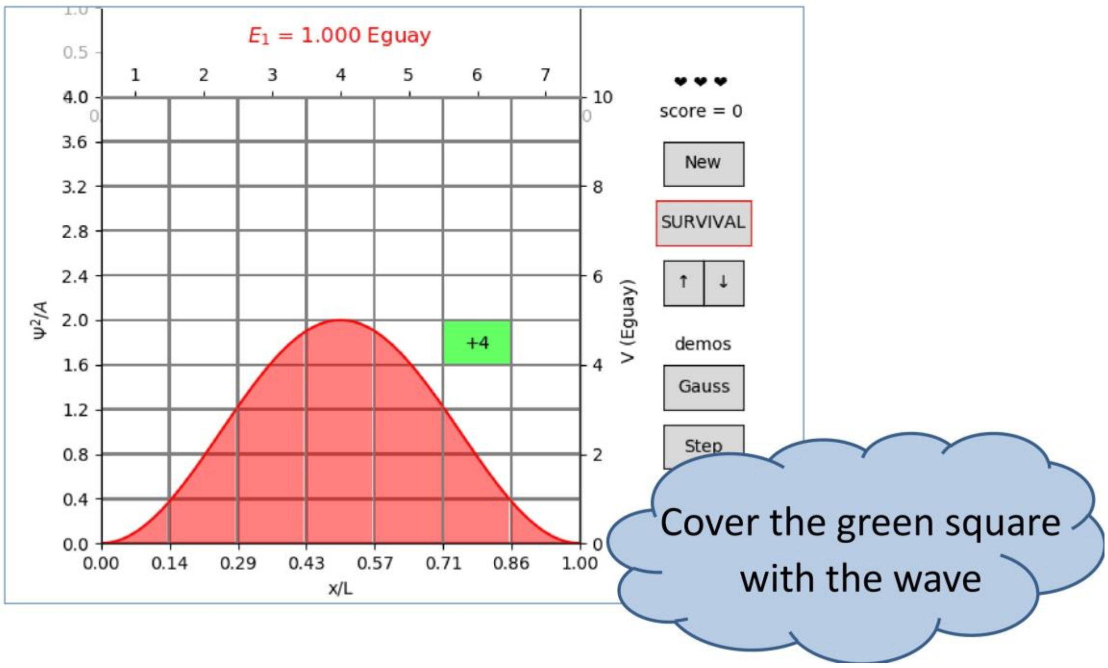
Demo3: (Finite) Wall



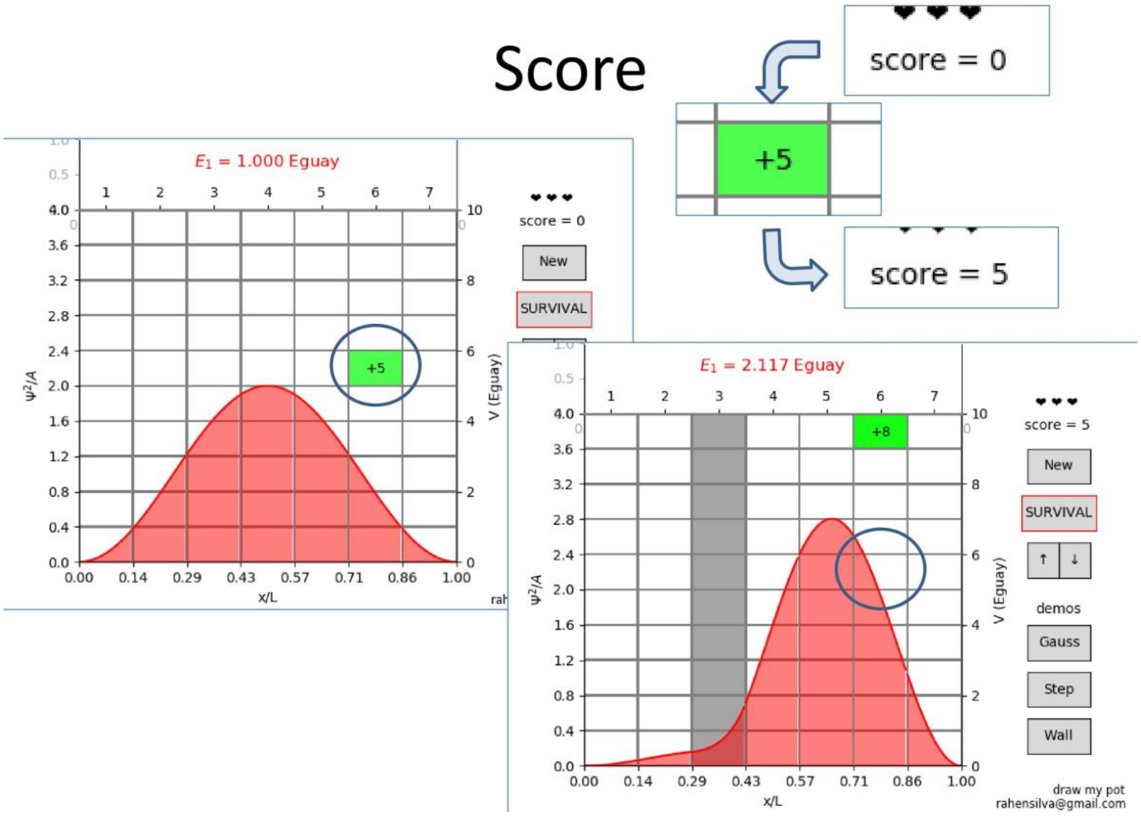
Change to SURVIVAL mode



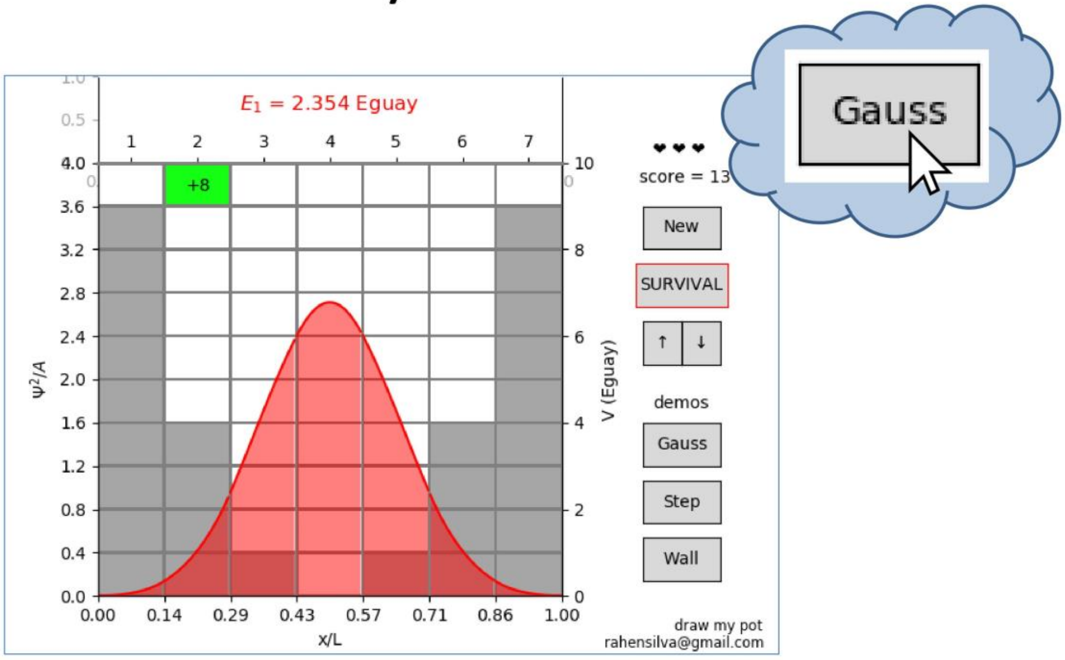
Get the ball



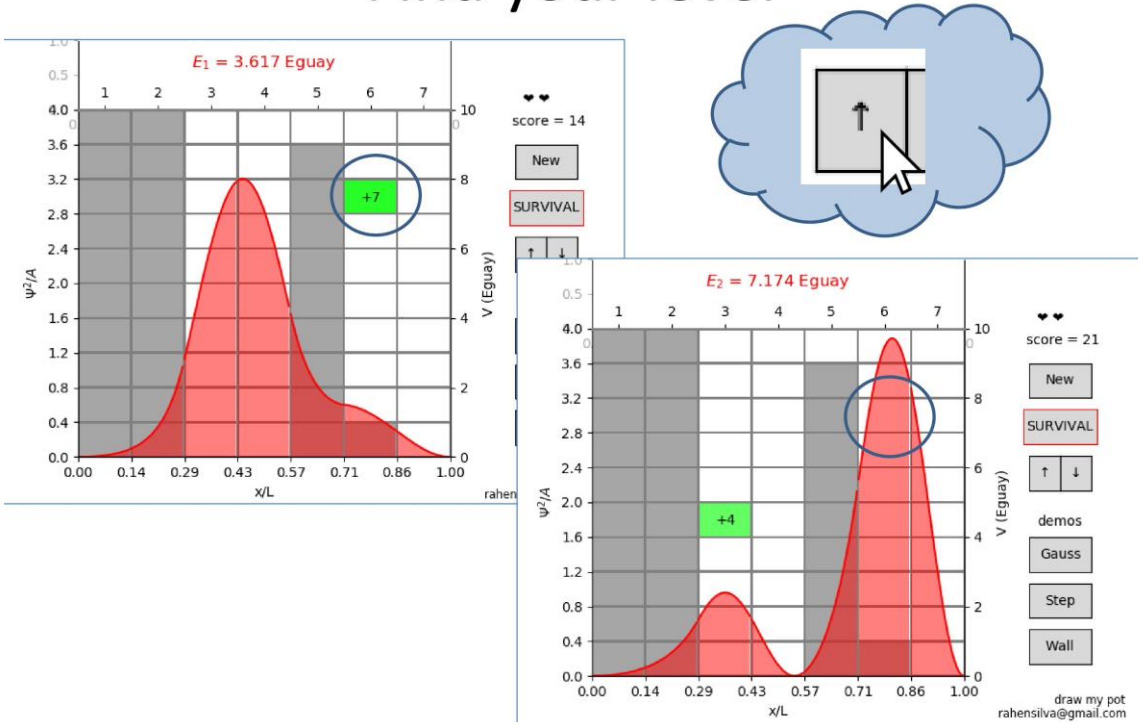
Score



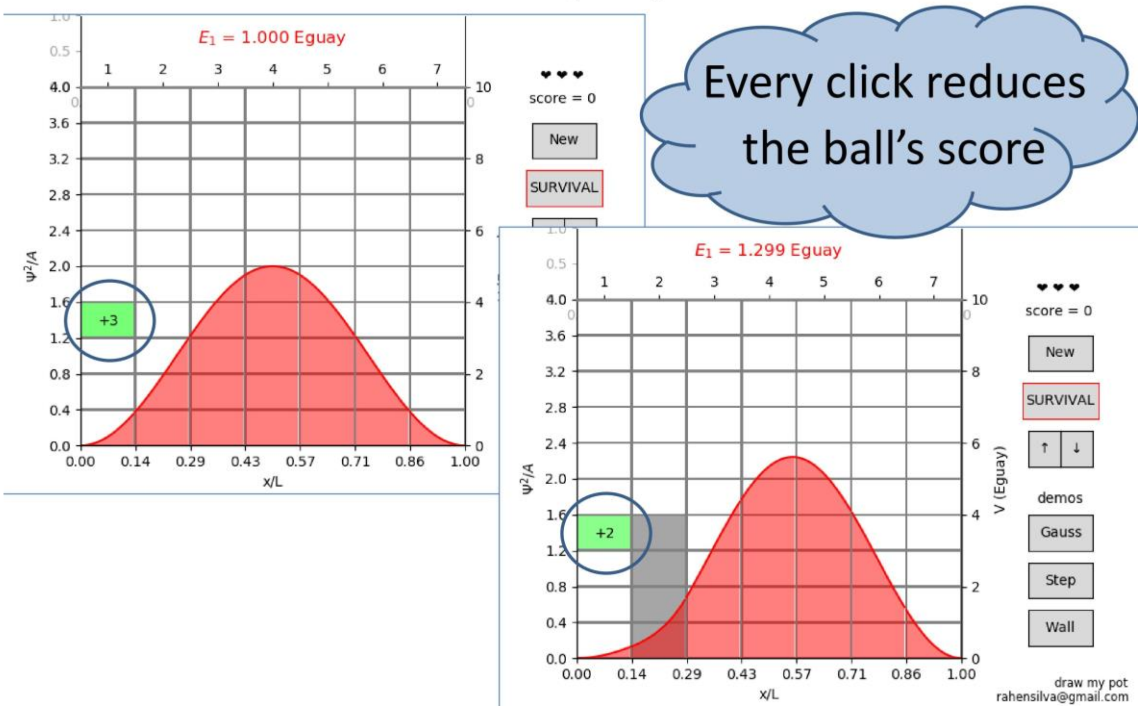
Use your habilities



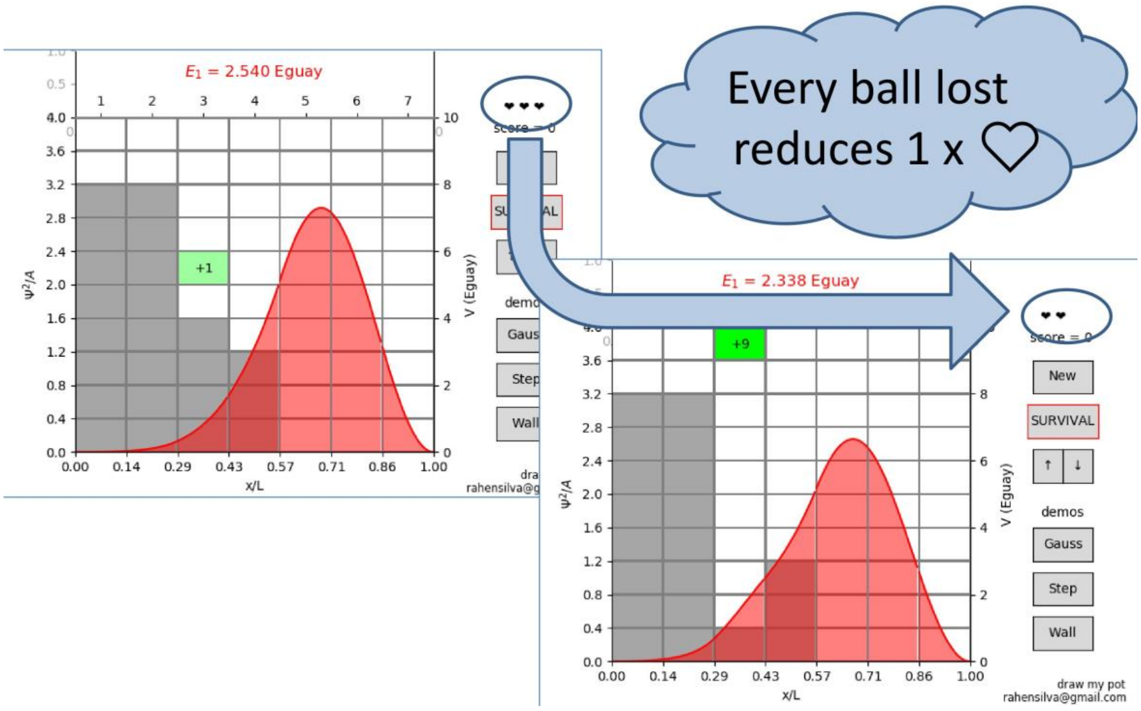
Find your level



Hurry up!

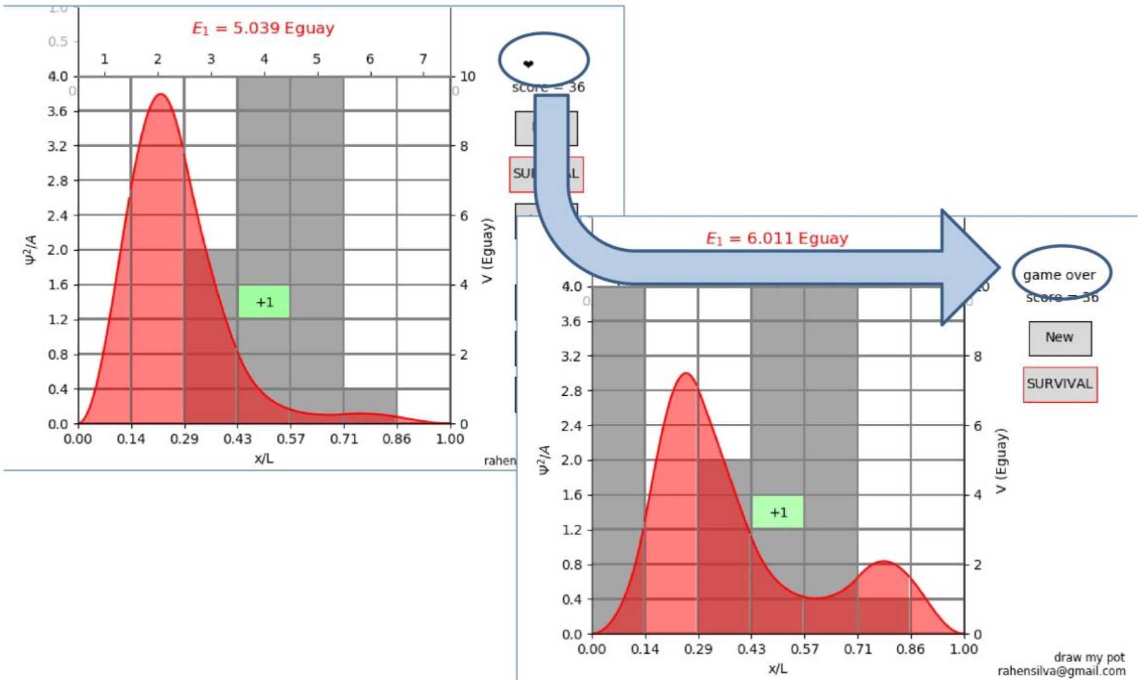


Caution!



Every ball lost reduces 1 x ♡

You have 3 x ♡



Annex 5: BJD resolution

Notes on the solution to 1D Schrdinger equation for piecewise potentials

bjd
Facultat de Física, Universitat de Barcelona, Diagonal 645, 08028 Barcelona, Spain.

Brief rendition of the way to solve the Schrdinger equation for piecewise potentials

We consider the time independent 1D Schrdinger equation, (in natural units)

$$-\frac{1}{2}\partial_x^2\psi(x) + V(x)\psi(x) = E\psi(x)\,. \tag{1}$$

Where the potential is taken to be piecewise constant, that is,

$$V(x) = V_k \; x_k < x < x_{k+1}\,. \tag{2}$$

where $\{x_k\}$ is a partition of the interval $[0, L]$, and $k = 0, \ldots, N$. For simplicity we take the utmost left and right potentials to be infinite, that is, the wave function is zero at $x = 0$ and $x = L$. For each interval, $[x_k, x_{k+1}]$ the general solution of the Schrdinger equation reads,

$$\psi_k(x) = A_k e^{i\kappa_k x} + B_k e^{-i\kappa_k x} \tag{3}$$

where $\kappa_k = \sqrt{2(E - V_k)}$.

First we consider the first and last intervals,

$$\begin{aligned}\psi_0(x) &= A_0 e^{i\kappa_0 x} + B_0 e^{-i\kappa_0 x}\\ \psi_N(x) &= A_N e^{i\kappa_N x} + B_N e^{-i\kappa_N x}\end{aligned} \tag{4}$$

the boundary conditions imply,

$$\begin{aligned}\psi_0(x) &= A_0\left(e^{i\kappa_0 x} - e^{-i\kappa_0 x}\right) = A_0\, 2\, i\, \sin(\kappa_0 x)\\ \psi_N(x) &= A_N e^{i\kappa_N L}\left(e^{i\kappa_N(x-L)} - e^{-i\kappa_N(x-L)}\right) = A_N\, 2\, i\, \sin(\kappa_N(x-L))\end{aligned} \tag{5}$$

If we consider two inner intervals we have to impose continuity of the wave function and of its first derivative, that reads,

$$\begin{aligned}\psi_k(x_{k+1}) &= \psi_{k+1}(x_{k+1})\\ \psi'_k(x_{k+1}) &= \psi'_{k+1}(x_{k+1})\end{aligned} \tag{6}$$

which read,

$$\begin{aligned}A_k e^{i\kappa_k x_{k+1}} + B_k e^{-i\kappa_k x_{k+1}} &= A_{k+1} e^{i\kappa_{k+1} x_{k+1}} + B_k e^{-i\kappa_{k+1} x_{k+1}}\\ A_k i\kappa_k e^{i\kappa_k x_{k+1}} - B_k i\kappa_k e^{-i\kappa_k x_{k+1}} &= A_{k+1} i\kappa_{k+1} e^{i\kappa_{k+1} x_{k+1}} - B_k i\kappa_{k+1} e^{-i\kappa_{k+1} x_{k+1}}\,.\end{aligned} \tag{7}$$

These can be written in matrix form as,

$$\begin{pmatrix} e^{i\kappa_k x_{k+1}} & e^{-i\kappa_k x_{k+1}}\\ i\kappa_k e^{i\kappa_k x_{k+1}} & -i\kappa_k e^{-i\kappa_k x_{k+1}} d\end{pmatrix}\begin{pmatrix}A_k\\ B_k\end{pmatrix} = \begin{pmatrix} e^{i\kappa_{k+1} x_{k+1}} & e^{-i\kappa_{k+1} x_{k+1}}\\ i\kappa_{k+1} e^{i\kappa_{k+1} x_{k+1}} & -i\kappa_{k+1} e^{-i\kappa_{k+1} x_{k+1}} d\end{pmatrix}\begin{pmatrix}A_{k+1}\\ B_{k+1}\end{pmatrix} \tag{8}$$

which can be written as,

$$\mathcal{M}(\kappa_k, x_{k+1})\phi_k = \mathcal{M}(\kappa_{k+1}, x_{k+1})\phi_{k+1} \tag{9}$$

where,

$$\mathcal{M}(\kappa, x) = \begin{pmatrix} e^{i\kappa x} & e^{-i\kappa x}\\ i\kappa e^{i\kappa x} & -i\kappa e^{-i\kappa x}\end{pmatrix} \qquad \phi_k = \begin{pmatrix}A_k\\ B_k\end{pmatrix} \tag{10}$$

This allows us to solve the wave function in the $k+1$ interval from the values in the k interval,

$$\phi_{k+1} = \mathcal{M}^{-1}(\kappa_{k+1}, x_{k+1})\mathcal{M}(\kappa_k, x_{k+1})\phi_k \tag{11}$$

where the inverse matrix reads,

$$\mathcal{M}^{-1}(\kappa, x) = \begin{pmatrix} \frac{1}{2}e^{-i\kappa x} & \frac{-i}{2\kappa}e^{-i\kappa x}\\ \frac{1}{2}e^{i\kappa x} & \frac{i}{2\kappa}e^{i\kappa x}\end{pmatrix} \tag{12}$$

A. how to find the eigenenergies?

To find the energy quantization we basicaly write the wave function in the last interval as a function of the wave function in the first interval,

$$\hat{\phi}_N = \mathcal{M}^{-1}(\kappa_N, x_N)\mathcal{M}(\kappa_{N-1}, x_N)\ldots\mathcal{M}^{-1}(\kappa_{j+1}, x_{j+1})\mathcal{M}(\kappa_j, x_{j+1})\ldots\mathcal{M}^{-1}(\kappa_1, x_1)\mathcal{M}(\kappa_0, x_1)\phi_0 \tag{13}$$

with $\phi_0 = \{A_0, -A_0\}$. This should indeed be equal to ϕ_N, that we wrote in Eq. (**??**).

Now for instance we do,

$$\hat{\phi}_N(1)/\hat{\phi}_N(2) - \phi_N(1)/\phi_N(2) = 0 \tag{14}$$

and look for values of E that solve the above equation. Notice in the above equation the values of A_0 and A_N.