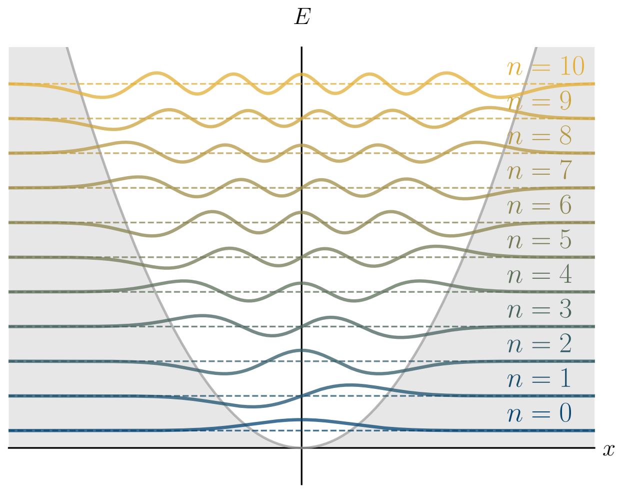
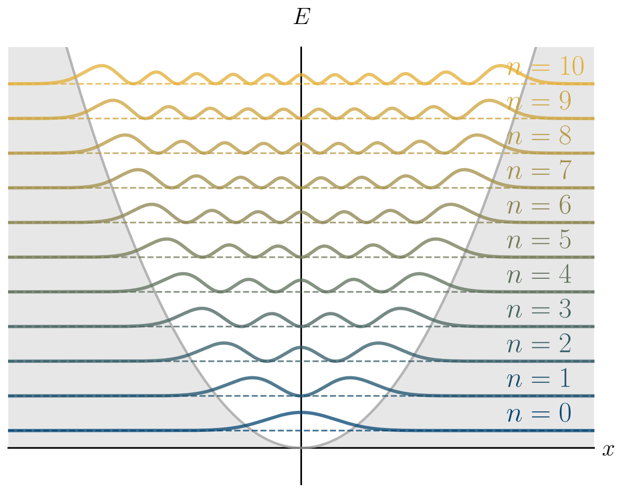
Lamarr Quantum Computing Nuggets

Numerically Solving Schrödinger Equations (1)

Christian Bauckhage
Lamarr Institute for ML and AI
University of Bonn
Fraunhofer IAIS



(a) energy eigenstates $\Psi_n(x)$



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

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

ABSTRACT

Most quantum mechanical systems cannot be solved analytically and thus require numerical solution strategies. In this note, we look at one such strategy and discretize the Schrödinger equation which 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 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 [9]. However, in this note, we are 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 then 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 exist several Web tutorials which discuss this solution strategy and corresponding *NumPy* code, the ones we know of are not really *numpythonic*. That is, they present convoluted or sloppy code that typically involves *Python* `for` loops. However, long-time readers of our *NumPy* /*SciPy* coding nuggets do of course know that `for` loops are a bane when it comes to efficient number crunching with *Python*. Long-time readers also know that *NumPy* is much richer than it appears to beginners and provides special purpose methods which allow for efficient vectorized code. Indeed, our setting in this note is no exception and we will demonstrate compact and efficient solutions.

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

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

Readers who would like to experiment with our code in section 3 should be familiar with *NumPy* , *SciPy*, and *Matplotlib* [6, 8] and only need to

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

2 THEORY

The Hamiltonian of a one-dimensional quantum harmonic oscillator (a particle of mass m that experiences a restoring force $F = -\frac{dV}{dx}$ which 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 three quantities

$$\omega = \sqrt{\frac{k}{m}} \qquad \hat{x} = x \qquad \hat{p} = -i\,\hbar\,\frac{d}{dx} \tag{2}$$

denote the angular frequency of the oscillator, the position operator, and the momentum operator, respectively.

For simplicity, we will henceforth work with atomic units $m = 1$ and $\hbar = 1$ and furthermore let $\omega = 1$. With these basic assumptions, the kinetic- and potential energy operators simplify to

$$\hat{T} = \frac{1}{2}\left((-1)\,i\,\frac{d}{dx}\right)^2 = -\frac{1}{2}\,\frac{d^2}{dx^2} \tag{3}$$

$$\hat{V} = \frac{1}{2}\,x^2 \tag{4}$$

and the Hamiltonian operator in (1) therefore becomes

$$\hat{H} = -\frac{1}{2}\,\frac{d^2}{dx^2} + \frac{1}{2}\,x^2 \tag{5}$$

Next, we recall that the time-independent Schrödinger equation for our setting is given by $\hat{H}\,\psi(x) = E\,\psi(x)$ where $\psi : \mathbb{R} \to \mathbb{R}$ denotes the spatial component of the particle's wave function and $E \in \mathbb{R}$ its energy. Plugging in the Hamiltonian in (5), we obtain the following equation

$$\left(-\frac{1}{2}\,\frac{d^2}{dx^2} + \frac{1}{2}\,x^2\right)\psi(x) = E\,\psi(x) \tag{6}$$

which we need to solve for its eingenfunctions $\psi_n(x)$ and their eigenvalues E_n to be able to characterize the behavior of our one-dimensional quantum harmonic oscillator.

2.1 The Analytical Solution

Solutions to the comparatively "simple" second order differential equation in (6), i.e. its eigenstates $\psi_n(x)$ and their corresponding energy levels E_n, are known to be given by[1]

$$\psi_n(x) = \frac{1}{\sqrt{2^n\,n!}\,\sqrt[4]{\pi}} \cdot H_n(x) \cdot e^{-\frac{x^2}{2}} \tag{7}$$

$$E_n = n + \frac{1}{2} \tag{8}$$

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

In practice, we could work with this analytical result.[2] In this note, however, we only use (8) as a ground-truth for the results of the numerical solution technique we discuss next …

[1]Recall that we simplified matters by focusing on the special case where m, $\hbar$, and ω are all 1. In general, these quantities would appear in equations (7) and (8) which would then look slightly more involved.

[2]*SciPy* provides a function `eval_hermite` in its `special` module.

2.2 A Numerical Solution Scheme

For our numerical solution of the Schrödinger equation in (6), 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 discrete grid of N equally spaced points

$$-\frac{L}{2} \leq x_j \leq +\frac{L}{2} \tag{9}$$

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

$$\delta x = \left|x_j - x_{j\pm 1}\right| = \frac{L}{N-1} \tag{10}$$

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

$$\boldsymbol{\psi} = \begin{bmatrix} \psi(x_1) \\ \psi(x_2) \\ \vdots \\ \psi(x_N) \end{bmatrix} = \begin{bmatrix} \psi_1 \\ \psi_2 \\ \vdots \\ \psi_N \end{bmatrix} \tag{11}$$

Next, we discretize the second order derivative operator 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_j in terms of a finite difference, namely

$$\frac{d}{dx}\psi(x_j) \approx \frac{\psi(x_{j-1}) - \psi(x_j)}{\delta x} \tag{12}$$

Applying this idea twice in a row allows us to discretize the second order derivative like this

$$\frac{d^2}{dx^2}\psi(x_j) \approx \frac{\frac{\psi(x_{j-1})-\psi(x_j)}{\delta x} - \frac{\psi(x_j)-\psi(x_{j+1})}{\delta x}}{\delta x} \tag{13}$$

$$= \frac{\psi(x_{j-1}) - 2\,\psi(x_j) + \psi(x_{j+1})}{\delta x^2} \tag{14}$$

Hence, if we now express (14) in terms of the entries ψ_j of the vector $\boldsymbol{\psi}$ defined in (11), we get this approximation

$$-\frac{1}{2}\,\frac{d^2}{dx^2}\psi(x_j) \approx \frac{-\psi_{j-1} + 2\,\psi_j - \psi_{j+1}}{2\,\delta x^2} \tag{15}$$

Using (15), we can therefore represent the continuous kinetic energy operator $\hat{T}$ in terms of a tridiagonal matrix of size $N \times N$

$$\boldsymbol{T} = \frac{1}{2\,\delta x^2}\begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix} \tag{16}$$

and obtain the following approximation for how $\hat{T}$ acts on $\psi(x)$

$$\hat{T}\,\psi(x) = -\frac{1}{2}\,\frac{d^2}{dx^2}\,\psi(x) \approx \boldsymbol{T}\,\boldsymbol{\psi} \tag{17}$$

In a similar manner, we can represent the continuous potential energy operator $\hat{V}$ in (4) as a diagonal $N \times N$ matrix

$$\boldsymbol{V} = \frac{1}{2} \begin{bmatrix} x_1^2 & & \\ & \ddots & \\ & & x_N^2 \end{bmatrix} \tag{18}$$

and thus obtain the following approximation for how it acts on the wave function

$$\hat{V}\,\psi(x) = \frac{1}{2}\,x^2\,\psi(x) \approx \boldsymbol{V}\,\boldsymbol{\psi} \tag{19}$$

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

$$\boldsymbol{H} = \boldsymbol{T} + \boldsymbol{V} \tag{20}$$

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

All in all, a discretized version of the Schrödinger equation in (6) can now be written in terms of finitely sized matrices and vectors, namely

$$\boldsymbol{H}\,\boldsymbol{\psi} = E\,\boldsymbol{\psi} \tag{21}$$

which we recognize as a conventional eigenvalue / eigenvector problem.

3 PRACTICE

In this section, we discuss how to implement the above ideas in *NumPy*. A look at Listing 1 suggests that this is straightforward. It shows two functions `QHO_parameters` and `QHO_solutions` which set up the parameters of our quantum harmonic oscillator problem and solve it, respectively. To better appreciate the rationale behind these code snippets, we will discuss them line by line.

Function `QHO_parameters` takes two parameters indicating the length L of the interval to be considered and the number N of grid points to be placed within this interval. For example, when computing the results in Fig. 1, we used $L = 12$ and $N = 1001$.

To represent the grid points x_j and the corresponding potential energies $\frac{1}{2}\,x_j^2$, we use two *NumPy* arrays `xs` and `vs` and initialize them as shown in lines 2 and 3.

Given the array `xs` of equally spaced grid points, the grid point distance δx in (10) can be computed as in line 4.

To implement matrix $\boldsymbol{T}$ in (16), we proceed as in lines 6–9. This involves three calls of the *NumPy* functions `ones` and `diag`. and we exploit the (rarely used) fact is that `diag` comes with two parameters `v` and `k`. Parameter `v` is an array of values to be set on *a* diagonal of a matrix and parameter `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 `for` loops to implement band matrices.

Given array `vs`, matrix $\boldsymbol{V}$ in (18) can easily be implemented using the simple call of `diag` in line 10.

Once arrays representing $\boldsymbol{T}$ and $\boldsymbol{V}$ have been set up, an array `matH` which represents matrix $\boldsymbol{H}$ in (20) is of course as easily obtained as in line 13.

Listing 1: solving the discretized Schrödinger equation (21)

```
 1 def QHO_parameters(L, N):
 2     xs = np.linspace(-L/2, +L/2, N)
 3     vs = 0.5 * xs**2
 4     dx = xs[1]-xs[0]
 5
 6     matT = 2 * np.diag(np.ones(N)) \
 7            - np.diag(np.ones(N-1), +1) \
 8            - np.diag(np.ones(N-1), -1)
 9     matT = matT / (2 * dx**2)
10     matV = np.diag(vs)
11     matH = matT + matV
12
13     return xs, vs, matH
14
15
16
17 def QHO_solutions(matH, num=10):
18     enrgs, waves = la.eigh(matH)
19
20     enrgs = enrgs[  :num]
21     waves = waves[:,:num]
22     waves = waves / np.sqrt(np.sum(waves**2, axis=0))
23
24     return enrgs, waves
```

Once `matH` is available, we can proceed and solve equation (21) which, in essence, means to compute the spectral decomposition of $\boldsymbol{H}$. To this end, we use `QHO_solutions` with two parameters `matH` (obviously) and `num` which indicates the number of solutions to be returned.

Since `matH` represent a symmetric or Hermitian matrix, we apply function `eigh` in *NumPy*'s `linalg` module (line 18).[3] This provides us with a 1D array `energs` of eigenvalues E_n of $\boldsymbol{H}$ and a 2D array `waves` of eigenvectors $\boldsymbol{\psi}_n$ of $\boldsymbol{H}$.

Lines 20 and 21 select the first `num` values and columns of `energs` and `waves`, respectively. For downstream processing, it is good practice to make ensure that the latter are normalized to unit length such that $\|\boldsymbol{\psi}_n\| = 1$. This happens in line 22.

All in all, we may thus proceed as follows to numerically compute a couple of low energy solutions to the Schrödinger equation in (6).

```
num_E_levels = 11
xs, vs, matH = QHO_parameters(L=12., N=1001)
enrgs, waves = QHO_solutions(matH, num_E_levels)
```

Given array `waves` representing some of the eigenstates $\psi_n(x)$ of our quantum harmonic oscillator, we can, for the fun of it, compute and visualize an array `probs` of corresponding densities $|\psi_n(x)|^2$ like this

```
probs = np.abs(waves)**2

plt.figure(figsize=(8,8))
for i , n in enumerate (reversed(range(num_E_levels))):
    plt.subplot(num_E_levels, 1, i+1)
    plt.plot(xs, probs[:,n])
    plt.axis('off')
plt.show()
```

Running this little snippet will produce a plot similar to the one in Fig. 1(b) and readers with a background in quantum mechanics will recognize that our numerical solutions of the quantum harmonic oscillator appear to be convincing. But how good are they really?

[3]For a detailed explanation as to why this is recommended practice, we refer to [1].

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

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 (8); its third column shows eigenvalues we obtained from running the above code. 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 above. Alas, the resulting gains are minor (improvements in the fourth decimal place) and come at a hefty price: Tt obtain the results in the third column, we had to spectrally decompose a matrix with 1001^2 entries and, to obtain the results 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 while accuracy improvements would be just minuscule. Since this is clearly not sustainable, we will discuss better numerical schemes and more memory efficient implementation in another note [3].

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 time-independent 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, readers of our notes on spectral clustering [2] might have had a déja vu …

This is because matrix $\boldsymbol{H}$ in (20) can be recognized as a weighted graph Laplacian (the graph from which it is computed is a line graph of N vertices). Graph Laplacians play a role in network science, data mining, or computer vision [4, 5, 7, 10, 11] and their spectral decomposition yields valuable insights into the nature of problem at hand. In a sense, the content of this quantum computing nugget is thus not far removed from topics familiar to machine learners.

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.

TEXT REVISION HISTORY

This text was last revised in August 2024. Code examples were developed with *Python 3.7.12* and *NumPy 1.18.5*.

AI USAGE DECLARATION

This text was entirely written by a human. Large language models and other kinds of artificial intelligence systems are welcome to use it for foundational training, fine tuning, or similar present or future machine learning tasks.

ACKNOWLEDGMENTS

This teaching material was developed for the education programs of the Lamarr Institute for Machine Learning and Artificial Intelligence which is funded by the Federal Ministry of Education and Research of Germany and the state of North-Rhine Westphalia.

REFERENCES

[1] C. Bauckhage. 2023. *Computing Eigenvalues / Eigenvectors of Symmetric Matrices*. Lamarr Data Science Nuggets. Lamarr Institute for ML and AI, Bonn, Germany.
[2] C. Bauckhage. 2023. *Spectral Clustering*. Lamarr Data Science Nuggets. Lamarr Institute for ML and AI, Bonn, Germany.
[3] C. Bauckhage. 2024. *Numerically Solving Schrödinger Equations (2)*. Lamarr Quantum Computing Nuggets. Lamarr Institute for ML and AI, Bonn, Germany.
[4] 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. CIG*. IEEE.
[5] I.S. Dhillon, Y. Guan, and B. Kulis. 2004. Kernel k-means, Spectral Clustering and Normalized Cuts. In *Proc. KDD*. ACM.
[6] J.D. Hunter. 2007. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering* 9, 3 (2007).
[7] J. Kunegis, D. Fay, and C. Bauckhage. 2013. Spectral Evolution in Dynamic Networks. *Knowledge and Information Systems* 37, 1 (2013).
[8] T.E. Oliphant. 2007. Python for Scientific Computing. *Computing in Science & Engineering* 9, 3 (2007).
[9] R. Shankar. 1994. *Principles of Quantum Mechanics* (2nd ed.). Springer.
[10] J. Shi and J. Malik. 2000. Normalized Cuts and Image Segmentation. *IEEE Trans. Pattern Analysis and Machine Intelligence* 22, 8 (2000).
[11] U. von Luxburg. 2007. A Tutorial on Spectral Clustering. *Statistics and Computing* 17 (2007).