2026-06

Application of the bootstrap method for solving simple quantum mechanical systems: a study of effectiveness

Del, Mollika Rani

Independent University, Bangladesh

Independent University Bangladesh

Department of Physical Sciences

Application of the Bootstrap Method for Solving Simple Quantum Mechanical Systems:

A Study of Effectiveness

PHY499 – Project in Physics

A thesis submitted in partial fulfillment of the

requirements for the degree of

Bachelor of Science in Physics

by

Mollika Rani Del

Student ID: 2010224

Supervisor:

M. Arshad Momen

Professor

Department of Physical Sciences

June 2026

Declaration

I, Mollika Rani Del, hereby declare that the thesis titled *Application of the Bootstrap Method for Solving Simple Quantum Mechanical Systems: A Study of Effectiveness* is my own original work, carried out under the supervision of Professor M. Arshad Momen, Department of Physical Sciences, Independent University, Bangladesh.

I further declare that this work, in whole or in part, has not been previously submitted for any degree or diploma at this or any other institution. No other sources or learning aids, other than those listed in this thesis, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references to all such work.

Signature of the student: _____

Certification of The Report

The report titled *Application of the Bootstrap Method for Solving Simple Quantum Mechanical Systems: A Study of Effectiveness* submitted by **Mollika Rani Del** (Student ID: 2010224) has been accepted as satisfactory in partial fulfillment of the requirements for the degree of BSc (Hons.) in Physics on 2nd July 2026.

Members of the Assessment Committee:

M. Arshad Momen, PhD _____
Dept. of Physical Sciences
Supervisor

Asma Begum, PhD _____
Dept. of Physical Sciences
Member

Jewel Kumar Ghosh, PhD _____
Dept. of Physical Sciences
Member

Rifat Ara Rouf, PhD _____
Dept. of Physical Sciences
Head of the Department

Approval

This is to certify that the thesis titled *Application of the Bootstrap Method for Solving Simple Quantum Mechanical Systems: A Study of Effectiveness*, submitted by Mollika Rani Del (Student ID: 2010224), has been examined and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Physics, Department of Physical Sciences, Independent University, Bangladesh.

Signature of the Supervisor: _____

Signature of the Head of the Department: _____

Signature of the Director, GSRIR: _____

Abstract

This thesis evaluates the effectiveness of the quantum mechanical bootstrap by applying it to two fundamental systems with known analytical solutions: the simple harmonic oscillator and the hydrogen atom. The bootstrap method is a consistency-based approach for determining the energy spectra of quantum mechanical systems without explicitly solving the Schrödinger equation.

The methodology involves deriving moment recursion relations from the Hamiltonian and canonical commutation relations, then constructing Hankel matrices from these moments. Physical energy eigenvalues are identified by imposing positivity constraints—requiring that these matrices be positive semi-definite, which ensures the moments correspond to a valid quantum state. Trial energies that violate this condition are systematically excluded, and the allowed regions converge toward the true spectrum as the matrix size increases.

Our results demonstrate that the bootstrap method successfully reproduces the known energy spectra for both systems. The findings confirm that the bootstrap provides a reliable alternative approach for spectral determination in quantum mechanics, suggesting its potential applicability to more complex systems where analytical solutions are unavailable.

Acknowledgments

First and foremost, I would like to thank my supervisor, Professor M. Arshad Momen, for his guidance and support throughout this project. His insights helped shape this work and kept me on the right track. I would like to acknowledge the Center for Computational and Data Sciences (CCDS) for providing the computational resources that made this work possible.

I am deeply grateful to my parents. Without their sacrifices and belief in me, I would not have made it this far. Everything I have achieved is because of them. I also want to thank my friends who stood by me during this journey.Their support meant more than they know. A special thanks to Proshanto for helping me with some of the simulations. His assistance saved me a lot of time and frustration.

Finally, I thank everyone who contributed to my education and growth over the years. This thesis is the result of many hands, not just my own.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Bootstrap method is a consistency-based method used to determine the energy spectra of some quantum mechanical systems without the need of explicitly solving the Schrödinger equation. In this method, we first derive the algebraic relation between moments of observables from the Hamiltonian and commutation relations, then apply some constraints to exclude unphysical solutions and converge to the allowed spectrum.

The bootstrap method has had a significant journey through physics. It first appeared in the late 1950's when Geoffrey Chew and his colleagues at Berkeley were struggling with a problem where particle accelerators kept discovering new particles called resonances. Such as mesons, baryons and more, creating what physicists called a "particle zoo".[2, 3]. The big question was which particles were truly fundamental and which were composite. Chew proposed what if no particle was more fundamental than any other? What if the entire system determined itself through pure self-consistency? [4].

This idea led to the S-matrix (scattering matrix) bootstrap approach in the 1960s [5]. Instead of solving differential equations or assuming fundamental building blocks, Chew's method imposed consistency conditions such as crossing symmetry, unitarity, and analyticity on particle scattering processes. The approach had early successes, including the prediction of the mass of the rho meson [6].

But by the 1970s, the bootstrap had faded from view. The discovery of quarks and the development of quantum chromodynamics (QCD) provided a simpler, more successful framework for understanding strong interactions [7]. But over the last decade, the bootstrap made a comeback in conformal field theory. People turned basic consistency ideas (like crossing and unitarity) into practical numerical constraints and got very sharp results [8, 9]. Similar ideas have been used in large N matrix models and matrix quantum mechanics to bound spectra without solving everything explicitly [10, 11]. Berenstein and Hulsey extended the bootstrap method to the Harmonic Oscillator and the Hydrogen atom, which

involve potentials in the form of a polynomial [11]. These studies made bootstrapping quantum mechanics a promising area of research because this approach might be useful in determining the spectrum of quantum systems that are otherwise challenging to solve analytically.

The method has been applied to Harmonic Oscillators [11], Anharmonic Oscillators [10], and Double-Well Potentials [12]. It has been also applied to solve for the spectrum of more complex 1-Dimensional Schrödinger equations such as periodic potentials [13, 14], Calabi-Yau model [15], Pöschl-Teller potential [16], and many more complex systems.

In this report, we will follow the work of Berenstein and Hulsey [11] and apply the bootstrap method to the Simple Harmonic Oscillator (SHO) and the Hydrogen atom. Our aim is to test the effectiveness of the bootstrap method to determine the energy spectrum of One-Dimensional Hamiltonians in these systems.

1.1 Research Objectives

The main objective is to test and evaluate the quantum mechanical bootstrap method.

- Assess how well the bootstrap approach works for determining the energy spectrum of one-dimensional Hamiltonians by applying it to problems with known analytical solutions.

- Compare the results obtained from the bootstrap method with the known analytical predictions for the hydrogen atom (Coulomb potential) and the simple harmonic oscillator.

- Characterize the accuracy, precision, and convergence properties of the bootstrap method for these specific quantum mechanical systems.

- Determine the effectiveness of the method as a potential tool for solving other, non-analytically solvable quantum mechanical problems.

1.2 Description of the Report

In this report, we introduce the quantum mechanical bootstrap method and show its effectiveness by applying it to two fundamental quantum systems. In **chapter 2**, we establish the mathematical and conceptual framework underlying the bootstrap approach. First, we introduce the key operator identities that form the backbone of the method. We then derive the moment recursion relation from the Hamiltonian and canonical commutation relation, which shows how higher-order moments can be generated from lower-order ones for any trial energy. Following that, we discuss the Hankel matrix structure and the positivity constraints that distinguish physical quantum states from unphysical ones. We end the chapter with the algorithmic structure of the bootstrap method and some important numerical considerations.

In chapters 3 and 4, we present the applications of the bootstrap method to two exactly solvable quantum systems: the Simple Harmonic Oscillator (SHO) and the Hydrogen atom (Coulomb potential). In **chapter 3**, we focus on the SHO where the moment recursion closes with only the energy as a free parameter. We derive the specific recursion relation for SHO moments, construct the Hankel matrix, and analyse how positivity constraints on even and odd sub-matrices progressively restrict the allowed energy values as the matrix size increases.

In chapter 4, we extend the method to the hydrogen atom which is a system where moments grow rapidly and the Coulomb singularity introduces numerical subtleties. We present results across multiple angular momentum quantum numbers and demonstrate that the bootstrap successfully reproduces the Bohr-Balmer spectrum with high precision. The width convergence plots also validate the method's effectiveness by showing exponential convergence with respect to Hankel matrix depth.

In **chapter 5**, we summarize the key findings from both applications, and discuss the convergence behaviour and numerical accuracy. We see why the bootstrap method is valuable. It reconstructs quantum spectra from algebraic consistency conditions rather than solving differential equations directly, which makes it suitable for systems where analytical solutions are unavailable. We acknowledge the numerical limitations while confirming that the bootstrap provides a powerful and systematic framework for determining quantum energy levels.

Chapter 2

Theoretical Foundation

In this chapter, we develop the theoretical foundation of the quantum mechanical bootstrap method used throughout this report. The main idea is to replace direct solution of the Schrödinger equation with a consistency-based formulation in terms of operator identities, moment relations, and positivity conditions.

We begin by deriving the moment recursion relations that connect expectation values of different order moments for any trial energy. These relations generate moment sequences, but additional constraints are required for physical validity . To impose those constraints, we introduce Hankel matrices constructed from moments and require positive semi-definiteness, which ensures compatibility with a normalizable quantum state. We then present the algorithmic structure used in computation and discuss practical numerical issues.

2.1 Moment Recursion

Moment recursion is a core component of the quantum mechanical bootstrap method, which helps in determining the energy spectrum by generating higher-order moments of position operators from lower-order ones and energy guesses. The moments are of the form $\langle x^n \rangle$.

The bootstrap approach generates a sequence of these moments recursively using operator identities satisfied by all energy eigenstates. The foundation of the method lies in two identities:

- $\mathcal{O}$ is a suitable operator (typically a monomial in x and p), which is time-independent.

- H is the Hamiltonian as such

$$H = \frac{p^2}{2m} + V(x)$$

- E is the energy
- $\langle O \rangle$ is the expectation value of the observable O in $|\psi\rangle$.
- $\langle HO \rangle$ is the expectation value of the product between the operator and the Hamiltonian $H \cdot O$ in $|\psi\rangle$
- $\langle [H, O] \rangle$ is the expectation value of the commutator $HO - OH$ in $|\psi\rangle$

$$\langle [H, \mathcal{O}] \rangle = 0 \tag{2.1}$$

$$\langle H\mathcal{O} \rangle = E\langle \mathcal{O} \rangle \tag{2.2}$$

2.2 Fundamental Operator Identities

2.2.1 Vanishing Commutator Identity

$$\langle [H, \mathcal{O}] \rangle = 0 \tag{2.3}$$

Derivation: Consider an energy eigenstate $|\psi\rangle$:

$$H|\psi\rangle = E|\psi\rangle$$

The expectation value of the commutator $[H, \mathcal{O}]$ in $|\psi\rangle$ is:

$$\langle [H, \mathcal{O}] \rangle = \langle \psi | H\mathcal{O} - \mathcal{O}H | \psi \rangle = \langle \psi | H\mathcal{O} | \psi \rangle - \langle \psi | \mathcal{O}H | \psi \rangle$$

Since $H|\psi\rangle = E|\psi\rangle$,

$$\langle \psi | H\mathcal{O} | \psi \rangle = \langle \psi | H(\mathcal{O}|\psi\rangle) \rangle = \langle H\psi | \mathcal{O} | \psi \rangle = E\langle \psi | \mathcal{O} | \psi \rangle$$

Similarly,

$$\langle \psi | \mathcal{O}H | \psi \rangle = \langle \psi | \mathcal{O}(H|\psi\rangle) \rangle = \langle \psi | \mathcal{O}(E|\psi\rangle) \rangle = E\langle \psi | \mathcal{O} | \psi \rangle$$

Therefore,

$$\langle [H, \mathcal{O}] \rangle = E\langle \psi | \mathcal{O} | \psi \rangle - E\langle \psi | \mathcal{O} | \psi \rangle = 0$$

2.2.2 Eigenvalue Identity

$$\langle H\mathcal{O} \rangle = E\langle \mathcal{O} \rangle \tag{2.4}$$

Derivation: For $|\psi\rangle$ such that $H|\psi\rangle = E|\psi\rangle$, we have:

$$\langle H\mathcal{O} \rangle = \langle \psi | H\mathcal{O} | \psi \rangle$$

$$\langle \psi | H\mathcal{O} | \psi \rangle = \langle H\psi | \mathcal{O} | \psi \rangle = E\langle \psi | \mathcal{O} | \psi \rangle = E\langle \mathcal{O} \rangle$$

Thus,

$$\langle H\mathcal{O} \rangle = E\langle \mathcal{O} \rangle$$

2.3 Moment Recursion Relation

By applying the above identities with appropriate choices of $\mathcal{O}$ (e.g x^s, $x^m p$), we can obtain recursion relations among the moments $\langle x^n \rangle$. For the systems considered in this work, the recursions are closed and depend only on E and other conserved quantum quantities.

First Relation

We start by choosing $O = x^s$ and computing the commutator $[H, x^s]$.

Given the Hamiltonian $H = \frac{p^2}{2} + V(x)$ (with $M = 1$):

The commutator $[H, x^s]$ splits as $[\frac{p^2}{2}, x^s] + [V(x), x^s]$

$[V(x), x^s] = 0$ Since $V(x)$ is a function of x, it commutes with x^s.

Recall the canonical commutation relation: $[x, p] = i$

Using the identities $[A^2, B] = A[A, B] + [A, B]A$ and if $[[A, B], B] = 0$ then $[A, B^n] = nB^{n-1}[A, B]$, we get:

$$[p^2, x^s] = p[p, x^s] + [p, x^s]p \tag{2.5}$$
$$= p(-is \cdot x^{s-1}) + (-is \cdot x^{s-1})p \tag{2.6}$$
$$= -is \cdot (p \cdot x^{s-1} + x^{s-1} \cdot p) \tag{2.7}$$
$$= -is \cdot (x^{s-1} \cdot p - i(s-1)x^{s-2} + x^{s-1} \cdot p) \tag{2.8}$$
$$= -is \cdot (2x^{s-1} \cdot p - i(s-1)x^{s-2}) \tag{2.9}$$
$$= -2is \cdot x^{s-1} \cdot p - s(s-1)x^{s-2}. \tag{2.10}$$

So,

$$[H, x^s] = [\frac{p^2}{2}, x^s] = -is \cdot x^{s-1} \cdot p - \frac{s(s-1)}{2}x^{s-2} \tag{2.11}$$

$$\langle [H, x^s] \rangle = 0 = \left\langle -is \cdot x^{s-1} \cdot p - \frac{s(s-1)}{2}x^{s-2} \right\rangle \tag{2.12}$$

$$s\langle x^{s-1}p \rangle = \frac{i}{2}s(s-1)\langle x^{s-2} \rangle \tag{2.13}$$

Second Relation

We choose $O = x^m p$ and apply $\langle [H, x^m p] \rangle = 0$:

$$[H, x^m p] = [\frac{p^2}{2}, x^m p] + [V(x), x^m p] \tag{2.14}$$

Using the product rule $[A, BC] = [A, B]C + B[A, C]$:

$$[\frac{p^2}{2}, x^m p] = [\frac{p^2}{2}, x^m]p + x^m[\frac{p^2}{2}, p] \tag{2.15}$$

$[\frac{p^2}{2}, p] = 0$ since $[p, p] = 0$

Using the result in (eq 2.11):

$$[\frac{p^2}{2}, x^m] = \left(-im \cdot x^{m-1} \cdot p - \frac{m(m-1)}{2}x^{m-2}\right)p \tag{2.16}$$
$$= -im \cdot x^{m-1} \cdot p^2 - \frac{m(m-1)}{2}x^{m-2}p \tag{2.17}$$

$$[V(x), x^m p] = [V(x), x^m]p + x^m[V(x), p]$$

$[V(x), x^m] = 0$ (functions of x commute)

using the identity $[f(x), p] = -if'(x)$: $[V(x), p] = -iV'(x)$

Therefore,

$$[V(x), x^m p] = x^m(-iV'(x)) = -ix^m V'(x) \tag{2.18}$$

So,

$$[H, x^m p] = -im \cdot x^{m-1} \cdot p^2 - \frac{m(m-1)}{2} x^{m-2} p - ix^m V'(x) \tag{2.19}$$

$$0 = \langle [H, x^m p] \rangle \tag{2.20}$$

$$0 = \left\langle -im \cdot x^{m-1} \cdot p^2 - \frac{m(m-1)}{2} x^{m-2} p - ix^m V'(x) \right\rangle \tag{2.21}$$

$$= -im\langle x^{m-1} p^2 \rangle - \frac{m(m-1)}{2} \langle x^{m-2} p \rangle - i\langle x^m V'(x) \rangle \tag{2.22}$$

$$m\langle x^{m-1} p^2 \rangle = -\frac{m(m-1)}{2} \cdot i\langle x^{m-2} p \rangle - \langle x^m V'(x) \rangle \tag{2.23}$$

$$= -\frac{m(m-1)}{2} \cdot i \cdot \frac{i(m-2)}{2} \langle x^{m-3} \rangle - \langle x^m V'(x) \rangle \tag{2.24}$$

$$= -\frac{m(m-1)}{2} \cdot \frac{i^2(m-2)}{2} \langle x^{m-3} \rangle - \langle x^m V'(x) \rangle \tag{2.25}$$

$$= -\frac{m(m-1)}{2} \cdot \frac{-1 \cdot (m-2)}{2} \langle x^{m-3} \rangle - \langle x^m V'(x) \rangle \tag{2.26}$$

$$= \frac{m(m-1)(m-2)}{4} \langle x^{m-3} \rangle - \langle x^m V'(x) \rangle \tag{2.27}$$

$$0 = m\langle x^{m-1} p^2 \rangle + \frac{m(m-1)(m-2)}{4} \langle x^{m-3} \rangle - \langle x^m V'(x) \rangle \tag{2.28}$$

Using the Second Identity with $O = x^{m-1}$

Now we apply $\langle HO \rangle = E\langle O \rangle$ with $O = x^{m-1}$:

$$\langle Hx^{m-1}\rangle = \left\langle \left(\frac{p^2}{2} + V(x)\right) x^{m-1} \right\rangle = \left\langle \frac{p^2}{2} x^{m-1} \right\rangle + \langle V(x) x^{m-1}\rangle \tag{2.29}$$

For $\langle \frac{p^2}{2} x^{m-1}\rangle$, we must be careful about operator ordering. In an expectation value within an eigenstate:

$$\left\langle \frac{p^2}{2} x^{m-1} \right\rangle = \frac{1}{2}\langle x^{m-1} p^2\rangle \tag{2.30}$$

$$\langle Hx^{m-1}\rangle = E\langle x^{m-1}\rangle = \frac{1}{2}\langle x^{m-1}p^2\rangle + \langle x^{m-1}V(x)\rangle \tag{2.31}$$

Eliminating Mixed Operators to Obtain the Final Recursion Relation

Multiplying eq. (2.31) by $2m$ and substituting the equation (2.27) in it, we get:

$$2mE\langle x^{m-1}\rangle = -\frac{m(m-1)(m-2)}{4}\langle x^{m-3}\rangle + \langle x^m V'(x)\rangle + 2m\langle x^{m-1}V(x)\rangle \tag{2.32}$$

$$0 = 2mE\langle x^{m-1}\rangle + \frac{m(m-1)(m-2)}{4}\langle x^{m-3}\rangle - \langle x^m V'(x)\rangle - 2m\langle x^{m-1}V(x)\rangle \tag{2.33}$$

This is the main recursion relation that captures the dynamics of the Hamiltonian.

Special Case: Virial Theorem

For $m = 1$, the relation becomes:

$$0 = 2E\langle x^0\rangle + \frac{1(0)(-1)}{4}\langle x^{-2}\rangle - \langle xV'(x)\rangle - 2\langle x^0 V(x)\rangle \tag{2.34}$$

Since $\langle x^0\rangle = 1$ and the second term is zero:

$$0 = 2E - \langle xV'(x)\rangle - 2\langle V(x)\rangle \tag{2.35}$$

$$2E = \langle xV'(x)\rangle + 2\langle V(x)\rangle \tag{2.36}$$

$$E = \frac{1}{2}\langle xV'(x)\rangle + \langle V(x)\rangle \tag{2.37}$$

This is the virial theorem.

The power of this approach is that it creates a recursion relation connecting different moments $\langle x^n\rangle$ for a given energy eigenstate, allowing the bootstrap method to determine the energy spectrum and moments without direct solution of the Schrödinger equation.

2.4 Positivity Constraints and Hankel Matrix

The recursion relation can generate moment sequences for any trial energy, but not all such sequences correspond to physical quantum states. This is because the recursion is derived purely from operator algebra, so it provides a formal set of moments even when the chosen energy does not belong to the true spectrum. However, a physical state must be represented by a normalizable wavefunction, and therefore its moments must arise from a positive probability distribution on configuration space. Similarly, the moment sequence must satisfy positivity constraints such as positive semi-definiteness of the Hankel matrix. Trial energies that fail these constraints may solve the recursion algebraically, but they cannot be associated with any valid quantum state.

This physical constraint translates into a mathematical condition through a simple observation. Consider any polynomial test function $\mathcal{O} = \sum_n c_n x^n$ where the coefficients $c_n \in \mathbb{C}$. The expectation of the products of the operators $\hat{O}$ and $\hat{O}^\dagger$ generate all the elements of the bootstrap matrix M, such that:

$$\langle \mathcal{O}^\dagger \mathcal{O}\rangle = \sum_{i,j} c_i^* c_j \langle x^{i+j}\rangle. \tag{2.38}$$

We notice that this expression is a norm in Hilbert space

$$\langle \mathcal{O}^\dagger \mathcal{O}\rangle \geq 0. \tag{2.39}$$

A proof of this is as follows: Let us suppose that $|\psi\rangle$ denotes the eigenstates of $\hat{O}$ with the eigenvalues λ,

$$\hat{O}|\psi\rangle = \lambda|\psi\rangle.$$

Since the operator $\hat{O}$ is not necessarily Hermitian, $\lambda \in \mathbb{C}$. If we evaluate the identity as the definition of an expectation value, we obtain

$$\langle\psi|\hat{O}^\dagger\hat{O}|\psi\rangle = \lambda^*\lambda = |\lambda|^2 \geq 0.$$

This shows that the eigenvalues of the operator $\hat{O}^\dagger\hat{O}$ are non-negative, which implies that the matrix is positive semi-definite.

This quadratic form defines a matrix with elements $(M_K)_{ij} = \langle x^{i+j}\rangle$ for $0 \leq i, j \leq K-1$. Such matrices, having constant entries along anti-diagonals, are called Hankel matrices. The general form of the Hankel matrix is:

$$M_K = \begin{pmatrix} a_0 & a_1 & a_2 & \cdots & a_{K-1} \\ a_1 & a_2 & a_3 & \cdots & a_K \\ a_2 & a_3 & a_4 & \cdots & a_{K+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{K-1} & a_K & a_{K+1} & \cdots & a_{2K-2} \end{pmatrix} \tag{2.40}$$

The positivity requirement is as follows: M_K must be positive semi-definite for all truncation sizes K. As we increase the matrix size K, we impose progressively stronger constraints. Each larger Hankel matrix contains the previous one as a principal submatrix, so the allowed parameter regions can only shrink, never expand. This monotonic convergence guarantees that excluded points stay excluded—a crucial feature for computational reliability.

Hankel matrices and their connection to positive measures were studied in depth by Hamburger and Stieltjes. The classical Hamburger moment problem states that a sequence of moments $\{a_n\}$ corresponds to a positive measure μ if all Hankel matrices M_K are positive semi-definite up to rank K. In the context of quantum mechanics, the positive measure μ corresponds to the probability density $|\psi(x)|^2$ on configuration space. A moment sequence is therefore realizable by a quantum state if and only if there exists a normalizable, non-negative probability distribution whose moments match the sequence. It was first shown on the half line $\mathbb{R}^+$ by Steiltjes and later extended to the full line $\mathbb{R}$ by Hamburger. [17] The distinction between the half-line (Stieltjes, for $r \in [0, \infty)$) and full-line (Hamburger, for $x \in (-\infty, \infty)$) cases is crucial for applications. The harmonic oscillator moments exist on the full real line and invoke Hamburger's result, while the hydrogen atom involves

radial moments on the half-line, requiring Stieltjes theory. The full-line case is generally more restrictive, fewer moment sequences qualify because negativity on either side of the origin must be accounted for.

2.5 Algorithmic Structure

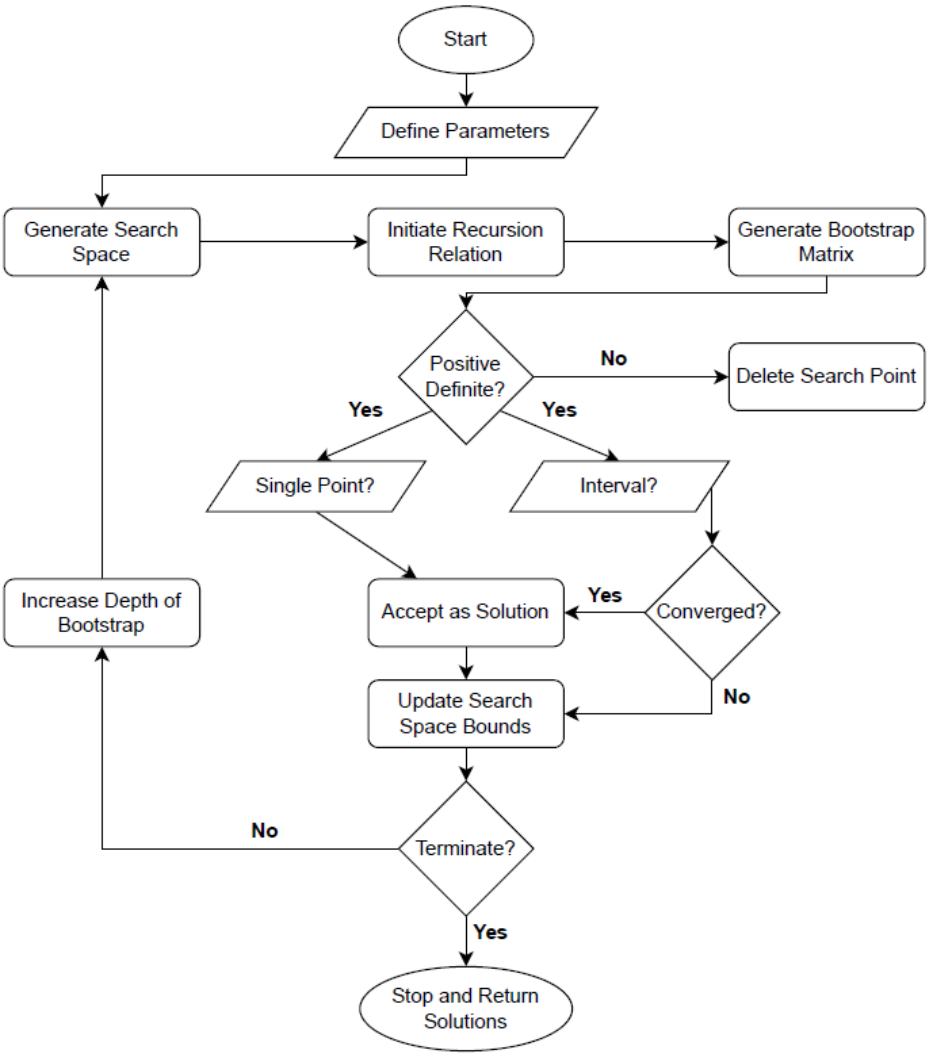

Figure 2.1: Bootstrap Algorithm Flowchart [1]

The bootstrap algorithm proceeds as follows:

1. **Grid the Search Space:** Choose a range of trial energies (and any additional parameters if required).
2. **Generate Moments:** For each trial energy, use the recursion relation to generate a finite sequence of moments.
3. **Construct the Hankel Matrix:** Build the $K \times K$ Hankel matrix from the first $2K - 2$ moments.
4. **Positivity Test:** Check if the Hankel matrix is positive definite. If not, discard the trial energy.
5. **Iterate at Higher Depth:** Increase K and repeat the process, further narrowing down the allowed energy regions.
6. **Convergence:** As K increases, the allowed regions in energy space shrink and converge toward the true quantum energy levels.

2.6 Numerical Considerations

- **Grid Resolution:** The energy search grid must be fine enough to resolve narrow allowed intervals.
- **Numerical Precision:** Moments can grow rapidly (factorially for the hydrogen atom), requiring high numerical precision to avoid false positives or negatives in the positivity test.
- **Rescaling:** To mitigate numerical instability, the Hankel matrix may be rescaled by the moments themselves without affecting positivity.

2.7 Initialization and Search Space

- **Search Space (S)**: The minimal set of initial parameters needed to start the recursion. Its dimension depends on the potential:

Potential	**Search Space** S	**Reason**
Harmonic Oscillator	$\{E\}$	$\langle x^2 \rangle = E$ from symmetry and recursion
Coulomb/Hydrogen	$\{E\}$	Virial theorem gives $\langle r^{-1} \rangle = -2E$
Cubic (gx^3)	$\{E, \langle x \rangle, \langle x^2 \rangle\}$	Symmetry-breaking terms require more data

Table 2.1: Minimal search space for different quantum potentials and the reasoning for each case.

- **Normalization:** $\langle x^0 \rangle = 1$ (wavefunction normalization)

Chapter 3

Harmonic Oscillator

The Simple Harmonic Oscillator (SHO) system, which is the simplest example among the systems we study, is defined in 1D Cartesian coordinates in the domain $\mathbb{R}$ with a search space containing only a single element, namely the energy. The SHO potential is given as:

$$V(x) = \frac{1}{2}mw^2x^2$$

where, m is the mass of the particle w is the angular frequency of the oscillator and x is the position coordinate.

For simplicity in the energy eigenvalues, we set $w = 1$ and $m = 1$. So, the potential energy becomes:

$$V(x) = \frac{1}{2}x^2$$

This potential describes a parabolic well ceneterd at $x = 0$, which is symmetric about the origin.

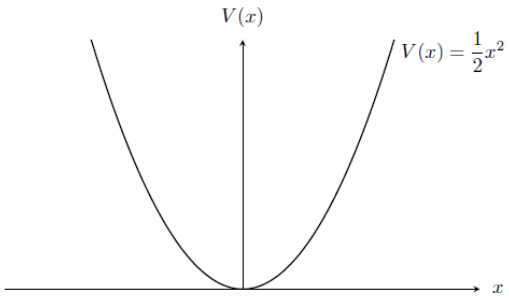

Figure 3.1: Graph of SHO potential

Figure 3.1 shows the graph of the function $V(x)$ in the configuration space. From the figure it is obvious that there are infinitely many bound states for the SHO system which

all have an energy $E \geq 0$.

3.1 Bound states

A bound state is a quantum state of a particle where:

- The particle is localized in space, meaning its wavefunction is concentrated in a finite region
- The particle cannot escape to infinity, it is confined by the potential well.
- The energy of the particle is less than the potential energy at infinity.

In mathematical terms, for a bound state, the wavefunction $\psi(x)$ is normalizable:

$$\int_{-\infty}^{\infty} |\psi(x)|^2 dx < \infty$$

which means the probability of finding a particle at ∞ is 0

Infinitely many bound states

Potentials such as SHO, extend to inifnity and typically support infinitely many bound states because the particle is always confined no matter how high its energy is.

3.2 The SHO Moment Recursion

We start with the 1D harmonic oscillator Hamiltonian (in units $m = 1$, $\omega = 1$, $\hbar = 1$)

$$H = \frac{p^2}{2} + \frac{1}{2}x^2 \tag{3.1}$$

and use the general operator identity valid in any energy eigenstate $|\psi\rangle$:

$$\langle [H, O] \rangle = 0 \tag{3.2}$$

We choose $O = x^s$. The commutator expands as:

$$[H, x^s] = \left[\frac{p^2}{2}, x^s\right] + \left[\frac{1}{2}x^2, x^s\right] \tag{3.3}$$

x^s commutes with x^2, so the second term vanishes:

$$[H, x^s] = \frac{1}{2}[p^2, x^s] \tag{3.4}$$

Thus, we need to compute $[p^2, x^s]$.

Recall the canonical commutation relation:

$$[p, x] = -i \tag{3.5}$$

and more generally,

$$[p, f(x)] = -i\frac{d}{dx}f(x) \tag{3.6}$$

So,

$$[p, x^s] = -isx^{s-1} \tag{3.7}$$

Now,

$$[p^2, x^s] = p[p, x^s] + [p, x^s]p \tag{3.8}$$
$$= p(-isx^{s-1}) + (-isx^{s-1})p \tag{3.9}$$
$$= -is(px^{s-1} + x^{s-1}p) \tag{3.10}$$

Expanding px^{s-1} using the same commutator:

$$[p, x^{s-1}] = -i(s-1)x^{s-2} \tag{3.11}$$

$$px^{s-1} = x^{s-1}p - i(s-1)x^{s-2} \tag{3.12}$$

Therefore,

$$[p^2, x^s] = -is[(x^{s-1}p - i(s-1)x^{s-2}) + x^{s-1}p] \tag{3.13}$$
$$= -is(2x^{s-1}p - i(s-1)x^{s-2}) \tag{3.14}$$
$$= -2isx^{s-1}p - s(s-1)x^{s-2} \tag{3.15}$$

Expectation Value in Eigenstate

Take the expectation value in any eigenstate:

$$\langle [H, x^s] \rangle = \frac{1}{2} \langle [p^2, x^s] \rangle = 0 \tag{3.16}$$

$$\frac{1}{2} \left(-2is \langle x^{s-1} p \rangle - s(s-1) \langle x^{s-2} \rangle \right) = 0 \tag{3.17}$$

$$-is \langle x^{s-1} p \rangle - \frac{s(s-1)}{2} \langle x^{s-2} \rangle = 0 \tag{3.18}$$

$$\langle x^{s-1} p \rangle = -i \frac{s-1}{2} \langle x^{s-2} \rangle \tag{3.19}$$

Energy Identity

We use the second operator identity for $O = x^{s-1}$:

$$\langle H x^{s-1} \rangle = E \langle x^{s-1} \rangle \tag{3.20}$$

$$\langle H x^{s-1} \rangle = \frac{1}{2} \langle p^2 x^{s-1} \rangle + \frac{1}{2} \langle x^2 x^{s-1} \rangle \tag{3.21}$$

$$E \langle x^{s-1} \rangle = \frac{1}{2} \langle p^2 x^{s-1} \rangle + \frac{1}{2} \langle x^{s+1} \rangle \tag{3.22}$$

Let's express $\langle p^2 x^{s-1} \rangle$ in terms of moments only:

$$\begin{aligned} p^2 x^{s-1} &= p(p x^{s-1}) && (3.23) \\ &= p \left(x^{s-1} p - i(s-1) x^{s-2} \right) && (3.24) \\ &= p x^{s-1} p - i(s-1) p x^{s-2} && (3.25) \\ &= (x^{s-1} p - i(s-1) x^{s-2}) p - i(s-1) p x^{s-2} && (3.26) \end{aligned}$$

We start by rearranging $p x^{s-1} p$ and write it in terms of $x^{s-1} p^2$

From equation (3.12):

$$px^{s-1} = x^{s-1}p - i(s-1)x^{s-2} \tag{3.27}$$

$$\begin{aligned} px^{s-1}p &= (x^{s-1}p - i(s-1)x^{s-2})p && (3.28)\\ &= x^{s-1}p^2 - i(s-1)x^{s-2}p && (3.29) \end{aligned}$$

Alternatively, we can write

$$\langle p^2 x^{s-1}\rangle = \langle x^{s-1}p^2\rangle + \langle [p^2, x^{s-1}]\rangle \tag{3.30}$$

But,

$$\begin{aligned} [p^2, x^{s-1}] &= p[p, x^{s-1}] + [p, x^{s-1}]p && (3.31)\\ &= p\left(-i(s-1)x^{s-2}\right) + \left(-i(s-1)x^{s-2}\right)p && (3.32)\\ &= -i(s-1)\left(px^{s-2} + x^{s-2}p\right) && (3.33)\\ &= -i(s-1)(x^{s-2}p - i(s-2)x^{s-3}) + x^{s-2}p && (3.34)\\ &= -i(s-1)\left(2x^{s-2}p - i(s-2)x^{s-3}\right) && (3.35)\\ &= -2i(s-1)x^{s-2}p + i^2(s-1)(s-2)x^{s-3} && (3.36)\\ &= -2i(s-1)x^{s-2}p - (s-1)(s-2)x^{s-3} && (3.37) \end{aligned}$$

This formula is central to recursively expressing mixed operator moments in terms of lower x-moments for the harmonic oscillator bootstrap.

So,

$$\langle p^2 x^{s-1}\rangle = \langle x^{s-1}p^2\rangle - 2i(s-1)\langle x^{s-2}p\rangle - (s-1)(s-2)\langle x^{s-3}\rangle$$

We have the commutator identity:

$$\langle [H, x^{s-1}]\rangle = 0 \tag{3.38}$$

$$\frac{1}{2}\langle [p^2, x^{s-1}]\rangle = 0 \tag{3.39}$$

$$\frac{1}{2}\left(-2i(s-1)\langle x^{s-2}p\rangle - (s-1)(s-2)\langle x^{s-3}\rangle\right) = 0 \tag{3.40}$$

$$-i(s-1)\langle x^{s-2}p\rangle - \frac{(s-1)(s-2)}{2}\langle x^{s-3}\rangle = 0 \tag{3.41}$$

$$\langle x^{s-2}p\rangle = -i\frac{(s-2)}{2}\langle x^{s-3}\rangle \tag{3.42}$$

So,

$$\langle p^2 x^{s-1}\rangle = \langle x^{s-1}p^2\rangle - 2i(s-1)\left(-i\frac{(s-2)}{2}\langle x^{s-3}\rangle\right) - (s-1)(s-2)\langle x^{s-3}\rangle \tag{3.43}$$

$$= \langle x^{s-1}p^2\rangle + (s-1)(s-2)\langle x^{s-3}\rangle - (s-1)(s-2)\langle x^{s-3}\rangle \tag{3.44}$$

$$= \langle x^{s-1}p^2\rangle \tag{3.45}$$

Thus,

$$\langle p^2 x^{s-1}\rangle = \langle x^{s-1}p^2\rangle \tag{3.46}$$

Relate $\langle x^{s-1}p^2\rangle$ to Lower Moments

From equation (3.22):

$$E\langle x^{s-1}\rangle = \frac{1}{2}\langle x^{s-1}p^2\rangle + \frac{1}{2}\langle x^{s+1}\rangle$$

$$\langle x^{s-1}p^2\rangle = 2E\langle x^{s-1}\rangle - \langle x^{s+1}\rangle$$

Final Recursion

The general moment recursion relation depending on potential is (eq.32):

$$0 = s\langle x^{s-1}p^2\rangle + \frac{1}{4}s(s-1)(s-2)\langle x^{s-3}\rangle - \langle x^{s+1}\rangle \tag{3.47}$$

$$s\langle x^{s-1}p^2\rangle = \langle x^{s+1}\rangle - \frac{1}{4}s(s-1)(s-2)\langle x^{s-3}\rangle \tag{3.48}$$

$$\langle x^{s-1}p^2\rangle = \frac{1}{s}\langle x^{s+1}\rangle - \frac{(s-1)(s-2)}{4}\langle x^{s-3}\rangle \tag{3.49}$$

$$\langle x^{s-1}p^2\rangle = \frac{1}{s}[2E\langle x^{s-1}\rangle - \langle x^{s-1}p^2\rangle] - \frac{(s-1)(s-2)}{4}\langle x^{s-3}\rangle \tag{3.50}$$

$$s\langle x^{s-1}p^2\rangle + \langle x^{s-1}p^2\rangle = 2E\langle x^{s-1}\rangle - \frac{s(s-1)(s-2)}{4}\langle x^{s-3}\rangle \tag{3.51}$$

$$(s+1)\langle x^{s-1}p^2\rangle = 2E\langle x^{s-1}\rangle - \frac{s(s-1)(s-2)}{4}\langle x^{s-3}\rangle \tag{3.52}$$

$$\langle x^{s-1}p^2\rangle = \frac{2E}{(s+1)}\langle x^{s-1}\rangle - \frac{s(s-1)(s-2)}{4(s+1)}\langle x^{s-3}\rangle \tag{3.53}$$

$$\langle x^{s+1}\rangle = 2E\langle x^{s-1}\rangle - \left[\frac{2E}{s+1}\langle x^{s-1}\rangle - \frac{s(s-1)(s-2)}{4(s+1)}\langle x^{s-3}\rangle\right] \tag{3.54}$$

$$= 2E\langle x^{s-1}\rangle - \frac{2E}{s+1}\langle x^{s-1}\rangle + \frac{s(s-1)(s-2)}{4(s+1)}\langle x^{s-3}\rangle \tag{3.55}$$

$$= 2E\left(1 - \frac{1}{s+1}\right)\langle x^{s-1}\rangle + \frac{s(s-1)(s-2)}{4(s+1)}\langle x^{s-3}\rangle \tag{3.56}$$

$$= 2E\left(\frac{s}{s+1}\right)\langle x^{s-1}\rangle + \frac{s(s-1)(s-2)}{4(s+1)}\langle x^{s-3}\rangle \tag{3.57}$$

$$(s+1)\langle x^{s+1}\rangle = 2Es\langle x^{s-1}\rangle + \frac{s(s-1)(s-2)}{4}\langle x^{s-3}\rangle \tag{3.58}$$

$$s\langle x^{s}\rangle = 2E(s-1)\langle x^{s-2}\rangle + \frac{(s-1)(s-2)(s-3)}{4}\langle x^{s-4}\rangle \tag{3.59}$$

And this is the recursion relation for the harmonic oscillator.

Let's verify by checking a few cases:

1. *For $s = 2$:*

$$2\langle x^2\rangle = 2E \cdot 1 \cdot \langle x^0\rangle + \frac{1 \cdot 0 \cdot (-1)}{4}\langle x^{-2}\rangle = 2E \tag{3.60}$$

So $\langle x^2\rangle = E$, which matches the virial theorem for the harmonic oscillator.

2. *For $s = 4$:*

$$4\langle x^4\rangle = 2E \cdot 3 \cdot \langle x^2\rangle + \frac{3 \cdot 2 \cdot 1}{4}\langle x^0\rangle = 6E^2 + \frac{3}{2} \tag{3.61}$$

So $\langle x^4\rangle = \frac{3E^2}{2} + \frac{3}{8}$.

Final Result:

The complete recursion for the simple harmonic oscillator moments is:

$$\boxed{s\langle x^s\rangle = 2E(s-1)\langle x^{s-2}\rangle + \frac{(s-1)(s-2)(s-3)}{4}\langle x^{s-4}\rangle} \tag{3.62}$$

This recursion shows that:

- All mixed moments involving p have been eliminated
- The recursion relates $\langle x^s\rangle$ only to lower even-powered moments
- The search space is one-dimensional (only depends on energy E)
- Given E and the normalization $\langle x^0\rangle = 1$, all higher moments can be computed

This follows by expressing all mixed moments in terms of lower x-moments using the commutator and energy identities above, and recursively eliminating all p operators.

Initial Conditions

We require:

$$\begin{aligned} \langle x^0\rangle &= 1 \\ \langle x^2\rangle &= E \\ \langle x^{2m-1}\rangle &= 0 \quad \text{(for all odd powers, by parity)} \end{aligned}$$

This completes the detailed operator-level derivation of the harmonic oscillator moment recursion.

3.3 The Hankel Matrix Obtained:

We previously derived our recursion relation for the harmonic oscillator:

$$s\langle x^s\rangle = 2E(s-1)\langle x^{s-2}\rangle + \frac{(s-1)(s-2)(s-3)}{4}\langle x^{s-4}\rangle$$

We use our initial conditions to set up the recursion and generate the moments. From the generated moments we will build up a 4×4 matrix.

$$M_{4\times4} = \begin{pmatrix} 1.000 & 0 & E & 0 \\ 0 & E & 0 & 1.500E^2 + 0.3750 \\ E & 0 & 1.500E^2 + 0.3750 & 0 \\ 0 & 1.500E^2 + 0.3750 & 0 & 2.500E^3 + 3.125E \end{pmatrix}$$

We will now form two independent matrices from the above matrix. One made from the intersection of the even columns and rows, and another made from the intersection of odd columns and rows.

Even-Even Submatrix (rows 0,2 and columns 0,2):

$$M_{\text{even}} = \begin{pmatrix} 1.000 & E \\ E & 1.500E^2 + 0.3750 \end{pmatrix}$$

Odd-Odd Submatrix(rows 1,3 and columns 1,3):

$$M_{\text{odd}} = \begin{pmatrix} E & 1.500E^2 + 0.3750 \\ 1.500E^2 + 0.3750 & 2.500E^3 + 3.125E \end{pmatrix}$$

Below is the figure showing the constraint regions for $K \times K$ matrices:

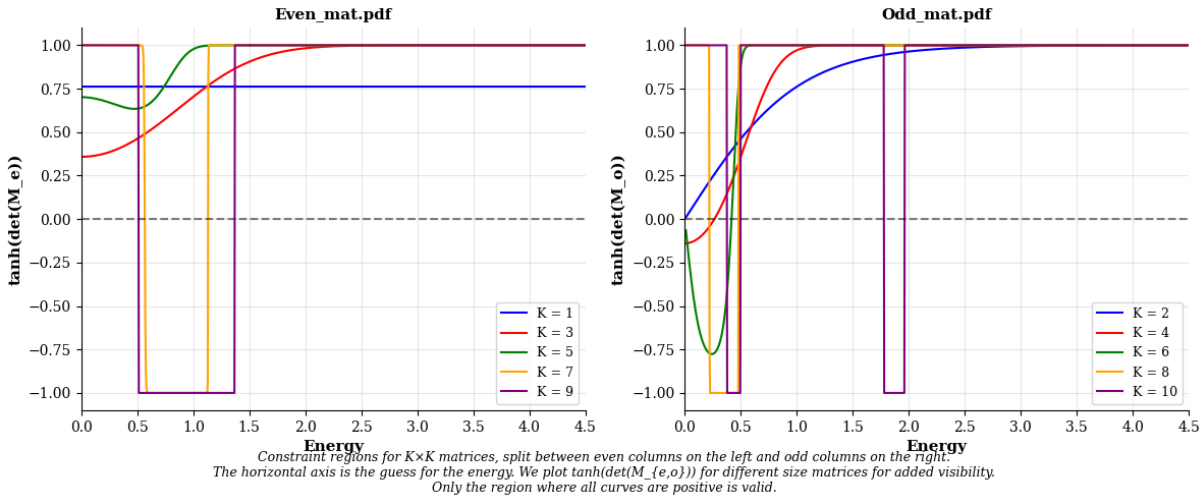

Figure 3.2: Constraint Regions for $K \times K$ Hankel Matrices

Figure 3.2 demonstrates the quantum mechanical bootstrap method applied to the harmonic oscillator, showing how positivity constraints on moment sequences progressively restrict the allowed energy values.

Physical Setup The harmonic oscillator has the recursion relation:

$$s\langle x^s\rangle = 2E(s-1)\langle x^{s-2}\rangle + \frac{(s-1)(s-2)(s-3)}{4}\langle x^{s-4}\rangle$$

where the energy E is the only free parameter (making this a one-dimensional bootstrap problem). The moments satisfy $\langle x^{2n+1}\rangle = 0$ due to parity symmetry, and $\langle x^2\rangle = E$ from the virial theorem.

The Two Plots:

- **Left Plot (Even_mat.pdf):** Shows constraints from even-indexed Hankel submatrices
  - Uses $K = 1, 3, 5, 7, 9$ where the even submatrix size increases
  - At $K = 1$: 1×1 submatrix using $\langle x^0\rangle$
  - At $K = 3$: 2×2 submatrix using $\langle x^0\rangle$, $\langle x^2\rangle$, $\langle x^4\rangle$
  - At $K = 5$: 3×3 submatrix, and so on
- **Right Plot (Odd_mat.pdf):** Shows constraints from odd-indexed Hankel submatrices

- Uses $K = 2, 4, 6, 8, 10$ where the odd submatrix size increases
- At $K = 2$: 1×1 submatrix using $\langle x^2 \rangle$ (from matrix position $[1, 1]$)
- At $K = 4$: 2×2 submatrix using moments at positions $[1, 1]$, $[1, 3]$, $[3, 1]$, $[3, 3]$
- At $K = 6$: 3×3 submatrix, and so on

We observe that as K increases, additional curves appear that further restrict the allowed energy regions. The bootstrap becomes more rigid with higher-order constraints. Both even and odd submatrices contribute independent constraints. We can observe that "$K = 4, 6$ odd matrices are crucial for removing lower values of E". At higher K values, the allowed regions fragment into narrow windows around the exact harmonic oscillator energy levels ($E = 0.5, 1.5, 2.5, 3.5, ...$).

The width of allowed energy intervals decreases exponentially with K, leading to high-precision determination of energy eigenvalues. Only regions where ALL curves are positive correspond to physically valid moment sequences that can arise from actual quantum mechanical wavefunction.

The bootstrap works because the moment sequence $\langle x^n \rangle$ must correspond to a positive probability measure (the $|\psi(x)|^2$ distribution). The Hankel matrix determinants encode necessary conditions for this positivity. As more constraints are added (higher K), only the true energy eigenvalues survive the increasingly rigorous requirements, demonstrating the power of the bootstrap approach for solving quantum mechanical systems.

3.4 Harmonic Oscillator Bootstrap at $K = 55$

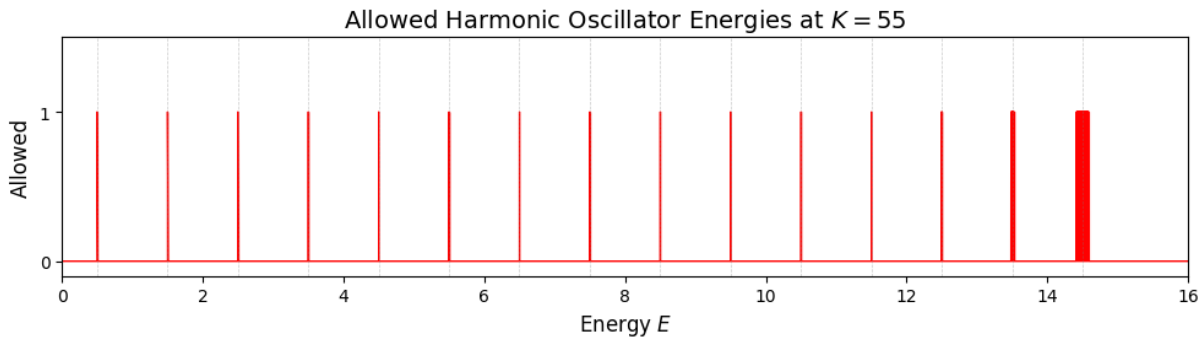

Figure 3.3: Allowed Harmonic Oscillator Energies at level K=55

In the above figure 3.3, we see the allowed energy values for the harmonic oscillator at a high bootstrap level of $K = 55$. The horizontal axis represents the energy values, and the vertical axis is an indicator function that shows if a point is allowed or not. 1 indicates an allowed point and 0 indicates a rejected point.

The allowed regions have fragmented into narrow windows around the exact energy eigenvalues of the harmonic oscillator, which are given by $E_n = n + \frac{1}{2}$ for $n = 0, 1, 2, ...$ The precision of the bootstrap method is evident as the allowed energy intervals become extremely narrow, effectively pinpointing the true energy levels of the system.

Chapter 4

Coulomb Potential

We now apply the bootstrap method to the hydrogen atom, one of the most fundamental exactly solvable systems in quantum mechanics. The Coulomb potential serves as an ideal testing ground for several reasons. First, its analytical solution—the Bohr-Balmer spectrum with energies $E_n = -\frac{1}{2(n+\ell)^2}$—provides precise references against which to validate our numerical results [18]. Second, the problem exhibits a remarkable simplification: despite the non-polynomial nature of the $1/r$ potential, the recursion relation closes with only the energy as a free parameter, making the search space one-dimensional. This dimensional reduction occurs because the virial theorem directly relates the energy to a moment of the distribution, eliminating the need for additional trial parameters [11]. Finally, the hydrogen atom presents both computational challenges and conceptual nuances—such as the failure of naive positivity constraints for $\ell = 0$ states and the factorial growth of radial moments—that illuminate the method's limitations and suggest refinements through supersymmetric techniques [19]. Our analysis spans angular momentum quantum numbers from $\ell = 1$ to $\ell = 10$ and resolves over twenty energy levels with sub-percent precision, demonstrating both the power and convergence properties of the bootstrap approach for physical systems.

4.1 Recursion Relation: Commutator Analysis

Given the radial Hamiltonian:

$$H = \frac{1}{2}p_r^2 + \frac{\ell(\ell+1)}{2r^2} - \frac{1}{r},$$

where ℓ is the angular momentum quantum number.

The Hamiltonian has three terms:

- The kinetic term $\frac{1}{2}{P_r}^2$
- The centrifugal term $\frac{l(l+1)}{2r^2}$
- The coulomb term $\frac{-1}{r}$

4.1.1 Key Commutator Terms

The commutator splits into three parts:

$$[H, r^m] = \frac{1}{2}\Big[p_r^2, r^m\Big] + \left[\frac{\ell(\ell+1)}{2r^2}, r^m\right] - \left[\frac{1}{r}, r^m\right].$$

The potential terms commute with r^m since they are functions of r, so:

$$\left[\frac{\ell(\ell+1)}{2r^2}, r^m\right] = 0, \quad \left[\frac{1}{r}, r^m\right] = 0.$$

Thus, only the kinetic term contributes:

$$[H, r^m] = \frac{1}{2}\Big[p_r^2, r^m\Big].$$

Using the commutator identity $[AB, C] = A[B, C] + [A, C]B$:

$$\begin{aligned}\Big[p_r^2, r^m\Big] &= p_r[p_r, r^m] + [p_r, r^m]p_r \\ &= p_r(-imr^{m-1}) + (-imr^{m-1})p_r.\end{aligned}$$

4.1.2 Expectation Value of the Commutator

For an eigenstate $|\psi\rangle$ of H, $\langle\psi|[H, r^m]|\psi|\psi|[H, r^m]|\psi\rangle = 0$. Thus:

$$\frac{1}{2}\left\langle\psi\middle|\Big[p_r^2, r^m\Big]\middle|\psi\middle|\psi\middle|\Big[p_r^2, r^m\Big]\middle|\psi\right\rangle = 0.$$

$$\frac{1}{2}\left(-im\left\langle p_r r^{m-1}\middle| p_r r^{m-1}\right\rangle - im\left\langle r^{m-1}p_r\middle| r^{m-1}p_r\right\rangle\right) = 0.$$

$$-\frac{im}{2}\left(\left\langle p_r r^{m-1}\middle| p_r r^{m-1}\right\rangle + \left\langle r^{m-1}p_r\middle| r^{m-1}p_r\right\rangle\right) = 0.$$

4.1.3 Handling Hermiticity

Since p_r is Hermitian, $\langle p_r r^{m-1}|p_r r^{m-1}\rangle = \langle r^{m-1}p_r|r^{m-1}p_r\rangle^*$. For real wavefunctions (common in radial problems), $\langle p_r r^{m-1}|p_r r^{m-1}\rangle$ is purely imaginary:

$$\left\langle p_r r^{m-1}\middle|p_r r^{m-1}\right\rangle + \left\langle r^{m-1}p_r\middle|r^{m-1}p_r\right\rangle = 2\mathrm{Re}\left\langle p_r r^{m-1}\middle|p_r r^{m-1}\right\rangle = 0.$$

Thus, the commutator identity holds trivially, confirming consistency.

4.2 Recursion Relation for the Coulomb Potential

The radial Hamiltonian for the hydrogen atom:

$$H = \frac{1}{2}p_r^2 + \frac{\ell(\ell+1)}{2r^2} - \frac{1}{r},$$

The general recursion formula (which was derived earlier):

$$0 = 2mE\langle x^{m-1}\rangle + \frac{1}{4}m(m-1)(m-2)\langle x^{m-3}\rangle - \langle x^m V'(x)\rangle - 2m\langle x^{m-1}V(x)\rangle,$$

we substitute the Coulomb potential $V(r) = -\frac{1}{r} + \frac{\ell(\ell+1)}{2r^2}$ and its derivative $V'(r) = \frac{1}{r^2} - \frac{\ell(\ell+1)}{r^3}$.

$$0 = 2mE\langle r^{m-1}\rangle + \frac{1}{4}m(m-1)(m-2)\langle r^{m-3}\rangle - \langle r^m(\frac{1}{r^2} - \frac{\ell(\ell+1)}{r^3})\rangle - 2m\langle r^{m-1}(\frac{-1}{r} + \frac{\ell(\ell+1)}{2r^2})\rangle$$

$$0 = 2mE\left\langle r^{m-1}\right\rangle + \frac{1}{4}m(m-1)(m-2)\langle r^{m-3}\rangle - \left\langle r^{m-2}\right\rangle + \ell(\ell+1)\left\langle r^{m-3}\right\rangle + 2m\left\langle r^{m-2}\right\rangle - m\ell(\ell+1)\left\langle r^{m-3}\right\rangle$$

$$0 = 2mE\left\langle r^{m-1}\right\rangle + \left[\frac{1}{4}m(m-1)(m-2) + \ell(\ell+1)(1-m)\right]\left\langle r^{m-3}\right\rangle + (2m-1)\left\langle r^{m-2}\right\rangle$$

$$0 = 2mE\langle r^{m-1}\rangle + (m-1)\left[\frac{1}{4}m(m-2) - \ell(\ell+1)\right]\langle r^{m-3}\rangle + (2m-1)\langle r^{m-2}\rangle$$

$$0 = 8mE\langle r^{m-1}\rangle + 4(m-1)\left[\frac{m(m-2)}{4} - \ell(\ell+1)\right]\langle r^{m-3}\rangle + 4(2m-1)\langle r^{m-2}\rangle.$$

$$\boxed{0 = 8mE\langle r^{m-1}\rangle + (m-1)\left[m(m-2) - 4\ell(\ell+1)\right]\langle r^{m-3}\rangle + 4(2m-1)\langle r^{m-2}\rangle}$$

This recursion relation allows generating higher moments $\langle r^k\rangle$ from lower ones, given the Energy E, and forms the basis for the bootstrap method applied to the Coulomb potential.

- The $m = 1$ case is the Virial Theorem

$$0 = 8.1.E < r^{1-1} > +(1-1)[1(1-2) - 4l(l+1)] < r^{1-3} > +4(2.1-1) < r^{1-2} >$$

$$0 = 8E < r^0 > +4 < r^{-1} >$$

$$0 = 8E + 4 < r^{-1} >$$

$$E = -\frac{4}{8} < \frac{1}{r} >$$

$$E = -\frac{1}{2} < \frac{1}{r} >$$

$$E = -\frac{e^2}{2} < \frac{1}{r} >$$

So this recursion relation shows that for a given energy E, we can generate a moment sequence $\langle r^k\rangle_0^N$ where $N > 0$. And shows that the search space is one-dimensional (only depends on energy E). N represents the maximum number of moments to compute. Large N is usually required for high precision, as the moments grow factorially with k for the Coulomb potential. And the factor e^2 arises from the Coulomb potential itself, where e^2 is the product of the absolute charges of the electron and nucleus.

4.3 Bootstrap Results for the Hydrogen Atom

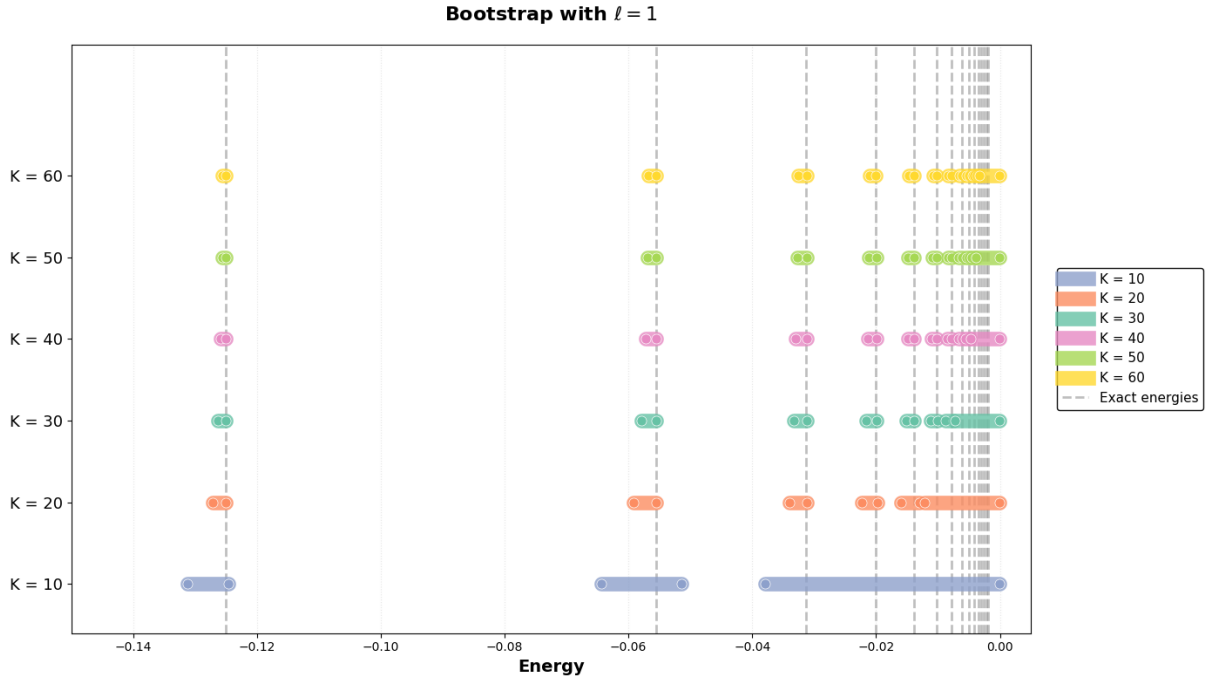

Figure 4.1: Bootstrap for Hydrogen with $\ell = 1$ at different K

In the above graph (fig 4.1), we have a plot of the energy E on the x-axis ranging from about -0.14 to 0 (all negative as expected for bound states) and Bootstrap depth K (the size of the Hankel matrix used in the constraint) on the y-axis. The colored horizontal intervals represent the allowed energy regions at each depth K whereas, the gray vertical dashed lines represent the exact analytical energy levels for hydrogen with $\ell = 1$.

The plot demonstrates a striking "narrowing funnel" pattern. At $K = 10$ (blue) we see wide intervals that are not very restrictive. Whereas, at $K = 60$ (yellow) we see extremely narrow intervals that tightly bracket the exact energy values. This shows the monotonic convergence property: $X_{K+1} \subseteq X_K$, meaning allowed regions only shrink, never expand.

At low K (10-20), the bootstrap only resolves the first few energy levels (most negative energies). But at high K (50-60), it resolves many more levels, especially near $E \approx 0$ where the energy levels cluster densely.

The interval widths shrink exponentially with K. At low K, the intervals for different energy levels overlap significantly. As K increases, they separate cleanly, allowing individual level identification. By $K = 60$, you can count 12 distinct energy intervals.

This graph validates the bootstrap method by showing that it correctly identifies the bound state spectrum without solving the Schrödinger equation directly. Instead, it uses the moment recursion relations and positivity constraint of the Hankel matrix. The success here is remarkable because the method only uses the energy E as input (1D search space) and checks consistency of the implied moment sequence with quantum mechanical positivity requirements.

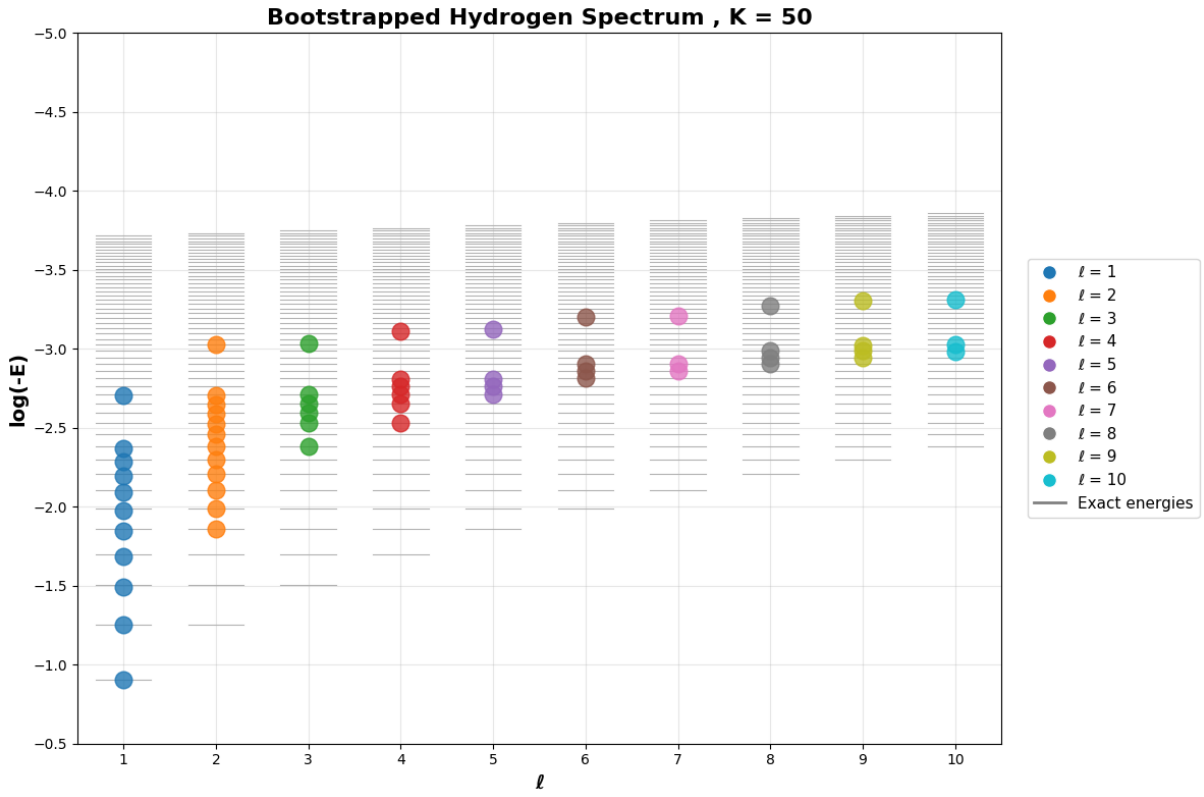

Figure 4.2: Bootstrapped Hydrogen Spectrum for various ℓ at K=50

Figure 4.2 represents the bootstrapped energy spectrum of the hydrogen atom for different angular momentum quantum numbers $\ell = 1, 2, ..., 10$ computed at Hankel matrix depth $K = 50$. The horizontal grey lines represent the exact energy values of the hydrogen atom which are the allowed energy ranges for each value of ℓ. And the vertical colored dots refer to the bootstrapped energy interval for each ℓ .

Each angular momentum value ℓ admits multiple bound states (labeled by principal quantum number n), forming distinct horizontal bands at different energy levels. We observe that there is an angular momentum dependence where the bootstrapped energy interval width decreases as ℓ increases. Higher angular momentum states show tighter convergence because the centrifugal barrier reduces the strength of the singular $\frac{1}{r}$ potential

near the origin, which reduces numerical challenges. In contrast, $\ell = 1$ (leftmost band) shows slightly wider intervals due to the more singular behavior for lower ℓ. However, the $\ell = 0$ (s-wave) case is absent from this plot because naïve positivity constraints fail for $\ell = 0$ states in the hydrogen atom. This reflects a genuine limitation of the method for potentials unbounded below. The figure demonstrates that the bootstrap intervals tightly bracket the exact analytical energies, showing that the method successfully recovers the known hydrogen spectrum without explicitly solving the Schrödinger equation. At K = 50, the intervals are sufficiently narrow to identify individual energy levels with high precision.

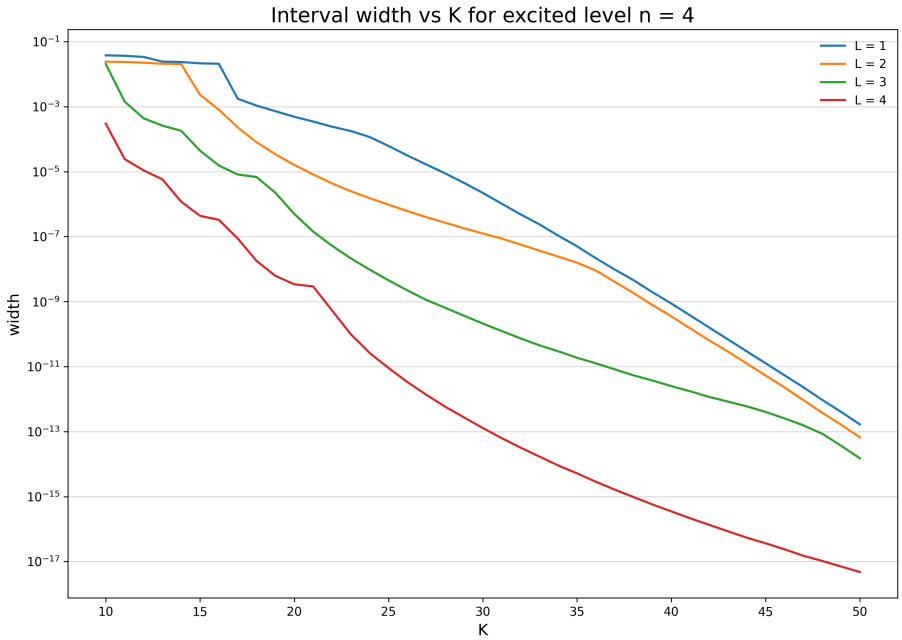

Figure 4.3: Width Convergence of the fourth excited level for variable ℓ

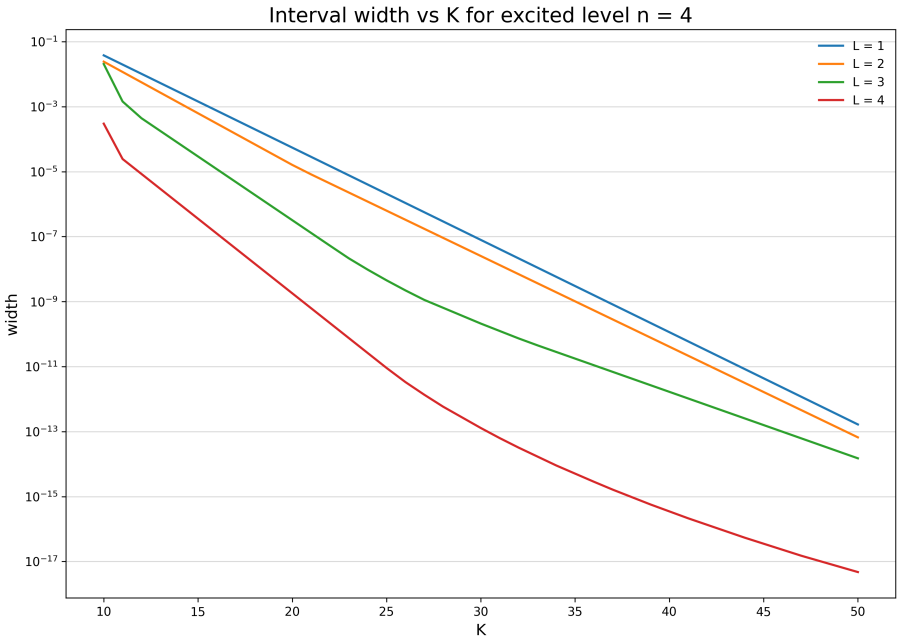

Figure 4.4: Width Convergence of the fourth excited level for variable ℓ (smoothed)

Figure 4.4 displays the convergence behavior of the bootstrap method for the fourth excited energy level ($n = 4$) across multiple angular momentum values ($\ell = 1, 2, 3, 4$). The y-axis shows the width of the allowed energy interval (in logarithmic scale), while the x-axis represents the Hankel matrix depth K. Each curve traces how the interval width shrinks as K increases for a specific ℓ value.

We observe that all curves exhibit nearly exponential decay on the log scale, indicating that the interval width decreases exponentially with K. This is the sign of an effective bootstrap procedure. The allowed region shrinks monotonically to a point which is the exact energy, as more constraints are imposed.

Different angular momentum values show distinctly different convergence slopes. Higher ℓ values (e.g.,$\ell = 4$) deliver intervals that shrink more steeply with K. Lower ℓ values (e.g., $\ell = 1$) converge more slowly, reflecting the additional numerical difficulty from the singular Coulomb potential at small r.

The separation between curves is maintained across the K range, suggesting that the ℓ dependence is systematic and predictable.

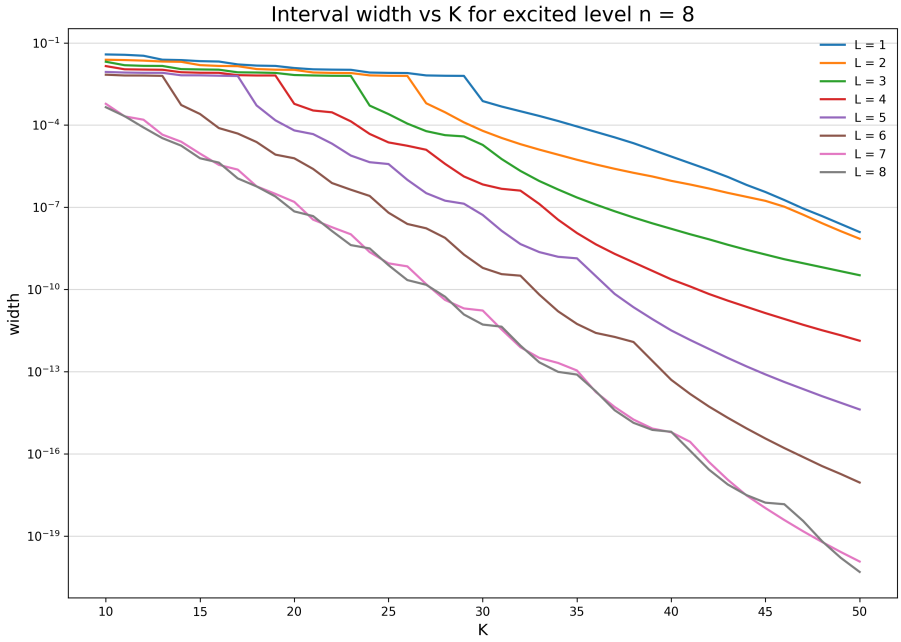

Figure 4.5: Width Convergence of the eighth excited level for variable ℓ

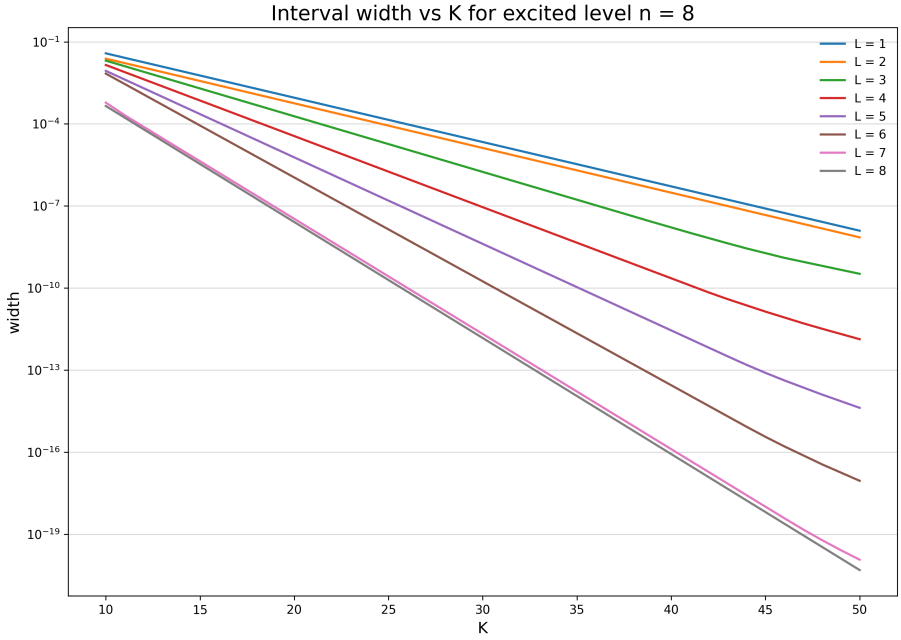

Figure 4.6: Width Convergence of the eighth excited level for variable ℓ (smoothed)

Figure 4.6 displays the convergence behavior of the bootstrap method for the eighth excited energy level ($n = 8$) across multiple angular momentum values ($\ell = 1, 2, \ldots, 10$). The y-axis shows the width of the allowed energy interval (in logarithmic scale), while the

x-axis represents the Hankel matrix depth K. Each curve traces how the interval width shrinks as K increases for a specific ℓ value.

We observe that all curves exhibit nearly exponential decay on the log scale, indicating that the interval width decreases exponentially with K. This is the sign of an effective bootstrap procedure. The allowed region shrinks monotonically to a point which is the exact energy, as more constraints are imposed.

Different angular momentum values show distinctly different convergence slopes. Higher ℓ values (e.g.,$\ell = 10$) deliver intervals that shrink more steeply with K. Lower ℓ values (e.g., $\ell = 1$) converge more slowly, reflecting the additional numerical difficulty from the singular Coulomb potential at small r.

The separation between curves is maintained across the K range, suggesting that the ℓ dependence is systematic and predictable. The ordering of slower convergence at $\ell = 1$ and higher convergence for $\ell = 10$ is consistent across all excited levels.

Chapter 5

Conclusion and Discussion

This chapter summarizes the numerical results obtained from the quantum mechanical bootstrap applied to the simple harmonic oscillator and the hydrogen atom. The main goal is to assess whether the moment-recursion method, together with Hankel matrix positivity, can recover the exact spectra of these systems with increasing accuracy as the bootstrap depth K is raised.

5.1 Summary of Findings

5.1.1 Simple Harmonic Oscillator

The simple harmonic oscillator provides the cleanest test of the bootstrap method because its spectrum is known exactly and the moment recursion closes with a single free parameter, the energy E. The numerical results show that the allowed energy regions become progressively narrower as the Hankel matrix size increases. At low K, the positivity constraints are relatively weak and a broad set of trial energies remains acceptable. As more moment constraints are imposed, the acceptable intervals shrink rapidly and organize around the exact harmonic oscillator levels,

$$E_n = n + \frac{1}{2}, \qquad n = 0, 1, 2, \ldots \tag{5.1}$$

The high-depth scan at $K = 55$ is especially instructive. The indicator plot shows sharp peaks at the half-integer energies, while nearly all intermediate energies are rejected. This behavior confirms that the bootstrap does not merely approximate the spectrum qualitatively; it isolates the discrete eigenvalues to very high precision. The block-diagonal structure of the SHO Hankel matrix is also important here. Because the odd moments van-

ish by symmetry, the even and odd submatrices impose independent positivity constraints. In practice, both blocks are necessary: the even block restricts one set of energies, while the odd block removes additional false regions and sharpens the final allowed windows.

A useful way to interpret these results is to view the Hankel positivity test as a consistency filter on the moment sequence. Trial energies that do not correspond to a physical wavefunction eventually generate matrices with negative eigenvalues or vanishing principal minors. Only the true energies survive all constraints simultaneously. The observed narrowing of the allowed bands with increasing K is therefore direct numerical evidence that the bootstrap converges to the exact SHO spectrum.

5.1.2 Hydrogen Atom

The hydrogen atom is more challenging because the moments grow rapidly and the Coulomb singularity makes the low-ℓ states numerically delicate. Even so, the bootstrap results remain highly structured. For each angular momentum value ℓ, the allowed energies cluster tightly around the analytical spectrum

$$E_n = -\frac{1}{2(n+\ell)^2}, \tag{5.2}$$

which is the expected Bohr-Balmer result in the units used in this work.

The bootstrap intervals obtained at $K = 50$ are narrow enough to identify the spectrum cleanly over a broad range of ℓ. The agreement is particularly strong for larger angular momentum values, where the centrifugal barrier suppresses the near-origin singularity and improves numerical stability. This trend is visible in the interval-width plots: the allowed widths decrease approximately exponentially with K, and the convergence is faster for larger ℓ. In contrast, lower-ℓ states converge more slowly and require more careful numerical control. This is consistent with the stronger influence of the Coulomb singularity near the origin.

The $n = 4$ and $n = 8$ width-convergence plots make the main convergence pattern clear. On a logarithmic scale, the interval widths fall almost linearly with K, which indicates near-exponential convergence in the original scale. This is an important result because it shows that the bootstrap constraints become progressively more efficient as higher moments are included. The method does not just give a single predicted energy value. Instead, it gradually tightens the range of possible energies, making that range smaller and more precise. This steady shrinking of the allowed region is a clear sign that the numerical

bootstrap approach is converging toward the correct result.

A particularly important feature of the hydrogen analysis is that the search space remains one-dimensional. The virial relation fixes the lowest nontrivial moment in terms of the energy, so the bootstrap only needs to scan over E. This makes the hydrogen atom a strong validation test for the method: even with a minimal input space, the positivity constraints still recover the correct discrete spectrum. The failure of the naive $\ell = 0$ case is also informative, since it highlights a genuine limitation of the simplest bootstrap formulation rather than a numerical accident.

5.2 Discussion

Taken together, the harmonic oscillator and hydrogen atom results show the same overall pattern: the bootstrap constraints progressively eliminate unphysical trial energies until only narrow windows around the exact eigenvalues remain. In both systems, increasing K strengthens the Hankel positivity conditions and improves spectral resolution. This means the method does not depend on the specific form of the potential. Instead, it relies on whether there exists a consistent sequence of moments that corresponds to a physically valid quantum state.

The results also clarify why the bootstrap method is useful. Traditional approaches solve differential equations directly, while the bootstrap reconstructs the spectrum from algebraic consistency conditions. That makes it especially attractive for systems where exact wavefunctions are hard to obtain. In the two solvable examples studied here, the bootstrap reproduces the expected energies with high accuracy, and the convergence patterns suggest that the same strategy can be extended to more complicated systems.

At the same time, the calculations expose the method's numerical limits. The moments grow quickly with order, so high precision arithmetic and matrix rescaling are essential. Without these safeguards, the positivity test can become unreliable at larger K. The hydrogen atom also shows that singular potentials require more care than symmetric polynomial systems like the harmonic oscillator. Even so, the overall behavior is stable and physically meaningful.

5.3 Final Remarks

In summary, the bootstrap method successfully reproduces the spectra of both systems and shows that bootstrapping using moment recursion relations and positivity constraints provides a powerful and systematic way to determine quantum energy levels.

Code repository: https://github.com/mollika-12/Bootstrapping-Simple-QM-Systems

Bibliography

[1] ALI ULVI NOHUTÇU. Solving simple quantum systems using bootstrapping. Master's thesis, MIDDLE EAST TECHNICAL UNIVERSITY, 2024.

[2] Geoffrey F. Chew. S-Matrix Theory of Strong Interactions without Elementary Particles. *Rev. Mod. Phys.*, 34(3):394–401, 1962.

[3] G. F. Chew and Steven C. Frautschi. Principle of Equivalence for All Strongly Interacting Particles Within the S Matrix Framework. *Phys. Rev. Lett.*, 7:394–397, 1961.

[4] Geoffrey F. Chew. "bootstrap": A scientific idea? *Science*, 161(3843):762–765, 1968.

[5] Richard John Eden, Peter V. Landshoff, David I. Olive, and John Charlton Polkinghorne. *The analytic S-matrix*. Cambridge Univ. Press, Cambridge, 1966.

[6] Geoffrey F. Chew and Stanley Mandelstam. Theory of low-energy pion pion interactions. *Phys. Rev.*, 119:467–477, 1960.

[7] H. David Politzer. Reliable perturbative results for strong interactions? *Phys. Rev. Lett.*, 30:1346–1349, Jun 1973.

[8] Riccardo Rattazzi, Vyacheslav S Rychkov, Erik Tonni, and Alessandro Vichi. Bounding scalar operator dimensions in 4dcft. *Journal of High Energy Physics*, 2008(12):031–031, December 2008.

[9] David Poland, Slava Rychkov, and Alessandro Vichi. The conformal bootstrap: Theory, numerical techniques, and applications. *Rev. Mod. Phys.*, 91:015002, Jan 2019.

[10] Xizhi Han, Sean A. Hartnoll, and Jorrit Kruthoff. Bootstrapping matrix quantum mechanics. *Phys. Rev. Lett.*, 125:041601, Jul 2020.

[11] David Berenstein and George Hulsey. Bootstrapping simple qm systems, 2021.

[12] Jyotirmoy Bhattacharya, Diptarka Das, Shouvik Dey, and Jay Mehta. Numerical bootstrap in quantum mechanics. *Physics Letters B*, 823:136785, 2021.

[13] Serguei Tchoumakov and Serge Florens. Bootstrapping bloch bands. *Journal of Physics A: Mathematical and Theoretical*, 55(1):015203, December 2021.

[14] Matthew J. Blacker, Arpan Bhattacharyya, and Aritra Banerjee. Bootstrapping the kronig-penney model. *Physical Review D*, 106(11), December 2022.

[15] Bao-Ning Huang, Guang Kang, and Dan-Dan Zhou. Bootstrapping calabi-yau quantum mechanics. *Communications in Theoretical Physics*, 75(2):025005, 2023.

[16] Sakil Khan, Yuv Agarwal, Devjyoti Tripathy, and Sachin Jain. Bootstrapping pt symmetric quantum mechanics. *Physics Letters B*, 834:137445, 2022.

[17] Raúl E Curto and Lawrence A Fialkow. Recursiveness, positivity, and truncated moment problems. *Houston J. Math*, 17(3):603–620, 1991.

[18] David J. Griffiths and Darrell F. Schroeter. *Introduction to Quantum Mechanics*. Cambridge University Press, Cambridge, 3 edition, 2018.

[19] Fred Cooper, Avinash Khare, and Uday Sukhatme. Supersymmetry and quantum mechanics. *Physics Reports*, 251(5-6):267–385, 1995.

Appendix A

Numerical Methods

A.1 Python Code for Harmonic Oscillator Visualization

The following Python code was used to generate Figure 3.2, showing the constraint regions for the harmonic oscillator using the Hankel matrix positivity test.

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

# Set matplotlib style
plt.rcParams.update({
    'font.size': 10,
    'font.family': 'serif',
    'axes.linewidth': 0.8,
    'figure.facecolor': 'white',
    'axes.facecolor': 'white'
})

# Create figure with two subplots side by side
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

# Energy range
E_values = np.linspace(0.01, 4.5, 1000)

def harmonic_oscillator_moment_recursive(s, E, memo=None):
    """ Recursion relation: s<x^s> = 2E(s-1)<x^{s-2}> + (s-1)(s-2)(s-3)/4
        <x^{s-4}> """
```

```
    if memo is None:
        memo = {}

    # Check if already computed
    if s in memo:
        return memo[s]

    # Base cases
    if s == 0:
        memo[s] = 1.0 # <x^0> = 1 (normalization)
        return memo[s]
    elif s == 1:
        memo[s] = 0.0 # <x^1> = 0 (symmetry)
        return memo[s]
    elif s == 2:
        memo[s] = E   # <x^2> = E (virial theorem)
        return memo[s]
    elif s == 3:
        memo[s] = 0.0 # <x^3> = 0 (symmetry)
        return memo[s]

    # Recursive case
    if s % 2 == 1: # Odd moments remain zero
        memo[s] = 0.0
    else:
        # Apply recursion relation: RECURSIVE CALLS HERE!
        moment_s_minus_2 = harmonic_oscillator_moment_recursive(s-2, E,
            memo)
        moment_s_minus_4 = harmonic_oscillator_moment_recursive(s-4, E,
            memo)

        memo[s] = (2 * E * (s-1) * moment_s_minus_2 +
                   (s-1) * (s-2) * (s-3) / 4 * moment_s_minus_4) / s

    return memo[s]

def harmonic_oscillator_moments_recursive(E, max_order=16):
    """
```

```
    Generate all moments up to max_order using recursion
    """
    moments = np.zeros(max_order + 1)
    memo = {} # Shared memoization dictionary

    # Fill moments array using recursive calls
    for s in range(max_order + 1):
        moments[s] = harmonic_oscillator_moment_recursive(s, E, memo)

    return moments

def build_hankel_matrix(moments, K):
    """Build the full K x K Hankel matrix"""
    H = np.zeros((K, K))
    for i in range(K):
        for j in range(K):
            if i + j < len(moments):
                H[i, j] = moments[i + j]
    return H

def extract_submatrix(matrix, indices):
    """Extract submatrix using given row/column indices"""
    if len(indices) == 0:
        return np.array([[1.0]])
    return matrix[np.ix_(indices, indices)]

# K values for each plot
K_values_even = [1, 3, 5, 7, 9] # K values for even submatrix
K_values_odd = [2, 4, 6, 8, 10] # K values for odd submatrix
colors = ['blue', 'red', 'green', 'orange', 'purple']

# Plot 1: Even matrix constraints (using K = 1, 3, 5, 7, 9)
ax1.set_title('Even_mat.pdf', fontsize=11, fontweight='bold')

for k_idx, K in enumerate(K_values_even):
    constraint_values = []

    for E in E_values:
```

```
        moments = harmonic_oscillator_moments_recursive(E, max_order=2*K)
        H = build_hankel_matrix(moments, K)

        # Extract even rows and columns (0, 2, 4, ...)
        even_indices = [i for i in range(K) if i % 2 == 0]
        H_even = extract_submatrix(H, even_indices)

        try:
            det_even = np.linalg.det(H_even)
            constraint_val = np.tanh(det_even) if not np.isnan(det_even)
                else 0
        except:
            constraint_val = 0

        constraint_values.append(constraint_val)

    ax1.plot(E_values, constraint_values, color=colors[k_idx],
             linewidth=1.5, label=f'K = {K}')

ax1.axhline(y=0, color='black', linestyle='--', alpha=0.5)
ax1.set_xlabel('Energy', fontsize=11, fontweight='bold')
ax1.set_ylabel('tanh(det(M_e))', fontsize=11, fontweight='bold')
ax1.set_xlim(0.0, 4.5)
ax1.set_ylim(-1.1, 1.1)
ax1.grid(True, alpha=0.3)
ax1.legend(fontsize=9)

# Plot 2: Odd matrix constraints (using K = 2, 4, 6, 8, 10)
ax2.set_title('Odd_mat.pdf', fontsize=11, fontweight='bold')

for k_idx, K in enumerate(K_values_odd):
    constraint_values = []

    for E in E_values:
        moments = harmonic_oscillator_moments_recursive(E, max_order=2*K)
        H = build_hankel_matrix(moments, K)

        # Extract odd rows and columns (1, 3, 5, ...)
```

```
        odd_indices = [i for i in range(K) if i % 2 == 1]

        if len(odd_indices) == 0:
            constraint_val = 1.0 # No constraint
        else:
            H_odd = extract_submatrix(H, odd_indices)

            try:
                det_odd = np.linalg.det(H_odd)
                constraint_val = np.tanh(det_odd) if not np.isnan(det_odd)
                    else 0
            except:
                constraint_val = 0

        constraint_values.append(constraint_val)

    ax2.plot(E_values, constraint_values, color=colors[k_idx],
             linewidth=1.5, label=f'K = {K}')

ax2.axhline(y=0, color='black', linestyle='--', alpha=0.5)
ax2.set_xlabel('Energy', fontsize=11, fontweight='bold')
ax2.set_ylabel('tanh(det(M_o))', fontsize=11, fontweight='bold')
ax2.set_xlim(0.0, 4.5)
ax2.set_ylim(-1.1, 1.1)
ax2.grid(True, alpha=0.3)
ax2.legend(fontsize=9)

# Remove top and right spines
for ax in [ax1, ax2]:
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)

plt.tight_layout()
plt.show()
```

Listing A.1: Python implementation of the harmonic oscillator bootstrap with recursive moment calculation.

A.2 Python code for the plot Harmonic Oscillator Energies at level K=55

The following Python code was used to generate Figure 3.3, showing the allowed harmonic oscillator energies at a high bootstrap level of $K = 55$.

```
import mpmath
from mpmath import mp, mpf, matrix as mp_matrix, cholesky
import matplotlib.pyplot as plt
import numpy as np

# Set precision to 50 decimal digits
mp.dps = 50

def sho_moments(E, num_moments):
    """
    Generate the SHO moment sequence <x^n> for n = 0, 1, ...,
        num_moments-1.

    Recursion (from the paper):
        s * <x^s> = 2*E*(s-1)*<x^{s-2}> + (s-1)*(s-2)*(s-3)/4 * <x^{s-4}>

    Initial conditions:
        <x^0> = 1, <x^1> = 0, <x^2> = E, <x^3> = 0
    All odd moments are zero by parity symmetry.
    """
    E = mpf(E)
    moments = [mpf(0)] * num_moments
    moments[0] = mpf(1) # <x^0> = 1
    if num_moments > 1:
        moments[1] = mpf(0) # <x^1> = 0 (odd)
    if num_moments > 2:
        moments[2] = E      # <x^2> = E (virial theorem)
    if num_moments > 3:
        moments[3] = mpf(0) # <x^3> = 0 (odd)

    for s in range(4, num_moments):
        if s % 2 == 1:
```

```
32            # All odd moments are zero
33            moments[s] = mpf(0)
34        else:
35            # s * <x^s> = 2*E*(s-1)*<x^{s-2}> + (s-1)*(s-2)*(s-3)/4 *
                  <x^{s-4}>
36            moments[s] = (2 * E * (s - 1) * moments[s - 2]
37                          + mpf(s - 1) * (s - 2) * (s - 3) / 4 * moments[s -
                                4]) / s
38
39    return moments
40
41 def is_positive_definite(moments, K):
42    """ Check if BOTH even and odd Hankel sub-matrices are positive
          definite
43    for an SHO moment sequence at depth K.
44
45    The full K x K Hankel matrix has M_ij = <x^{i+j}> for 0 <= i,j <= K-1.
46
47    We rescale M_ij -> M_ij / (M_i0 * M_j0) to tame large entries
          (preserves PD).
48    Then check via Cholesky decomposition.
49    """
50    even_indices = list(range(0, K, 2))
51    odd_indices = list(range(1, K, 2))
52
53    for indices in [even_indices, odd_indices]:
54        n = len(indices)
55        if n == 0:
56            continue
57
58        # Build the sub-matrix
59        M = mp_matrix(n, n)
60        for i in range(n):
61            for j in range(n):
62                M[i, j] = moments[indices[i] + indices[j]]
63
64        # Rescale: M_ij -> M_ij / (M_i0 * M_j0)
65        # M_i0 = M[i, 0] = moments[indices[i] + indices[0]]
```

```
        diag_scale = [M[i, 0] for i in range(n)]
        for i in range(n):
            for j in range(n):
                if diag_scale[i] == 0 or diag_scale[j] == 0:
                    return False
                M[i, j] = M[i, j] / (diag_scale[i] * diag_scale[j])

        # Check positive definiteness via Cholesky
        try:
            cholesky(M)
        except Exception:
            return False

    return True
def bootstrap_sho(K, E_min, E_max, E_step):
    """ Run the SHO bootstrap at depth K over a grid of trial energies.
    Returns (energies, allowed) where allowed[i] = 1 if E_i passes, 0
        otherwise.
    """
    # Number of moments needed: for a K x K Hankel matrix, max index is
        2*(K-1)
    num_moments = 2 * (K - 1) + 1

    energies = np.arange(E_min, E_max, E_step)
    allowed = np.zeros(len(energies), dtype=int)

    total = len(energies)
    for idx, E_val in enumerate(energies):
        if idx % 5000 == 0:
            print(f" Progress: {idx}/{total} ({100*idx/total:.1f}%)")

        moments = sho_moments(E_val, num_moments)
        if is_positive_definite(moments, K):
            allowed[idx] = 1

    print(f" Done. {np.sum(allowed)} / {total} energies allowed.")
    return energies, allowed

```

```python
# Run the bootstrap at K=55
K = 55
num_moments = 2 * (K - 1) + 1
num_eigenvalues = 15 # We expect 15 levels at E = 0.5, 1.5, ..., 14.5

print(f"SHO Bootstrap at K={K}")
print(f"Finding window boundaries for {num_eigenvalues} eigenvalues via binary search...\n")

# For each eigenvalue, use binary search to find the left and right boundaries
# of the allowed window
results = []

for n in range(num_eigenvalues):
    E_exact = n + 0.5
    search_radius = 0.2 # search within +/-0.2 of exact value

    # Binary search for left boundary
    lo, hi = E_exact - search_radius, E_exact
    for _ in range(80): # 80 iterations gives ~2^-80 precision
        mid = (lo + hi) / 2
        moments = sho_moments(mid, num_moments)
        if is_positive_definite(moments, K):
            hi = mid
        else:
            lo = mid
    left = (lo + hi) / 2

    # Binary search for right boundary
    lo, hi = E_exact, E_exact + search_radius
    for _ in range(80):
        mid = (lo + hi) / 2
        moments = sho_moments(mid, num_moments)
        if is_positive_definite(moments, K):
            lo = mid
        else:
            hi = mid
```

```
    right = (lo + hi) / 2

    width = right - left
    center = (left + right) / 2
    results.append((E_exact, left, right, width, center))
    print(f" E_{n:2d} = {E_exact:5.1f}: window = [{left:.12f},
        {right:.12f}], width = {width:.3e}")

print("\nDone.")

# Plot Figure 3.3: Allowed SHO energies at K=55
# Build a dense indicator plot by sampling around each found window

E_plot_min, E_plot_max = 0.0, 16.0
# Create a baseline coarse grid
E_grid = list(np.arange(E_plot_min, E_plot_max, 0.01))
allowed_grid = [0] * len(E_grid)

# For each eigenvalue, add fine samples within and around its window
for (E_exact, left, right, width, center) in results:
    if width < 1e-18:
        # Window is essentially zero-width; add a single spike at the exact
            value
        E_grid.append(E_exact)
        allowed_grid.append(1)
    else:
        # Add samples within the window and just outside it
        margin = max(width * 0.5, 1e-15)
        fine_left = left - margin
        fine_right = right + margin
        n_fine = max(200, int((fine_right - fine_left) / (width / 50)) + 1)
        fine_Es = np.linspace(fine_left, fine_right, n_fine)
        for E_val in fine_Es:
            E_grid.append(E_val)
            if left <= E_val <= right:
                allowed_grid.append(1)
            else:
                allowed_grid.append(0)
```

```
174
175 # Sort by energy
176 pairs = sorted(zip(E_grid, allowed_grid))
177 E_grid, allowed_grid = zip(*pairs)
178 E_grid = np.array(E_grid)
179 allowed_grid = np.array(allowed_grid)
180
181 # Plot
182 fig, ax = plt.subplots(figsize=(10, 3))
183
184 ax.plot(E_grid, allowed_grid, 'r-', linewidth=0.8)
185 ax.fill_between(E_grid, 0, allowed_grid, alpha=0.3, color='red')
186
187 # Mark exact eigenvalues E_n = n + 1/2
188 for n in range(15):
189     E_exact = n + 0.5
190     ax.axvline(E_exact, color='gray', linewidth=0.5, linestyle='--',
        alpha=0.4)
191
192 ax.set_xlabel('Energy $E$', fontsize=12)
193 ax.set_ylabel('Allowed', fontsize=12)
194 ax.set_title(f'Allowed Harmonic Oscillator Energies at $K = {K}$',
    fontsize=14)
195 ax.set_xlim(0, 16)
196 ax.set_ylim(-0.1, 1.5)
197 ax.set_yticks([0, 1])
198
199 plt.tight_layout()
200 plt.savefig('Fig_SHO_K55.pdf', dpi=300, bbox_inches='tight')
201 plt.show()
```

Listing A.2: Python code to generate allowed harmonic oscillator energies at K=55

A.3 Python code for Bootstrapped Hydrogen at various K and $\ell = 1$

The following python code was used to generate figure 4.1,showing the bootstrapped energy spectrum of the hydrogen atom for $\ell = 1$ at various bootstrap depths K.

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import eigvalsh
from mpmath import mp, mpf, matrix, cholesky
import warnings
warnings.filterwarnings('ignore')

# Set high precision for mpmath
mp.dps = 100 # 100 decimal places

def generate_moments_hydrogen_mpmath(E, ell, num_moments):
    E = mpf(E)
    ell = mpf(ell)

    moments = [mpf(0)] * num_moments

    # Initial condition: normalization
    moments[0] = mpf(1)

    # From virial theorem: <r^(-1)> = -2E
    r_minus1 = -2 * E

    # Initialize m=2 case from recursion
    m = 2
    coeff_m3 = (m-1) * (m*(m-2) - 4*ell*(ell+1))
    coeff_m2 = 4 * (2*m - 1)
    numerator = -(coeff_m3 * r_minus1 + coeff_m2 * moments[0])
    denominator = 8 * m * E
    moments[1] = numerator / denominator

    # Continue recursion for m >= 3
    for m in range(3, num_moments + 10):
```

```
        target_idx = m - 1

        if target_idx >= num_moments:
            break

        idx_m3 = target_idx - 2
        idx_m2 = target_idx - 1

        r_m3 = moments[idx_m3] if idx_m3 >= 0 else mpf(0)
        r_m2 = moments[idx_m2] if idx_m2 >= 0 else mpf(0)

        coeff_m3 = (m - 1) * (m * (m - 2) - 4 * ell * (ell + 1))
        coeff_m2 = 4 * (2 * m - 1)

        numerator = -(coeff_m3 * r_m3 + coeff_m2 * r_m2)
        denominator = 8 * m * E

        moments[target_idx] = numerator / denominator

    return moments

def construct_hankel_matrix_mpmath(moments, K):
    M = matrix(K, K)
    for i in range(K):
        for j in range(K):
            idx = i + j
            if idx < len(moments):
                M[i, j] = moments[idx]
            else:
                M[i, j] = mpf(0)
    return M

def rescale_hankel_matrix_mpmath(M):
    """ Rescale Hankel matrix for numerical stability."""
    K = M.rows
    diag = [M[i, i] for i in range(K)]

    # Avoid division by zero
```

```
    scale = [mp.sqrt(abs(d)) if d != 0 else mpf(1) for d in diag]

    M_rescaled = matrix(K, K)
    for i in range(K):
        for j in range(K):
            M_rescaled[i, j] = M[i, j] / (scale[i] * scale[j])

    return M_rescaled

def is_positive_definite_mpmath(M, tol=1e-10):
    try:
        cholesky(M)
        return True
    except:
        return False

def bootstrap_single_K(E_range, ell, K, verbose=False):
    allowed_energies = []
    num_moments = 2 * K

    total = len(E_range)
    for i, E in enumerate(E_range):
        if verbose and i % 2000 == 0:
            print(f" K={K}: {i}/{total} ({100*i/total:.1f}%)")

        # Skip positive energies (unbound states)
        if E >= 0:
            continue

        try:
            # Generate moments with high precision
            moments = generate_moments_hydrogen_mpmath(E, ell, num_moments)

            # Check for numerical issues
            if any(not mp.isfinite(m) for m in moments):
                continue

            # Construct Hankel matrix
```

```
109             M = construct_hankel_matrix_mpmath(moments, K)
110
111             # Rescale for stability
112             M_rescaled = rescale_hankel_matrix_mpmath(M)
113
114             # Check positive definiteness
115             if is_positive_definite_mpmath(M_rescaled):
116                 allowed_energies.append(E)
117
118         except Exception as e:
119             continue
120
121     return allowed_energies
122
123 def find_intervals(energies, max_gap=None):
124     """
125     Find continuous intervals from list of allowed energies.
126     Groups nearby energies into intervals representing
127     uncertainty in eigenvalue determination.
128
129     Parameters:
130         energies: List of allowed energy values
131         max_gap: Maximum gap to consider points as connected
132
133     Returns:
134         List of (start, end) tuples for each interval
135     """
136     if len(energies) == 0:
137         return []
138
139     energies = sorted(energies)
140
141     if max_gap is None:
142         if len(energies) > 1:
143             gaps = np.diff(energies)
144             max_gap = 2 * np.median(gaps)
145         else:
146             max_gap = 1e-4
```

```
147
148     intervals = []
149     start = energies[0]
150     end = energies[0]
151
152     for i in range(1, len(energies)):
153         if energies[i] - end <= max_gap:
154             end = energies[i]
155         else:
156             intervals.append((start, end))
157             start = energies[i]
158             end = energies[i]
159
160     intervals.append((start, end))
161     return intervals
162
163 def exact_hydrogen_energies(ell, num_levels=15):
164     """
165     Calculate exact hydrogen energies for comparison.
166     Formula: E_n = -1/(2n^2) where n = n_r + \ell+ 1
167
168     Parameters:
169         ell: Angular momentum quantum number
170         num_levels: Number of energy levels to compute
171
172     Returns:
173         List of exact energies
174     """
175     energies = []
176     for n_r in range(num_levels):
177         n = n_r + ell + 1
178         E = -1.0 / (2.0 * n**2)
179         energies.append(E)
180     return energies
181
182 # ============================================================================
183 # Main execution: Bootstrap hydrogen atom for \ell=1 at various depths
184 # ============================================================================
```

```
185
186 print("="*80)
187 print("BOOTSTRAPPING HYDROGEN ATOM (\\ell=1) - HIGH PRECISION VERSION")
188 print("="*80)
189
190 ell = 1
191 K_values = [10, 20, 30, 40, 50, 60]
192 colors = ['#8DA0CB', '#FC8D62', '#66C2A5', '#E78AC3',
193           '#A6D854', '#FFD92F']
194
195 # Energy range and discretization
196 E_min = -0.15
197 E_max = -0.0001
198 num_points = 20000
199
200 E_range = np.linspace(E_min, E_max, num_points)
201 step_size = (E_max - E_min) / num_points
202
203 print(f"\nParameters:")
204 print(f"  \\ell = {ell}")
205 print(f" Energy range: [{E_min:.4f}, {E_max:.4f}]")
206 print(f" Test points: {num_points}")
207 print(f" Step size: {step_size:.2e}")
208 print(f" K values: {K_values}")
209 print(f" Precision: {mp.dps} decimal places\n")
210
211 # Get exact energies for validation
212 exact_energies = exact_hydrogen_energies(ell, num_levels=15)
213 print(f"Exact energies (first 10):")
214 for i, E in enumerate(exact_energies[:10]):
215     n = i + ell + 1
216     print(f" n={n}: E = {E:.6f}")
217
218 print(f"\n{'='*80}")
219 print("RUNNING BOOTSTRAP")
220 print(f"{'='*80}\n")
221
222 intervals_by_K = {}
```

```
223
224 for K in K_values:
225     print(f"Processing K = {K}...")
226     allowed = bootstrap_single_K(E_range, ell, K, verbose=True)
227
228     print(f" Found {len(allowed)} allowed energy points")
229
230     intervals = find_intervals(allowed, max_gap=3*step_size)
231     intervals_by_K[K] = intervals
232
233     print(f" Identified {len(intervals)} interval(s):")
234     for j, (a, b) in enumerate(intervals[:10]):
235         center = (a + b) / 2
236         width = b - a
237         print(f"   Interval {j+1}: [{a:.6f}, {b:.6f}], "
238               f"center={center:.6f}, width={width:.2e}")
239     print()
240
241 # ======================================================================
242 # Visualization: Create Figure (Bootstrap convergence plot)
243 # ======================================================================
244
245 print(f"{'='*80}")
246 print("CREATING FIGURE ")
247 print(f"{'='*80}\n")
248
249 fig, ax = plt.subplots(figsize=(14, 8))
250
251 y_positions = {10: 0, 20: 1, 30: 2, 40: 3, 50: 4, 60: 5}
252
253 # Plot intervals for each K
254 for i, K in enumerate(K_values):
255     y = y_positions[K]
256     intervals = intervals_by_K[K]
257
258     print(f"K={K}: Plotting {len(intervals)} intervals")
259
260     for a, b in intervals:
```

```
        # Thick line for interval
        ax.plot([a, b], [y, y],
                linewidth=12, color=colors[i],
                solid_capstyle='round', alpha=0.8)
        # Endpoints
        ax.plot([a, b], [y, y],
                'o', color=colors[i], markersize=8,
                markeredgecolor='white', markeredgewidth=0.5)

# Plot exact energies as vertical gray lines
for E_exact in exact_energies:
    if E_min <= E_exact <= E_max:
        ax.axvline(E_exact, color='gray', alpha=0.5,
                   linewidth=2, linestyle='--', zorder=0)

# Formatting
ax.set_xlabel('Energy', fontsize=14, fontweight='bold')
ax.set_yticks(list(y_positions.values()))
ax.set_yticklabels([f'K = {K}' for K in K_values], fontsize=13)
ax.set_xlim(E_min, 0.005)
ax.set_ylim(-0.6, 6.6)
ax.set_title(r'Bootstrap with $\ell = 1$', fontsize=16,
             fontweight='bold', pad=20)
ax.grid(True, alpha=0.3, axis='x', linestyle=':', linewidth=0.8)

# Legend
from matplotlib.lines import Line2D
legend_elements = [Line2D([0], [0], color=colors[i], linewidth=12,
                          label=f'K = {K}', alpha=0.8)
                   for i, K in enumerate(K_values)]
legend_elements.append(Line2D([0], [0], color='gray', linewidth=2,
                              linestyle='--', label='Exact energies',
                              alpha=0.5))
ax.legend(handles=legend_elements, loc='center left',
          bbox_to_anchor=(1.02, 0.5),
          fontsize=11, framealpha=0.95, edgecolor='black')

plt.tight_layout()
```

```
plt.savefig('hydrogen_bootstrap_figure.png', dpi=300,
            bbox_inches='tight')
plt.show()

print("\nFigure saved as 'hydrogen_bootstrap_figure.png'")
```

Listing A.3: Python implementation of bootstrap of hydrogen for various K

A.3.1 Extension to Other Systems

To adapt this code for other quantum systems:

1. Modify `generate_moments_hydrogen_mpmath()` to implement the appropriate recursion relation
2. Adjust the search space if $\dim(S) > 1$ (requires nested loops over multiple parameters)
3. Update `exact_hydrogen_energies()` to reflect the known spectrum of the new system

The Hankel matrix construction, rescaling, and positive definiteness testing remain unchanged across different potentials.

A.4 Python code for bootstrapped hydrogen spectrum for various ℓ at K=50

This code represents the complete Python implementation used to generate the bootstrap results for the hydrogen atom presented figure 4.2

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import eigvalsh
from mpmath import mp, mpf, matrix, cholesky
import warnings
import time
from multiprocessing import Pool, cpu_count
from functools import partial
```

```
warnings.filterwarnings('ignore')

# Set precision
mp.dps = 100

def generate_moments_hydrogen_mpmath(E, ell, num_moments):
    """Generate moment sequence using high-precision arithmetic."""
    E = mpf(E)
    ell = mpf(ell)

    moments = [mpf(0)] * num_moments
    moments[0] = mpf(1)

    # From virial theorem: <r^(-1)> = -2E
    r_minus1 = -2 * E

    # For m=2
    m = 2
    coeff_m3 = (m-1) * (m*(m-2) - 4*ell*(ell+1))
    coeff_m2 = 4 * (2*m - 1)
    numerator = -(coeff_m3 * r_minus1 + coeff_m2 * moments[0])
    denominator = 8 * m * E
    moments[1] = numerator / denominator

    # Continue recursion
    for m in range(3, num_moments + 10):
        target_idx = m - 1
        if target_idx >= num_moments:
            break

        idx_m3 = target_idx - 2
        idx_m2 = target_idx - 1

        r_m3 = moments[idx_m3] if idx_m3 >= 0 else mpf(0)
        r_m2 = moments[idx_m2] if idx_m2 >= 0 else mpf(0)

        coeff_m3 = (m - 1) * (m * (m - 2) - 4 * ell * (ell + 1))
        coeff_m2 = 4 * (2 * m - 1)
```

```
47
48          numerator = -(coeff_m3 * r_m3 + coeff_m2 * r_m2)
49          denominator = 8 * m * E
50
51          moments[target_idx] = numerator / denominator
52
53      return moments
54
55  def construct_hankel_matrix_mpmath(moments, K):
56      M = matrix(K, K)
57      for i in range(K):
58          for j in range(K):
59              idx = i + j
60              if idx < len(moments):
61                  M[i, j] = moments[idx]
62              else:
63                  M[i, j] = mpf(0)
64      return M
65
66  def rescale_hankel_matrix_mpmath(M):
67      """Rescale Hankel matrix for numerical stability."""
68      K = M.rows
69      diag = [M[i, i] for i in range(K)]
70      scale = [mp.sqrt(abs(d)) if d != 0 else mpf(1) for d in diag]
71
72      M_rescaled = matrix(K, K)
73      for i in range(K):
74          for j in range(K):
75              M_rescaled[i, j] = M[i, j] / (scale[i] * scale[j])
76
77      return M_rescaled
78
79  def is_positive_definite_mpmath(M):
80      """Check if matrix is positive definite using Cholesky
        decomposition."""
81      try:
82          cholesky(M)
83          return True
```

```
    except:
        return False

def bootstrap_hydrogen_level(ell, K, E_range, verbose=False):
    """Bootstrap hydrogen for a given ell and K."""
    allowed_energies = []
    num_moments = 2 * K

    total = len(E_range)
    for i, E in enumerate(E_range):
        if verbose and i % 1000 == 0:
            print(f"   ℓ={ell} Progress: {i}/{total} ({100*i/total:.1f}%)")

        if E >= 0:
            continue

        try:
            moments = generate_moments_hydrogen_mpmath(E, ell, num_moments)

            if any(not mp.isfinite(m) for m in moments):
                continue

            M = construct_hankel_matrix_mpmath(moments, K)
            M_rescaled = rescale_hankel_matrix_mpmath(M)

            if is_positive_definite_mpmath(M_rescaled):
                allowed_energies.append(E)

        except:
            continue

    return allowed_energies

def find_energy_intervals(energies, max_gap=None):
    """Find energy intervals and return their centers and widths."""
    if len(energies) == 0:
        return [], []

```

```
    energies = sorted(energies)

    if max_gap is None:
        if len(energies) > 1:
            gaps = np.diff(energies)
            max_gap = 2 * np.median(gaps)
        else:
            max_gap = 1e-6

    intervals = []
    start = energies[0]
    end = energies[0]

    for i in range(1, len(energies)):
        if energies[i] - end <= max_gap:
            end = energies[i]
        else:
            intervals.append((start, end))
            start = energies[i]
            end = energies[i]

    intervals.append((start, end))

    # Calculate centers and half-widths for error bars
    centers = [(a + b) / 2 for a, b in intervals]
    half_widths = [(b - a) / 2 for a, b in intervals]

    return centers, half_widths

def exact_hydrogen_energies(ell, num_levels=20):
    """Exact hydrogen energies: E_n = -1/(2n^2) where n = n_r + ℓ+ 1"""
    energies = []
    for n_r in range(num_levels):
        n = n_r + ell + 1
        E = -1.0 / (2.0 * n**2)
        energies.append(E)
    return energies

```

```
160 # ================================================
161 # PARALLEL PROCESSING WRAPPER
162 # ================================================
163
164 def process_single_ell(ell, K, E_min, E_max, num_points, step_size,
        verbose=False):
165     """
166     Process a single ℓvalue - designed to be called in parallel.
167     Returns tuple: (ell, centers, half_widths, computation_time)
168     """
169     ell_start = time.time()
170
171     print(f"Processing \ell = {ell}...")
172
173     E_range = np.linspace(E_min, E_max, num_points)
174     allowed = bootstrap_hydrogen_level(ell, K, E_range, verbose=verbose)
175
176     print(f" \ell={ell}: Found {len(allowed)} allowed points")
177
178     centers, half_widths = find_energy_intervals(allowed,
            max_gap=3*step_size)
179
180     ell_time = time.time() - ell_start
181     print(f" \ell={ell}: Identified {len(centers)} energy level(s) in
            {ell_time/60:.2f} minutes")
182
183     return (ell, centers, half_widths, ell_time)
184
185 print("="*80)
186 print("CREATING BOOTSTRAPPED HYDROGEN SPECTRUM (PARALLEL)")
187 print("="*80)
188
189 # Parameters
190 K = 50
191 ell_values = list(range(1, 11)) # ℓ= 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
192
193 # Energy range
194 E_min = -0.15
```

```
E_max = -0.0001
num_points = 100000
step_size = 1.5e-6

# CPU information
available_cores = cpu_count()
num_workers = min(len(ell_values), available_cores) # Don't use more
    workers than tasks

print(f" K = {K}")
print(f" ℓ values: {ell_values}")
print(f" Energy range: [{E_min:.4f}, {E_max:.4f}]")
print(f" Test points: {num_points}")
print(f" Precision: {mp.dps} decimal places")
print(f" Available CPU cores: {available_cores}")
print(f" Using {num_workers} parallel workers")
print("="*80)
print()

# Bootstrap for each ℓIN PARALLEL
all_centers = {}
all_errors = {}

start_time = time.time()

# Create partial function with fixed parameters
process_func = partial(
    process_single_ell,
    K=K,
    E_min=E_min,
    E_max=E_max,
    num_points=num_points,
    step_size=step_size,
    verbose=True)

# Run in parallel using multiprocessing Pool
print(f" Starting parallel computation with {num_workers} workers...\n")

```

```
with Pool(processes=num_workers) as pool:
    results = pool.map(process_func, ell_values)

# Unpack results
for ell, centers, half_widths, ell_time in results:
    all_centers[ell] = centers
    all_errors[ell] = half_widths

total_time = time.time() - start_time

print("\n" + "="*80)
print(f" PARALLEL COMPUTATION COMPLETE")
print(f" Total wall-clock time: {total_time/60:.2f} minutes")
print(f" Speedup vs sequential: ~{len(ell_values):.0f}x (theoretical)")
print("="*80)

print("\n" + "="*80)
print("CREATING PLOT")
print("="*80)

for ell in ell_values:
    print(f"\ell={ell}: {len(all_centers[ell])} intervals found")

# Create plot
fig, ax = plt.subplots(figsize=(12, 8))

# Define colors
colors_list = plt.cm.tab10(np.linspace(0, 1, 10))

# Plot each (*$\ell$*)
for idx, ell in enumerate(ell_values):
    centers = all_centers[ell]

    if len(centers) > 0:
        for center in centers:
            y_value = np.log10(-center)
            ax.plot(ell, y_value, 'o', color=colors_list[idx],
                    markersize=12, alpha=0.8, zorder=5)
```

```
270
271         print(f" Plotted {len(centers)} points for \ell={ell}")
272
273 # Plot exact energies as horizontal lines
274 for ell in ell_values:
275     exact_energies_list = exact_hydrogen_energies(ell, num_levels=50)
276     for E_ex in exact_energies_list:
277         if E_min <= E_ex <= E_max:
278             y_ex = np.log10(-E_ex)
279             ax.plot([ell-0.3, ell+0.3], [y_ex, y_ex],
280                     'gray', linewidth=0.8, alpha=0.6, zorder=0)
281
282 # Format the plot
283 ax.set_xlabel('ℓ', fontsize=14, fontweight='bold')
284 ax.set_ylabel('log(-E)', fontsize=14, fontweight='bold')
285 ax.set_title(f'Bootstrapped Hydrogen Spectrum , K = {K}', fontsize=16,
        fontweight='bold')
286 ax.set_xticks(list(ell_values))
287 ax.set_xlim(0.5, 10.5)
288 ax.set_ylim(-5, -0.5)
289 ax.invert_yaxis()
290 ax.grid(True, alpha=0.3)
291
292 # Add legend
293 from matplotlib.lines import Line2D
294 legend_elements = [Line2D([0], [0], marker='o', color='w',
295                           markerfacecolor=colors_list[i], markersize=10,
296                           label=f'(*@$\ell$@*) = {ell}')
297                    for i, ell in enumerate(ell_values)]
298 legend_elements.append(Line2D([0], [0], color='gray', linewidth=2,
299                               label='Exact energies'))
300 ax.legend(handles=legend_elements, loc='center left',
        bbox_to_anchor=(1.02, 0.5), fontsize=11)
301
302 plt.tight_layout()
303 plt.savefig('hydrogen_spectrum_for_various_l.png', dpi=300,
        bbox_inches='tight')
304 plt.show()
```

A.5 Python code for width convergence of fourth and eighth excited levels for different ℓ

This code represents the complete Python implementation used to generate figures 4.3 and 4.5

```python
def width_scan_path(level_n, ell):
    return CACHE_DIR / f"width_v7_level{level_n}_ell{ell}.pkl"

def _merge_intervals(intervals):
    """Merge overlapping or touching intervals."""
    if not intervals:
        return []
    intervals = sorted(intervals, key=lambda pair: pair[0])
    merged = [intervals[0]]
    for low, high in intervals[1:]:
        last_low, last_high = merged[-1]
        if low <= last_high:
            merged[-1] = (last_low, max(last_high, high))
        else:
            merged.append((low, high))
    return merged


def _scan_connected_interval(ell, K, interval, num_points):
    """Scan one connected interval and return allowed sub-intervals."""
    lo, hi = interval
    if hi <= lo or num_points < 2:
        return []

    step = (hi - lo) / mpf(num_points - 1)
    energies = [lo + step * i for i in range(num_points)]
    allowed = [is_hydrogen_positive_definite(E, ell, K) for E in energies]

    bins = []
    current = []
    for idx, ok in enumerate(allowed):
        if ok:
```

```
            current.append(energies[idx])
        elif current:
            bins.append(current)
            current = []
    if current:
        bins.append(current)

    out = []
    for values in bins:
        left = max(lo, values[0] - step)
        right = min(hi, values[-1] + step)
        if right > left:
            out.append((left, right))

    return _merge_intervals(out)


def _multiband_check(ell, K, intervals, npts_per_band=50):
    """Re-grid each connected interval and union the resulting allowed
        windows."""
    if not intervals:
        return []

    collected = []
    for idx, interval in enumerate(intervals):
        points = npts_per_band * 10 if idx == len(intervals) - 1 else
            npts_per_band
        collected.extend(_scan_connected_interval(ell, K, interval, points))

    return _merge_intervals(collected)


def _interval_midpoint(iv):
    return (iv[0] + iv[1]) / 2


def _select_interval_for_target(intervals, target, previous_interval=None):
    """
```

```
    Choose one interval branch for width tracking.

    Priority:
    1) interval containing target,
    2) interval closest to previous branch midpoint,
    3) interval closest to target midpoint.
    """
    if not intervals:
        return None

    containing = interval_containing_energy(intervals, target)
    if containing is not None:
        return containing

    if previous_interval is not None:
        prev_mid = _interval_midpoint(previous_interval)
        return min(intervals, key=lambda iv: abs(_interval_midpoint(iv) -
            prev_mid))

    return min(intervals, key=lambda iv: abs(_interval_midpoint(iv) -
        target))


def _target_interval_via_recursive_scan(ell, target, K_min, K_max,
    min_energy, max_energy):
    """ Track one interval branch over K using recursive interval
        narrowing."""
    rows = []

    searched = _scan_connected_interval(
        ell=ell,
        K=K_min,
        interval=(min_energy, max_energy),
        num_points=1000, )

    tracked_prev = None
    for K in range(K_min, K_max + 1):
        if K > K_min:
```

```
103             searched = _multiband_check(ell=ell, K=K, intervals=searched,
                    npts_per_band=50)
104
105         tracked = _select_interval_for_target(searched, target,
                tracked_prev)
106         if tracked is None:
107             rows.append({
108                 "K": K,
109                 "width": None,
110                 "left": None,
111                 "right": None,
112                 "target": str(target),
113             })
114             tracked_prev = None
115             print(f"ell={ell}, K={K}: no interval selected")
116         else:
117             left, right = tracked
118             width = interval_width(tracked)
119             rows.append({
120                 "K": K,
121                 "width": str(width),
122                 "left": str(left),
123                 "right": str(right),
124                 "target": str(target),
125             })
126             tracked_prev = tracked
127             print(f"ell={ell}, K={K}: width={width}")
128
129     return rows
130
131
132 def track_interval_widths(level_n, ell_values, K_min=10, K_max=50,
        force=False):
133     """ Track interval widths vs K for a fixed hydrogen level n. """
134     results = {}
135     min_energy = PAPER_PARAMS["energy_min"]
136     max_energy = PAPER_PARAMS["energy_max"]
137
```

```
    for ell in ell_values:
        path = width_scan_path(level_n, ell)
        if path.exists() and not force:
            print(f"Loading cached widths for n={level_n}, ell={ell}")
            results[ell] = load_cache(path)
            continue

        target = exact_hydrogen_energy(level_n, ell)
        rows = _target_interval_via_recursive_scan(
            ell=ell,
            target=target,
            K_min=K_min,
            K_max=K_max,
            min_energy=min_energy,
            max_energy=max_energy,)

        payload = {
            "level_n": level_n,
            "ell": ell,
            "rows": rows,}
        save_cache(path, payload)
        results[ell] = payload

    return results


def decode_width_results(payload):
    decoded = []
    for row in payload["rows"]:
        decoded.append((row["K"], None if row["width"] is None else
            mpf(row["width"])))
    return decoded


def plot_width_convergence(width_payloads, title):
    fig, ax = plt.subplots(figsize=(10.6, 7.6))

    for ell in sorted(width_payloads):
```

```
        decoded = decode_width_results(width_payloads[ell])
        x = [K for K, width in decoded if width is not None]
        y = [float(width) for _, width in decoded if width is not None]
        ax.plot(x, y, linewidth=1.8, label=f"L = {ell}")

    ax.set_yscale("log")
    ax.set_xlabel("K", fontsize=13)
    ax.set_ylabel("width", fontsize=13)
    ax.set_title(title, fontsize=17)
    ax.grid(axis="y", color=PLOT_STYLE["grid"], linewidth=0.8)
    ax.legend(frameon=False)
    plt.tight_layout()
    return fig, ax

FIG3_RUN_PRODUCTION = True
FIG3_FORCE_RERUN = True
FIG4_RUN_PRODUCTION = True
FIG4_FORCE_RERUN = False

fig3_widths = {}
fig4_widths = {}


def track_widths_for_excited_level(excited_index, ell_values, force=False):
    """
    Track widths for a fixed excited level across varying ell.

    The paper's "excited level n = m" corresponds to fixed principal
        quantum
    number N = m + 1, with radial index depending on ell:
        radial_n(ell) = N - ell.
    """
    principal_N = excited_index + 1
    result = {}
    for ell in ell_values:
        radial_n = principal_N - ell
        if radial_n < 1:
            continue
```

```
212
213         payload = track_interval_widths(
214             level_n=radial_n,
215             ell_values=[ell],
216             K_min=PAPER_PARAMS["K_min"],
217             K_max=PAPER_PARAMS["K_max"],
218             force=force,)
219         result[ell] = payload[ell]
220         print(
221             f"excited={excited_index}, ell={ell}: using radial
                n={radial_n}, "
222             f"K-range {PAPER_PARAMS['K_min']}..{PAPER_PARAMS['K_max']}" )
223
224     return result
225
226
227 if FIG3_RUN_PRODUCTION:
228     fig3_widths = track_widths_for_excited_level(
229         excited_index=4,
230         ell_values=range(1, 5),
231         force=FIG3_FORCE_RERUN,
232     )
233 else:
234     print("FIG4.3_RUN_PRODUCTION is False.")
235
236 if FIG4_RUN_PRODUCTION:
237     fig4_widths = track_widths_for_excited_level(
238         excited_index=8,
239         ell_values=range(1, 9),
240         force=FIG4_FORCE_RERUN,
241     )
242 else:
243     print("FIG4.5_RUN_PRODUCTION is False. Set it to True to compute the
        n=8 convergence data.")
244
245 if fig3_widths:
246     fig3, ax3 = plot_width_convergence(fig3_widths, "Interval width vs K
        for excited level n = 4")
```

```
247     fig3.savefig(OUTPUT_DIR / "figure-4.3-width-n4.png", dpi=300,
            bbox_inches="tight")
248     plt.show()
249 else:
250     print("No Fig. 4.3 convergence results loaded yet.")
251 
252 if fig4_widths:
253     fig4, ax4 = plot_width_convergence(fig4_widths, "Interval width vs K
            for excited level n = 8")
254     fig4.savefig(OUTPUT_DIR / "figure-4.5-width-n8.png", dpi=300,
            bbox_inches="tight")
255     plt.show()
256 else:
257     print("No Fig. 4.5 convergence results loaded yet.")
258 
259 for level_n, payloads in [(4, fig3_widths), (8, fig4_widths)]:
260     print(f"\nChecking target containment for n={level_n}")
261     for ell in sorted(payloads):
262         target = exact_hydrogen_energy(level_n, ell)
263         misses = []
264         for row in payloads[ell]["rows"]:
265             if row["left"] is None:
266                 misses.append((row["K"], "missing"))
267                 continue
268             left = mp.mpf(row["left"])
269             right = mp.mpf(row["right"])
270             if not (left <= target <= right):
271                 misses.append((row["K"], float(target), float(left),
                        float(right)))
272         print(f" ell={ell}: misses={len(misses)}")
273         if misses:
274             print("  first miss:", misses[0])
275 
276 
277 fig2_scans = {}
278 fig2_summaries = {}
279 for ell in range(PAPER_PARAMS["ell_min"], PAPER_PARAMS["ell_max"] + 1):
280     scan = cached_full_scan(
```

```
        ell=ell,
        K=PAPER_PARAMS["K_max"],
        E_min=PAPER_PARAMS["energy_min"],
        E_max=PAPER_PARAMS["energy_max"],
        step=PAPER_PARAMS["energy_step"],
        force=False,
    )
    fig2_scans[ell] = scan
    fig2_summaries[ell] =
        summarize_intervals(decode_intervals(scan["intervals"]))
    print(f"ell={ell}: resolved {len(fig2_summaries[ell])} intervals")

def is_hydrogen_pd_offset(E, ell, K, offset):
    moments = hydrogen_moments(E, ell, 2 * (K - 1) + 1 + offset)
    M = mp.matrix(K, K)
    for i in range(K):
        for j in range(K):
            M[i, j] = moments[i + j + offset]
    try:
        cholesky(M)
        return True
    except Exception:
        return False


def quick_width_for_predicate(level_n, ell, K, pred):
    target = exact_hydrogen_energy(level_n, ell)
    if not pred(target):
        return None

    def locate(direction):
        lo = target
        hi = target
        step = mp.mpf('1e-10')
        if direction < 0:
            while pred(lo) and lo > PAPER_PARAMS['energy_min']:
                hi = lo
                lo -= step
```

```
318                 step *= 2
319             a, b = lo, hi
320             for _ in range(80):
321                 mid = (a + b) / 2
322                 if pred(mid):
323                     b = mid
324                 else:
325                     a = mid
326             return b
327         else:
328             while pred(hi) and hi < PAPER_PARAMS['energy_max']:
329                 lo = hi
330                 hi += step
331                 step *= 2
332             a, b = lo, hi
333             for _ in range(80):
334                 mid = (a + b) / 2
335                 if pred(mid):
336                     a = mid
337                 else:
338                     b = mid
339             return a
340 
341     left = locate(-1)
342     right = locate(+1)
343     return right - left
344 
345 for level_n, ell in [(4,1), (4,2), (8,1), (8,8)]:
346     w0 = quick_width_for_predicate(level_n, ell, 50, lambda E:
        is_hydrogen_pd_offset(E, ell, 50, 0))
347     w2 = quick_width_for_predicate(level_n, ell, 50, lambda E:
        is_hydrogen_pd_offset(E, ell, 50, 2))
348     print(f"n={level_n}, ell={ell}: offset0={w0}, offset2={w2}")
349 
350 def is_hydrogen_pd_stieltjes(E, ell, K):
351     moments = hydrogen_moments(E, ell, 2 * K)
352     M0 = mp.matrix(K, K)
353     M1 = mp.matrix(K, K)
```

```
    for i in range(K):
        for j in range(K):
            M0[i, j] = moments[i + j]
            M1[i, j] = moments[i + j + 1]
    try:
        cholesky(M0)
        cholesky(M1)
        return True
    except Exception:
        return False

for level_n, ell in [(4,1), (4,2), (8,1), (8,8)]:
    ws = quick_width_for_predicate(level_n, ell, 50, lambda E:
        is_hydrogen_pd_stieltjes(E, ell, 50))
    print(f"stieltjes n={level_n}, ell={ell}: width={ws}")

def local_intervals(pred, E_min, E_max, step):
    energies = []
    e = E_min
    while e <= E_max:
        energies.append(e)
        e += step
    flags = [pred(E) for E in energies]
    out = []
    inside = False
    start = None
    for i, ok in enumerate(flags):
        if ok and not inside:
            start = energies[i]
            inside = True
        if inside and ((not ok) or i == len(flags)-1):
            end = energies[i-1] if not ok else energies[i]
            out.append((start, end, end-start))
            inside = False
    return out

ell = 1
K = 50
```

```
391 target_n4 = exact_hydrogen_energy(4, ell)
392 window = mp.mpf('0.01')
393 iv_plain = local_intervals(lambda E: is_hydrogen_positive_definite(E, ell,
        K), target_n4-window, target_n4+window, mp.mpf('5e-6'))
394 print('plain intervals near n=4, ell=1, K=50:', len(iv_plain))
395 for a,b,w in iv_plain[:10]:
396     contains = (a <= target_n4 <= b)
397     print(' ', a, b, 'width=', w, 'contains target=', contains)
398
399 iv_st = local_intervals(lambda E: is_hydrogen_pd_stieltjes(E, ell, K),
        target_n4-window, target_n4+window, mp.mpf('5e-6'))
400 print('stieltjes intervals near n=4, ell=1, K=50:', len(iv_st))
401 for a,b,w in iv_st[:10]:
402     contains = (a <= target_n4 <= b)
403     print(' ', a, b, 'width=', w, 'contains target=', contains)
404
405 import pickle
406 from pathlib import Path
407
408 cache_dir = CACHE_DIR
409
410 def k50_width_from_file(path):
411     with open(path, 'rb') as f:
412         payload = pickle.load(f)
413     row50 = next(r for r in payload['rows'] if r['K'] == 50)
414     return None if row50['width'] is None else mp.mpf(row50['width'])
415
416 for version in ['width_level', 'width_v2_level', 'width_v3_level',
        'width_v4_level']:
417     p = cache_dir / f"{version}4_ell1.pkl"
418     if p.exists():
419         print(version, 'n4 ell1 K50 width =', k50_width_from_file(p))
420
421 for version in ['width_level', 'width_v2_level', 'width_v3_level',
        'width_v4_level']:
422     p = cache_dir / f"{version}8_ell8.pkl"
423     if p.exists():
424         print(version, 'n8 ell8 K50 width =', k50_width_from_file(p))
```

```
425
426 def edge_errors_at_K(width_payloads, level_n, K=50):
427     print(f"\nEdge errors at K={K}, n={level_n}")
428     for ell in sorted(width_payloads):
429         row = next(r for r in width_payloads[ell]['rows'] if r['K'] == K)
430         if row['left'] is None:
431             print(f" ell={ell}: missing")
432             continue
433         left = mp.mpf(row['left'])
434         right = mp.mpf(row['right'])
435         target = exact_hydrogen_energy(level_n, ell)
436         left_err = target - left
437         right_err = right - target
438         print(f" ell={ell}: width={right-left}, left_err={left_err},
              right_err={right_err}")
439
440 edge_errors_at_K(fig3_widths, 4)
441 edge_errors_at_K(fig4_widths, 8)
```