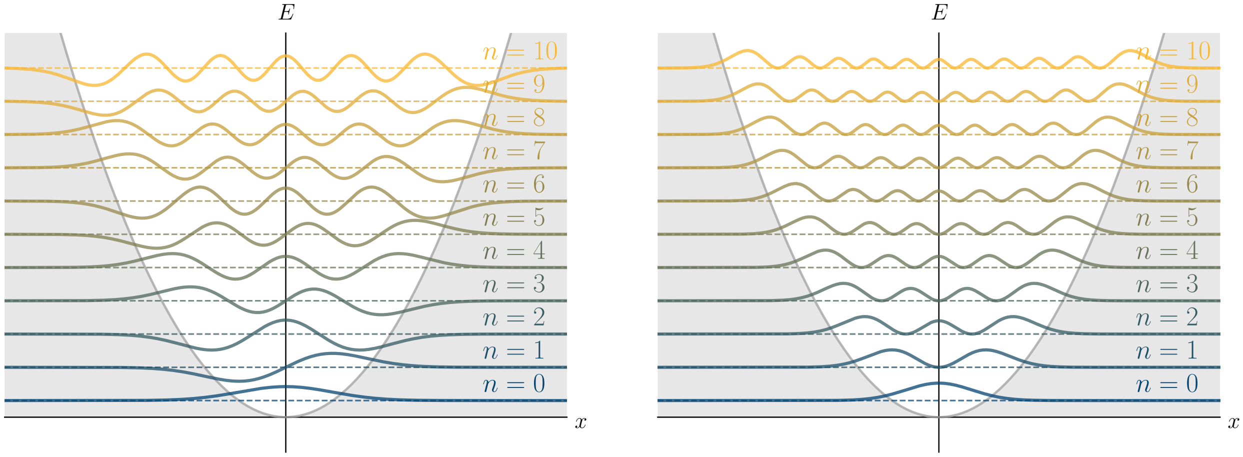
ML2R Coding Nuggets
Numerically Solving the Schrödinger Equation (Part 1)

Christian Bauckhage*
Machine Learning Rhine-Ruhr
University of Bonn
Bonn, Germany



energy eigenstates $\Psi_n(x)$

probability densities $|\Psi_n(x)|^2$

Figure 1: Illustration of energy eigenstates of a 1D quantum harmonic oscillator and corresponding probability densities.

ABSTRACT

Most quantum mechanical systems cannot be solved analytically and therefore require numerical solution strategies. In this note, we consider a simple such strategy and discretize the Schrödinger equation that governs the behavior of a one-dimensional quantum harmonic oscillator. This leads to an eigenvalue / eigenvector problem over finite matrices and vectors which we then implement and solve using standard *NumPy* functions.

1 INTRODUCTION

The one-dimensional quantum harmonic oscillator is an important model system in quantum mechanics. It is the quantum analog of the classical harmonic oscillator and one of the few quantum systems for which there existst an analytical solution [8]. However, this note is concerned with *numerical* solutions to the corresponding Schrödinger equation. Our goal is to use this arguably simple setting to familiarize ourselves with fundamental approximation techniques which will come in handy later.

Our specific approach will be to discretize the position variable of the quantum harmonic oscillator and to compute the spectral decomposition of the correspondingly discretized Hamiltonian of the system. As we shall see, this can be easily accomplished using standard *NumPy* methods.

While there already are several Web tutorials on this approach and its implementation using *NumPy*, the ones we know of are not really *numpythonic*. That is, they present convoluted or sloppy code that typically involves *Python* `for` loops. Long-time readers of this series do of course know that these are a bane when it comes to efficiency in numerical computing. Long-time readers also know that *NumPy* is much richer than it appears to the novice and provides special purpose methods that allow for writing efficient, vectorized code. Indeed, we will demonstrate that our current setting is no exception and allows for compact and efficient solutions.

As always, we will first review the necessary theory (in section 2) and then present practical implementation ideas and discuss their characteristics (in section 3).

Ideally, readers of this note should have a background in quantum mechanics; those who don't will have to take much of the theory and jargon in section 2 for granted.

Readers who would like to experiment with our code snippets in section 3 should be familiar with *NumPy* , *SciPy*, and *Matplotlib* [4, 7] and only need to

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

* 0000-0001-6615-2128

2 THEORY

The Hamiltonian of a one-dimensional quantum harmonic oscillator (a particle of mass m that experiences a restoring force $F = -\frac{dV}{dx}$ that is proportional to its displacement x from an equilibrium point) amounts to

$$\hat{H} = \hat{T} + \hat{V} = \frac{1}{2m}\,\hat{p}^2 + \frac{1}{2}\,m\,\omega^2\,\hat{x}^2 \tag{1}$$

Here, $\hat{T}$ and $\hat{V}$ are kinetic- and potential energy operators and the quantities

$$\omega = \sqrt{\frac{k}{m}} \qquad \hat{x} = x \qquad \hat{p} = -i\,\hbar\,\frac{d}{dx}$$

denote the angular frequency of the oscillator, the position operator, and the momentum operator, respectively. For simplicity, we will henceforth consider atomic units $m = 1$, $\hbar = 1$ and also let $k = 1$.

With these basic assumptions, the Hamiltonian in (1) becomes

$$\hat{H} = -\frac{1}{2}\,\frac{d^2}{dx^2} + \frac{1}{2}\,x^2 \tag{2}$$

and the time-independent Schrödinger equation $\hat{H}\,\Psi(x) = E\,\Psi(x)$ which governs the behavior of the system reads

$$\left(-\frac{1}{2}\,\frac{d^2}{dx^2} + \frac{1}{2}\,x^2\right)\,\Psi(x) = E\,\Psi(x) \tag{3}$$

where $\Psi(x)$ denotes the particle's wave function.

2.1 The Analytical Solution

Solutions (i.e. eigenvalues E_n and eigenfunctions $\Psi_n(x)$) to the second order differential equation in (3) are known to be given by

$$\Psi_n(x) = \frac{1}{\sqrt{2^n n!}\,\sqrt[4]{\pi}} \cdot H_n(x) \cdot e^{-\frac{x^2}{2}} \tag{4}$$

$$E_n = n + \frac{1}{2} \tag{5}$$

where $H_n(x)$ denotes the *physicist's* Hermite polynomial of order n and $n = 0, 1, 2, \ldots$

2.2 A Numerical Solution Scheme

For our numerical solution of the Schrödinger equation in (3), we assume that the movement of the quantum particle is confined to a one-dimensional interval $[-l/2, +l/2]$ of length l. On this interval, we consider a grid of N equally spaced points

$$-\frac{l}{2} \leq x_i \leq +\frac{l}{2} \tag{6}$$

such that the distance between any pair of neighboring grid points amounts to

$$\delta x = \frac{l}{N-1} \tag{7}$$

This discretization of the domain of the continuous position variable x allows us to represent the continuous wave function $\Psi(x)$ in terms of an N-dimensional vector

$$\boldsymbol{\psi} = \begin{bmatrix} \Psi(x_1) \\ \Psi(x_2) \\ \vdots \\ \Psi(x_N) \end{bmatrix} = \begin{bmatrix} \psi_1 \\ \psi_2 \\ \vdots \\ \psi_N \end{bmatrix} \tag{8}$$

Next, we discretize the second order derivative operator that features prominently in (3). To this end, we note that our domain discretization allows us to approximate first order derivatives of the wave function Ψ at grid points x_i in terms of finite differences, namely

$$\frac{d}{dx}\Psi(x_i) \approx \frac{\Psi(x_{i-1}) - \Psi(x_i)}{\delta x} \tag{9}$$

Applying this idea again allows us to discreteize the second order derivative as follows

$$\frac{d^2}{dx^2}\Psi(x_i) \approx \frac{\frac{\Psi(x_{i-1})-\Psi(x_i)}{\delta x} - \frac{\Psi(x_i)-\Psi(x_{i+1})}{\delta x}}{\delta x} \tag{10}$$

$$= \frac{\Psi(x_{i-1}) - 2\,\Psi(x_i) + \Psi(x_{i+1})}{\delta x^2} \tag{11}$$

In other words, if we express (11) in terms of entries ψ_i of the vector $\boldsymbol{\psi}$ defined in (8), we now have the following approximation

$$-\frac{1}{2}\,\frac{d^2}{dx^2}\Psi(x_i) \approx \frac{-\psi_{i-1} + 2\,\psi_i - \psi_{i+1}}{2\,\delta x^2} \tag{12}$$

Using (12), we can therefore represent the continuous kinetic energy operator $\hat{T}$ in terms of a tridiagonal matrix of size $N \times N$

$$\boldsymbol{T} = \frac{1}{2\,\delta x^2}\begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix} \tag{13}$$

and obtain the following approximation

$$\hat{T}\,\Psi(x) = -\frac{1}{2}\,\frac{d^2}{dx^2}\,\Psi(x) \approx \boldsymbol{T}\,\boldsymbol{\psi} \tag{14}$$

Similarly, we can approximate the continuous potential energy operator $\hat{V}$ as a diagonal $N \times N$ matrix

$$\boldsymbol{V} = \frac{1}{2}\begin{bmatrix} x_1^2 & & \\ & \ddots & \\ & & x_N^2 \end{bmatrix} \tag{15}$$

and thus obtain

$$\hat{V}\,\Psi(x) = \frac{1}{2}\,x^2\,\Psi(x) \approx \boldsymbol{V}\,\boldsymbol{\psi} \tag{16}$$

Putting all these considerations together, a discretized version of the Hamiltonian $\hat{H}$ of the quantum harmonic oscillator becomes

$$\boldsymbol{H} = \boldsymbol{T} + \boldsymbol{V} \tag{17}$$

and we note that this $N \times N$ matrix $\boldsymbol{H}$ is symmetric because it is a sum of two symmetric matrices,

Last but not least, a discretized version of the Schrödinger equation in (3) can now be written in terms of finitely sized matrices and vectors, namely

$$\boldsymbol{H}\,\boldsymbol{\psi} = E\,\boldsymbol{\psi} \tag{18}$$

which we recognize as a simple eigenvalue / eigenvector problem.

Listing 1: solving the discretized Schrödinger equation (18)

```
 1 l = 12.
 2 N = 1001
 3
 4 xs = np.linspace(-l/2, +l/2, n)
 5 vs = 0.5 * xs**2
 6
 7 dx = xs[1]-xs[0]
 8
 9 matV = np.diag(vs)
10
11 matT = 2 * np.diag(np.ones(N)) \
12        - np.diag(np.ones(N-1), +1) \
13        - np.diag(np.ones(N-1), -1)
14 matT /= (2 * dx**2)
15
16 matH = matT + matV
17
18 es, psis = la.eigh(matH)
19
20 psis /= np.sqrt(np.sum(psis**2, axis=0))
21
22 dens = np.abs(psis)**2
```

Table 1: Analytical- and numerical eigenenergies of a QHO

n	E_n analytical	E_n numerical	
		$N = 1001$	$N = 2001$
0	0.5	0.499995	0.499999
1	1.5	1.499977	1.499994
2	2.5	2.499941	2.499985
3	3.5	3.499887	3.499972
4	4.5	4.499815	4.499954
5	5.5	5.499726	5.499931
6	6.5	6.499618	6.499905
7	7.5	7.499493	7.499874
8	8.5	8.499355	8.499845
9	9.5	9.499231	9.499844
10	10.5	10.499231	10.499988

3 PRACTICE

In this section, we discuss how to implement the above ideas in *NumPy*. A look at Listing 1 suggests that this is actually straightforward. To better appreciate the rationale behind the individual steps of this piece of code, we will discuss it line by line.

To begin with, we need to set the length l of the interval to be considered and the number N of grid points we want to place within this interval. For example, when computing the results in Fig. 1, we used $l = 12$ and $N = 1001$ just as in lines 1 and 2.

To represent the grid points x_i and the corresponding potential energies $V(x_i) = \frac{1}{2}\,x_i^2$, we use two *NumPy* arrays `xs` and `vs` and initialize them as shown in lines 4 and 5.

Given the array `xs` containing equally spaced grid points, the grid point distance δx can be computed as in line 7.

Given array `vs`, matrix $\boldsymbol{V}$ can easily be implemented using the *NumPy* function `diag` (see line 9).

To implement matrix $\boldsymbol{T}$, we proceed as in lines 11–14. This involves the *NumPy* function `ones` and once again the function `diag`. At this point, we note that `diag` comes with two parameters `v` and `k` where `v` is an array of values to be set on a diagonal of a matrix and `k` is an integer (…, -1, 0, +1, …) indicating which diagonal is to be set. The default (`k=0`) is to consider the main diagonal, positive or negative choices of `k` indicate sub-diagonals above or below the main diagonal. In other words, those who know *NumPy* well do not need any `for` loops to implement an array `matH` that represents the Hamiltonian $\boldsymbol{H}$ of a quantum harmonic oscillator (see line 16).

Once `matH` is available, we can compute its spectral decomposition. Since `matH` represent a symmetric or Hermitian matrix, we apply function `eigh` in *NumPy*'s `linalg` module (line 18).[1] This provides us with a 1D array `es` of eigenvalues E_n of $\boldsymbol{H}$ and a 2D array `psis` of eigenvectors $\boldsymbol{\psi}_n$ of $\boldsymbol{H}$.

For downstream processing, it is good practice to normalize the latter such that $\|\boldsymbol{\psi}_n\| = 1$. This happens in line 20.

Finally, line 22 turns the array `psis` representing wave functions $\Psi(x)$ into an array `dens` representing probability densities $|\Psi(x)|^2$.

A simple *Matplotlib* recipe for plotting, say, the first 11 columns of the array of densities is

```
num = 11
plt.figure(figsize=(10,10))
for i, n in enumerate(reversed(range(num))):
    plt.subplot(num, 1, i+1)
    plt.plot(xs, dens[:,n])
    plt.axis('off')
plt.show()
```

Running this little script will produce a plot similar to the one in Fig. 1(b), albeit not quite as appealing.

To produce the result in Fig. 1, we computed arrays `psis` and `dens` just as shown in Listing 1 and plotted their first 11 columns in a fanciful manner. Readers with a background in quantum mechanics will recognize from Fig. 1 that our numerical solutions of the quantum harmonic oscillator appears to be convincing. But how good are they really?

A simple quality check consists in comparing our numerically obtained eigenvalues E_n to the analytically prescribed ones. Table 1 presents such a comparison. Its second column shows eigenvalues computed according to equation (5); its third column shows eigenvalues we obtained from running the code in Listing 1. Looking at these numbers, it seems that our rather simple (and rather coarse) numerical scheme yields fairly accurate results.

However, the fourth column of Tab. 1 suggests that even better results are possible, if we increase the number of grid points. To produce this column, we worked with $N = 2001$ grid points but otherwise proceeded as in Listing 1. Alas, the resulting gain is minor (improvements in the fourth decimal place) and comes at a hefty price. To obtain the numbers in the third column, we had to spectrally decompose a matrix with 1001^2 entries, to obtain the numbers in the fourth column, we had to work with a matrix about 4 times as big, namely with 2001^2 entries. If we were to continue to double the resolution of our grid, matrix sizes would continue to grow by a factor of four but accuracy improvements would be just minuscule. Since this is not sustainable, we will discuss further and much better numerical schemes in later notes.

[1] For an in-depth explanation as to why this is recommended practice, we refer to our earlier discussion in [1].

4 SUMMARY AND OUTLOOK

In this note, we discussed how to numerically determine eigenstates and eigenenergies of a one-dimensional quantum harmonic oscillator. The simple key idea was to discretize the domain of the position variable into a finite grid of equally spaced points and to use finite differences over this grid to obtain a discretized version of the Hamiltonian of the system. Approximated in terms of this discrete Hamiltonian, the Schrödinger equation for the quantum harmonic oscillator became an equation involving matrices and vectors of finite sizes and the corresponding eigenvalue / eigenvector problem could be solved using standard *NumPy* methods.

While the numerical scheme we discussed in this note is rather coarse and does not scale well to grids of higher resolution, it should ring a bell for people who have a background in machine learning. This is because matrix $\boldsymbol{H}$ in (17) can be recognized as a *weighted graph Laplcacian* (the graph from which it is computed is a line graph of N vertices).

Graph Laplcians play an important role in data mining, network analysis, or computer vision [2, 3, 5, 6, 9, 10] and their spectral decomposition yields valuable insights into the nature of problem under consideration. In a certain sense, the content of this note is thus not far removed from topics familiar to machine learning practitioners.

This is good to know because we will use connections like this in upcoming notes in order to build bridges between the seemingly unrelated areas of machine learning and quantum computing.

ACKNOWLEDGMENTS

This material was produced within the Competence Center for Machine Learning Rhine-Ruhr (**ML2R**) which is funded by the Federal Ministry of Education and Research of Germany (grant no. 01IS18038C). The authors gratefully acknowledge this support.

REFERENCES

[1] C. Bauckhage. 2015. NumPy / SciPy Recipes for Data Science: Eigenvalues / Eigenvectors of Covariance Matrices. researchgate.net. https://dx.doi.org/10.13140/RG.2.1.2307.5046.

[2] C. Bauckhage, R. Sifa, A. Drachen, C. Thurau, and F. Hadiji. 2014. Beyond Heatmaps: Spatio-Temporal Clustering using Behavior-Based Partitioning of Game Levels. In *Proc. Conf. on Computational Intelligence and Games*. IEEE.

[3] I.S. Dhillon, Y. Guan, and B. Kulis. 2004. Kernel k-means, Spectral Clustering and Normalized Cuts. In *Proc. KDD*. ACM.

[4] J.D. Hunter. 2007. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering* 9, 3 (2007).

[5] J. Kunegis, D. Fay, and C. Bauckhage. 2010. Network Growth and the Spectral Evolution Model. In *Proc. CIKM*. ACM.

[6] J. Kunegis, D. Fay, and C. Bauckhage. 2013. Spectral Evolution in Dynamic Networks. *Knowledge and Information Systems* 37, 1 (2013).

[7] T.E. Oliphant. 2007. Python for Scientific Computing. *Computing in Science & Engineering* 9, 3 (2007).

[8] R. Shankar. 1994. *Principles of Quantum Mechanics* (2nd ed.). Springer.

[9] J. Shi and J. Malik. 2000. Normalized Cuts and Image Segmentation. *IEEE Trans. Pattern Analysis and Machine Intelligence* 22, 8 (2000).

[10] U. von Luxburg. 2007. A Tutorial on Spectral Clustering. *Statistics and Computing* 17 (2007).