QSE 200 Sections

Ben Augenbraun

Oct 11, 2022

CONTENTS

I am collecting notes/notebooks used in the QSE 200 (Fall 2022) Sections on this webpage.

CHAPTER

ONE

SECTION 4: DISCRETE VARIABLE REPRESENTATION

Last week, we studied the connection between position and momentum space using Fourier transforms. We learned that we can work in either "position space" with basis states $\langle x' | x \rangle = \delta(x-x')$, or in "momentum space" with basis states $\langle k' | k \rangle = \delta(k-k')$. We also showed that these kets are connected by the relation

$$\langle k|x\rangle = (2\pi)^{-1/2} e^{-ikx}. \tag{1.1}$$

In this week's section, we will use what we learned to build and solve matrix representations of the Hamiltonian for arbitrary 1D potentials using a method called the "Discerete Variable Representation" (DVR). This is a method that is widely used in physics and chemistry to solve for the states and energies of potential energy surfaces needed to describe molecular vibrations, chemical reactions, and beyond.

Learning Goals: After this Section you should be able to:

- Understand when it is more natural to work in a "position" vs. "momentum" basis
- Use the sinc basis to express the Hamiltonian for an arbitrary 1D potential
- Find the eigenvalues and eigenvectors of Hamiltonians for arbtirary potentials using the discrete variable representation

1.1 Motivating the Sinc-Basis DVR

As with previous problems, we'll first introduce a grid of N equally spaced points on the domain $[x_\text{min}, x_\text{max}]$. The points can be specified as

$$x_j = x_\text{min} + (j-1)\Delta x \tag{1.2}$$

where

$$\Delta x = \frac{x_\text{max} - x_\text{min}}{N-1}. \tag{1.3}$$

Next, we want to set up some basis ${|\phi_i\rangle}$ that allows us to express our wavefunction as a superposition of convenient basis functions. Putting this into math based on last week's section, we want a way to implement the expansion

$$\psi(x) = \langle x|\psi\rangle = \sum_{i=1}^{n} c_i \langle x|\phi_i\rangle = \sum_{i=1}^{n} c_i \phi_i(x). \tag{1.4}$$

There are many possible reasonable choices of this basis, and there is a vast literature of DVR methods that use different choices (some that can't even be written as nice analytical functions!). But let's think about what we would want in a general DVR basis set.

Question: What nice properties might we want?

Answer: Some example of useful properties would be:

- We want the basis to satisfy $\langle x_i | x_j \rangle \propto \delta_{i,j}$

- It would be nice if the basis functions were "well behaved" so that we can evaluate derivatives, Fourier transforms, etc.

One example of a set of functions that satisfy these desiderata are the *sinc functions*:

$$\phi_j(x) = \frac{1}{\sqrt{\Delta x}} \frac{\sin[\pi(x - x_j)/\Delta x]}{\pi(x - x_j)/\Delta x} = \frac{1}{\sqrt{\Delta x}} \text{sinc}[\pi(x - x_j)/\Delta x]. \tag{1.5}$$

First of all, let's explore why these basis functions are useful. It's best to start this discussion graphically, so let's use some code to implement the basis and plot the basis functions.

```
import numpy as np
import matplotlib.pyplot as plt
```

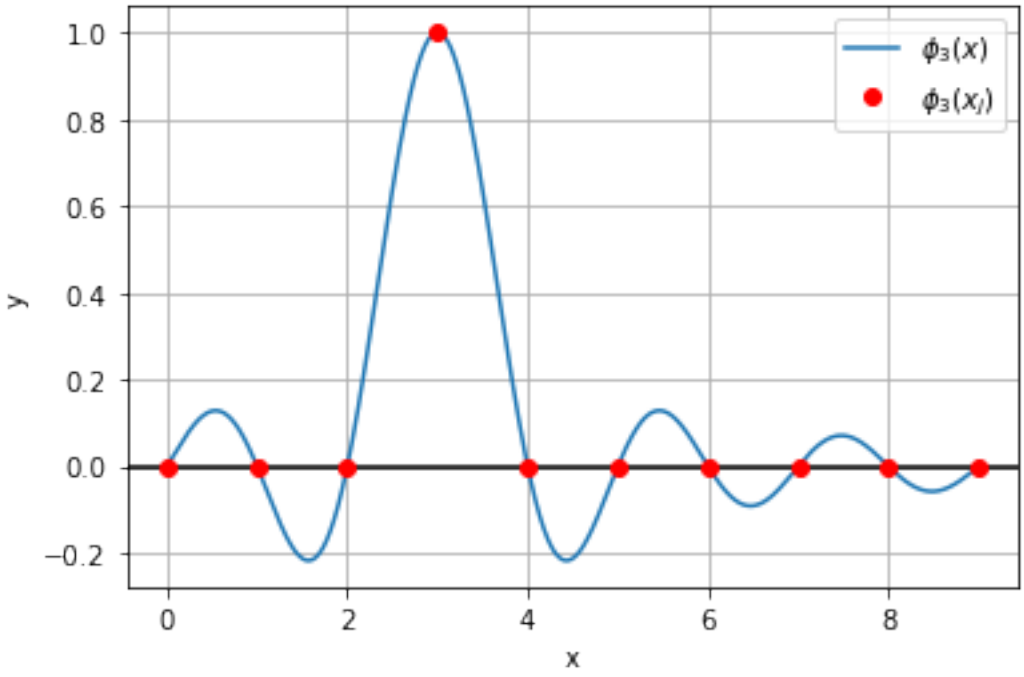
In the space below, we give you some code that defines a grid of x points and a function that implements the sinc-basis. Use these pieces of code to explore the sinc-basis functions and learn about their properties. You are looking to answer questions like the following:

- What is the value of a basis function ϕ_j at grid point x_j?
- What is the value of a basis function ϕ_j at a grid point x_{j+1}? At x_{j+2}?
- In general, how do the basis functions relate to one another?

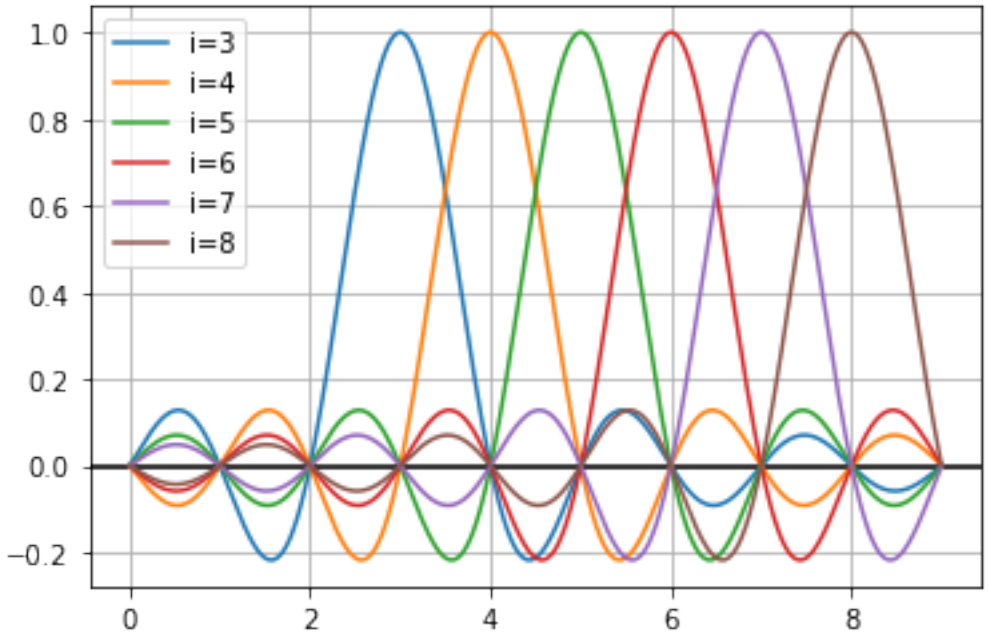
```
# Parameters that we set
xmin = 0
xmax = 9
N = 10

# Grid spacing and position-space grid points
Delta_x = (xmax-xmin)/(N-1)
xs = np.linspace(xmin,xmax,N)
xs_fine = np.arange(xmin,xmax,Delta_x/1000)

# Definition of the sinc-basis functions
def phi(x,xj):
    return 1/np.sqrt(Delta_x) * np.sinc((x-xj)/Delta_x)
```

There are two useful ways that we can look at these basis functions. First of all, we can plot an example like $\phi_3(x_j)$ to look at the $i=3$ basis function evaluated at each grid point. If we overlay this with a plot of $\phi_3(x)$ evaluated on a much finer grid, we see that the sinc-basis is defined such that a given basis function ϕ_i is nonzero at grid point x_i, and its oscillations ensure that it passes through zero at every other grid point!

Question: Does it matter that the sinc function takes on nonzero values away from the $i\neq j$ grid points?



It can also be useful to look at a series of these sinc functions centered at different grid points. Notice where they all pass through zero, and where each individual basis function doesn't pass through zero.



1.1.1 Interpreting the Sinc-Basis

Question: In light of these plots, how can we interpret the sinc-basis functions as compared to a set of Dirac delta functions?

Answer: This is a finite resolution basis that is like the closest possible approximation to a delta function that we can make for a given range on the number of momentum basis functions that our spatial grid can accomodate. As the plots above show, we means this in the sense that these basis states satisfy

$$\phi_j(x_k) \propto \delta_{k,j}, \tag{1.6}$$

so a given basis function only gives a nonzero contribution to a function at the point at which that basis function is centered.

The sinc basis has a nice relationship to the Fourier transforms that we discussed last week. In particular, the sinc basis functions can be written as follows:

$$\phi_j(x) = \frac{\sqrt{\Delta x}}{2\pi} \int_{-\pi/\Delta x}^{\pi/\Delta x} e^{ik(x-x_j)} dk \tag{1.7}$$

Question: Can you justify this form qualitatively? How does it relate to Fourier transforms that we studied last week? And why are the limits on the integral those numbers?

1.2 Deriving the Kinetic Energy Matrix

Now that we have a useful basis, we can try to solve problems with it. We still need to learn how to express the Hamiltonian in this basis. Since the Hamiltonian is

$$H = -\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + V(x) \tag{1.8}$$

we need to figure out how to express the second derivative operator and the $V(x)$ operator in our chosen basis.

Let's start with the kinetic energy term. We want to derive a kinetic energy matrix with matrix elements

$$T_{ij} = -\frac{\hbar^2}{2m}\left\langle \phi_i \left| \frac{d^2}{dx^2} \right| \phi_j \right\rangle. \tag{1.9}$$

It turns out this is where the Fourier transform expression is super helpful. We can go through the following steps:

$$\int_{-\infty}^{\infty} \phi_i^*(x)\frac{d^2}{dx^2}\phi_j(x)dx = \frac{\Delta x}{(2\pi)^2}\int_{-\infty}^{\infty} dx \int_{-\pi/\Delta x}^{\pi/\Delta x} e^{-ik(x-x_i)}dk \int_{-\pi/\Delta x}^{\pi/\Delta x}\left(\frac{d^2}{dx^2}e^{ik'(x-x_j)}\right)dk' \tag{1.10}$$

Because we've expressed the basis function in terms of plane waves, evaluating $\frac{d^2}{dx^2}$ is really easy: it just corresponds to pulling down a factor of $(ik)^2$. That gets us to

$$\frac{\Delta x}{(2\pi)^2}\int_{-\pi/\Delta x}^{\pi/\Delta x}(ik)^2 e^{ik(x_i-x_j)}dk. \tag{1.11}$$

If you work through these integrals, you will find that:

$$\boxed{T_{i,j=i} = \frac{\hbar^2}{2m\Delta x^2}\frac{\pi^2}{3}} \tag{1.12}$$

and

$$\boxed{T_{i,j\neq i} = \frac{\hbar^2}{2m\Delta x^2}\frac{2(-1)^{i-j}}{(i-j)^2}} \tag{1.13}$$

Task: Now fill in the code below to make a function that builds the kinetic energy operator and the potential energy operator.

```
# work in some convenient units
hbar = 1
m = 1
```

```
# Function to make the kinetic energy operator
def make_T(x):
    Delta_x = x[1]-x[0]
    N = xs.shape[0]
    Tmat = np.zeros((N,N))

    # now loop over kinetic energy matrix and fill in matrix elements
    for i in range(N):
        for j in range(N):
            if i==j:
                Tmat[i,j] = (hbar**2/(2*m*Delta_x**2)) * (-1)**(i-j) * (np.pi**2)/3
            else:
                Tmat[i,j] = (hbar**2/(2*m*Delta_x**2)) * (-1)**(i-j) * 2/(i-j)**2

    return Tmat
```

1.3 Deriving the Potential Energy Matrix

For the potential energy matrix, we need to express $V(x)$ in matrix form. Since $V(x)$ can be expanded as a power series in the position operator $\hat{x}$, let's first look at the matrix form of $\hat{x}$ itself. We want the matrix elements $x_{ij} = \langle \phi_i | x | \phi_j \rangle$. In other words, we want

$$\int_{-\infty}^{\infty} \phi_i(x)^* x \phi_j(x) dx = \frac{\Delta x}{(2\pi)^2} \int_{-\pi/\Delta x}^{\pi/\Delta x} dk \int_{-\pi/\Delta x}^{\pi/\Delta x} dk' e^{ikx_i - ik'x_j} \int_{-\infty}^{\infty} x e^{ix(k'-k)} dx. \tag{1.14}$$

In the last term, note that we can replace x by $\frac{d}{d(ik')}$ and then pull the derivative outside of the integral over x (since it doesn't depend on x anymore). Working through the integrals after that step, we get to

$$x_{ij} = \frac{\Delta x}{2\pi} x_i \int_{-\pi/\Delta x} \pi/\Delta x dk' e^{ik'(x_i - x_j)} = x_i \delta_{i,j}. \tag{1.15}$$

The important conclusion is that $\hat{x}$ is diagonal in this basis! So the potential energy, which can be expanded in powers of x is also diagonal in this basis! Knowing this, we can just make the potential energy operator by building a diagonal matrix from $V(x)$. The function `make_V` does this, and then the function `make_H` just calls the kinetic and potential energy functions and puts them together.

```
# Function to make the potential energy operator
def make_V(x,Vfunc):
    Vmat = np.diag(Vfunc(x))
    return Vmat

# Function to make the full Hamiltonian
def make_H(x,Vfunc):
    return make_T(x) + make_V(x,Vfunc)
```

1.4 Testing the DVR code

That was a lot of work to get our Hamiltonian matrix. But now let's see the payoff! We'll see that the DVR code is incredibly accurate, especially compared to the finite differences method that we used before. As always let's test our code by using a harmonic oscillator potential. Let's start with a 20 point grid and see how the calculations do for energies and wavefunctions.

```
xs = np.linspace(-10,10,20)
```

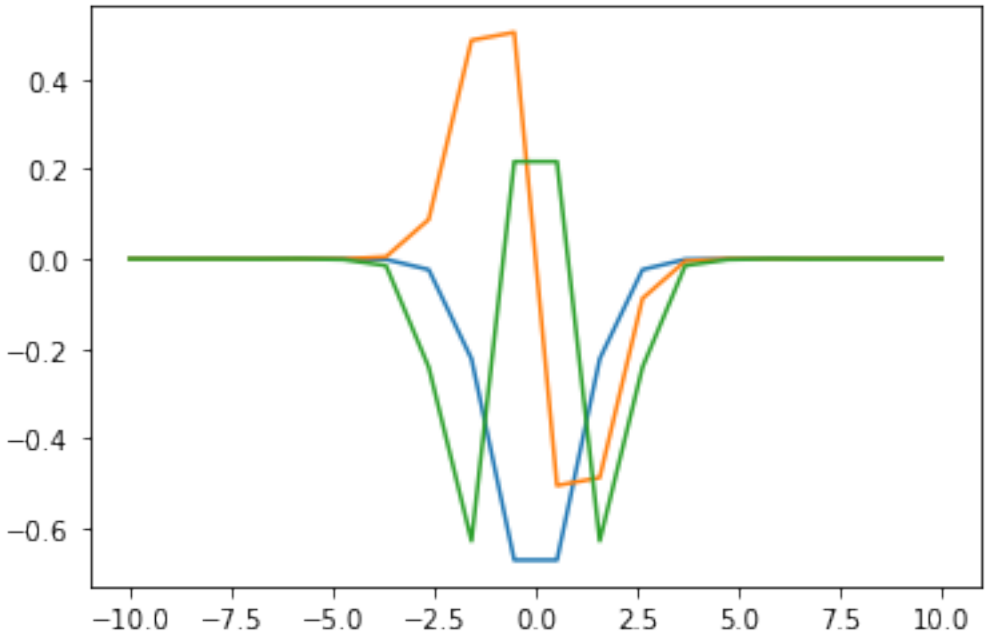
```
def Vharmonic(x):
    return 0.5*x**2
```

```
Ham = make_H(xs,Vharmonic)
vals, vecs = np.linalg.eigh(Ham)
vals[0:4]
```

```
array([0.50042652, 1.49199271, 2.54377448, 3.30864454])
```

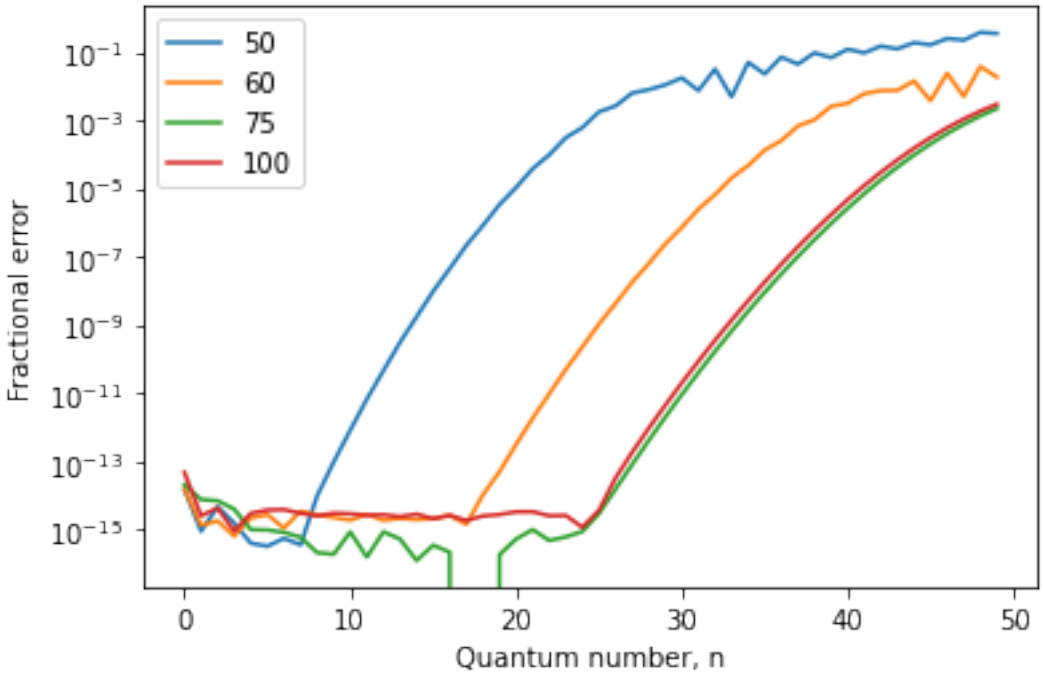
```
plt.plot(xs,vecs[:,0:3])
```

```
[<matplotlib.lines.Line2D at 0x7fdb1af5e8e0>,
 <matplotlib.lines.Line2D at 0x7fdb1af5e910>,
 <matplotlib.lines.Line2D at 0x7fdb1af5ea30>]
```



Let's look at the accuracy of our solutions as a function of quantum number.

```
Text(0, 0.5, 'Fractional error')
```



Question: Explain the differences between the case with 50 grid points and the case with 60 grid points. Why is the agreement better to higher n for 60 grid points? Also, why does the agreement seem to always get worse above $n \sim 25$, even if the number of grid points is increased?

1.4.1 Comparing DVR to Finite Differences

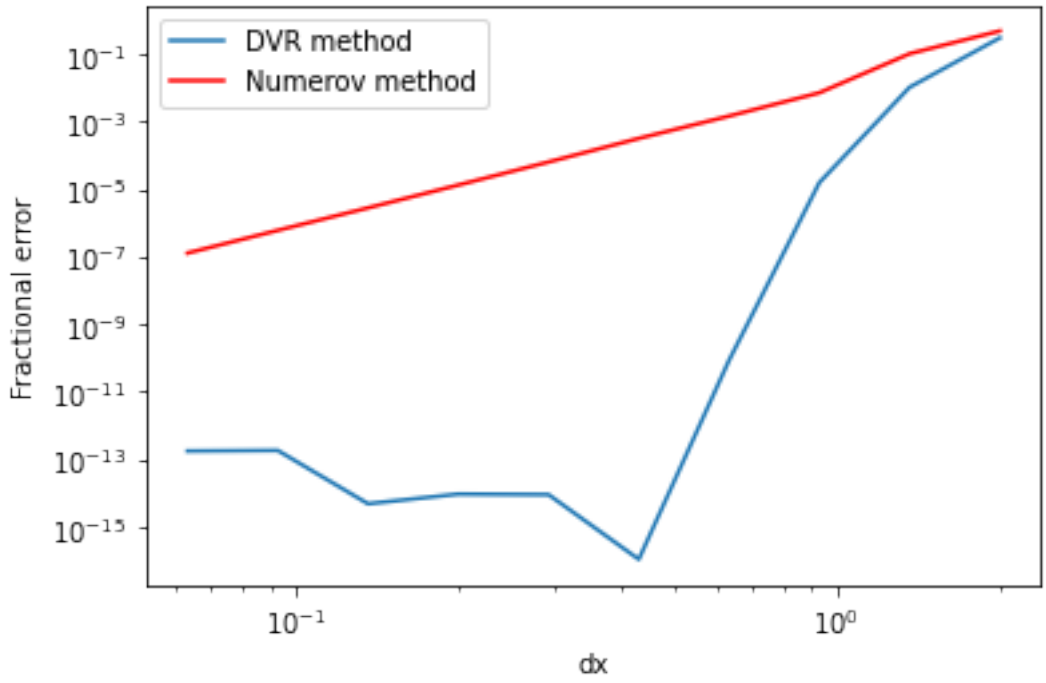
Let's look at the accuracy of the numerical solution as a function of the number of grid points. We can compare this to the results of HW 1 to see how the DVR method does against finite differences.

In this block, write a function that computes the fractional error in the SHO eigenstates as a function of the number of grid points used.

This next block is already written for you– nothing you need to do!— to implement and solve the same problem using the Numerov method. The code is taken from the HW1 solutions.

Now, make a plot of the relative errors comparing the numerov and DVR methods for the ground state as a function of the grid spacing Δx.

```
Text(0, 0.5, 'Fractional error')
```



Clearly the DVR method is much better than the Numerov technique! For reasonable grid sizes, the DVR method gets the ground-state energy with almost 10 orders of magnitude better relative agreement!! (And remember, the Numerov method was already much better than a "plain" second-order finite differences approximation to the Hamiltonian…)

Question: Why do you think the DVR method stops getting more accurate at a certain step size? What does this tell you about setting up simulations?

CHAPTER

TWO

SECTION 6: AN INTUITIVE PICTURE OF BAND STRUCTURE

```
import numpy as np
import matplotlib.pyplot as plt
from ipywidgets import interact, FloatSlider
from IPython.display import display
```

In this Section, we are going to explore the emergence of band structure using simple models that you have studied before. We will consider a series of potential barriers (equally spaced) to simulate the periodic potential that electrons experience inside a crystal. By varying the heights of these barriers, we can smoothly transition between two simple limits: (1) a single, wide square well and (2) a series of narrow, uncoupled square wells. Band structure emerges between these two extremes.

Learning Goals After this Section, you should:

- Be able to explain the qualitative origins of energy bands
- Explain the origin of energy spacings *within* bands as compared to *between* bands
- Describe the shape of electronic wavefunctions at the edge of each Brillouin zone
- Explain the connection between bound state and scattering methods of characterizing the energy levels of the periodic potential

2.1 Qualitative Solution

Before we dive into numerical solutions, let's think about the following qualitative situation: we have a square well of width L. Within the well, we place a series of barriers that partition the square well into N narrower wells with width a. Each barrier has height β and thickness b.

Let's start by considering the barriers to be infinitely thin (basically, delta functions). Now, as a class, let's work through the following questions:

1. What are the energies and eigenstates of the wide well if the barriers are not present ($\beta=0$)?
2. When $\beta \rightarrow \infty$, what should the wavefunctions look like?
3. As we increase β to take on finite values, what will happen to the wavefunctions? How will they look? What will their ordering be like relative to the cases we just described?
4. What analogies could we make between excitation of different bands and excitations of atoms in a solid?

After going through these qualitative arguments, we can now look at what the numerics tell us.

2.2 Building the potential V(x)

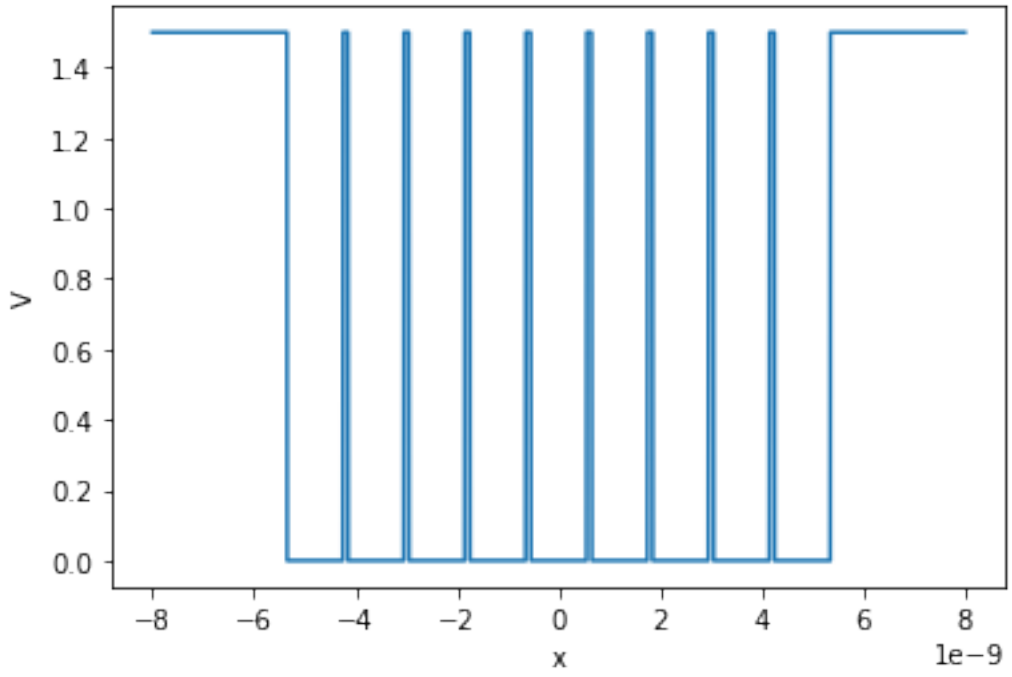
First we'll write some functions that build our model potential. Our model includes the following parameters:

- L: width of the enclosing square well
- a: width of each narrow square well
- b: thickness of each internal barrier
- N: number of narrow square wells
- β: heigh of each barrier

```
# This function will return another function that depends only on x.
# That's a convenient way to get a function of x that can go into our DVR code.
# fullwidth allows us to embed the entire potential in a wider "framing" square well
def make_potential(N, a, b, beta):
    L = N*a+(N-2)*b
    if N > 0:
        x0s = np.arange(1,N)*(a+b)-(b) - L/2 # locations of barrier centers
        def fn(x):
            # builds a barrier of height beta and width b centered at each x0 value:
            barriers = beta*np.sum([np.heaviside(x-x0s[n]+b/2 , 0.5) - np.
↪heaviside((x-x0s[n]-b/2), 0.5) for n in range(N-1)], axis=0)
            return barriers + beta * ((x < -L/2-b/2) | (x>L/2+b/2))
    else:
        a = L
        def fn(x):
            return np.zeros(len(x)) + beta * ((x < -L/2) | (x>L/2))

    return fn
```

```
f_test = make_potential(9, 1.1e-9, 0.1e-9, 1.5)
x = np.linspace(-8,8, 5000)*1e-9
plt.plot(x,f_test(x).T)
plt.xlabel("x")
plt.ylabel("V")
```

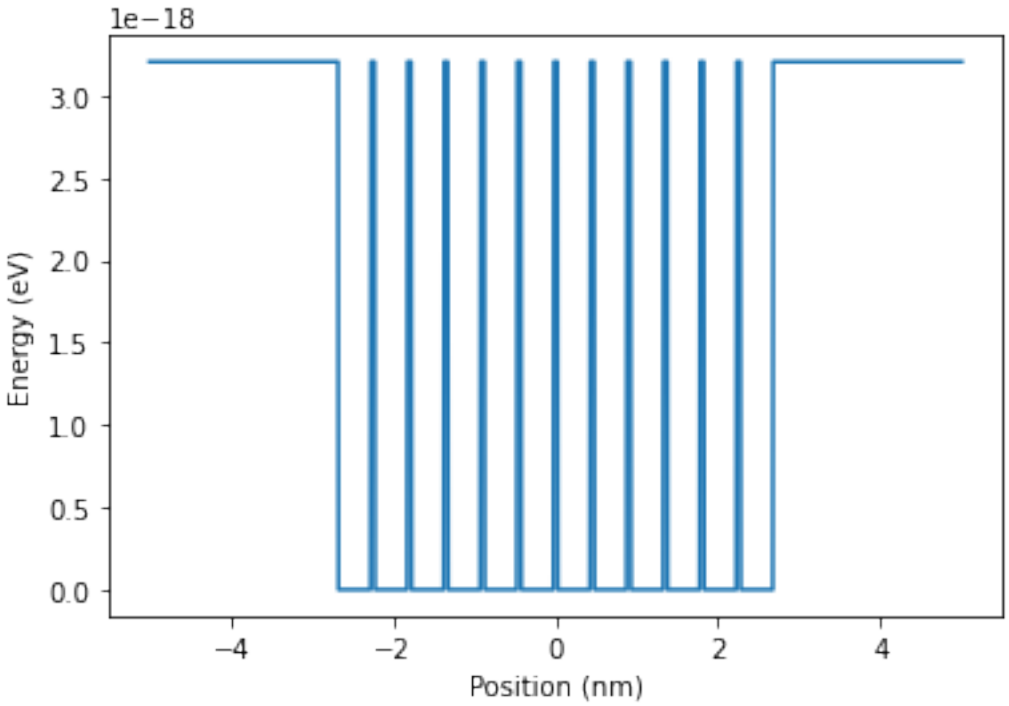
```
Text(0, 0.5, 'V')
```



2.3 Making the DVR code

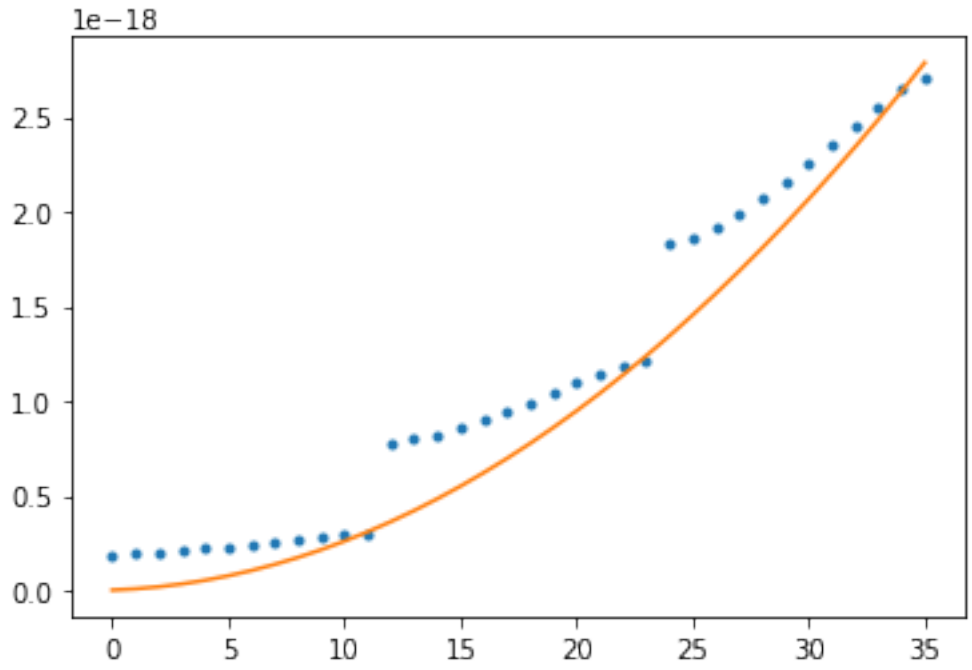
Next, we set up the DVR code from Section 4 to build the Hamiltonian for our system. This code is the same as we have used previously.

Let's test this on a potential consisting of 12 wells that are fairly deep. After building the potential, we construct the Hamiltonian, evaluate the eigenvectors/eigenvalues, and then plot.



We can now plot the numerically computed eigenvalues. Let's overlay these values with the energy levels that would occur if there were no barriers inside the potential—the energy levels of the "wide" finite square well.

```
[<matplotlib.lines.Line2D at 0x7f8b70556e80>]
```



Notice that the periodic potential leads to a structure highly reminiscient of band structure: the energy levels roughly follow the parabolic shape expected from the wide square well (or, from a different perspective, for the free particle), but the levels are broken up into bands with the characteristic shape that we see in class. And there are band gaps between

these bands.

Question: What will the plot look like if we increase the well separation to the point that the wells are essentially fully uncoupled? Will there be energy degeneracies? How many levels will share the same energy? What happens as the number of wells is increased–which energy gaps "close" and which remain?

To answer these questions, perform a "numerical experiment" and compare your results to the appropriate square well. How do the numbers compare?

Question: What happens if the well separation is decreased to essentially zero? How do these energies compare to the appropriate finite square well? What about the comparison to free particle energies?

Let's look at the wavefunctions to understand this even better. It will be useful to plot the actual wavefunctions and have some interactivity. The code below makes a plot with a slider that allows you to look at all of the bound eigenstates for a given potential. Once you set the number of wells, their width, and their barriers, you can scan through the eigenstates and look at the behavior.

For example, we can start with 12 wells that are each 0.8 nm wide, 4 eV high, and have a 0.05 nm barrier between them. For this set of parameters, use the interactive plotting to gain some insights about the potential.

```
N = 12 # number of wells
a = 0.8e-9
b = 0.05e-9 # wall thickness in nm
beta = 4 / 6.24150907e18 # well height in J
V_wells = make_potential(N, a, b, beta)
xs = np.linspace(-8, 8, 900)*1e-9

Ham=make_H(xs,V_wells)

vals, vecs = np.linalg.eigh(Ham)
# make sure ground state is upward going
vecs[:,0] = vecs[:,0] * np.sign(vecs[np.argmax(np.abs(vecs[:,0])),0])
# number of bound states:
nbound = np.sum([vals<beta])


interact_wavefunction();
```

```
interactive(children=(FloatSlider(value=0.0, description='Eigenstate', max=32.0,␣
↪step=1.0), Output()), _dom_cl…
```

Question: Using the interactive graphs above, how does the behavior of each wavefunction within a single band conform to our expectations from Bloch's theorem? How do the wavefunctions differ from band to band? What do you expect to happen as $N \rightarrow \infty$?

2.4 Transfer matrix approach

There's a totally different way to look at band structure: by studying the scattering properties of the potential. This is a complementary method to learn about a potentials bound states. Scattering studies are widely used in many fields of physics/chemistry, from studying the solid state to studying atoms and molecules. Let's see what we can learn about electronic band structure from this point of view using the transfer matrix program that you wrote on HW2.

First of all, let's go over the idea of the transfer matrix approach.

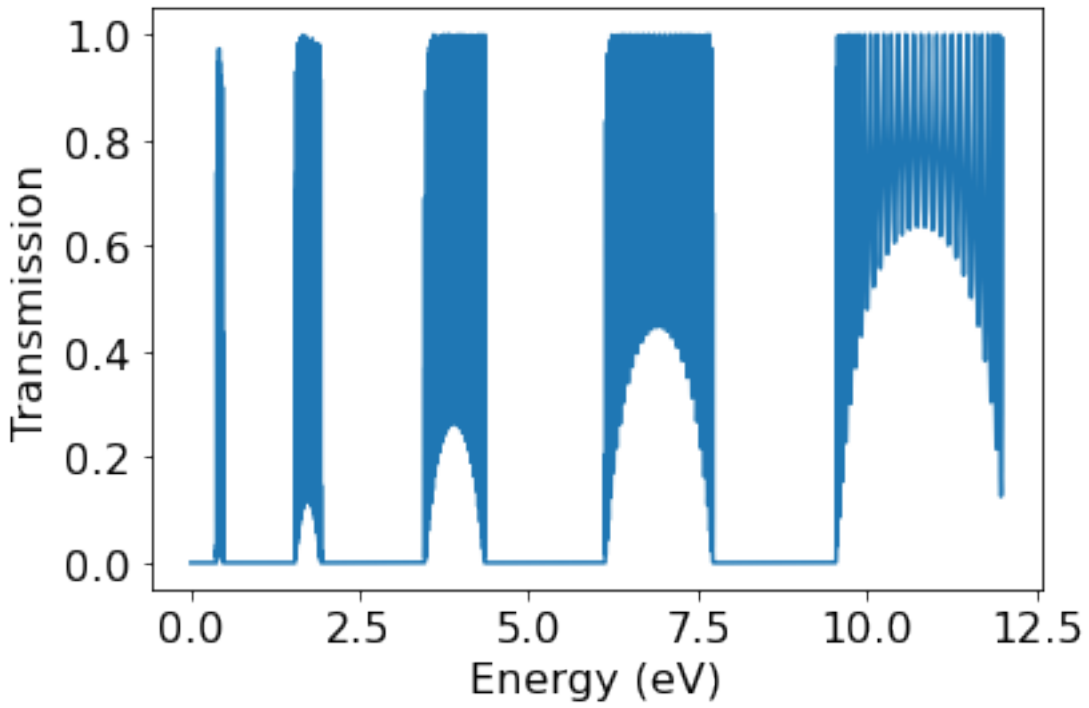
Set up approach, talk about different role of propagation vs. transmission matrices.

The code to implement each of these matrices is copied below.

```
def DMat(k1, k2):
    res = np.zeros((2,2),dtype=np.complex_)
    res[0,0] = (1 + k2/k1)/2
    res[0,1] = (1 - k2/k1)/2
    res[1,0] = res[0,1]
    res[1,1] = res[0,0]
    return res

def PMat(k, L):
    res = np.zeros((2,2),dtype=np.complex_)
    res[0,0] = np.exp(-1j * k * L)
    res[1,1] = np.exp(1j * k * L)
    return res
```

Question: What do the peaks here represent? What is the physics going on when N=2? N=3? Large N?

```
a = 0.8e-9
b = 0.1e-9

Es = np.arange(0.01, 12.0, 0.001) / 6.24150907e18
width_barrier = b
width_gap = a
Vb=10/ 6.24150907e18

Nwells=30


Ttrans = np.zeros(Es.size)
i = 0
for E in Es:
    klow = np.emath.sqrt(2 * m * E/hbar**2)
    khigh = np.emath.sqrt(2 * m * (E - Vb)/hbar**2)
    res_mat = DMat(klow, khigh) @ PMat(khigh, width_barrier) @ DMat(khigh, klow) @␣
 ↪PMat(klow, width_gap)
    U, V = np.linalg.eig(res_mat)
    diag_res_mat = np.diag([U[0],U[1]])
    res_mat = np.linalg.matrix_power(diag_res_mat,Nwells)
    res_mat = V @ res_mat @ np.linalg.inv(V)
    Ttrans[i] = 1 - np.abs(res_mat[1, 0])**2 / np.abs(res_mat[0,0])**2
    i = i + 1

plt.plot(Es * 6.24150907e18, Ttrans)
plt.xlabel("Energy (eV)",fontsize=16)
plt.ylabel("Transmission",fontsize=16)
plt.tick_params(axis='both', which='major', labelsize=16)
```



Let's compare the transmission peaks to the bound state energy of our finite square well.

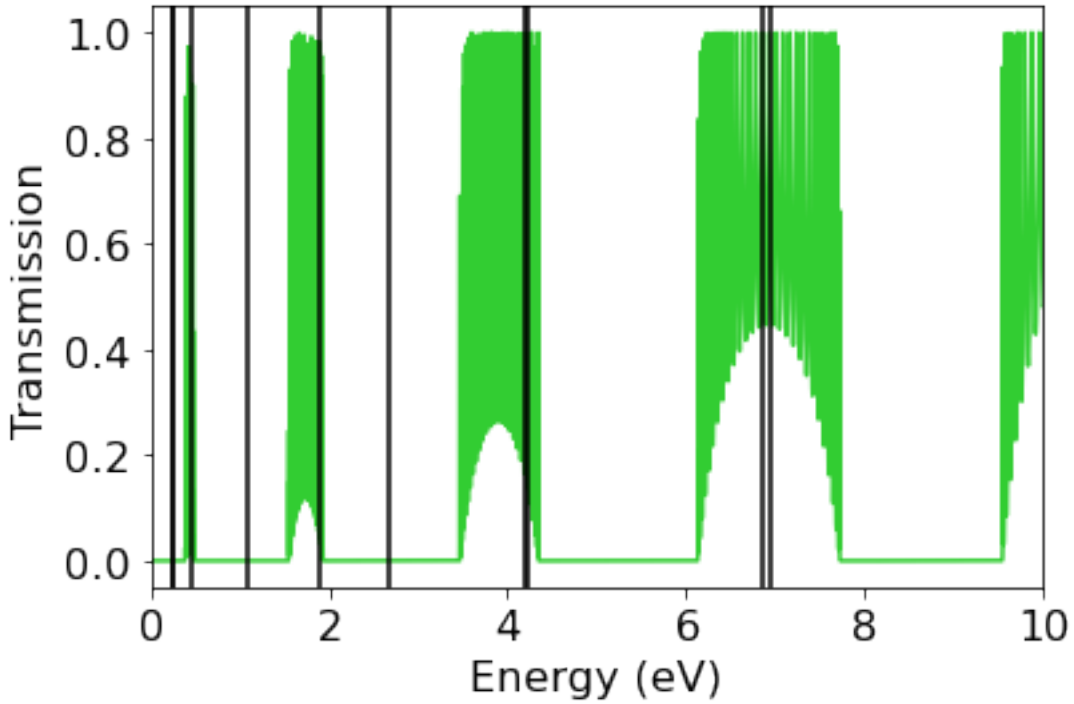
```
V_wells = make_potential(Nwells, a, b, beta)
xs = np.linspace(-15, 15, 1500)*1e-9

Ham=make_H(xs,V_wells)

vals, vecs = np.linalg.eigh(Ham)
```

Now let's compare the transmission fraction to the band energies.

Question: How can we easily determine the limits of each energy band? (How does it compare to the number of wells being simulated?

```
band_limits = [0]
for n in range(5):
    band_limits.append(n*Nwells-1)
    band_limits.append(n*Nwells)

plt.plot(Es * 6.24150907e18, Ttrans, color="limegreen")
[plt.axvline(vals[i]*6.24150907e18, color="black") for i in band_limits]
plt.xlabel("Energy (eV)",fontsize=16)
plt.ylabel("Transmission",fontsize=16)
plt.tick_params(axis='both', which='major', labelsize=16)
plt.xlim(0,10)
```

```
(0.0, 10.0)
```


Transmission
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
Energy (eV)