UNIVERSITY OF CALIFORNIA

SANTA CRUZ

FORMALIZATION OF WEYL'S LAW
AND
ITS IMPLEMENTATION INTO LEAN CODE

A thesis submitted in partial satisfaction of the
requirements for the degree of

MASTER OF ARTS

in

MATHEMATICS

by

ZhiBin Huang

December 2025

The Thesis of ZhiBin Huang
is approved:

_____
Professor Pedro F Morales
Almazan

_____
Professor Francois Monard

_____
Professor Bob Leo Hingtgen

_____
Peter Biehl
Vice Provost and Dean of Graduate Studies

Contents

Abstract

Formalization of Weyl's Law and its implementation into Lean code

ZHIBIN HUANG

This thesis explores the formalization of Weyl's Law and its implementation in the Lean 4 proof assistant, bridging classical mathematical analysis with modern computational verification techniques. Under the guidance of Professor Pedro Morales-Almazan at UCSC, this research investigates the asymptotic growth behavior of eigenvalues in the Laplace eigenvalue problem across various bounded domains including intervals, rectangles, and disks.

The project systematically develops the theoretical foundations of the Laplace operator in one, two, and three dimensions, addressing the eigenvalue problem $-\Delta u = \lambda u$ across different coordinate systems and geometries. In this work, we consider the Laplacian with Dirichlet boundary conditions, which ensure that the eigenvalue problem is well defined and admits a discrete spectrum. A significant contribution lies in the translation of classical mathematical proofs into formally verified code using Lean 4, supported by contemporary large language models.

Through this work, we demonstrate the challenges and opportunities in formalizing complex mathematical theories, highlighting the interplay between human mathematical insight and computational verification. The research contributes to the growing body of formally verified mathematics while providing insights into the future of mathematical proof in the age of computational assistants. This thesis represents ongoing work in understanding the deep nature of mathematical proof and establishes a foundation for further formalization of spectral theory results in proof assistants.

Acknowledgments

I would like to express my sincere gratitude to my advisor, Professor Pedro Morales-Almazan, We started the project in March 2025 until December 2025! During the summer, his continuous support, patience, and immense knowledge answered a lot of questions I had. His guidance helped me throughout the research and writing of this thesis. I finally accomplished the goal!

I would like to thank Advisor Lyss Melton and Vice Chair Longzhi Lin for answering my thesis preparation questions and checking in with me. I would also like thank to Professor Francois Monard and Professor Bob Leo Hingtgen for being on my thesis Committee! Finally, thank you to the entire mathematics department at UCSC for providing a stimulating and supportive environment for learning and research.

Furthermore, I would like to acknowledge the developers of Lean and mathlib for creating such powerful tools for formal mathematics, and the creators of the deepseek model for their open-source contributions that assisted in this work.

Finally, I must express my profound gratitude to my family and friends for their unwavering support and encouragement throughout my studies.

Chapter 1: Preface

In Fall 2024, I began my graduate studies in mathematics at UCSC, with my background in computer sciences and mathematics. It was my great honor to work under the guidance of my advisor, Professor Pedro Morales-Almazan, and a new "crazy" idea began to take shape, one that bridges classical analysis with modern proof technology. This thesis project is still early work. Our goal is not to rush toward completion but to understand deeply what it means to "proof" mathematics. We discussed a lot of ideas and found that a shared interest topic was Weyl's Theorem.

The thesis will bring the idea of the Laplace eigenvalue problem gives us a sequence of eigenvalues, and that Weyl's law describes the asymptotic growth rate of these eigenvalues, depending on the dimension of the domain where the equation is defined. Therefore, we also check the Laplace equation in bounded domains, for example, a segment, a rectangle, a disk, etc. While we studied those topics in a physical book and notes, formalizing this result within a proof assistant like Lean—guided and supported by modern large language models—offered an entirely new mode of engagement. To be honest, it's taken a long time for us and it's not easy to "translate" between the mathematical proof language and LEAN.

Inspired by the amazing work and open resources online and motivated by contemporary tools like Lean 4, mathlib, and LLM Deepseek, our project represents the theory from the ground up. This involved defining the Laplace operator in 1D, 2D, and 3D settings, expressing associated eigenvalue problems ($-\Delta u = \lambda u$), and attempting to formalize parts of Weyl's Law itself. In doing so, we encountered not only the expected difficulties of formalization such as definitions, domains and calculation.

This project couldn't have been a success without the support and mentorship of Professor Pedro Morales-Almazan, whose patience and mathematical insight guided each step. I also acknowledge the influence of the UCSC teaching and research environment, which continues to foster innovation at the intersection of classical and computational thought. In my master's thesis, I will present all my research work. Thank you.

ZhiBin Huang
Santa Cruz, California, USA
March 2025

Introduction

Have you ever wondered how scientists figured out the rules behind light and heat? This story begins almost 200 years ago and connects mathematics, physics, and thermodynamics in a fascinating way.

The Beginning: A "Perfect" Object

In 1859 [1], a scientist named Kirchhoff proposed an interesting idea: an idealized object he called a "blackbody." This blackbody had a special characteristic: it could absorb all light shining on it and also emit light of all wavelengths. Most importantly, the light it emitted **depended only on its temperature** and had nothing to do with what material it was made of or its shape.

Another way you can understood as: whether you use an iron block or a ceramic cup, heating them to the same temperature will make them glow with the same "color" of light. This concept started a great milestone and research for a "universal formula" to describe this light.

Progress: Inspiration from Sound

Later, in the late 19th century [1], another great scientist, Lord Rayleigh, entered the scene. He mainly studied sound, like how sound vibrates and reflects in a square room. He discovered something interesting: as the frequency of sound (which you can think of as the pitch) gets higher, the number of possible vibration modes increases very rapidly, following the **cube** of the frequency.

A simple analogy: a container can hold far more high-frequency "sound particles" than low-frequency ones. Through calculation, he found an approximate rule:

$$N(\nu) \sim V\nu^3 \tag{1}$$

So what's that mean: (The number of vibration modes N is roughly equal to the room's Volume V, multiplied by the frequency ν cubed). From this formula, we can see that higher frequencies lead to a greater "density" of modes. Specifically, the density is proportional to the square of the frequency:

$$D(\nu) = \frac{dN}{d\nu} \sim \nu^2 \tag{2}$$

(The mode density D is roughly equal to the frequency ν squared). This discovery was very powerful because it was the first time a link was made between the **geometric size of a space (volume)** and a **physical phenomenon (the frequency of sound or light)**, which also led the concepts later become "Weyl's Law."

Conflict

Scientists quickly applied Rayleigh's ideas about sound to light (electromagnetic waves). Based on classical physics, they derived a formula to describe blackbody radiation, called the "Rayleigh-Jeans law [1]":

$$\rho(\nu, T) = \frac{8\pi\nu^2}{c^3} k_B T \tag{3}$$

(The energy density ρ is proportional to the frequency ν squared and the temperature T). But this formula had a big problem! It predicted that when an object emits high-frequency light (like ultraviolet light), the energy would become infinite. This is obviously impossible, as otherwise everything around us would be scorched by infinite energy. This absurd conclusion became known in history as the "**Ultraviolet Catastrophe**." Classical physics had hit a solid wall.

The Savior: Planck's "Quantum" Breakthrough

Then, in 1900, Max Planck proposed a groundbreaking solution. He found a new formula that perfectly matched all the experimental data [1]:

$$\rho_P(\nu, T) = \frac{8\pi\nu^2}{c^3} \cdot \frac{h\nu}{e^{h\nu/k_B T} - 1} \tag{4}$$

This formula looks complex, but it hid an even more revolutionary idea: **energy does not change continuously, but comes in tiny packets**. Planck suggested that light can only be absorbed or emitted in minimum "packets" of energy. The size of each energy packet is equal to $h\nu$ (where h is a constant, and ν is the frequency). It's like you can't pay with any small amount of change, but only with fixed coins like 10 cents, 50 cents, or 1 dollar. This concept of "energy coins" (quanta) forcefully reined in the energy that was rushing towards "infinity," solving the ultraviolet catastrophe. Planck's formula became the starting point of quantum mechanics.

Unification: Mathematics Accomplish the Deal – Weyl's Law

So, the problem is: Was Rayleigh's original estimate correct? The physicist Sommerfeld conjectured that the number of high-frequency vibration modes in a container **depends only on its volume, not its specific shape**. This echoed Kirchhoff's original idea. Finally, in 1911, the mathematician Hermann Weyl provided a clear **mathematical proof**, whose conclusion is the famous "**Weyl's Law**." For a d-dimensional space, it is expressed as [1]:

$$N(\lambda) \sim \frac{\omega_d}{(2\pi)^d} |\Omega| \lambda^{d/2} \quad \text{as } \lambda \to \infty \tag{5}$$

(The total number of vibration modes N below a frequency threshold λ is determined mainly by the volume of the space $|\Omega|$ and λ raised to the power of $d/2$). This law mathematically confirmed that the basic premise used by Rayleigh and Planck was completely correct.

Chapter 2: The Laplace–Beltrami Operator

Chapter 2.1: Dirichlet eigenvalues of the Laplacian

The basic eigenvalue problem with Dirichlet boundary conditions is

$$-\Delta u = \lambda u \quad \text{in } D, \qquad u = 0 \quad \text{on } \partial D,$$

where $D \subset \mathbb{R}^n$ is an open, bounded domain with piecewise smooth boundary ∂D.

This problem asks for nontrivial solutions $u \neq 0$ and scalars λ such that the Laplacian acts like a "frequency" operator.

1. u **are called** eigenfunctions.
2. λ are the corresponding eigenvalues.

In physics, this problem describes the vibration modes of a membrane or drumhead that is fixed along its boundary. The condition $u = 0$ on ∂D means the membrane cannot move at the edge.

Chapter 2.2: Spectral theorem for compact self-adjoint operators

In mathematics, the Dirichlet Laplacian $-\Delta$ is an unbounded, self-adjoint operator on $L^2(D)$, but its *resolvent* $(-\Delta + I)^{-1}$ is compact and self-adjoint. This property allows us to apply the spectral theorem for compact, self-adjoint operators to the Dirichlet eigenvalue problem. Therefore, the spectral theorem guarantees that:

1. All Dirichlet eigenvalues λ are *real and positive*.
2. Each eigenspace is *finite-dimensional*.
3. The eigenvalues can be arranged in increasing order.

Thus, the spectrum is discrete [4]:

$$0 < \lambda_1 < \lambda_2 \leqslant \lambda_3 \leqslant \cdots \to \infty.$$

Each λ_k has finite multiplicity, and the eigenfunctions $\{u_k\}$ form an orthonormal basis of $L^2(D)$.

Thus, the spectrum consists only of discrete eigenvalues, since for the Dirichlet Laplacian the resolvent is compact and therefore the essential spectrum is empty. Physically, the eigenfunctions correspond to the vibration modes of a membrane fixed along its boundary, while the eigenvalues represent the squared frequencies of vibration. This discrete spectral structure provides the foundation for studying asymptotic phenomena such as Weyl's law.

Chapter 2.3: Laplace Operator Formula in Cartesian and Curvilinear coordinates

The Laplacian is a differential operator named after Pierre-Simon Laplace. Its most general and elegant definition is as the divergence of the gradient of a function:

$$\Delta f = \bigtriangledown^2 f = \bigtriangledown \cdot (\bigtriangledown f)$$

In my own understanding of that notation:

- ∇f is the gradient, which captures the rate and direction of change of the scalar function f, resulting in a vector field. The gradient points in the direction of maximal increase of the function, while its negative gives the direction of steepest decrease.
- $\nabla \cdot$ is the divergence, which measures the magnitude of a source or sink at a given point in a vector field, resulting in a scalar.

Another way to understand the definition is: the Laplacian Δf at a point measures the difference between the average value of f in a small neighborhood around that point and the value of f at the point itself. However, its explicit form depends on the number of dimensions and the geometry (coordinate system) of the space.

1. In Cartesian coordinates, the general formula of the Laplace operator:

$$\Delta f = \sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_i^2}.$$

However, in curvilinear coordinates such as polar (2D) or cylindrical(3D) and spherical (3D), the formula will be different from the Cartesian coordinates; therefore, in curvilinear coordinates, we will use the formula named the Orthogonal Curvilinear formula:

2. Orthogonal Curvilinear coordinate formula:

$$\Delta f = \frac{1}{h_1 h_2 \cdots h_n} \sum_{i=1}^{n} \frac{\partial}{\partial q_i} \left(\frac{h_1 h_2 \cdots h_n}{h_i^2} \frac{\partial f}{\partial q_i} \right),$$

where q_i are the orthogonal curvilinear coordinates (e.g. r, θ, ϕ in spherical coordinates), and h_i are the corresponding *scale factors* that describe how unit distances stretch in each coordinate direction:

$$h_i = \left| \frac{\partial \mathbf{r}}{\partial q_i} \right|.$$

For example, in spherical coordinates (r, θ, ϕ) we have $h_1 = 1$, $h_2 = r$, and $h_3 = r \sin\theta$.

Of course, it doesn't look very clear at first, but we will use both formulas in the next chapter.

Chapter 2.4: Explicit Forms in Different Dimensions and Geometries

In this section, we illustrate the Laplacian in 1D - 3D :

- 1D: interval case (such as intervals and vibrations of a string):

  There is only one variable, x. The Laplacian simplifies to the ordinary second derivative:

$$\Delta f = \frac{d^2 f}{dx^2}$$

- 2D Cartesian Coordinates (such as surface, plane, vibrating drumhead):

  In the Cartesian plane, with variables x and y, the Laplacian is the sum of the second partial derivatives:

$$\Delta f = \frac{d^2 f}{dx^2} + \frac{d^2 f}{dy^2}$$

- 2D Curvilinear coordinates: (such as circular and heat diffusion from a point source) In polar coordinates, (r, ϕ) is more efficient. The Laplacian transforms to:

$$\Delta f = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial f}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 f}{\partial \theta^2} = \frac{\partial^2 f}{\partial r^2} + \frac{1}{r}\frac{\partial f}{\partial r} + \frac{1}{r^2}\frac{\partial f}{\partial \theta^2}.$$

The choice of coordinate system in 3D is paramount and is dictated by the symmetry of the problem. Using the appropriate system can turn a intractable partial differential equation into a separable one.

- 3D Cartesian Coordinates:

  In Cartesian coordinates (x, y, z) , the Laplacian extends naturally:

$$\Delta f = \frac{d^2 f}{dx^2} + \frac{d^2 f}{dy^2} + \frac{d^2 f}{dz^2}$$

- 3D Curvilinear coordinates: Cylindrical Coordinates (r, ϕ, z),

  In the cylindrical case, separation of variables leads to Bessel functions of the first kind

$$\Delta f = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial f}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 f}{\partial \phi^2} + \frac{\partial^2 f}{\partial z^2}$$

- 3D Curvilinear coordinates: Spherical coordinates (r, θ, ϕ),

  Using spherical coordinates leads to spherical Bessel functions

$$\Delta f = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial f}{\partial r}\right) + \frac{1}{r^2 \sin\theta}\frac{\partial}{\partial \theta}\left(\sin\theta\frac{\partial f}{\partial \theta}\right) + \frac{1}{r^2 \sin^2\theta}\frac{\partial^2 f}{\partial \phi^2}.$$

Chapter 2.5: Analytical Solutions of Eigenvalue Problems

1D Dirichlet problem on an interval

In the one–dimensional Dirichlet case, all eigenvalues are simple, in contrast with higher-dimensional settings in which angular symmetry leads to multiplicities.

We first work on the interval $D = [0, L]$. The Dirichlet eigenvalue problem reads

$$-\Delta f = \lambda f \quad \Longrightarrow \quad -\frac{d^2 f}{dx^2} = \lambda f, \qquad f(0) = f(L) = 0.$$

As I discussed with Professor Pedro, the function that works will be $f(x) = sin(n\pi x)$:

$$f(x) = \sin\Big(\frac{n\pi x}{L}\Big), \qquad n = 1, 2, 3, \ldots.$$

Let's us now verify this.

- First derivative:
$$f'(x) = \frac{n\pi}{L}\cos\Big(\frac{n\pi x}{L}\Big).$$
- Second derivative:
$$f''(x) = -\left(\frac{n\pi}{L}\right)^2 \sin\Big(\frac{n\pi x}{L}\Big) = -\left(\frac{n\pi}{L}\right)^2 f(x).$$

  Substituting into the eigenvalue equation gives

$$-f''(x) = \left(\frac{n\pi}{L}\right)^2 f(x),$$

so f is indeed an eigenfunction with eigenvalue

$$\lambda_n = \left(\frac{n\pi}{L}\right)^2, \qquad n = 1, 2, 3, \ldots.$$

Equivalently, solving the ODE in full: the general solution of $-f'' = \lambda f$ is

$$f(x) = A\cos(\sqrt{\lambda}\,x) + B\sin(\sqrt{\lambda}\,x)$$

The condition $f(0) = 0$ implies $A = 0$, so $f(x) = B\sin(\sqrt{\lambda}\,x)$
The boundary condition $f(L) = 0$ then requires $\sin(\sqrt{\lambda}\,L) = 0$, $\sqrt{\lambda}\,L = n\pi$ for $n \in \mathbb{Z}$.

Thus the Dirichlet spectrum on $[0, L]$ is

$$\lambda_n = \left(\frac{n\pi}{L}\right)^2, \quad f_n(x) = \sin\Big(\frac{n\pi x}{L}\Big), \qquad n = 1, 2, 3, \ldots.$$

where λ_n is eigenvalue and $f_n(x)$ is eigenfunction.

2D Dirichlet problem on a rectangle

Let $D = [0, a] \times [0, b]$. The Dirichlet eigenvalue problem for the Laplacian is

$$-\Delta f = \lambda f, \qquad -\Delta f = -\frac{\partial^2 f}{\partial x^2} - \frac{\partial^2 f}{\partial y^2},$$

with boundary condition $f(0, y) = f(a, y) = f(x, 0) = f(x, b) = 0$,
For all admissible (x, y). Here λ is a constant eigenvalue.
As I discussed with Professor Pedro, let's make an assumption, assume

$$f(x, y) = X(x)\, Y(y).$$

Substituting into the PDE gives

$$-\Big(X''(x)Y(y) + X(x)Y''(y)\Big) = \lambda X(x)Y(y).$$

Dividing both sides by $X(x)Y(y)$

$$-\frac{X''(x)}{X(x)} - \frac{Y''(y)}{Y(y)} = \lambda.$$

Since the left-hand side is a sum of a function of x only and a function of y only.
Each term must be constant. therefore;

$$u + v = \lambda \qquad -\frac{X''(x)}{X(x)} = u \text{ and } -\frac{Y''(y)}{Y(y)} = v.$$

Both X and Y, the ideas is same as the 1D, so we could use the result from 1D Dirichlet problem:

$$X''(x) + uX(x) = 0, \qquad X(0) = X(a) = 0,$$
$$Y''(y) + vY(y) = 0, \qquad Y(0) = Y(b) = 0.$$

From the 1D theory, the solutions are

$$X_n(x) = \sin\Big(\frac{n\pi x}{a}\Big), \quad u = \Big(\tfrac{n\pi}{a}\Big)^2, \quad n = 1, 2, \ldots,$$

$$Y_m(y) = \sin\Big(\frac{m\pi y}{b}\Big), \quad v = \Big(\tfrac{m\pi}{b}\Big)^2, \quad m = 1, 2, \ldots.$$

Thus the eigenfunctions and eigenvalues of the 2D Dirichlet problem on the rectangle are

$$\text{Eigenfunction: } f_{n,m}(x, y) = \sin\Big(\tfrac{n\pi x}{a}\Big)\, \sin\Big(\tfrac{m\pi y}{b}\Big)$$

$$\text{Eigenvalues: } \lambda_{n,m} = \Big(\tfrac{n\pi}{a}\Big)^2 + \Big(\tfrac{m\pi}{b}\Big)^2, \quad n, m = 1, 2, 3, \ldots.$$

2D Dirichlet problem in polar coordinates

We now focus on a domain whose boundary comprises part of a circle; unlike the normal rectangular, we should use polar coordinates. In polar coordinates $x = r\cos\theta$ and $y = r\sin\theta$.

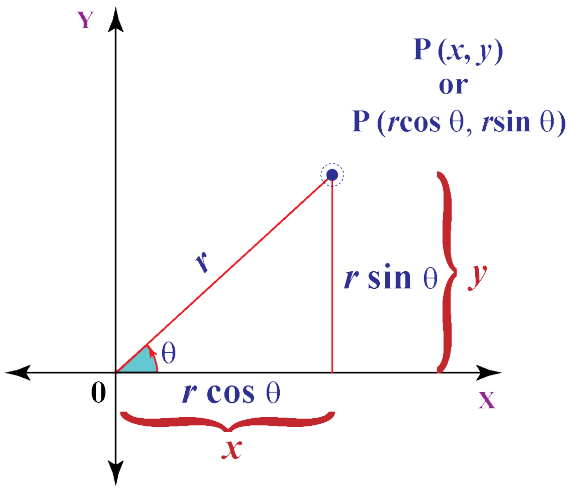


Figure 1: polar coordinates: $x = r\cos\theta$ and $y = r\sin\theta$

Moreover, Using the idea of the Pythagorean theorem: $a^2 + b^2 = c^2$, observing the figure on top, we also found out that $r^2 = (r\cos\theta)^2 + (r\sin\theta)^2 = x^2 + y^2$. So $r = \sqrt{x^2+y^2}$ and $\theta = tan^{-1}(y/x)$.

Now, consider Laplace's operator

$$-\Delta f = \lambda f, \qquad -\Delta f = -\frac{\partial^2 f}{\partial x^2} - \frac{\partial^2 f}{\partial y^2}$$

Since x, y are Cartesian coordinates and r, θ are standard polar coordinates on the plane. To determine Laplace's operator in polar coordinates, apply the chain rule and get:

$$\frac{d}{dx} = \frac{dr}{dx}\frac{d}{dr} + \frac{d\theta}{dx}\frac{d}{d\theta}$$

$$\frac{d}{dy} = \frac{dr}{dy}\frac{d}{dr} + \frac{d\theta}{dy}\frac{d}{d\theta}$$

We pause here, use the fact on top, and figure out some math calculation:

$$\frac{dr}{dx} = \frac{d}{dx}(\sqrt{x^2+y^2}) = \frac{d}{dx}(x^2+y^2)^{1/2} = \frac{1}{2}(x^2+y^2)^{-1/2}\cdot 2x = \frac{x}{\sqrt{x^2+y^2}} = \frac{x}{r} = \frac{r\cos\theta}{r} = \cos\theta$$

$$\frac{d\theta}{dx} = \frac{d}{dx} tan^{-1}\left(\frac{y}{x}\right) = \frac{1}{1+(y/x)^2}\left(\frac{-y}{x^2}\right) = -\frac{y}{x^2+y^2} = \frac{-\sin\theta}{r}$$

$$\frac{dr}{dy} = \frac{d}{dy}(\sqrt{x^2+y^2}) = \frac{d}{dy}(x^2+y^2)^{1/2} = \frac{1}{2}(x^2+y^2)^{-1/2}\cdot 2y = \frac{y}{\sqrt{x^2+y^2}} = \frac{y}{r} = \frac{r\sin\theta}{r} = \sin\theta$$

$$\frac{d\theta}{dy} = \frac{d}{dy} tan^{-1}\left(\frac{y}{x}\right) = \frac{1}{1+(y/x)^2}\left(\frac{1}{x}\right) = \frac{x}{x^2+y^2} = \frac{\cos\theta}{r}$$

Good! We are now ready to move to more calculations:

$$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} = \frac{d}{dx}\frac{d}{dx} + \frac{df}{dy}\frac{d}{dy} = \left(\frac{dr}{dx}\frac{d}{dr} + \frac{d\theta}{dx}\frac{d}{d\theta}\right)^2 + \left(\frac{dr}{dy}\frac{d}{dr} + \frac{d\theta}{dy}\frac{d}{d\theta}\right)^2$$

$$\Delta = \left(\cos\theta\frac{\partial}{\partial r} - \frac{\sin\theta}{r}\frac{\partial}{\partial\theta}\right)^2 + \left(\sin\theta\frac{\partial}{\partial r} + \frac{\cos\theta}{r}\frac{\partial}{\partial\theta}\right)^2$$

$$= \cos^2\theta\frac{\partial^2}{\partial r^2} - \frac{2\cos\theta\sin\theta}{r}\frac{\partial^2}{\partial r\partial\theta} + \frac{\sin^2\theta}{r^2}\frac{\partial^2}{\partial\theta^2} + \sin^2\theta\frac{\partial^2}{\partial r^2} + \frac{2\cos\theta\sin\theta}{r}\frac{\partial^2}{\partial r\partial\theta} + \frac{\cos^2\theta}{r^2}\frac{\partial^2}{\partial\theta^2} + \frac{1}{r}\frac{\partial}{\partial r}$$

The factor $\frac{1}{r}dr$ arises from rewriting the Laplacian in polar coordinates, where the Jacobian of the change of variables introduces an additional r dependent term.
Finally, we obtain the Laplacian in polar coordinates:

$$\boxed{= \frac{\partial^2}{\partial r^2} + \frac{1}{r}\frac{\partial}{\partial r} + \frac{1}{r^2}\frac{\partial^2}{\partial\theta^2}.}$$

It looks like a lot of work already. Is there a simple and second way to calculate a higher dimension? The answer is Yes! Orthogonal Curvilinear coordinate formula:

$$\Delta f = \frac{1}{h_r h_\theta}\sum_{i=1}^{2}\frac{\partial}{\partial q_i}\left(\frac{h_r h_\theta}{h_i^2}\frac{\partial u}{\partial q_i}\right).$$

Let $x = r\cos\theta$, $y = r\sin\theta$, $h_r = 1$ and $h_\theta = r$.

$$\Delta f = \frac{1}{r}\left[\frac{\partial}{\partial r}\left(\frac{r}{1^2}\frac{\partial f}{\partial r}\right) + \frac{\partial}{\partial\theta}\left(\frac{r}{r^2}\frac{\partial f}{\partial\theta}\right)\right].$$

$$\Delta f = \frac{1}{r}\frac{\partial}{\partial r}(r f_r) + \frac{1}{r}\frac{\partial}{\partial\theta}\left(\frac{1}{r}f_\theta\right) = f_{rr} + \frac{1}{r}f_r + \frac{1}{r^2}f_{\theta\theta}.$$

$$\boxed{\Delta f = \frac{1}{r}\frac{\partial}{\partial r}(r\,\partial_r f) + \frac{1}{r^2}\partial_\theta^2 f = \frac{\partial^2}{\partial r^2} + \frac{1}{r}\frac{\partial}{\partial r} + \frac{1}{r^2}\frac{\partial^2}{\partial\theta^2}.}$$

We had now accomplished the first step. using the result and start from the Dirichlet eigenvalue equation in polar coordinates:

$$-\Delta f = \lambda f \quad \Longrightarrow \quad -\Big(\frac{\partial^2}{\partial r^2} + \frac{1}{r}\frac{\partial}{\partial r} + \frac{1}{r^2}\frac{\partial^2}{\partial \theta^2}\Big) f(r,\theta) = \lambda f(r,\theta).$$

Assume $f(r,\theta) = R(r) \cdot \Theta(\theta)$, compute each term and substitute:

$$\begin{aligned}\Delta f &= f_{rr} + \frac{1}{r} f_r + \frac{1}{r^2} f_{\theta\theta} \\ &= R''(r)\Theta(\theta) + \frac{1}{r} R'(r)\Theta(\theta) + \frac{1}{r^2} R(r)\Theta''(\theta).\end{aligned}$$

Thus the PDE equation becomes

$$-\Big[R''(r)\Theta(\theta) + \frac{1}{r} R'(r)\Theta(\theta) + \frac{1}{r^2} R(r)\Theta''(\theta)\Big] = \lambda R(r)\Theta(\theta).$$

To play around the PDE, let's divide both sides by $R(r)\Theta(\theta)$

$$-\frac{R''(r)}{R(r)} - \frac{1}{r}\frac{R'(r)}{R(r)} - \frac{1}{r^2}\frac{\Theta''(\theta)}{\Theta(\theta)} = \lambda.$$

Multiply by r^2 to clear denominators:

$$-r^2\frac{R''(r)}{R(r)} - r\frac{R'(r)}{R(r)} - \frac{\Theta''(\theta)}{\Theta(\theta)} = \lambda r^2.$$

Rearrange so that the θ-dependent part is isolated:

$$\frac{\Theta''(\theta)}{\Theta(\theta)} = -r^2\frac{R''(r)}{R(r)} - r\frac{R'(r)}{R(r)} - \lambda r^2.$$

The left-hand side depends only on θ, while the right-hand side depends only on r. Hence both sides must equal the same constant. Introduce the separation constant $-m^2$

$$\frac{\Theta''(\theta)}{\Theta(\theta)} = -m^2, \qquad m \in \mathbb{R}.$$

$$\Theta''(\theta) + m^2\Theta(\theta) = 0.$$

Plugging back the $\frac{\Theta''}{\Theta} = -m^2$ back into the separated equation gives us:

$$-r^2\frac{R''(r)}{R(r)} - r\frac{R'(r)}{R(r)} + m^2 = \lambda r^2,$$

hence,

$$r^2 R''(r) + rR'(r) + \big(\lambda r^2 - m^2\big) R(r) = 0.$$

As I discussed with Professor Pedro, I realized this topic was new to me; Professor Pedro introduced a direction for deeper study: the Bessel equation.

Let's us continue the previous result:

$$r^2\frac{d^2R}{dr^2} + r\frac{dR}{dr} + (k^2r^2 - m^2)R = 0, \quad \text{where} \quad k^2 = \lambda$$

The bounded solutions at $r = 0$ are Bessel functions of the first kind:

$$R_m(r) = J_m(kr)$$

Applying the Dirichlet boundary condition $f(R, \theta) = 0$ requires:

$$J_m(kR) = 0$$

Let $k_{m,n}$ denote the n-th positive zero of $J_m(x)$. Then:

$$k_{m,n} = \frac{j_{m,n}}{R}, \quad \lambda_{m,n} = \left(\frac{j_{m,n}}{R}\right)^2$$

The eigenfunctions are:

$$f_{m,n}(r, \theta) = J_m\left(\frac{j_{m,n}}{R}r\right)\left[A\cos(m\theta) + B\sin(m\theta)\right]$$

- Case 1 when $m = 0$:

  $$f_{0,n}(r) = J_0\Big(\tfrac{j_{0,n}}{R}\,r\Big).$$

  The angular solution is constant ($\Theta_0(\theta) = 1$), giving a one-dimensional eigenspace for each radial index n.

- Case 2 when $m \geqslant 1$:

  $$f^{\cos}_{m,n}(r, \theta) = J_m\Big(\tfrac{j_{m,n}}{R}\,r\Big)\cos(m\theta),$$
  $$f^{\sin}_{m,n}(r, \theta) = J_m\Big(\tfrac{j_{m,n}}{R}\,r\Big)\sin(m\theta).$$

  Because the angular part provides both $\cos(m\theta)$ and $\sin(m\theta)$ components, making each eigenvalue $\lambda_{m,n}$ at least double it. (In other words, we should always multiplicity 2 when $m = 1$ because the result for *cos* and *sin* is repeat.).

3D Dirichlet problem on a rectangular box

Let $D = [0,a] \times [0,b] \times [0,c]$. The Dirichlet eigenvalue problem for the Laplacian is

$$-\Delta f = \lambda f, \qquad \Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2},$$

with boundary conditions
$f(0,y,z) = f(a,y,z) = f(x,0,z) = f(x,b,z) = f(x,y,0) = f(x,y,c) = 0.$

Assume a product form $f(x,y,z) = X(x)\,Y(y)\,Z(z)$
Substituting into the PDE gives

$$-\big(X''(x)Y(y)Z(z) + X(x)Y''(y)Z(z) + X(x)Y(y)Z''(z)\big) = \lambda X(x)Y(y)Z(z).$$

Dividing by $X(x)Y(y)Z(z)$

$$-\frac{X''(x)}{X(x)} - \frac{Y''(y)}{Y(y)} - \frac{Z''(z)}{Z(z)} = \lambda.$$

Each term depends on a single variable, so set

$$u + v + w = \lambda. \qquad -\frac{X''(x)}{X(x)} = u, \qquad -\frac{Y''(y)}{Y(y)} = v, \qquad -\frac{Z''(z)}{Z(z)} = w,$$

Each factor satisfies a 1D Dirichlet problem:

$$X''(x) + u\,X(x) = 0, \quad X(0) = X(a) = 0,$$
$$Y''(y) + v\,Y(y) = 0, \quad Y(0) = Y(b) = 0,$$
$$Z''(z) + w\,Z(z) = 0, \quad Z(0) = Z(c) = 0.$$

From the 1D results,

$$X_n(x) = \sin\Big(\frac{n\pi x}{a}\Big), \quad u = \Big(\tfrac{n\pi}{a}\Big)^2, \quad n = 1,2,\ldots,$$

$$Y_m(y) = \sin\Big(\frac{m\pi y}{b}\Big), \quad v = \Big(\tfrac{m\pi}{b}\Big)^2, \quad m = 1,2,\ldots,$$

$$Z_k(z) = \sin\Big(\frac{k\pi z}{c}\Big), \quad w = \Big(\tfrac{k\pi}{c}\Big)^2, \quad k = 1,2,\ldots.$$

Thus the eigenfunctions and eigenvalues on the rectangular box are

$$f_{n,m,k}(x,y,z) = \sin\Big(\frac{n\pi x}{a}\Big)\,\sin\Big(\frac{m\pi y}{b}\Big)\,\sin\Big(\frac{k\pi z}{c}\Big),$$

$$\lambda_{n,m,k} = \Big(\tfrac{n\pi}{a}\Big)^2 + \Big(\tfrac{m\pi}{b}\Big)^2 + \Big(\tfrac{k\pi}{c}\Big)^2, \qquad n,m,k = 1,2,3,\ldots.$$

3D Curvilinear coordinates: Cylindrical Coordinates

To make an easy, as we see the example at 2D polar, we will apply the Orthogonal Curvilinear coordinate formula. Let $x = r\cos\theta$, $y = r\sin\theta$, $z = z$, with scale factors

$$h_r = 1, \quad h_\theta = r, \quad h_z = 1.$$

Thus

$$\Delta f = \frac{1}{h_r h_\theta h_z}\sum_{i=1}^{3}\frac{\partial}{\partial q_i}\left(\frac{h_r h_\theta h_z}{h_i^2}\frac{\partial f}{\partial q_i}\right), \qquad (q_1, q_2, q_3) = (r, \theta, z).$$

$$\Delta f = \frac{1}{r}\left[\frac{\partial}{\partial r}\left(\tfrac{r}{1^2}f_r\right) + \frac{\partial}{\partial \theta}\left(\tfrac{r}{r^2}f_\theta\right) + \frac{\partial}{\partial z}\left(\tfrac{r}{1^2}f_z\right)\right].$$

$$\Delta f = \frac{1}{r}\frac{\partial}{\partial r}(rf_r) + \frac{1}{r}\frac{\partial}{\partial \theta}\left(\tfrac{1}{r}f_\theta\right) + \frac{1}{r}\frac{\partial}{\partial z}(rf_z).$$

$$\Delta f = f_{rr} + \frac{1}{r}f_r + \frac{1}{r^2}f_{\theta\theta} + f_{zz}.$$

$$\boxed{\Delta f = \frac{\partial^2 f}{\partial r^2} + \frac{1}{r}\frac{\partial f}{\partial r} + \frac{1}{r^2}\frac{\partial^2 f}{\partial \theta^2} + \frac{\partial^2 f}{\partial z^2}.}$$

Starting from the cylindrical Laplacian and consider the Dirichlet eigenproblem

$$-\Delta f = \lambda f \quad \text{in } D, \qquad f|_{\partial D} = 0,$$

where $D = \{0 \le r \le R,\ 0 \le \theta < 2\pi,\ 0 \le z \le L\}$.

Assume $f(r, \theta, z) = R(r)\,\Theta(\theta)\,Z(z)$ and substituting into $-\Delta f = \lambda f$:

$$\frac{\partial^2 f}{\partial r^2} + \frac{1}{r}\frac{\partial f}{\partial r} + \frac{1}{r^2}\frac{\partial^2 f}{\partial \theta^2} + \frac{\partial^2 f}{\partial z^2} = \Delta R(r)\,\Theta(\theta)\,Z(z).$$

Computing each derivatives separately:

$$f_r = R'(r)\Theta(\theta)Z(z), \quad f_{rr} = R''(r)\Theta(\theta)Z(z),$$

$$f_\theta = R(r)\Theta'(\theta)Z(z), \quad f_{\theta\theta} = R(r)\Theta''(\theta)Z(z),$$

$$f_z = R(r)\Theta(\theta)Z'(z), \quad f_{zz} = R(r)\Theta(\theta)Z''(z).$$

Substituting back, the PDE becomes

$$R''(r)\Theta Z + \frac{1}{r}R'(r)\Theta Z + \frac{1}{r^2}R\Theta'' Z + R\Theta Z'' \;=\; -\lambda R\Theta Z.$$

Dividing both side by $R(r)\Theta(\theta)Z(z)$ and we get

$$\frac{R''(r)}{R(r)} + \frac{1}{r}\frac{R'(r)}{R(r)} + \frac{1}{r^2}\frac{\Theta''(\theta)}{\Theta(\theta)} + \frac{Z''(z)}{Z(z)} \;=\; -\lambda.$$

Now each term depends on a single variable; use the idea from the 2D polar case to introduce separation constants:

$$\frac{\Theta''(\theta)}{\Theta(\theta)} = -m^2, \qquad \frac{Z''(z)}{Z(z)} = -\Big(\frac{k\pi}{L}\Big)^2,$$

which leave the radial equation

$$R''(r) + \frac{1}{r}R'(r) + \Big(\lambda - \Big(\tfrac{k\pi}{L}\Big)^2 - \tfrac{m^2}{r^2}\Big)R(r) = 0.$$

This is again the Bessel equation.

$$-\frac{R''}{R} - \frac{1}{r}\frac{R'}{R} - \frac{1}{r^2}\frac{\Theta''}{\Theta} - \frac{Z''}{Z} = \lambda.$$

Rearrange and separate the z-dependence:

$$-\frac{Z''(z)}{Z(z)} = \mu, \qquad -\frac{\Theta''(\theta)}{\Theta(\theta)} = m^2,$$

so that

$$-\frac{R''}{R} - \frac{1}{r}\frac{R'}{R} + \frac{m^2}{r^2} = \lambda - \mu.$$

2. Axial (z) equation. The z-equation is

$$Z''(z) + \mu Z(z) = 0,$$

with Dirichlet $Z(0) = Z(L) = 0$. Nontrivial solutions will be

$$\mu = \left(\frac{k\pi}{L}\right)^2, \qquad k = 1, 2, 3, \ldots,$$

3. Angular equation. The θ-equation is

$$\Theta''(\theta) + m^2\Theta(\theta) = 0,$$

with 2π-periodicity, hence $m \in \mathbb{Z}$ and

$$\Theta_m(\theta) = \begin{cases} 1, & m = 0, \\ \cos(m\theta),\ \sin(m\theta), & m \geq 1. \end{cases}$$

4. Radial equation (Bessel). Let

$$\lambda - \mu = \lambda - \Big(\frac{k\pi}{L}\Big)^2.$$

The Bessel equation

$$r^2R'' + rR' + (\kappa^2 r^2 - m^2)R = 0.$$

Its regular solution at $r = 0$ is $J_m(\kappa r)$.

Imposing the Dirichlet boundary condition $R(R) = 0$ gives the quantization

$$J_m(\kappa R) = 0.$$

Let $j_{m,n}$ denote the n-th positive zero of J_m. Then

$$\kappa_{m,n} = \frac{j_{m,n}}{R}.$$

Combining axial and radial quantizations yields the eigenvalues

$$\boxed{\lambda_{m,n,k} \;=\; \left(\frac{j_{m,n}}{R}\right)^2 + \left(\frac{k\pi}{L}\right)^2, \;\; m = 0,1,2,\ldots,\; n = 1,2,\ldots,\; k = 1,2,\ldots}$$

and corresponding eigenfunctions (up to normalization)

$$\boxed{f_{m,n,k}(r,\theta,z) = J_m\Big(\frac{j_{m,n}}{R}\,r\Big) \times \begin{cases} \sin\big(\frac{k\pi z}{L}\big), & m = 0, \\ \cos(m\theta)\,\sin\big(\frac{k\pi z}{L}\big), \quad \sin(m\theta)\,\sin\big(\frac{k\pi z}{L}\big), & m \geq 1. \end{cases}}$$

3D Curvilinear coordinates: Spherical Coordinates

To make an easy, as we see the example at 2D polar, we will apply the Orthogonal Curvilinear coordinate formula. Let

$$x = r\sin\theta\cos\phi, \quad y = r\sin\theta\sin\phi, \quad z = r\cos\theta,$$

with scalar factors

$$h_r = 1, \quad h_\theta = r, \quad h_\phi = r\sin\theta.$$

$$\Delta f = \frac{1}{h_r h_\theta h_\phi}\sum_{i=1}^{3}\frac{\partial}{\partial q_i}\left(\frac{h_r h_\theta h_\phi}{h_i^2}\frac{\partial f}{\partial q_i}\right), \qquad (q_1, q_2, q_3) = (r, \theta, \phi).$$

$$\Delta f = \frac{1}{r^2\sin\theta}\left[\frac{\partial}{\partial r}(r^2\sin\theta\, f_r) + \frac{\partial}{\partial\theta}\left(\tfrac{r^2\sin\theta}{r^2}\, f_\theta\right) + \frac{\partial}{\partial\phi}\left(\tfrac{r^2\sin\theta}{(r\sin\theta)^2}\, f_\phi\right)\right].$$

$$\Delta f = \frac{1}{r^2}\frac{\partial}{\partial r}(r^2 f_r) + \frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}(\sin\theta f_\theta) + \frac{1}{r^2\sin^2\theta}f_{\phi\phi}.$$

$$\boxed{\Delta f = \frac{1}{r^2}\frac{\partial}{\partial r}\Big(r^2\frac{\partial f}{\partial r}\Big) + \frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}\Big(\sin\theta\frac{\partial f}{\partial\theta}\Big) + \frac{1}{r^2\sin^2\theta}\Big(\frac{d^2 f}{d\phi^2}\Big)}$$

Just like the previous example, we assume separation of variables

$$f(r, \theta, \phi) = R(r)\, Y(\theta, \phi),$$

Plugging into the eigenvalue problem

$$-\Delta f = \lambda f \Longrightarrow -\left[\frac{Y}{r^2}\frac{d}{dr}\Big(r^2 R'(r)\Big) + \frac{R}{r^2\sin\theta}\frac{\partial}{\partial\theta}\Big(\sin\theta\, Y_\theta\Big) + \frac{R}{r^2\sin^2\theta}Y_{\phi\phi}\right] = \lambda RY.$$

Dividing through by RY and multiplying by r^2, we get

$$\frac{1}{R}\frac{d}{dr}\Big(r^2 R'(r)\Big) + \lambda r^2 = \frac{1}{Y}\left[-\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\Big(\sin\theta\, Y_\theta\Big) - \frac{1}{\sin^2\theta}Y_{\phi\phi}\right].$$

Since the left side depends only on r and the right side only on (θ, ϕ), both must equal a separation constant, which we denote $\ell(\ell+1)$. Thus we got a ugly answer as

$$\frac{1}{R}\frac{d}{dr}\Big(r^2 R'(r)\Big) + \lambda r^2 = \ell(\ell+1), \qquad \frac{1}{Y}\left[\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\Big(\sin\theta\, Y_\theta\Big) + \frac{1}{\sin^2\theta}Y_{\phi\phi}\right] = -\ell(\ell+1),$$

where $\ell = 0, 1, 2, \ldots$.

Angular part. The angular equation defines the spherical harmonics

$$Y_\ell^m(\theta,\phi), \qquad m=-\ell,\dots,\ell,$$

with multiplicity $2\ell+1$.

Radial part. The radial equation becomes

$$r^2R''(r)+2rR'(r)+\big(\lambda r^2-\ell(\ell+1)\big)R(r)=0.$$

See! It's the Bessel function, or we should name this the spherical Bessel equation, with solutions

$$R(r)=j_\ell(\sqrt{\lambda}\,r),$$

where j_ℓ is the spherical Bessel function. The Dirichlet boundary condition $R(R)=0$ requires

$$j_\ell(\sqrt{\lambda}\,R)=0.$$

Thus, the eigenvalues are determined by the zeros $\alpha_{\ell,n}$ of the spherical Bessel function j_ℓ:

$$\boxed{\lambda_{\ell,n}=\left(\tfrac{\alpha_{\ell,n}}{R}\right)^2, \qquad \ell=0,1,2,\dots,\ n=1,2,\dots}$$

with eigenfunctions

$$\boxed{f_{\ell,m,n}(r,\theta,\phi)=j_\ell\Big(\tfrac{\alpha_{\ell,n}}{R}r\Big)\,Y_\ell^m(\theta,\phi), \qquad m=-\ell,\dots,\ell.}$$

Chapter 2.6 : Relationship between the Laplace operator and Weyl's Law

An easy way to understand Weyl's law is: it describes the asymptotic behavior of eigenvalues of the Laplace–Beltrami operator. In its simplest form, it states the asymptotic growth of the eigenvalues of the Laplacian on bounded domains with Dirichlet and Neumann boundary conditions. For compact Riemannian manifolds, it also describes the counting function of the eigenvalues associated with the Laplace operator.

As we read in Chapter 2.1, for a bounded domain $\Omega \subset \mathbb{R}^d$ with Dirichlet boundary conditions, the eigenvalue problem is:

$$-\Delta u = \lambda u \quad \text{in } \Omega, \quad u|_{\partial\Omega} = 0$$

Let $0 < \lambda_1 \leqslant \lambda_2 \leqslant \lambda_3 \leqslant \cdots \to \infty$ be the eigenvalues[5].
The notation represent the eigenvalue counting function is:

$$N(\lambda) = \#\{j : \lambda_j \leqslant \lambda\}$$

then the Weyl's law states that as $\lambda \to \infty$ [5]:

$$N(\lambda) \sim \frac{\omega_d}{(2\pi)^d} \mathrm{Vol}(\Omega) \lambda^{d/2}$$

where ω_d is the volume of the unit ball in $\mathbb{R}^d$, and $\mathrm{Vol}(\Omega)$ is the volume of the domain.

What's that mean? Why is it important?

- As we know already, Weyl's law describes the asymptotic eigenvalue behavior of the Laplace operator
- It applies to both Dirichlet and Neumann boundary conditions (with the same leading term)
- On Riemannian manifolds (M, g), Weyl's law becomes:

  $$N(\lambda) \sim \frac{\omega_d}{(2\pi)^d} \mathrm{Vol}_g(M) \lambda^{d/2}$$

  where $\mathrm{Vol}_g(M)$ is the volume with respect to the metric g.

Let's take a look at the details and a couple of examples of Weyl's Law in different dimensions (Chapter 3)

Chapter 3: Weyl Asymptotic Law

Chapter 3.1: Statement and meaning of Weyl's Law

Perfect, after chapter 2. We've seen why Weyl's Law matters, let's take a closer look at the general formula[6].

The mathematical expression of **Weyl's Law** is:

$$\boxed{N(\lambda) \sim \frac{\omega_d}{(2\pi)^d}\mathrm{Vol}(\Omega)\lambda^{d/2}}$$

Here are the meanings of notations:

- $N(\lambda)$: Number of eigenvalues of the Laplace operator $-\Delta$ (with specified boundary conditions) that are less than or equal to λ.
- d: Dimension of domain Ω (for example: $d = 1, 2, 3$).
    1. $d = 1$: interval (length),
    2. $d = 2$: planar region (area),
    3. $d = 3$: spatial domain (volume).
- $\mathrm{Vol}(\Omega)$: Volume of the domain Ω.
    A. 1 dimension is interval,
    B. 2 dimensions is area
    C. 3 dimensions is volume
- ω_d: Volume of d - dimensional unit ball in $\mathbb{R}^d$ [6]:

$$\omega_d = \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)}$$

The Γ is represented as the Gamma function: $\Gamma(n) = (n-1)!$ for integer n. More generally by the integral formula

$$\Gamma(t) = \int_0^\infty x^{t-1}e^{-x}\,dx, \qquad t > 0.$$

More details of ω_d in different dimension are provided:

1. $d = 1, \omega_1 = 2$,
2. $d = 2, \omega_2 = \pi$,
3. $d = 3, \omega_3 = \frac{4\pi}{3}$,

Chapter 3.2: Weyl's Law in 1D, 2D, and 3D: Explicit Examples

Chapter 3.2.1: Weyl's Law for 1 dimension

Example 0.0.1 (1D Dirichlet Laplacian)**.** The eigenvalues of the Laplacian operator $-\frac{d^2}{dx^2}$ on the interval $[0, L]$, with Dirichlet boundary conditions $u(0) = u(L) = 0$, are given by

$$\lambda_n = \left(\frac{n\pi}{L}\right)^2, \quad n = 1, 2, 3, \ldots \tag{6}$$

The eigenvalue counting function, which tells us how many eigenvalues are less than or equal to a given λ, is

$$N(\lambda) = \left\lfloor \frac{L\sqrt{\lambda}}{\pi} \right\rfloor \sim \frac{L\sqrt{\lambda}}{\pi} \quad \text{as } \lambda \to \infty. \tag{7}$$

Verification of Weyl's Law for 1 Dimensional Case: Interval $[0, L]$

In this case, $d = 1$. we apply the general formula:

$$N(\lambda) \sim \frac{\omega_d}{(2\pi)^d} \mathrm{Vol}(\Omega) \lambda^{d/2} \Longrightarrow N(\lambda) \sim \frac{\omega_1}{(2\pi)^1} \mathrm{Vol}(\Omega) \lambda^{1/2}$$

To calculate the ω_1, recall the general formula for Gamma function:

$$\omega_1 = \frac{\pi^{1/2}}{\Gamma(\frac{1}{2}+1)} = \frac{\pi^{1/2}}{\Gamma(\frac{3}{2})} = \frac{\sqrt{\pi}}{\frac{1}{2}\Gamma(\frac{1}{2})} = \frac{2\sqrt{\pi}}{\sqrt{\pi}} = 2$$

Now we have $\omega_1 = 2$ (unit interval length) and
$\mathrm{vol}(\Omega) = L$ because the interval = $[0, L]$.
Substituting into:

$$N(\lambda) \sim \frac{\omega_1}{(2\pi)^1} \mathrm{Vol}(\Omega) \lambda^{1/2} = \frac{2}{2\pi} L\sqrt{\lambda} = \frac{L\sqrt{\lambda}}{\pi}$$

which matches the asymptotic behavior in (7).

To better understand how 1D Laplacian case and 1D Weyl's law are growth, Professor Pedro and I wrote the code in Python (To view the Python code, please go to the appendix)

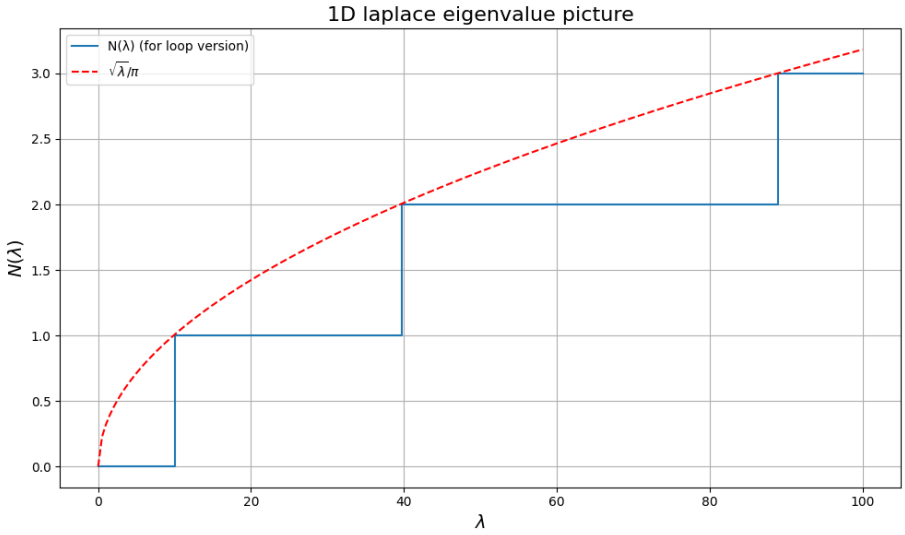


Figure 2: The behavior of N(λ)

As you can see above, the Weyl asymptotic law in one dimension for the Laplace operator with Dirichlet boundary conditions on the interval $[0, 1]$. We started by setting the domain length $a = 1.0$ and created 200 evenly spaced λ values from 0 to 100. For each λ value, I implemented a counting algorithm that goes through integer values of n, checking when the eigenvalues $\lambda_n = (n\pi)^2$ drop below the current λ threshold. The loop continues until we find the first eigenvalue that exceeds λ, giving us the exact count of eigenvalues below that value.

I chose to plot $N(\lambda)$ using a step function because it perfectly captures the discrete, staircase nature of how eigenvalues accumulate—you can actually see each new eigenvalue being added to the count. To compare with theory, I also plotted the predicted Weyl asymptotic $N(\lambda) \sim \sqrt{\lambda}/\pi$ as a red dashed line. The maps give an eigenvalue sequence:

- $\lambda_1 = 1^2\pi^2 \approx 9.87$
- $\lambda_2 = 2^2\pi^2 = 4\pi^2 \approx 39.48$
- $\lambda_3 = 3^2\pi^2 = 9\pi^2 \approx 88.83$
- $\lambda_4 = 4^2\pi^2 = 16\pi^2 \approx 157.91$

The behavior of $N(\lambda)$ in the figure directly corresponds to this discrete eigenvalue spectrum. Each horizontal segment represents an interval of λ values where no new eigenvalues appear, while each vertical jump marks the threshold where another eigenvalue enters the count. For example, when λ crosses approximately 9.87, $N(\lambda)$ jumps from 0 to 1 as λ_1 becomes included. Similarly, the function jumps to 2 when λ exceeds 39.48 (λ_2), to 3 at 88.83 (λ_3), and to 4 at 157.91 (λ_4). If we look closer to the picture, the red dashed curve representing $\sqrt{\lambda}/\pi$ provides the Weyl asymptotic approximation to this function. Therefore, for large λ, the number of eigenvalues below λ grows as $\frac{L}{\pi}\sqrt{\lambda} = \frac{\sqrt{\lambda}}{\pi}$ when $L = 1$.

Chapter 3.2.2: Weyl's Law for 2 dimension

Example 0.0.2 (2D Dirichlet Laplacian)**.** The eigenvalues of the Laplacian $-\Delta = -\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)$ on the rectangle $[0, a] \times [0, b]$ with Dirichlet boundary conditions are:

$$\lambda_{m,n} = \pi^2 \left(\frac{m^2}{a^2} + \frac{n^2}{b^2}\right), \quad m, n \in \mathbb{N} \tag{8}$$

The counting function $N(\lambda)$ corresponds to the number of integer lattice points (m, n) in the first quadrant satisfying:

$$\frac{m^2}{a^2} + \frac{n^2}{b^2} \leqslant \frac{\lambda}{\pi^2} \tag{9}$$

Asymptotically, this behaves like [5]:

$$N(\lambda) \sim \frac{ab}{4\pi}\lambda \tag{10}$$

Verification of Weyl's Law in 2 Dimensions: Rectangle $[0, a] \times [0, b]$

In this case, we set $d = 2$, and the domain Ω is the area: $\text{Area}(\Omega) = ab$.
Apply the general formula of Weyl's Law:

$$N(\lambda) \sim \frac{\omega_d}{(2\pi)^d}\text{Area}(\Omega)\lambda^{d/2} \Longrightarrow N(\lambda) \sim \frac{\omega_2}{(2\pi)^2}\, ab\, \lambda$$

To compute ω_2, the area of the 2D unit ball (unit disk):

$$\omega_2 = \frac{\pi^1}{\Gamma(1+1)} = \frac{\pi}{\Gamma(2)} = \frac{\pi}{1} = \pi$$

Substituting into the formula:

$$N(\lambda) \sim \frac{\pi}{4\pi^2} \cdot ab \cdot \lambda = \frac{ab\lambda}{4\pi}$$

This matches the explicit asymptotic result in (10),
thus confirming the validity of Weyl's Law in two dimensions.

To better understand how the 2D Laplacian case and 2D Weyl's law grow, Professor Pedro and I wrote the code in Python (To view the Python code, please go to the appendix)

We now explores the two-dimensional Weyl asymptotic law for the Laplace operator with Dirichlet boundary conditions on the square domain $[0,1] \times [0,1]$. In this case, let us set $a = b = 1$ in our base case. The eigenvalues of the Laplacian in this setting take the form

$$\lambda_{m,n} = \pi^2 \left(\left(\frac{m}{a}\right)^2 + \left(\frac{n}{b}\right)^2 \right), \qquad m, n \in \mathbb{N},$$

The algorithm proceeds by generating a set of λ thresholds between 0 and 320. For each threshold λ, it iterates over integer pairs (m, n) up to a fixed cutoff and counts how many eigenvalues $\lambda_{m,n}$ fall below λ. This produces the eigenvalue counting function $N(\lambda)$, stored as a list and then visualized. The resulting staircase plot of $N(\lambda)$ accurately reflects the discrete nature of eigenvalue accumulation as λ increases. In parallel, the code computes the Weyl asymptotic prediction

$$N(\lambda) \sim \frac{ab}{4\pi}\lambda,$$

where ab is the area of the rectangle. The red dashed line shows this theoretical linear growth, and its strong agreement with the computed counting function confirms the two-dimensional Weyl law: the number of eigenvalues below λ grows proportionally to λ in two dimensions.

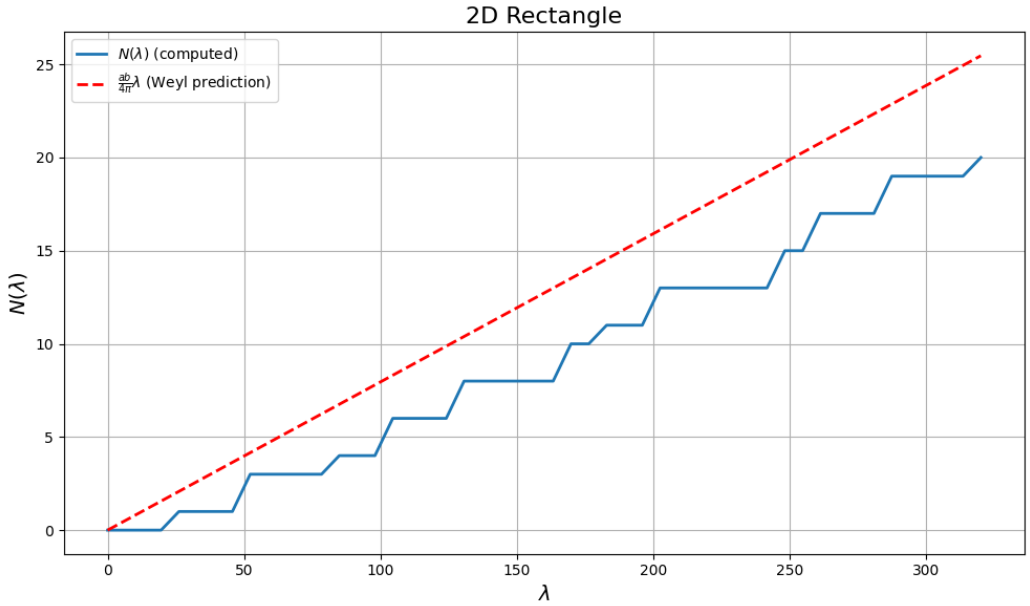


Figure 3: Case A: $\lambda \in [0, 320]$

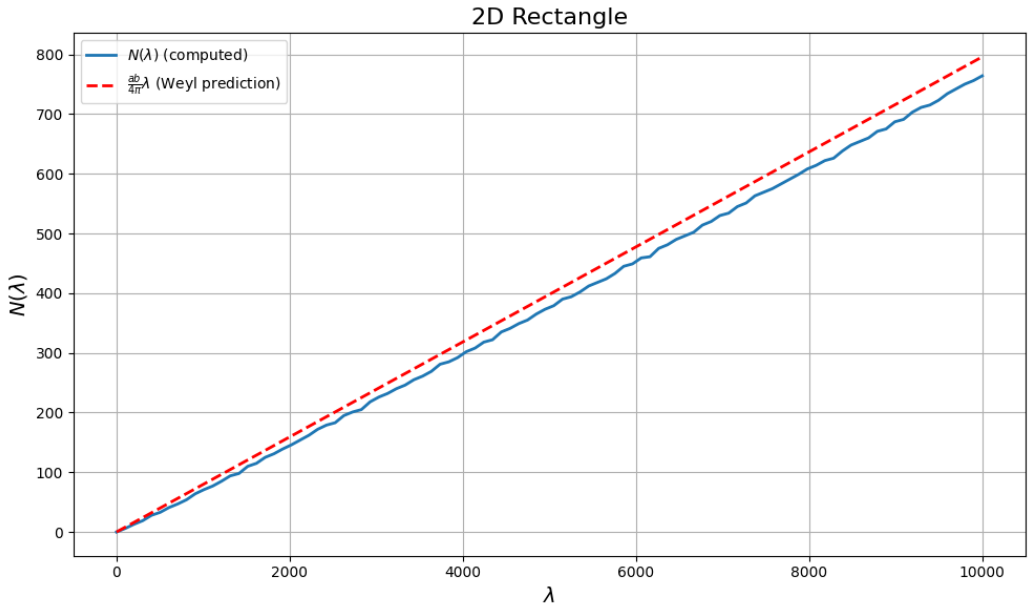


Figure 4: Case B: $\lambda \in [1, 10000]$

As you already see from the figure above, we code the counting experiment for the square $[0,1] \times [0,1]$ with the same code but two different choices of the sampling range for λ:

- Case A: $\lambda \in [0, 320]$ sampled at 50 points, i.e. `np.linspace(0,320,50)`.
- Case B: $\lambda \in [1, 10000]$ sampled at 100 points, i.e. `np.linspace(1,10000,100)`.

Both experiments use the same discrete eigenvalue generator

$$\lambda_{m,n} = \pi^2\big((m/a)^2 + (n/b)^2\big) = \lambda_{m,n} = \pi^2(m^2 + n^2), \qquad m, n \in \mathbb{N}, \qquad a = b = 1,$$

and the same truncation $m, n \leq 50$.

Observed behaviour.

- In Case A (small λ range) the plotted counting function $N(\lambda)$ follows the Weyl line reasonably well but does not lie exactly on it: the staircase nature and individual eigenvalue locations are visible and create noticeable deviations from the smooth Weyl curve.
- In Case B (much larger λ range) the computed $N(\lambda)$ and the Weyl prediction appear much closer: the smooth linear Weyl curve is an excellent approximation across the sampled range.

I am listing the first few distinct eigenvalues (in increasing order) we get

- $\lambda_1 = \lambda_{1,1} = 2\pi^2 \approx 19.7392$,
- $\lambda_2 = \lambda_{1,2} = \lambda_{2,1} = 5\pi^2 \approx 49.3480$ (multiplicity 2),
- $\lambda_3 = \lambda_{2,2} = 8\pi^2 \approx 78.9568$,
- $\lambda_4 = \lambda_{1,3} = \lambda_{3,1} = 10\pi^2 \approx 98.6960$ (multiplicity 2).

The idea is similar previous result when I calculate the 1D.The eigenvalues of the Laplacian on the unit square with Dirichlet boundary conditions are given by $\lambda_{m,n} = \pi^2(m^2 + n^2)$ for positive integers m, n. To obtain the first few distinct eigenvalues, we list the smallest values of $m^2 + n^2$. The lowest eigenvalue arises from $(m, n) = (1, 1)$, giving $\lambda_{1,1} = 2\pi^2$. The next case comes from $(1, 2)$ or $(2, 1)$, both producing $5\pi^2$, so this eigenvalue has multiplicity two. Taking $(2, 2)$ yields $8\pi^2$, which is distinct, while $(1, 3)$ and $(3, 1)$ give $10\pi^2$, again of multiplicity two. In general, multiplicity occurs whenever different pairs (m, n) yield the same sum $m^2 + n^2$, so the ordering of the spectrum depends on sorting these sums in increasing order.

The summary I discussed with Professor Pedro.
By the explanation and code in the appendix, the two numerical experiments on the square $[0, 1] \times [0, 1]$ demonstrate the connection between the Laplace operator spectrum and Weyl's law. For example, for small λ values (Case A), the staircase structure of the counting function $N(\lambda)$ reveals fluctuations around the smooth Weyl prediction, reflecting the discrete nature of individual eigenvalues. Although Case A does not show us how similar the Laplace operator spectrum and Weyl's law are. Of course, we can look closer for a larger domain like λ (Case B), where these fluctuations become negligible, and $N(\lambda)$ aligns almost perfectly with the linear Weyl curve. This confirms the asymptotic validity of Weyl's law: as $\lambda \to \infty$, the growth of eigenvalues of the Laplace operator is controlled by the leading term $\frac{|\Omega|}{4\pi}\lambda$. It's not done yet. 2D also covers the polar coordinates, which we will now move on to!

Example 0.0.3 (Dirichlet Laplacian on a Disk)**.** Consider the unit disk

$$\Omega = \{(x, y) \in \mathbb{R}^2 : x^2 + y^2 \leqslant R^2\}$$

with Dirichlet boundary conditions. By separation of variables in polar coordinates, the radial equation reduces to a Bessel differential equation. The admissible eigenvalues are determined by the zeros of the Bessel functions:

$$J_m(\sqrt{\lambda}\, R) = 0, \qquad m \in \mathbb{Z}_{\geqslant 0}, \tag{11}$$

where J_m is the Bessel function of order m. Denoting the k-th positive zero of J_m by $j_{m,k}$, the eigenvalues are given by

$$\lambda_{m,k} = \left(\frac{j_{m,k}}{R}\right)^2.$$

Each eigenvalue has multiplicity one when $m = 0$, while for $m > 0$ the angular dependence provides two independent eigenfunctions $\cos(m\theta)$ and $\sin(m\theta)$, hence the multiplicity is two.

Verification of Weyl's Law in 2 dimension Circle:

For $d = 2$, the volume is $\mathrm{vol}(\Omega) = \mathrm{Area}(\Omega) = \pi R^2$.

Apply the general formula Weyl's Law:

$$N(\lambda) \sim \frac{\omega_d}{(2\pi)^d}\mathrm{Area}(\Omega)\lambda^{d/2} \Longrightarrow N(\lambda) \sim \frac{\omega_2}{(2\pi)^2}\pi R^2\lambda$$

To compute ω_2, the area of the 2D unit ball (unit disk):

$$\omega_2 = \frac{\pi^1}{\Gamma(1+1)} = \frac{\pi}{\Gamma(2)} = \frac{\pi}{1} = \pi$$

Substituting into the formula:

$$N(\lambda) \sim \frac{\pi}{4\pi^2}\pi R^2\lambda = \frac{R^2}{4}\lambda$$

To better understand how the 2D Laplacian case and 2D Weyl's law grow, Professor Pedro and I wrote the code in Python (To view the Python code, please go to the appendix)

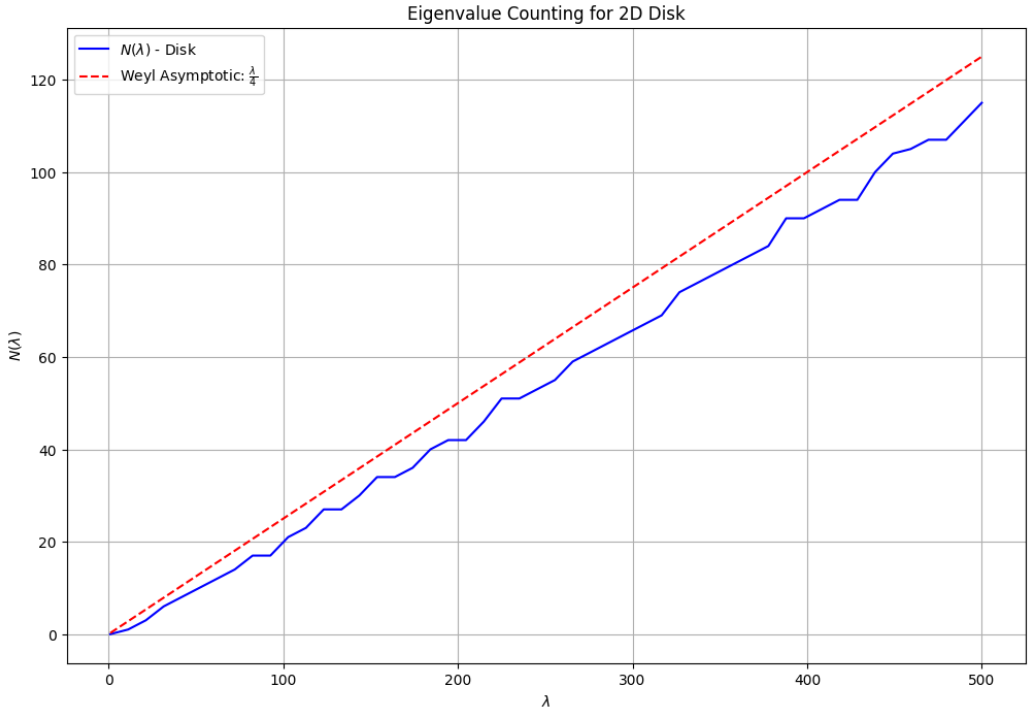


Figure 5: $\lambda \in [0, 500]$

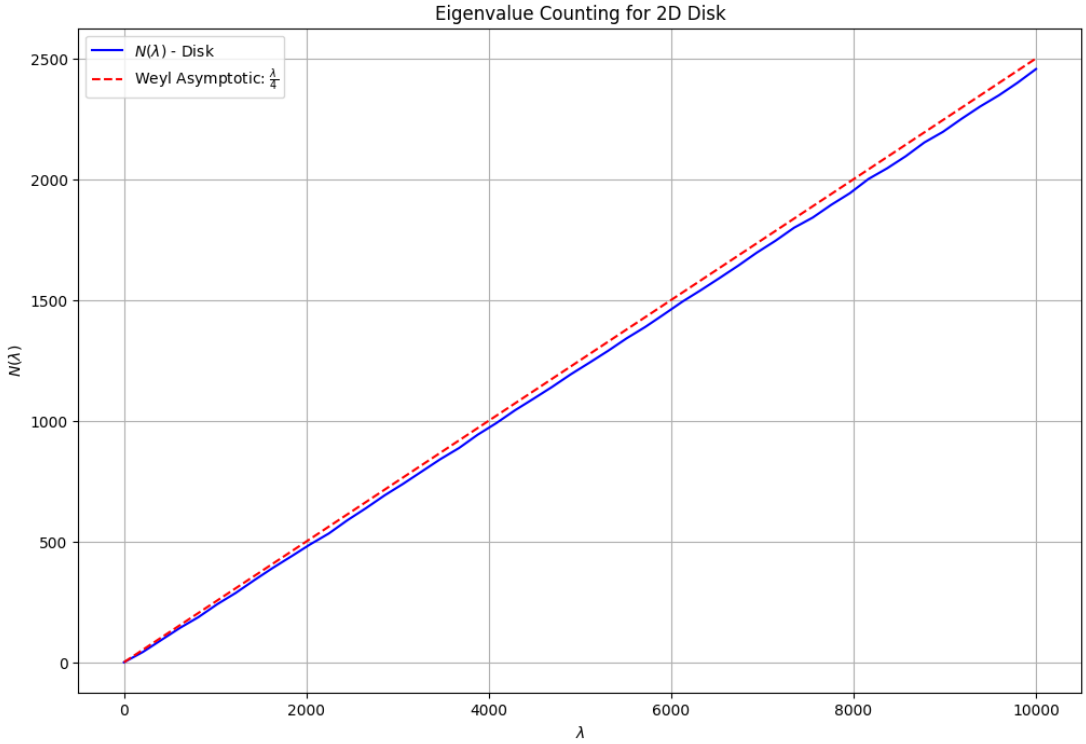


Figure 6: $\lambda \in [0, 10000]$

Before we go over more details, recall the Dirichlet Laplacian eigenvalues:

$$\lambda_{m,n} = j_{m,n}^2, \qquad R = 1,$$

where $j_{m,n}$ denotes the n-th positive zero of the Bessel function $J_m(x)$. The first few eigenvalues (in increasing order) will looks like the following:

- $\lambda_1 = j_{0,1}^2 \approx (2.4048)^2 \approx 5.783$ (multiplicity 1),
- $\lambda_2 = j_{1,1}^2 \approx (3.8317)^2 \approx 14.682$ (multiplicity 2),
- $\lambda_3 = j_{2,1}^2 \approx (5.1356)^2 \approx 26.375$ (multiplicity 2),
- $\lambda_4 = j_{0,2}^2 \approx (5.5201)^2 \approx 30.472$ (multiplicity 1),
- $\lambda_5 = j_{3,1}^2 \approx (6.3802)^2 \approx 40.707$ (multiplicity 2).

The multiplicity pattern follows from the angular dependence of the eigenfunctions. When $m = 0$, the eigenfunction involves only the radial mode $J_0(j_{0,n}r)$, leading to multiplicity 1. For $m > 0$, the angular dependence introduces two independent modes, $\cos(m\theta)$ and $\sin(m\theta)$, so that each eigenvalue $\lambda_{m,n}$ with $m > 0$ has multiplicity 2. This explains the observed alternation between single and double multiplicities in the eigenvalue list.

In the case of the two-dimensional disk with Dirichlet boundary conditions, the eigenvalues of the Laplacian take the form $\lambda_{m,n} = \left(\frac{j_{m,n}}{R}\right)^2$, where $j_{m,n}$ denotes the n-th positive zero of the Bessel function $J_m(x)$. The Python code implements this formula by looping over angular indices $m \geqslant 0$, computing sufficiently many Bessel zeros using `scipy.special.jn_zeros`, and checking which eigenvalues fall below a given cutoff λ.

The counting rule distinguishes between $m = 0$, which contributes multiplicity one, and $m > 0$, where both $\cos(m\theta)$ and $\sin(m\theta)$ solutions exist, giving multiplicity two. By evaluating this count across a range of λ-values, we obtain the eigenvalue counting function $N(\lambda)$ for the disk. The plot compares this exact counting function (blue curve) with the two-dimensional Weyl asymptotic law $N(\lambda) \sim \frac{A}{4\pi}\lambda$, where the area of the unit disk is $A = \pi$, so that the asymptotic line is simply $\lambda/4$. The numerical results show that while the exact spectrum exhibits oscillations due to the arithmetic structure of Bessel zeros, the growth of $N(\lambda)$ closely follows the Weyl prediction for large λ. When I set the eigenvalue counting function for the disk on a smaller range of λ (a few hundred), the staircase nature of the spectrum is visible: the contributions from individual Bessel zeros appear clearly, and the curve of $N(\lambda)$ shows oscillations above and below the smooth Weyl prediction $\lambda/4$. This reflects how messy and unbalanced the Bessel zeros are at low energy. Additionally, when I exchange the domain for a very large λ (for example, up to ten thousand), the line will become almost the same compared to the overall linear growth.

Chapter 3.2.3: Weyl's Law for 3 dimension

Example 0.0.4 (3D Dirichlet Laplacian)**.** For a rectangular box $[0,a] \times [0,b] \times [0,c]$, the eigenvalues of $-\Delta$ with Dirichlet boundary conditions are given by:

$$\lambda_{k,m,n} = \pi^2 \left(\frac{k^2}{a^2} + \frac{m^2}{b^2} + \frac{n^2}{c^2} \right), \quad k,m,n \in \mathbb{N} \tag{12}$$

The counting function satisfies the asymptotic relation:

$$N(\lambda) \sim \frac{abc}{6\pi^2} \lambda^{3/2} \tag{13}$$

Verification of Weyl's Law in Three Dimensions: Box $[0,a] \times [0,b] \times [0,c]$

In this case, $d = 3$ and $\mathrm{Vol}(\Omega) = abc$. Weyl's Law gives:

$$N(\lambda) \sim \frac{\omega_d}{(2\pi)^d} \mathrm{Vol}(\Omega) \lambda^{d/2} \Longrightarrow N(\lambda) \sim \frac{\omega_3}{(2\pi)^3} \, abc \, \lambda^{3/2}$$

Now we compute the volume of the 3D unit ball:

$$\omega_3 = \frac{\pi^{3/2}}{\Gamma(\frac{3}{2}+1)} = \frac{\pi^{3/2}}{\Gamma(\frac{5}{2})} = \frac{\pi^{3/2}}{\frac{3}{4}\sqrt{\pi}} = \frac{4}{3}\pi$$

Substituting into the formula:

$$N(\lambda) \sim \frac{\frac{4}{3}\pi}{8\pi^3} \cdot abc \cdot \lambda^{3/2} = \frac{abc}{6\pi^2} \lambda^{3/2}$$

which matches the expected asymptotic formula in (13), verifying Weyl's Law in three dimensions.

To better understand how the 3D Laplacian case and 3D Weyl's law grow, Professor Pedro and I wrote the code in Python (To view the Python code, please go to the appendix)

The behavior is analogous to the 1 dimensional and 2 dimensional cases, but in 3 dimensions, the structure is complex due to the larger number of possible permutations. But the idea is similar.

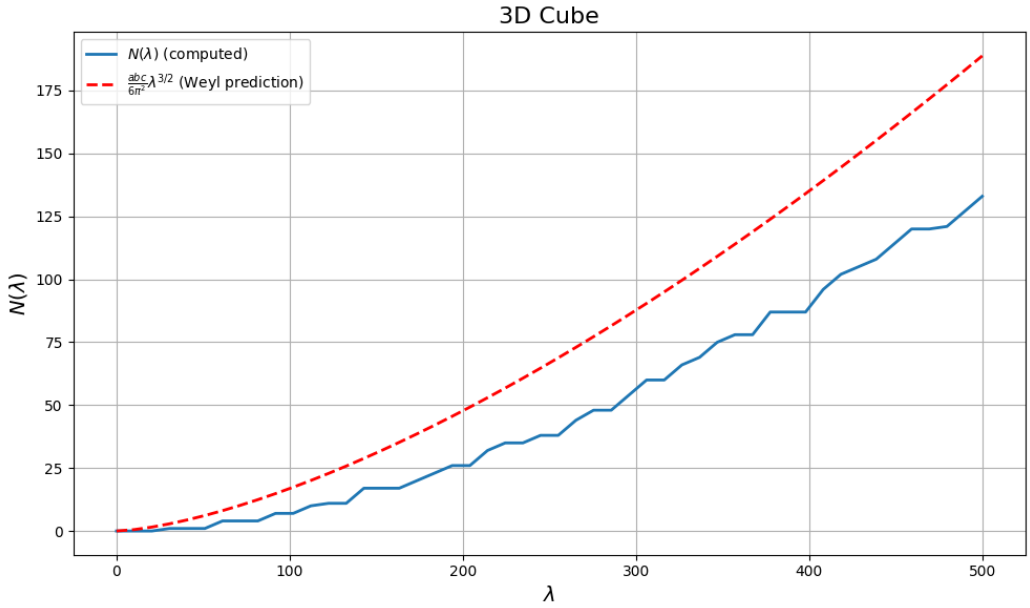


Figure 7: $\lambda \in [0, 500]$

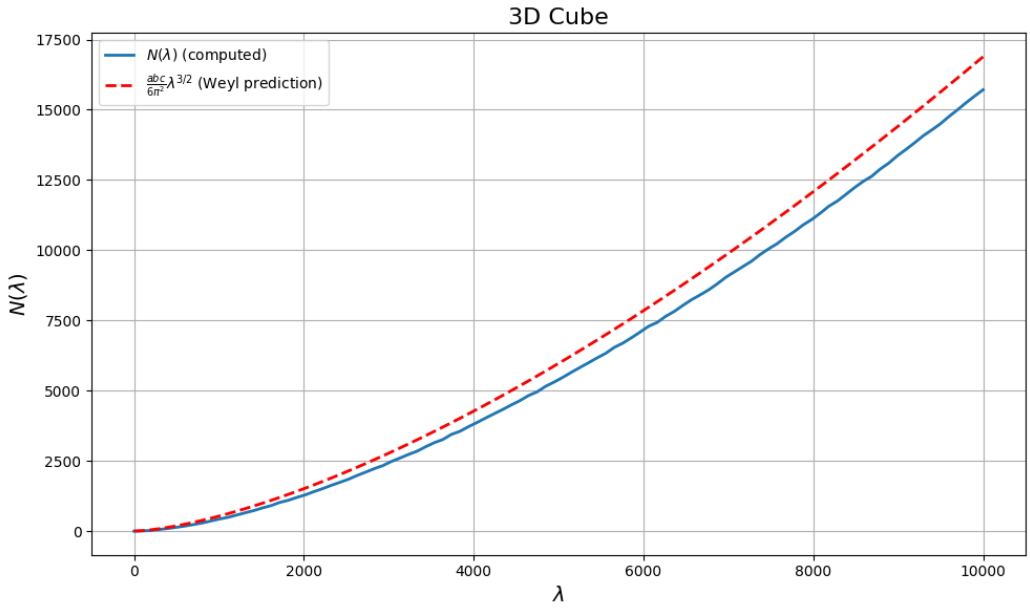


Figure 8: $\lambda \in [0, 10000]$

As you see the pictures, we use Python and compute the eigenvalue counting function for the 3D Laplace operator on a unit cube with Dirichlet boundary conditions.

The base case is let $a = b = c = 1$, recall the unit cube with Dirichlet boundary conditions:

$$\lambda_{m,n,k} = \pi^2\big(m^2 + n^2 + k^2\big), \qquad m, n, k \in \mathbb{Z}_{>0}.$$

The first few eigenvalues (in increasing order) are:

- $\lambda_1 = \lambda_{1,1,1} = 3\pi^2 \approx 29.6088$ (multiplicity 1),
- $\lambda_2 = \lambda_{1,1,2} = \lambda_{1,2,1} = \lambda_{2,1,1} = 6\pi^2 \approx 59.2176$ (multiplicity 3),
- $\lambda_3 = \lambda_{1,2,2} = \lambda_{2,1,2} = \lambda_{2,2,1} = 9\pi^2 \approx 88.8264$ (multiplicity 3),
- $\lambda_4 = \lambda_{1,1,3} = \lambda_{1,3,1} = \lambda_{3,1,1} = 11\pi^2 \approx 108.384$ (multiplicity 3).

The multiplicity arises whenever different ordered triples (m, n, k) yield the same sum $m^2 + n^2 + k^2$. For instance, the sum 6 corresponds to the permutations of $(1, 1, 2)$, giving multiplicity 3. Similarly, the sum 9 corresponds to the permutations of $(1, 2, 2)$, again with multiplicity 3. The spectrum is obtained by sorting these sums in increasing order.

I start by generating λ values from 1 to 500 and 1 to 10,000 and count how many eigenvalues $\lambda_{k,m,n} = \pi^2 \left(\frac{k^2}{a^2} + \frac{m^2}{b^2} + \frac{n^2}{c^2}\right)$ fall below each λ using triple nested loops. The picture shows the actual eigenvalue count $N(\lambda)$, while the red dashed line shows the 3D Weyl asymptotic prediction $N(\lambda) \sim \frac{V}{6\pi^2}\lambda^{3/2}$, where V = 1 is the volume of the unit cube. In both cases (from 1 to 500 and from 1 to 10,000), we can observe that as λ increases toward infinity, the blue computed curve and the red theoretical Weyl prediction become nearly equal. which shows the growth of the number of eigenvalues is indeed controlled by the Weyl law.

Same like the 2 dimensional, the 3 dimensional also covers the two different coordinates, which are the cylinder and spherical. We will now move on to!

Example 0.0.5 (3D Dirichlet Laplacian on Ball)**. Cylinder:** the eigenvalues take the form

$$\lambda = \left(\frac{j_{m,k}}{a}\right)^2 + \left(\frac{n\pi}{L}\right)^2, \quad m \in \mathbb{Z},\ k, n \in \mathbb{N}, \tag{14}$$

where $j_{m,k}$ represent the k-th zero of the Bessel function J_m.

Spherical: For the ball

$$\Omega = \{(x, y, z) \in \mathbb{R}^3 : x^2 + y^2 + z^2 \leqslant R^2\},$$

the eigenvalues are determined by the zeros of spherical Bessel functions:

$$\sqrt{\lambda}R = j_{\ell+1/2,k}, \quad \ell = 0, 1, 2, \ldots, \quad k = 1, 2, 3, \ldots, \tag{15}$$

where $j_{\nu,k}$ are the zeros of J_ν.

Verification of Weyl's Law (Cylinder and spherical)
In this case, For $d = 3$, we apply the Weyl's Law:

$$N(\lambda) \sim \frac{\omega_d}{(2\pi)^d}\mathrm{Vol}(\Omega)\lambda^{d/2} \Longrightarrow N(\lambda) \sim \frac{\omega_3}{(2\pi)^3} \cdot \mathrm{Vol}(\Omega) \cdot \lambda^{3/2}$$

Now we compute the volume of the 3D unit ball:

$$\omega_3 = \frac{\pi^{3/2}}{\Gamma(\frac{3}{2}+1)} = \frac{\pi^{3/2}}{\Gamma(\frac{5}{2})} = \frac{\pi^{3/2}}{\frac{3}{4}\sqrt{\pi}} = \frac{4}{3}\pi$$

Substituting into the formula:

$$N(\lambda) \sim \frac{\omega_3}{(2\pi)^3} \cdot \frac{4}{3}\pi R^3\lambda^{3/2} = \frac{\frac{4}{3}\pi}{8\pi^3} \cdot \frac{4}{3}\pi R^3\lambda^{3/2} = \frac{R^3\lambda^{3/2}}{9\pi}\sqrt{\lambda}.$$

- **Cylinder:** The domain volume is $\mathrm{Vol}(\Omega) = \pi a^2 L$. Hence,

$$N(\lambda) \sim \frac{\frac{4}{3}\pi}{(2\pi)^3} \cdot \pi a^2 L\, \lambda^{3/2}.$$

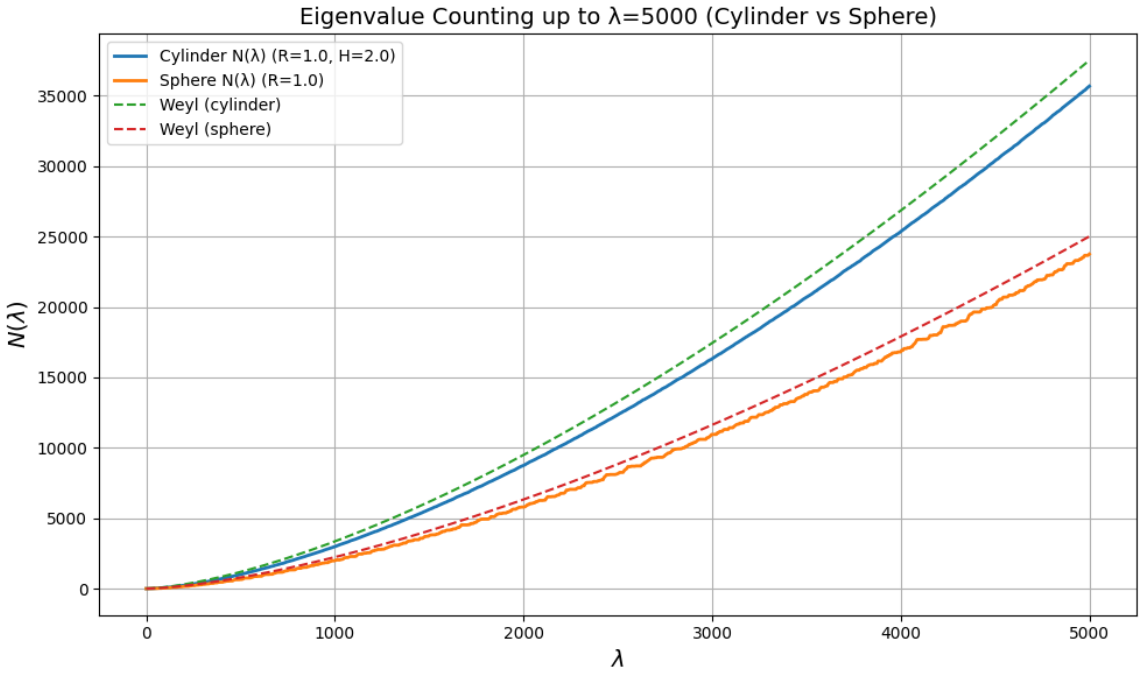
- **Spherical:** The domain volume is $\mathrm{Vol}(\Omega) = \frac{4}{3}\pi R^3$. Hence,

$$N(\lambda) \sim \frac{\frac{4}{3}\pi}{(2\pi)^3} \cdot \frac{4}{3}\pi R^3\, \lambda^{3/2}.$$

To better understand how the 3D Laplacian case and 3D Weyl's law grow, Professor Pedro and I wrote the code in Python (To view the Python code, please go to the appendix)

To be honest, putting this together wasn't easy, especially the part where I computed the zeros of the spherical Bessel functions. It took a lot of time working on it, which was much harder than the previous 4 Python codes (1D,2D,3D), and I spent an amount of time debugging it. Lucky, the result could help reader see how the eigenvalue distributions differ between a cylinder and a sphere in three dimensions. let's go over the details.

The most important core idea is actually pretty intuitive: for the cylinder, eigenvalues come from combining radial Bessel zeros and axial sine modes, like $\lambda = \left(\frac{j_{m,n}}{R}\right)^2 + \left(\frac{k\pi}{H}\right)^2$, while for the sphere, everything is determined by the zeros of the spherical Bessel function, $\lambda = \left(\frac{z_{l,n}}{R}\right)^2$. But when it came to implementation, the details piled up, like correctly accounting for the multiplicities associated with angular momentum indices m and l: for the cylinder, $m = 0$ gives singlets and $m > 0$ gives doublets; for the sphere, each l gives $2l + 1$ degeneracy. Moreover, the cylinder's volume is $\pi R^2 H$, the sphere's is $\frac{4}{3}\pi R^3$, and it's this volume factor V that really governs the main trend in eigenvalue counting! Even though the two shapes are totally different, in the large-λ limit the distribution of eigenvalues only depends on the volume.



What really amazed me was the final graph result: before I tried a small λ like 400, the graph looked hugely different. The graph looks hugely different. I realized that I might need a larger λ. So when I extended λ up to 5000, even though both the cylinder and sphere $N(\lambda)$ curves show that familiar staircase structure (since eigenvalues are discrete), it indeed shows that its respective Weyl asymptotics $\frac{V}{6\pi^2}\lambda^{3/2}$ almost perfectly.

Chapter 4: Lean and Large Language Models (LLMs)

When I start this Thesis Project with Professor Pedro Morales-Almazan. We discussed and explored these capabilities by attempting to formalize mathematical concepts in Lean 4. specifically, defining the Laplace operator and approaching a formal statement of Weyl's Law. Throughout this process, I experienced with both online and local LLMs to assist in writing, debugging, and understanding Lean code.

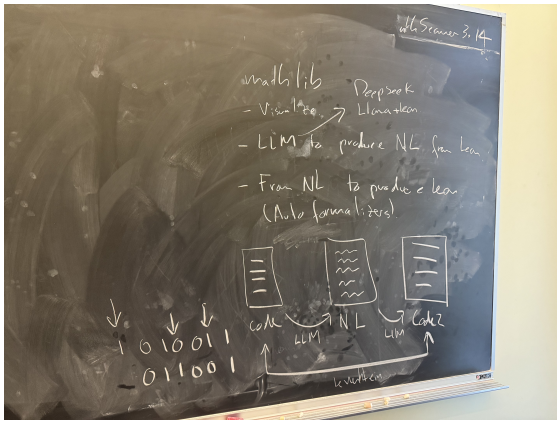


Figure 9: The prototype ideas 1

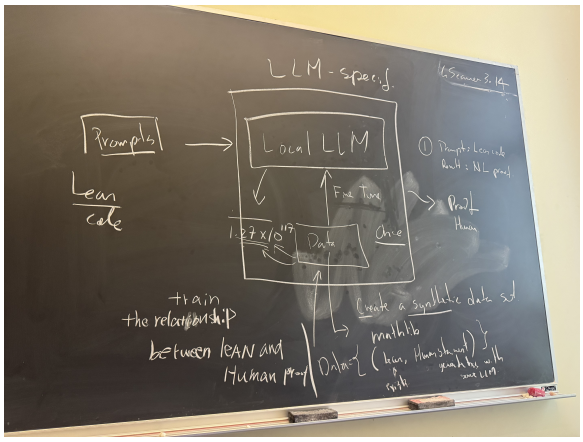


Figure 10: The prototype ideas 2

We are two explorers preparing to chart a new path in an unfamiliar mathematical landscape called Lean 4. I still remember that afternoon: sunlight pouring through the office window as we stood by the blackboard, debating how to translate the classical concept of the Laplace operator into the Lean language. It felt like trying to build a big Lego set—full of challenges. I had a hard time trying to figure out the Lean 4 code.

In the first couple of weeks, I became a researcher working with YouTube videos, Lean code library, and local LLM assistants. I downloaded and experienced both online and local LLMs to assist in writing, debugging, and understanding Lean code. Sometimes these LLM assistants behaved like cool guys, quickly producing LEAN 4 code structures; other times they played as confused helpers, offering some beautiful "fakecode" that Lean's compiler failed. Every time I had to step into the role of detective, I carefully checked each detail and code.

Through this project, I worked with Professor Morales-Almazan to debug the LEAN 4 code. This collaboration with LLM felt less like replacing human reasoning and more like having a thought-provoking partner— who is quick, smart, but also confused.

Chapter 4.1: Introduction of LLMs

LLM, which is also known as Large Language Models, emerged around 2018. A large language model is based on an algorithm that applies neural network techniques with a large number of parameters (Well, it's typically billions of weights or more) to process and understand human languages or text using self-supervised learning techniques. to process and understand human languages or text using self-supervised learning techniques. Its mission includes text generation, translation, machine translation, summary writing, image generation from text, machine coding, chatbots, and communication.

Chapter 4.2: Introduction to LEAN 4 and its Mathlib

In the modern Math community, do you know what is one of the greatest "magic" for Math? Yes! As you read the title, the answer will be LEAN. It is a universal functional programming language and open resource for the mathematics community. Lean isn't just another language; it's powerful a triple tool. It's a powerful functional programming language, a collaborative, open source open to all mathematicians in the world. it more likely a unique bridges that connect the human language, mathematical notation, and programming code. Honestly, a lot of mathematicians are using it now to explore ideas and then keep updating the LEAN and Mathlib.

The benefits of using Lean:

1. When mathematicians write their proofs step by step in LEAN, it allows rigorous verification of mathematical theorems.

2. It provides a logical framework for converting mathematical theorems into code and verifying their correctness Correct conceptual errors while deepening their understanding of the research object, such as Theorem, Example. etc

Lean 4 is the fourth generation of the Lean family. It builds on the success of its predecessor, Lean 3, and goes one step further: it runs faster, more easier to use, and best of all, it makes the two things of "writing proofs" and "writing programs" more comfortable.

Mathlib (mathematics library) is a Lean mathematical tool library maintained by the Lean community, which collects and updates thousands of pre-written definitions, theorems, and proofs around the world. For example, basic addition, subtraction, multiplication and division to calculus, linear algebra, topology and even more advanced mathematical content, Mathlib covers everything.

An easy way to understand their relationship is like this: Lean 4 is "paper and pen", allowing you to write rigorous mathematical content; Mathlib is "a large dictionary + a collection of proofs by mathematicians.", You can directly import theorems or definitions that others have proved, without repeating the work.

Chapter 4.2.1: Getting Started with Lean 4 — Manual Setup and Basic Usage

There are two ways for you to start interacting with Lean. If you want to try Lean without installing it, you can try the online Lean website. It's free and quick. However, If you want to work on a bigger project, I recommend installing it on your own computer, which will give you the most satisfactory experience. For my thesis project, I set up Lean 4 using a combination of tools provided by the Lean community and GitHub. Unlike the standard way, which often involves initializing a new Lean project using the `lake` build system, I followed a more direct way to accomplish this.

First Step: Installed Visual Studio Code (VS Code)

On your device, use your browser(e.g., Chrome, Edge, Safari) to search for Visual Studio Code download, the first one will usually be the correct link to download. In addition, I also recommend to install **Git**, which will be use a lot for control system that Lean's package, uses to download dependencies such as `mathlib` and upload the project at your Github. I will go over step by step guide for setting up these tools on **Windows** and **macOS**.

1. Download and Install Visual Studio Code

1. Search for `Visual Studio Code download`: https://code.visualstudio.com

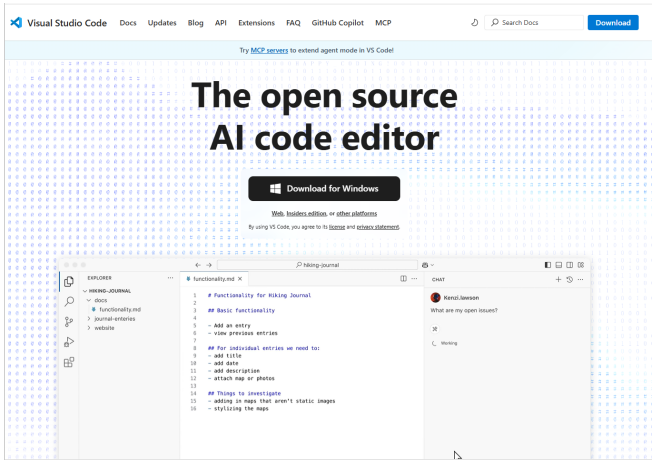


Figure 11: Visual Studio Code page

2. Click on the **Download** button on the top right. The website will detect your system:
   - On **Windows**, download the `.exe` installer.
   - On **macOS**, download the `.dmg` installer.

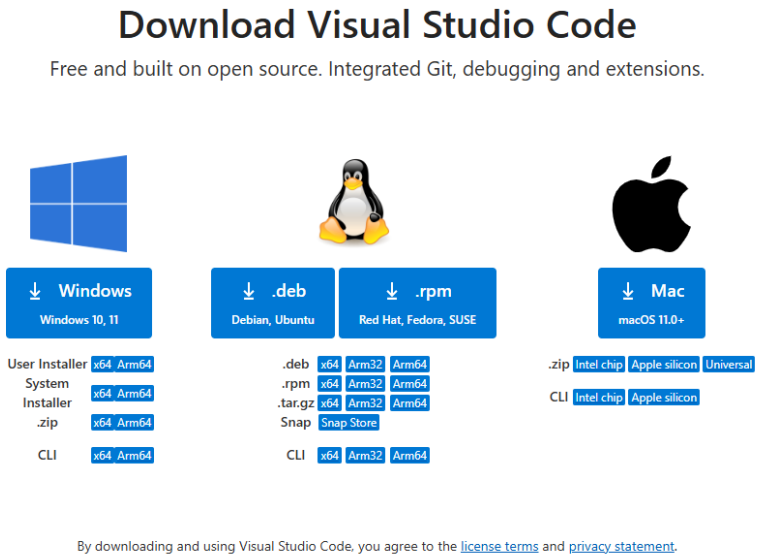


Figure 12: Visual Studio Code download page

3. Run the installer:
   - **Windows**: Double-click the installer and follow the setup wizard.
   - **MacOS**: Open the `.dmg` file and drag the VS Code icon to the `Applications` folder.

Second Step: Install the Lean 4 Extension in VS Code

1. Open VS Code and the leftest column, click on the **Extensions**, then search for `Lean 4`.

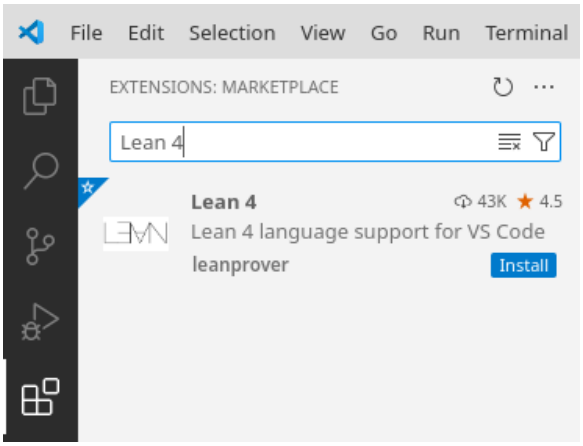


Figure 13: The Extensions and search Lean

2. Click **Install and follow the setup guide for Elan**.

   Open the Lean 4 setup guide by creating a new text file using 'File > New Text File', Clicking on the ∀ - symbol in the top right and selecting 'Documentation".

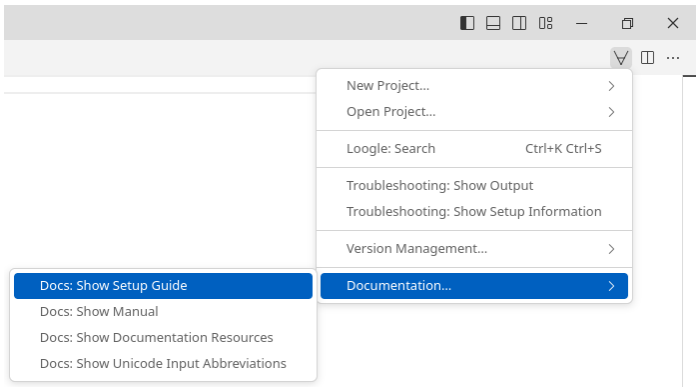


Figure 14: Setup Guide

3. **Follow the Lean 4 and Elan setup guide. It will:**

   - Walk you through learning resources for Lean
   - Teach new users how to set up Lean's dependencies on your platform
   - Make sure Lean 4 is installed and check the version
   - Make sure to install Elan (the Lean version manager)

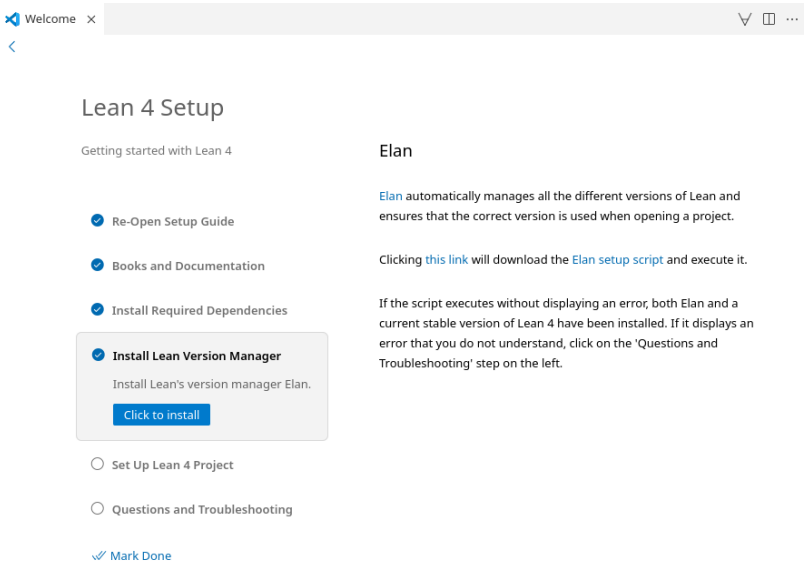


Figure 15: LEAN 4 and Elan Setup Guide

Third Step: Installed Git

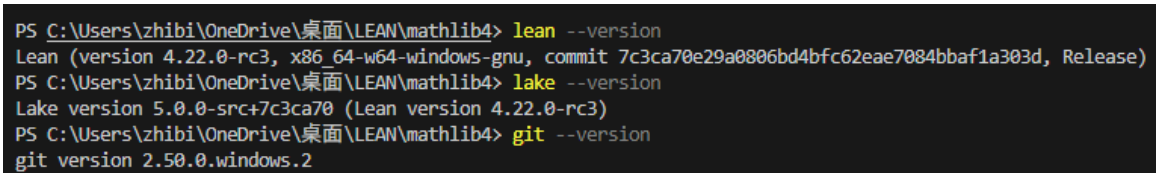
Git is a control system that offers numerous advantages for developers, teams, and organizations. It's perfect to manage your LEAN project. Of Course, it is used by Lean's package manager to fetch libraries such as `mathlib`.

- **Windows**:
  1. Visit `https://git-scm.com/download/win`
  2. The installer will download automatically.
     Run the file and follow the setup instructions (default options are sufficient).
  3. Open `Command Prompt` and type `git --version` to confirm the installation.
- **MacOS**:
  1. Visit `https://git-scm.com/download/mac`.
  2. If Git is not installed, macOS will prompt you to install the `Xcode Command Line Tools`. Click `Install` and follow the prompts.

Verifying Your Setup

To ensure all step above are correctly installed, open VS Code and the integrated terminal, then type the following commands:

1. lean –version
2. lake –version
3. git –version



As you can see the picture above, if each command returns a version number, your setup is complete and ready to use for Lean 4 development with mathlib.

More direct way to approach the Mathlib

Unlike the standard workflow, which often involves initializing a new Lean project using the `lake` build system, I use a more direct way. As you all read the above step, I installed Visual Studio Code (VS Code) and added the Lean 4 extensions. This extension provides syntax highlighting, proof state visualization, and interactive feedback within the editor. Instead of using `lake` to generate a new project, I git cloned the official `mathlib4` from GitHub:

git clone https://github.com/leanprover-community/mathlib4

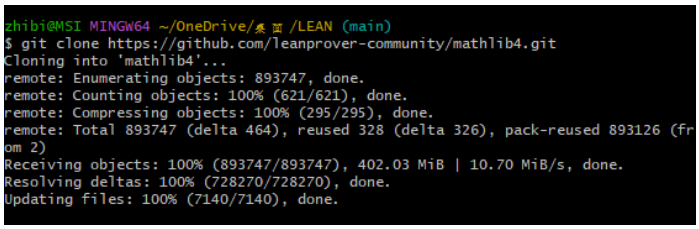


Figure 16: git clone the website

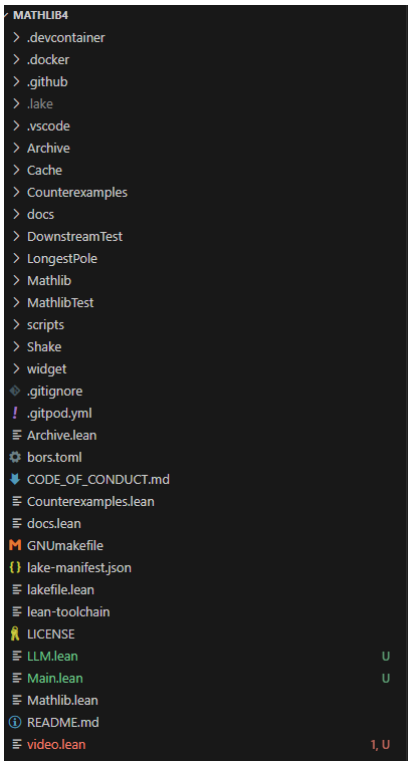


Figure 17: Mathlib at VS Code

After cloning, I opened the entire `mathlib4` directory in VS Code. Then, I created my own Lean files (name.lean) and wrote Lean code directly inside the library's source files. I understand this way does not sound professional. But it allowed us to immediately access the device and use Mathlib's existing modules without additional dependency configuration.

Most important thing: I was still able to define functions, prove basic lemmas, and begin modeling differential operators like the Laplacian using `mathlib`'s existing formalized structures. Of course, if I need to update my Mathlib, I just need to run `git pull` again.

While that method offers better project isolation and build reproducibility, the approach I used is sufficient for small-scale research projects and experimentation. This setup provided a friendly environment for students and new beginners to learn Lean syntax, explore `mathlib`, and develop initial attempts to formalize results such as the eigenvalue problem and Weyl's Law.

Chapter 4.2.2: How to write and run basic Lean code

We can now move on to a new chapter. As I talked about in the previous chapter, Lean is a functional programming language and theorem prover designed for formalizing mathematics and verifying logical arguments.

Import the Libraries

A perfect Lean file begins with import statements, and some readers may be confused. Why? An easy way to understand is that importing different statements will bring a lot of definitions and theorems from external libraries such as mathlib4.

When a user writes Lean code, especially when working on formal mathematics using `mathlib4`, it is common to begin by importing relevant modules. As you can see in the picture below, I had imported a lot of different libraries.

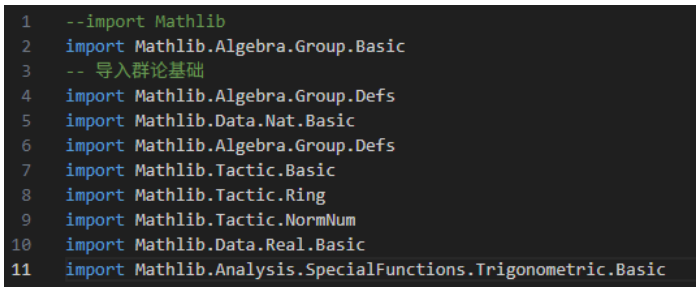


Figure 18: Different import

- `import Mathlib.Algebra.Group.Basic`
  This module provides basic definitions and properties of algebraic groups. It includes group axioms, group operations, and basic lemmas that are foundational in abstract algebra.

- `import Mathlib.Algebra.Group.Defs`
  This import contains core typeclass definitions for groups, monoids, and semigroups. It introduces the abstract structure (such as `Group`, `CommGroup`, `Monoid`) without additional theorems.

- `import Mathlib.Data.Nat.Basic`
  Includes basic facts and operations on natural numbers ($\mathbb{N}$). It defines addition, multiplication, ordering, and basic number-theoretic results.

- `import Mathlib.Tactic.Basic`
  This module provides basic tactics used in constructing proofs. Examples include `intro`, `assumption`, `exact`, and simple case analysis.

- `import Mathlib.Tactic.Ring`
  Provides the `ring` tactic, which is useful for automatically solving equalities involving ring expressions. It works in any ring-like structure.

- `import Mathlib.Tactic.NormNum`
  This enables the `norm_num` tactic, which evaluates numeric expressions (e.g., simplifying $3 + 5 \times 2$ to 13) and proves numeric equalities automatically.

- `import Mathlib.Data.Real.Basic`
  Offers foundational definitions and properties of real numbers ($\mathbb{R}$), including arithmetic, ordering, and limits. It is essential for calculus and analysis.

- `import Mathlib.Analysis.SpecialFunctions.Trigonometric.Basic`
  This module introduces basic definitions and properties of trigonometric functions such as sin, cos, and tan, as well as their relationships and derivatives in real analysis.

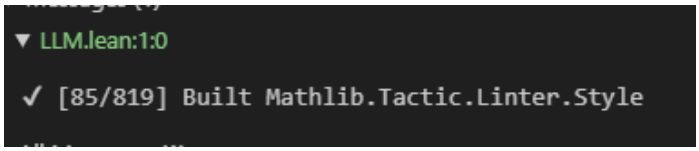
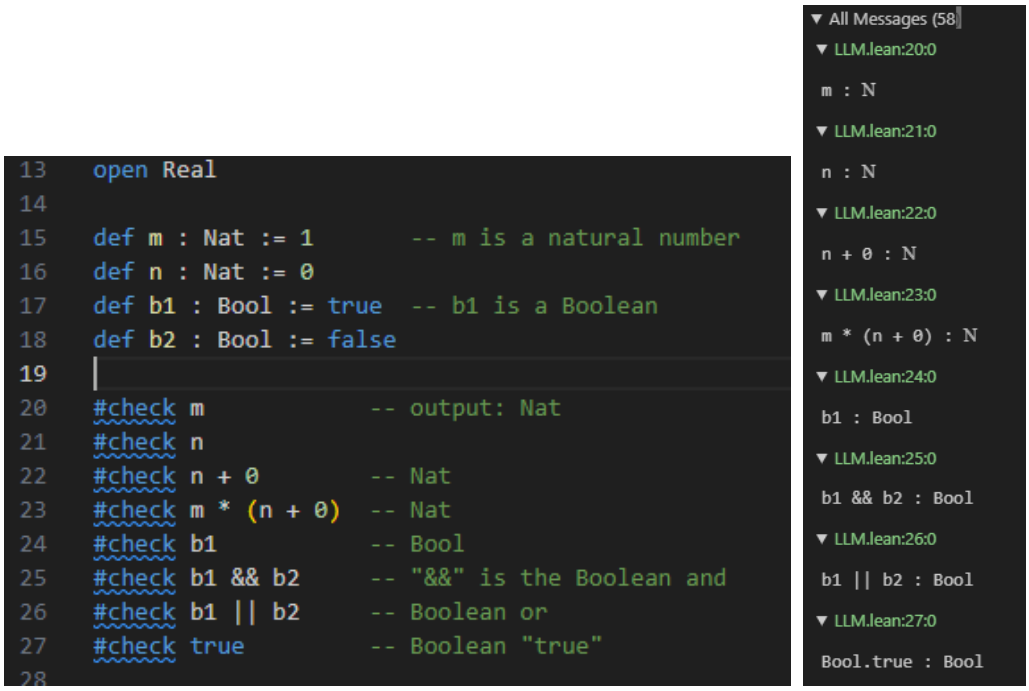


Figure 19: Importing more libraries also means it takes a longer time for your device to upload.

These imports collectively allow the user to work with number theory, abstract algebra, real analysis, and proof automation within a single Lean environment. Of course, there are more libraries to import, but I believe it should be good enough for new users to try their first Lean project.

Writing and Evaluating Basic Lean Code

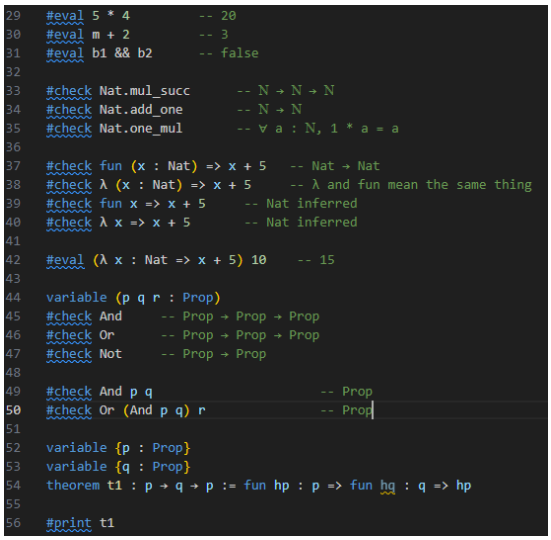
It's time to learn some basic code, I will try my best to explain the code. The following Lean code demonstrates basic data types, arithmetic operations, Boolean logic, function expressions, and simple propositional logic.

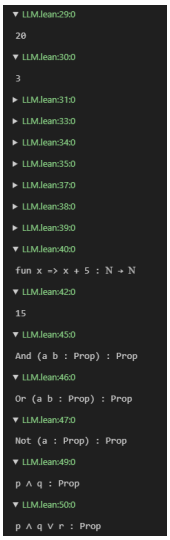


```
13  open Real
14
15  def m : Nat := 1       -- m is a natural number
16  def n : Nat := 0
17  def b1 : Bool := true  -- b1 is a Boolean
18  def b2 : Bool := false
19
20  #check m               -- output: Nat
21  #check n
22  #check n + 0           -- Nat
23  #check m * (n + 0)     -- Nat
24  #check b1              -- Bool
25  #check b1 && b2        -- "&&" is the Boolean and
26  #check b1 || b2        -- Boolean or
27  #check true            -- Boolean "true"
28
```

```
▼ All Messages (58)
▼ LLM.lean:20:0
m : N
▼ LLM.lean:21:0
n : N
▼ LLM.lean:22:0
n + 0 : N
▼ LLM.lean:23:0
m * (n + 0) : N
▼ LLM.lean:24:0
b1 : Bool
▼ LLM.lean:25:0
b1 && b2 : Bool
▼ LLM.lean:26:0
b1 || b2 : Bool
▼ LLM.lean:27:0
Bool.true : Bool
```

- `open Real`
  Opens the namespace `Real`, allowing direct access to real number definitions and theorems.
- `def m : Nat := 1`, `def n : Nat := 0`
  Defines two natural number variables: `m` is set to 1, and `n` is 0.
- `def b1 : Bool := true`, `def b2 : Bool := false`
  Defines two Boolean values: `b1` is `true` and `b2` is `false`.
- `#check` expressions
  The `#check` command displays the type of an expression. For example:
  - `#check m` returns `Nat`
  - `#check b1 && b2` checks the type of the Boolean conjunction, which is `Bool`
  - `#check Nat.one_mul` returns the type `∀ a : ℕ, 1 * a = a`





- `#eval` expressions
  The `#eval` command evaluates expressions. For example:
  - `#eval 5 * 4` returns `20`
  - `#eval m + 2` returns `3`
  - `#eval b1 && b2` returns `false`
- **Anonymous functions (lambdas)**
  Lean supports lambda expressions using either `fun` or the lambda symbol λ. For example:
  - `fun (x : Nat) => x + 5` defines a function from `Nat` to `Nat`
  - `#eval (λ x : Nat => x + 5) 10` evaluates to `15`
- **Logical operations**
  Lean includes propositional logic operations:
  - `And`, `Or`, and `Not` correspond to logical conjunction, disjunction, and negation respectively.
  - `variable (p q r : Prop)` declares three propositional variables.
  - `#check And p q` checks that combining two propositions yields another proposition.
- **Simple theorem construction**
  The statement below defines a simple theorem that returns the first argument given both `p` and `q`:

$$\text{theorem t1} : p \to q \to p := \text{fun hp} : p => \text{fun hq} : q => hp$$

Proves that if both `p` and `q` are true, then `p` must be true. This is a basic example of constructive logic using lambda expressions to represent assumptions. These examples

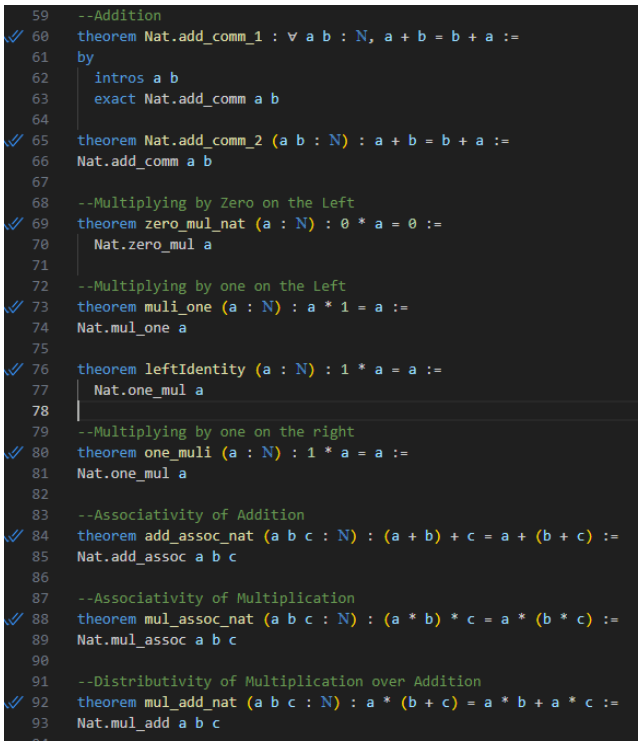
demonstrate Lean's expressive capability in both computation and formal proof construction, providing a foundation for writing more advanced mathematical logic.

Theorem and Example

Lean 4 enables users to verify logical statements. Two important keywords for this are `theorem` and `example`.

- "Theorem" in Lean allows users to formally state and prove the results using logic and definitions already available in the system or defined by the user.
- "Example" is used to state a proposition that you want to prove, but without giving it a name. It's like a one time unnamed theorem for learning or showing small logical truths quickly.

Both of them support the use of concrete examples through `#check` and `#eval` commands. Therefore, using both `theorem` and `example`, users can move fluidly between informal exploration and fully formalized reasoning. The following theorems represent the basic properties in Lean, such as commutativity, associativity, identity, and distributivity.



```
59  --Addition
60  theorem Nat.add_comm_1 : ∀ a b : ℕ, a + b = b + a :=
61  by
62    intros a b
63    exact Nat.add_comm a b
64
65  theorem Nat.add_comm_2 (a b : ℕ) : a + b = b + a :=
66  Nat.add_comm a b
67
68  --Multiplying by Zero on the Left
69  theorem zero_mul_nat (a : ℕ) : 0 * a = 0 :=
70    Nat.zero_mul a
71
72  --Multiplying by one on the Left
73  theorem muli_one (a : ℕ) : a * 1 = a :=
74  Nat.mul_one a
75
76  theorem leftIdentity (a : ℕ) : 1 * a = a :=
77    Nat.one_mul a
78
79  --Multiplying by one on the right
80  theorem one_muli (a : ℕ) : 1 * a = a :=
81  Nat.one_mul a
82
83  --Associativity of Addition
84  theorem add_assoc_nat (a b c : ℕ) : (a + b) + c = a + (b + c) :=
85  Nat.add_assoc a b c
86
87  --Associativity of Multiplication
88  theorem mul_assoc_nat (a b c : ℕ) : (a * b) * c = a * (b * c) :=
89  Nat.mul_assoc a b c
90
91  --Distributivity of Multiplication over Addition
92  theorem mul_add_nat (a b c : ℕ) : a * (b + c) = a * b + a * c :=
93  Nat.mul_add a b c
```

Figure 22: Theorem

A theorem is used to declare a mathematical result.

- The syntax theorem name:

  statement := proof defines a named fact, where:

1 The name is your chosen label for the theorem.

2 The statement is the proposition you're proving,

3 Proof is how it's proven (often using previously established results, such as Nat.add assoc).

I had coded a lot of "examples" to show a variety of logical and arithmetic statements using Lean 4's `example` keyword. These examples are inspired by topics covered in UCSC's Math 100, such as propositional logic, basic equivalences, and natural number arithmetic. The use of `example` allows us to explore logical truths or arithmetic identities without formally naming each result as a theorem. This is ideal for learning and for quickly checking the validity of propositions in Lean.

```
-- other properties
open Classical
example : (p → (q → r)) ↔ (p ∧ q → r) := sorry
example : ((p ∨ q) → r) ↔ (p → r) ∧ (q → r) := sorry
example : ¬(p ∨ q) ↔ ¬p ∧ ¬q := sorry
example : ¬p ∨ ¬q → ¬(p ∧ q) := sorry
example : ¬(p ∧ ¬p) := sorry
example : p ∧ ¬q → ¬(p → q) := sorry
example : ¬p → (p → q) := sorry
example : (¬p ∨ q) → (p → q) := sorry
example : p ∨ False ↔ p := sorry
example : p ∧ False ↔ False := sorry
example : (p → q) → (¬q → ¬p) := sorry

example : (p → q ∨ r) → ((p → q) ∨ (p → r)) := sorry
example : ¬(p ∧ q) → ¬p ∨ ¬q := sorry
example : ¬(p → q) → p ∧ ¬q := sorry
example : (p → q) → (¬p ∨ q) := sorry
example : (¬q → ¬p) → (p → q) := sorry
example : p ∨ ¬p := sorry
example : (((p → q) → p) → p) := sorry


variable (a b c : Nat)

example : a + 0 = a := Nat.add_zero a
example : 0 + a = a := Nat.zero_add a
example : a * 1 = a := Nat.mul_one a
example : 1 * a = a := Nat.one_mul a
example : a + b = b + a := Nat.add_comm a b
example : a + b + c = a + (b + c) := Nat.add_assoc a b c
example : a * b = b * a := Nat.mul_comm a b
example : a * b * c = a * (b * c) := Nat.mul_assoc a b c
example : a * (b + c) = a * b + a * c := Nat.mul_add a b c
example : a * (b + c) = a * b + a * c := Nat.left_distrib a b c
example : (a + b) * c = a * c + b * c := Nat.add_mul a b c
example : (a + b) * c = a * c + b * c := Nat.right_distrib a b c
```

Figure 23: Example

At the first part, I begin with examples based on classical propositional logic. The following statements rely on common logical rules such as De Morgan's laws, the law of the excluded middle, and contraposition. The `open Classical` command enables classical logic features in Lean.

- De Morgan's Laws:$\neg(p \vee q) \leftrightarrow \neg p \wedge \neg q$.
- Contraposition: $(p \to q) \to (\neg q \to \neg p)$.
- Classical tautologies such as $p \vee \neg p$.

At the second part, I also explore examples that involve basic identities over natural numbers, using `Nat.add_comm` and `Nat.mul_assoc`. These are consistent with topics from Math 100 that discuss algebraic properties of operations.

The second part properties also reflect:

- The identity elements for addition and multiplication.
- Commutativity and associativity of addition and multiplication.
- Distributive laws of multiplication over addition.

Chapter 4.2.3: Comparing Human LEAN Coded Proofs vs LLM Generated Proofs

This Chapter mainly focuses on the comparison, also highlighting how web based access to high-parameter models can significantly outperform smaller, locally run versions, especially in tasks involving formal logic, theorem proving, and symbolic reasoning. In recent years, formal proof assistants such as `Lean 4` have gained prominence in both mathematics and computer science communities due to their ability to verify mathematical proofs with rigorous precision. At the same time, the advent of large language models (LLMs) has introduced new possibilities for automating theorem proving and formal code generation.

In our project, Professor Pedro Morales-Almazan and I investigate and compare human written Lean 4 proofs to those generated by several LLM-based systems, including

(1) DeepSeek + llama 8B

(2) Qwen 14B

(3) llama 3.2 8B

(4) DeepSeek Website page

Due to limited hardware capacity on my personal computer, I was unable to run large-scale models like Deepseek + LLaMA 128B locally. Instead, I accessed the web version, which relies on large scale backend infrastructure, typically involving distributed clusters of GPUs (such as A100 or H100), optimized with parallelized tensor computation. This makes the web based Deepseek model the most intelligent and capable among the ones I tested.

Among the local four LLMs, the Deepseek + LLaMA 8B model is ”smarter” than the others because it shows the best ability in proof generation, followed by Qwen, and then by the base LLaMA 7B. To be honest, I personally think Deepseek’s alignment with logic, build, and math prover is impressive, likely due to its domain-specific pretraining on code and mathematics.

By analyzing the syntactic correctness, semantic soundness, and readability of the generated Lean code, we aim to evaluate the current capabilities and limitations of LLMs in formal mathematical reasoning. Our study draws on selected couple laws and theorems from UCSC’s MATH 100 curriculum, emphasizing foundational logic and equivalences such as the commutativity of conjunction. We hope this work contributes to the understanding of how AI can augment or assist formal mathematical education and research.

Code comparison of Local LLM, Webpage LLM, Human Written.

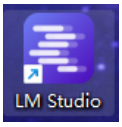


Figure 24: LM Studio

To run local models, I installed an app called **LM Studio**, which provides an easy to use interface for downloading and interacting with a wide variety of open-source LLMs. LM Studio allows users to select models such as LLaMA, Qwen, and Deepseek in different parameter sizes (e.g., 7B, 8B), and run them locally using quantized formats optimized for lower-end hardware (such as 4-bit GGUF versions). This platform made it convenient for me to experiment with different LLMs and directly compare their performance in generating Lean 4 proofs. While local inference is limited by hardware constraints such as RAM and VRAM capacity LM Studio nonetheless enabled meaningful exploration of smaller models and served as a practical tool for testing and benchmarking formal reasoning tasks.

When the user opens the LM Studio, on the left side, there is a magnifier icon(search button). which allows users to explore more open resource LLMs. By clicking on it, users can search for and download models such as `LLaMA`, `Qwen`, or `Deepseek` in different configurations (e.g., 7B, 8B, 13B, or even 70B+). However, if you are using your personal device, I strongly recommend not downloading too big llm.

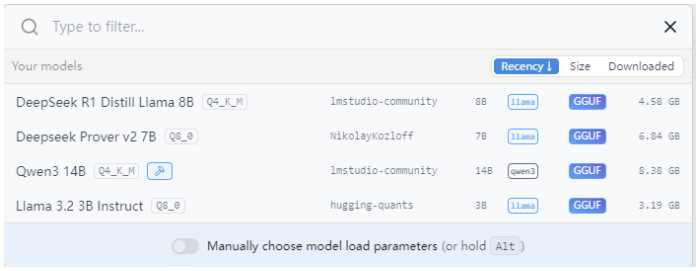


Figure 25: LM Studio

The reason is that for those using personal computers or laptops with limited RAM and VRAM, large models such as `Deepseek + LLaMA 128B`, as downloading or running these can easily crash or freeze your system. These larger models typically require professional GPUs or server-grade hardware with high memory bandwidth and support for multi-GPU parallelization. In my case, I am using a personal laptop equipped with 16GB of RAM. While this configuration is not perfect for machine learning tasks, it is ok for running small to medium sized models such as `LLaMA 7B` or `Deepseek 8B`. Therefore, please remember that the models like `8B` or `7B` are far more manageable for regular users.

Code Comparison 1: Associativity of Addition in Lean

```
--Human Written Code LLM
------------------
variable (R : Type*) [Ring R]

--Human Code
#check (add_assoc : ∀ a b c : R, a + b + c = a + (b + c))

--The code I wrote
example (a b c : R) : (a + b) + c = a + (b + c) :=
  add_assoc a b c

example (a b c : ℕ) : (a + b) + c = a + (b + c) :=
Nat.add_assoc a b c

--Webpage Deepseek Code
example (x y z : R) : x + (y + z) = (x + y) + z :=
  (add_assoc x y z).symm

-- Local LLM
example (a b c : ℕ) : (a + b) + c = a + (b + c) :=
linarith [add_assoc]

-- After fix
example (a b c : ℕ) : (a + b) + c = a + (b + c) :=
by exact add_assoc a b c
```

Figure 26: Example code

In the first example, we consider the associativity of addition over the natural numbers and a general ring R. The human-written code is straightforward and minimal, using the known theorem `add_assoc` directly:

```
#check (add_assoc :  a b c : R, a + b + c = a + (b + c))
example (a b c : R) : (a + b) + c = a + (b + c) :=
  add_assoc a b c
```

For natural numbers:

```
example (a b c : N) : (a + b) + c = a + (b + c) :=
  Nat.add_assoc a b c
```

The webpage-based large language model (LLM), such as DeepSeek or ChatGPT, demonstrates a deeper understanding by using symmetry and invoking the theorem in the reverse direction:

```
example (x y z : R) : x + (y + z) = (x + y) + z :=
  (add_assoc x y z).symm
```

This version is not only logically valid but also shows a more nuanced grasp of how to manipulate theorems in Lean.
In contrast, a local 8B LLM attempted the proof using:

```
example (a b c : N) : (a + b) + c = a + (b + c) :=
  linarith [add_assoc]
```

This attempt is flawed for two reasons. First, the tactic linarith is not suitable for proving associativity directly; it is generally used for linear arithmetic inequalities, not for rewriting based on named lemmas. Second, the identifier add assoc is not recognized in this context unless explicitly imported or defined, and even if available, linarith does not use it in a meaningful way. This indicates that the local LLM lacks contextual understanding of tactic applicability and lemma usage in Lean. As a result, it gives the impression of being "not very smart" or superficial in reasoning.

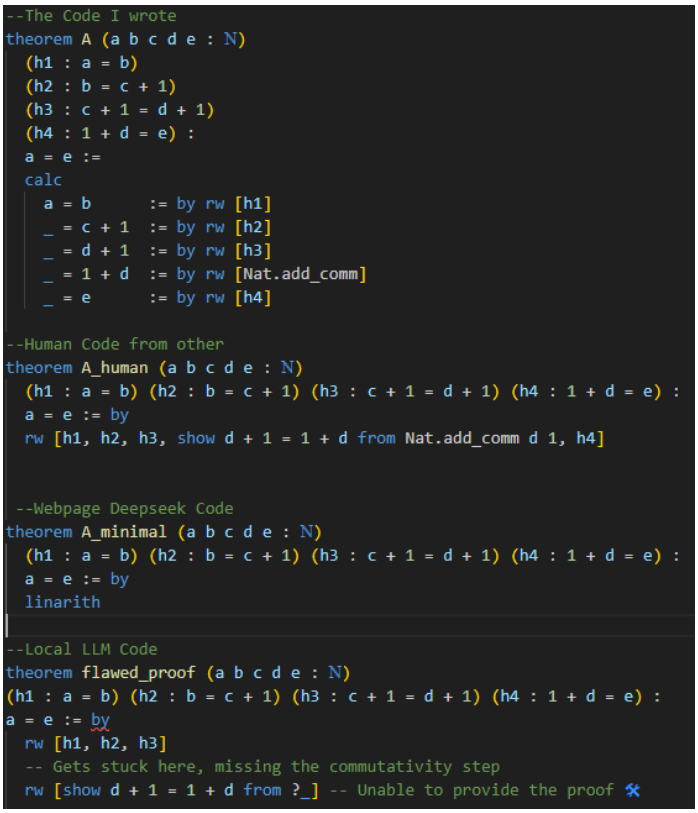
After minimal human correction, the fixed version becomes:

```
example (a b c : N) : (a + b) + c = a + (b + c) :=
  by exact add_assoc a b c
```

This comparison shows that the human-written and web LLM-generated code is more accurate and idiomatic, whereas the local LLM tends to misuse tactics or demonstrate weaker reasoning. Furthermore, both human and web-based LLM solutions are cleaner and better aligned with Lean's proof idioms.

When working with a local LLM model (8B), I noticed that it often failed to understand simple Lean tactics or produced incorrect code. For example, when attempting to prove the associativity of addition using `linarith [add_assoc]`, the local LLM generated responses with logical or syntax errors, such as referencing unknown identifiers or applying irrelevant lemmas. In contrast, the web-based LLM was noticeably more intelligent and helpful. It quickly identified the correct approach, suggested cleaner alternatives like using `(add_assoc x y z).symm`, and presented more elegant and readable proofs. Additionally, both the human-written code and the webpage LLM's suggestions maintained a neat, structured style that the local LLM struggled to replicate. Overall, the web LLM demonstrated a higher level of mathematical understanding and clarity in its Lean code output.

Code Comparison 2: Equality Chain Proofs in Lean



```
--The Code I wrote
theorem A (a b c d e : ℕ)
  (h1 : a = b)
  (h2 : b = c + 1)
  (h3 : c + 1 = d + 1)
  (h4 : 1 + d = e) :
  a = e :=
  calc
    a = b      := by rw [h1]
    _ = c + 1  := by rw [h2]
    _ = d + 1  := by rw [h3]
    _ = 1 + d  := by rw [Nat.add_comm]
    _ = e      := by rw [h4]

--Human Code from other
theorem A_human (a b c d e : ℕ)
  (h1 : a = b) (h2 : b = c + 1) (h3 : c + 1 = d + 1) (h4 : 1 + d = e) :
  a = e := by
  rw [h1, h2, h3, show d + 1 = 1 + d from Nat.add_comm d 1, h4]


 --Webpage Deepseek Code
theorem A_minimal (a b c d e : ℕ)
  (h1 : a = b) (h2 : b = c + 1) (h3 : c + 1 = d + 1) (h4 : 1 + d = e) :
  a = e := by
  linarith

--Local LLM Code
theorem flawed_proof (a b c d e : ℕ)
(h1 : a = b) (h2 : b = c + 1) (h3 : c + 1 = d + 1) (h4 : 1 + d = e) :
a = e := by
  rw [h1, h2, h3]
  -- Gets stuck here, missing the commutativity step
  rw [show d + 1 = 1 + d from ?_] -- Unable to provide the proof 🛠
```

In example theorem A, the code is more complex compared to the first example. We explore different approaches to proving equality chains through a series of transitive relations. Two human written codes with two different styles of handling such proofs Let's take a step by step structure:

```
theorem A (a b c d e : \mathbb{N})
(h1 : a = b)
(h2 : b = c + 1)
(h3 : c + 1 = d + 1)
(h4 : 1 + d = e) :
a = e :=
calc
  a = b      := by rw [h1]
  _ = c + 1  := by rw [h2]
  _ = d + 1  := by rw [h3]
  _ = 1 + d  := by rw [Nat.add_comm]
  _ = e      := by rw [h4]
```

The purpose I wrote this code is to provide clear, readable proof steps that closely mirror mathematical reasoning. Of course, it also looks stupid because of each extra step. But I recommend it for every new user since it makes the proof easy to follow and verify.

The second human written I explore online and it approach uses a more compact rewrite:

```
theorem B (a b c d e : \mathbb{N})
(h1 : a = b)
(h2 : b = c + 1)
(h3 : c + 1 = d + 1)
(h4 : 1 + d = e) : a = e :=
by rw [h1, h2, h3, Nat.add_comm, h4]
```

This version demonstrates efficiency and familiarity with Lean's rewrite system. In Lean, the code `rw` stands for *rewrite*. It is used to replace one expression with another, based on an equality that has already been established in the context. Let's take a closer look at the last line:

```
by rw [h1, h2, h3, Nat.add_comm, h4]
```

it rewrites:

1. `rw [h1]` replaces a with b.
2. `rw [h2]` replaces b with $c + 1$.
3. `rw [h3]` replaces $c + 1$ with $d + 1$.
4. `rw [Nat.add_comm]` uses the commutativity of addition to rewrite $d + 1$ as $1 + d$.
5. `rw [h4]` replaces $1 + d$ with e.

At the end, the goal $a = e$ is established. Thus, the `rw` is a short cut way of expressing the familiar process of applying equalities step by step to transform one side of an equation into the other.

A webpage LLM suggests a more direct approach:

```
theorem A_minimal (a b c d e : \mathbb{N})
(h1 : a = b) (h2 : b = c + 1) (h3 : c + 1 = d + 1) (h4 : 1 + d = e) :
a = e := by
  linarith
```

This solution applies Lean's code `linarith`, which looks at all the assumptions (equalities, inequalities) in the context of the current proof and tries to prove the current goal. It is particularly good at handling linear relations such as the addition, subtraction, and multiplication of constants. LLM uses this code to let computers handle the tedious details, allowing humans to focus on step logic.

In a local 8B LLM produced a flawed attempt:

```
theorem flawed_proof (a b c d e : \mathbb{N})
(h1 : a = b) (h2 : b = c + 1) (h3 : c + 1 = d + 1) (h4 : 1 + d = e) :
a = e := by
  rw [h1, h2, h3]
  -- Gets stuck here, missing the commutativity step
  rw [show d + 1 = 1 + d from ?_] -- Unable to provide the proof
```

The Local LLM lean code fails for two main reasons. First, it correctly applies the initial rewrites, but then stalls at the commutativity step. Second, it uses an incomplete show statement without providing the necessary proof term. You may read the following theorem B I debug with Professor Pedro. I will not cover the details because they have the same idea as Theorem A.

```
--The Code I wrote
theorem B (a b c d e : N)
  (h1 : a = b)
  (h2 : b = c + 1)
  (h3 : c + 1 = d + 1)
  (h4 : 1 + d = e) : a = e :=
  by rw [h1, h2, h3, Nat.add_comm, h4]

-- Human Code (More explicit)
theorem B_explicit (a b c d e : N)
  (h1 : a = b) (h2 : b = c + 1) (h3 : c + 1 = d + 1) (h4 : 1 + d = e) :
  a = e := by
  rw [h1, h2, h3, Nat.add_comm d 1, h4]

--Webpage DeepSeek Code
theorem B_auto (a b c d e : N)
  (h1 : a = b) (h2 : b = c + 1) (h3 : c + 1 = d + 1) (h4 : 1 + d = e) :
  a = e := by
  linarith

--Local 8B LLM Attempt
theorem B_flawed (a b c d e : N)
  (h1 : a = b) (h2 : b = c + 1) (h3 : c + 1 = d + 1) (h4 : 1 + d = e) :
  a = e := by
  rw [h1, h2, h3]
  -- Stuck here, doesn't handle the commutativity properly
  sorry
  --Webpage Deepseek Code
```

Overall, we saw two examples of comparison show that human written Lean code tends to be more intentional and pedagogically sound, while advanced web based LLMs can generate correct but potentially opaque solutions. Meanwhile, smaller local LLMs are not smart enough and sometimes they still struggle with the logical reasoning required for non-trivial mathematical proofs.

Chapter 5: Weyl Law direct proof and prototype format in Lean 4

Chapter 5.1: Proof of Weyl aymptotic

We all know that the goal of Weyl's law is to determine the rate of growth of eigenvalues of the Laplacian. We will prove the Weyl aymptotic in 2 dimensions, which is also called as "Dirichlet Neumann bracketing". Then higher dimensional proof is similar. There are 3 steps we need to establish in order to prove the Weyl law asymptotic.

Step 1: Rectangular domains

From Chapter 3.2.2 (10), we know that:

> For the rectangle $(0, L) \times (0, M)$ with area Area $= LM$, the Dirichlet eigenvalues $\lambda_1, \lambda_2, \lambda_3, \ldots$ (in increasing order) satisfy [5]:
>
> $$\lambda_n \sim \frac{4\pi n}{\text{Area}} \quad \text{as } n \to \infty$$
>
> The same asymptotic holds for Neumann eigenvalues μ_n. We give a proof for Dirichlet eigenvalues[5]

Proof. For any $\alpha > 0$, define the eigenvalue counting function:

$$N(\alpha) = \#\{\text{eigenvalues} \leqslant \alpha\} = \#\left\{(j,k) \in \mathbb{N} \times \mathbb{N} : \frac{j^2}{L^2} + \frac{k^2}{M^2} \leqslant \frac{\alpha}{\pi^2}\right\}$$

Starting from the eigenvalue condition:

$$\lambda_{j,k} = \frac{j^2}{L^2} + \frac{k^2}{M^2} \leqslant \frac{\alpha}{\pi^2}$$

Multiply both sides by L^2M^2:

$$M^2j^2 + L^2k^2 \leqslant \frac{\alpha L^2M^2}{\pi^2}$$

Divide both sides by $\frac{\alpha L^2M^2}{\pi^2}$ and simplify this and we get:

$$\frac{M^2j^2}{\frac{\alpha L^2M^2}{\pi^2}} + \frac{L^2k^2}{\frac{\alpha L^2M^2}{\pi^2}} \leqslant 1 \Longrightarrow \frac{j^2}{\frac{\alpha L^2}{\pi^2}} + \frac{k^2}{\frac{\alpha M^2}{\pi^2}} \leqslant 1$$

Let $a = \frac{\sqrt{\alpha}L}{\pi}$ and $b = \frac{\sqrt{\alpha}M}{\pi}$. simplify it around:

$$a^2 = \frac{\alpha L^2}{\pi^2}, \quad b^2 = \frac{\alpha M^2}{\pi^2}$$

Substitute back:

$$\frac{j^2}{a^2} + \frac{k^2}{b^2} \leqslant 1$$

This is exactly the equation of an ellipse in the (j,k)-plane with semi-axes a and b. As we see the picture below:

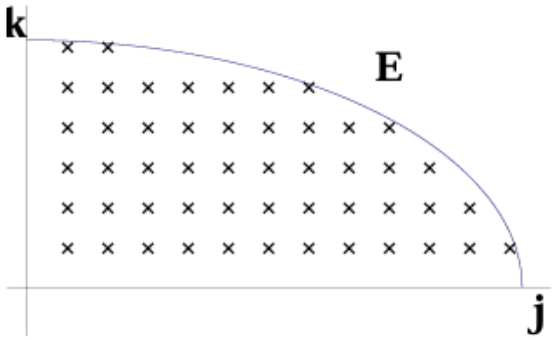


Figure 27: upper bound

This means (j,k) lies in the ellipse $E : \frac{x^2}{a^2} + \frac{y^2}{b^2} \leqslant 1$ in the first quadrant.
Thus:

$$N(\alpha) = \#\{(j,k) \in \mathbb{N} \times \mathbb{N} : (j,k) \in E\}$$

We connect each lattice point $(j,k) \in E$ with the square

$$S(j,k) = [j-1, j] \times [k-1, k]$$

Because the ellipse E is convex and $(j,k) \in E$, the entire square $S(j,k)$ lies within E. for any $(x,y) \in S(j,k)$, we have $x \leqslant j$ and $y \leqslant k$, so:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} \leqslant \frac{j^2}{a^2} + \frac{k^2}{b^2} \leqslant 1$$

These squares are disjoint and all lie within the first quadrant of ellipse E. Therefore, the total area of these squares is at most the area of E in the first quadrant.

Each square has area 1, so the total area is $N(\alpha)$. The total area of ellipse E is πab, so the area in the first quadrant is $\frac{1}{4}\pi ab$. Substituting a and b:

$$\frac{1}{4}\pi ab = \frac{1}{4}\pi \cdot \frac{\sqrt{\alpha}L}{\pi} \cdot \frac{\sqrt{\alpha}M}{\pi} = \frac{LM}{4\pi}\alpha = \frac{\text{Area}}{4\pi}\alpha$$

Therefore:

$$N(\alpha) \leqslant \text{area of E in first quadrant: } \frac{1}{4}\pi ab \leqslant \textbf{upper bound: } \frac{\text{Area}}{4\pi}\alpha$$

In the reverse direction, consider the union of all squares $S(j,k)$ with $(j,k) \in E$.
It covers the ellipse E shifted down and left by one unit, restricted to the first quadrant:

$$\bigcup S(j,k) \supset (E-(1,1)) \cap (\text{first quadrant})$$

For any $(x,y) \in (E-(1,1)) \cap (\text{first quadrant})$, it's true that $(x+1, y+1) \in E$ and $x \geqslant 0, y \geqslant 0$. Then there exist integers j,k and we set $j = \lfloor x+1 \rfloor$ and $k = \lfloor y+1 \rfloor$ such that $(x,y) \in S(j,k)$ and $(j,k) \in E$. The area of $(E-(1,1)) \cap (\text{first quadrant})$ equals the area of E in the region $u \geqslant 1, v \geqslant 1$, where $u = x+1, v = y+1$. The area of E in the first quadrant is $\frac{1}{4}\pi ab$.

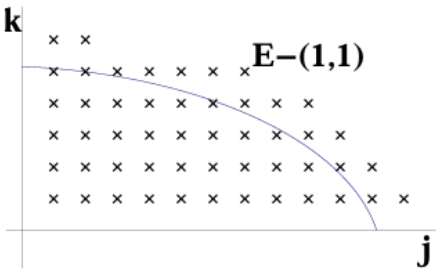


Figure 28: lower bound

Removing the regions $u < 1$ and $v < 1$ reduces the area by at most $a+b$. Thus:

$$\text{Area of union} \geqslant \frac{1}{4}\pi ab - a - b$$

Because the squares are disjoint and each has area 1, the area of the union is exactly $N(\alpha)$.

$$\frac{1}{4}\pi ab = \frac{\text{Area}}{4\pi}\alpha, \quad a+b = \frac{\sqrt{\alpha}(L+M)}{\pi} = \frac{\text{Perimeter}}{2\pi}\sqrt{\alpha}$$

Therefore:

$$N(\alpha) \geqslant \textbf{lower bound: } \frac{\text{Area}}{4\pi}\alpha - \frac{\text{Perimeter}}{2\pi}\sqrt{\alpha}$$

Finally, we combining the upper and lower bounds:

$$\frac{\text{Area}}{4\pi}\alpha - \frac{\text{Perimeter}}{2\pi}\sqrt{\alpha} \leqslant N(\alpha) \leqslant \frac{\text{Area}}{4\pi}\alpha$$

As $\alpha \to \infty$, the $\sqrt{\alpha}$ term becomes negligible compared to the α term:

$$\lim_{\alpha\to\infty} \frac{N(\alpha)}{\frac{\text{Area}}{4\pi}\alpha} = 1 \Rightarrow N(\alpha) \sim \frac{\text{Area}}{4\pi}\alpha$$

In this first part proof of Weyl's Law, we already to show two things: the **upper bound** $N(\alpha) \leqslant \frac{\text{Area}}{4\pi}\alpha$ is obtained by showing that all point squares fit inside the ellipse, while the **lower bound** $N(\alpha) \geqslant \frac{\text{Area}}{4\pi}\alpha - \frac{\text{Perimeter}}{2\pi}\sqrt{\alpha}$ is obtained by showing that the shifted ellipse is covered by the point squares. Finally, we combined them and got the complete formula $N(\alpha) \sim \frac{\text{Area}}{4\pi}\alpha$. □

Step 2: The finite union of rectangles

Let $R_1, R_2, \ldots, R_n$ be disjoint rectangular domains and define two related domains[5]:

$$\Omega = \bigcup_{m=1}^{n} R_m \quad \text{(disjoint union)}$$

$$\overline{\Omega} = \text{Int}\left(\bigcup_{m=1}^{n} \overline{R_m}\right) \quad \text{(connected interior of the union)}$$

We need to recall some significant theorem that are essential for this proof: Spectral Theory and Dirichlet monotonicity. Let's quickly review over those results:

Key Theorems for Step 2 from Richard S. Laugesen's notes[5]

1. **Spectral Theory on Finite Unions of Domains:** For a finite disjoint union of domains $\Omega = \bigcup_{m=1}^{n} R_m$, the spectrum of the Laplacian is the union of the spectra of each component. Moreover, eigenfunctions on each R_m can be extended by zero to the entire domain, and the eigenvalue counting functions are additive:

$$N_\Omega(\lambda) = \sum_{m=1}^{n} N_{R_m}(\lambda).$$

2. **Dirichlet Monotonicity (Theorem 11.2):** If $\Omega_1 \subset \Omega_2$, then the Dirichlet eigenvalues are monotone decreasing with respect to domain inclusion:

$$\lambda_j(\Omega_2) \leqslant \lambda_j(\Omega_1), \quad \forall j \geq 1.$$

3. **Lemma 2.2 (Inversion of Asymptotics):** Let $c > 0$ be fixed. Then

$$N(\alpha) \sim \frac{\alpha}{c} \quad \text{as } \alpha \to \infty \quad \Longleftrightarrow \quad \lambda_n \sim cn \quad \text{as } n \to \infty.$$

Combining these results, we can construct a sequence of rectangular domains that approximate Ω from the inside and outside. By Dirichlet monotonicity, their eigenvalues provide natural lower and upper bounds for the eigenvalues on Ω:

$$\underline{\lambda}_j \leqslant \lambda_j(\Omega) \leqslant \overline{\lambda}_j, \quad \forall j \geq 1.$$

Since the counting functions for rectangles admit an explicit Weyl asymptotic, the same leading term carries over to Ω by the inversion lemma. Hence, the rectangular exhaustion method yields the desired Weyl asymptotic formula.
From Step 1 (Weyl's law for a single rectangle, Dirichlet case), for each R_m we have:

$$N_{\text{Dir}}(\alpha; R_m) \sim \frac{|R_m|}{4\pi}\,\alpha \quad \text{as } \alpha \to \infty.$$

Applying the additivity property, we obtain:

$$N_{\mathrm{Dir}}(\alpha;\Omega) = \sum_{m=1}^{n} N_{\mathrm{Dir}}(\alpha;R_m) \sim \sum_{m=1}^{n} \frac{|R_m|}{4\pi}\,\alpha = \frac{|\Omega|}{4\pi}\,\alpha.$$

By Lemma 2.2 (Inversion of Asymptotics), inverting the relation between counting function and eigenvalues yields the Dirichlet eigenvalue asymptotic on Ω[5]:

$$\lambda_j(\Omega) \sim \frac{4\pi j}{|\Omega|} \qquad (j \to \infty).$$

Thus, Weyl's law (Dirichlet case) holds for any finite union of rectangular domains. This Dirichlet-only result provides the lower/upper bounds required when we later approximate an arbitrary domain by such unions (via Dirichlet monotonicity).

Step 3: Approximation of arbitrary domains

The proof for arbitrary domains follows the textbook in *Courant and Hilbert, §VI.4.4*. We partition the domain G using a square grid and analyze the contributions from interior squares and boundary regions separately[5] [9].

Divide the plane into squares of side a, decomposing G into:

- h interior squares $Q_1, Q_2, \ldots, Q_h$
- r boundary domains $E_1, E_2, \ldots, E_r$

The boundary domains are of two types (Figures 5 and 6 in the text)[9] :

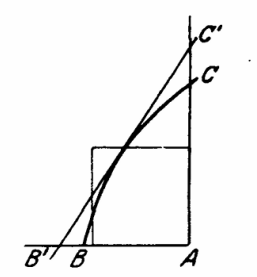


Figure 5

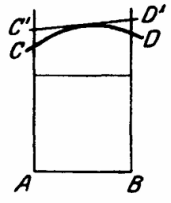


Figure 6

- Type 1: Bounded by two perpendicular line segments and a boundary curve segment
- Type 2: Bounded by one partition line segment, two perpendicular segments, and a boundary curve segment

For the boundary domains E_i, after transforming them to standard shapes, one obtains a similar estimate:

$$A_{E_i}(\lambda) < c_1 a^2 \lambda + c_2 a \sqrt{\lambda}.$$

Applying the comparison theorems in the text (analogous to Theorems 2, 4, and 5 in *Courant and Hilbert*), we have:

$$\sum_{i=1}^{h} A_i(\lambda) \leqslant A(\lambda) \leqslant \sum_{i=1}^{h} A_i(\lambda) + \sum_{i=1}^{r} A_{E_i}(\lambda),$$

where $A(\lambda)$ is the Dirichlet eigenvalue counting function for Ω.
Substituting the above estimates gives:

$$\text{Lower bound:} \quad A(\lambda) \geq \frac{ha^2}{4\pi}\lambda + \theta_1 c_1 ha\sqrt{\lambda},$$

$$\text{Upper bound:} \quad A(\lambda) \leq \frac{ha^2}{4\pi}\lambda + \theta_2 c_2 ha\sqrt{\lambda} + \theta_3 c_3 ra^2\lambda + \theta_4 c_4 ra\sqrt{\lambda}.$$

Observe that as $a \to 0$:

- $ha^2 \to |\Omega|$ (the area of the domain),
- $ra^2 \to 0$ (since $r < C/a$, so $ra^2 < Ca \to 0$),
- The $\sqrt{\lambda}$ terms are of lower order compared to λ terms.

Therefore, taking $\lambda \to \infty$, the leading terms dominate, and we obtain:

$$\lim_{\lambda\to\infty} \frac{4\pi A(\lambda)}{\lambda|\Omega|} = 1.$$

Hence, the two-dimensional Dirichlet Weyl's Law follows:

$$A(\lambda) \sim \frac{|\Omega|}{4\pi}\lambda,$$

which completes the proof for arbitrary bounded domains. The same idea extends to higher dimensions by partitioning into cubes instead of squares.

Chapter 5.2: LEAN 4 prototype for Weyl law by ZhiBin and Professor Pedro

With the mathematical proof of Weyl's Law now complete for the two-dimensional case, we turn to the practical challenge of formalizing this result in the Lean 4 theorem prover.

Professor Pedro and I are currently developing a formalization of Weil's law for Dirichlet eigenvalues in Lean 4, following the three-step proof, which is the same as the previous section 5.1. While we have built a framework, it will require the development of a significant amount of new math library before a full formalization can be achieved.

Our prototype framework come with the result for rectangular domains via explicit eigenvalue computations, then extends it to finite disjoint unions of rectangles using spectral additivity, and finally approximates arbitrary bounded domains via rigorous region approximations.

However, this project presents significant challenges due to shortcomings in the current Mathlib4 library. A lot of fundamental math theorems, such as spectral theory—including a rigorous treatment of Dirichlet eigenvalues, the space of eigenfunctions, and region approximation theory are not yet defined by other mathematicians. I hope that in the future, if more mathlib has been added on and smart math major colleagues could apply the framework and accomplish the goal! Thank you!

Project Page and (LEAN 4 prototype code)

Chapter 6: Appendix

Professor Pedro and I worked together on a Python code that explores Weyl's Law and the Laplacian operator. The code computes and easily to view the eigenvalues of the Laplacian and compares the numerical results with the Weyl's Law. Below are the details of the python code, and all user can also visit the two links at the end for the full code and further discussion.

Reader could also review the python code in the following webpage, it has a clear explain each code and details.
1) Python code 1
2) Python code 2

1 dimensional Weyl Law Python Code

```
# 1D Weyl law example
a = 1.0  # interval [0, a]

Lambda_vals = np.linspace(0, 100, 200)  # lambda from 0 to 100
N_vals = []  # store N(lambda)

for Lambda in Lambda_vals:
    count = 0
    n = 1
    while True:
        lambda_n = (n * np.pi / a) ** 2
        if lambda_n < Lambda:
            count += 1
            n += 1
        else:
            break
    N_vals.append(count)
```

2 dimensional Weyl Law Python Code

```
#2D Retangle weyl law
import numpy as np
import matplotlib.pyplot as plt

a = b = 1.0

Lambda_vals = np.linspace(0, 320, 50)
#Lambda_vals = np.linspace(1, 10000, 100)

N_vals = []

m_max = 50
n_max = 50

for Lambda in Lambda_vals:
    count = 0
    for m in range(1, m_max):
        for n in range(1, n_max):
            lambda_mn = np.pi**2 * ((m / a)**2 + (n / b)**2)
            if lambda_mn < Lambda:
                count += 1
    N_vals.append(count)

Weyl_line = (a * b / (4 * np.pi)) * Lambda_vals
```

2 dimensional polar coordinate Weyl Law Python Code

```
# 2D polar coordinate wey law
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
from scipy.special import jn_zeros

def count_disk_eigenvalues(lam, R=1):
    count = 0
    sqrt_lam = np.sqrt(lam)
    max_order = int(sqrt_lam * R) + 10

    for m in range(0, max_order):    # m = 0,1,2,...
        num_zeros = int(sqrt_lam * R / np.pi) + 20
        zeros = jn_zeros(m, num_zeros)

        for z in zeros:
            eigenvalue = (z / R)**2
            if eigenvalue <= lam:
                count += 2 if m > 0 else 1
            else:
                break
    return count

#lambda_vals_disk = np.linspace(1, 500, 50)
lambda_vals_disk = np.linspace(1, 10000, 50)

N_disk=[count_disk_eigenvalues(lam)
for lam in tqdm(lambda_vals_disk)]

asymptotic_disk = np.array(lambda_vals_disk) * (np.pi/(4*np.pi))
```

3 dimensional Weyl Law Python Code

```
#3D Cube weyl law

import numpy as np
import matplotlib.pyplot as plt

a = b = c = 1.0

#Lambda_vals = np.linspace(0, 500, 50)
Lambda_vals = np.linspace(1, 10000, 100)

N_vals = []

m_max = 100
n_max = 100
l_max = 100

for Lambda in Lambda_vals:
   count = 0
    for m in range(1, m_max):
      for n in range(1, n_max):
        for l in range(1, l_max):
        lambda_mnl = np.pi**2 * ((m/a)**2 + (n/b)**2 + (l/c)**2)
        if lambda_mnl < Lambda:
            count += 1
    N_vals.append(count)

Weyl_line = ((a * b * c)/ (6 * (np.pi)**2 )) * Lambda_vals**(1.5)
```

3 dimensional Cylinder and sphere

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import jn_zeros, spherical_jn
from scipy.optimize import brentq
from math import pi

def spherical_bessel_zeros(l, n_zeros, x_max=None):
    zeros = []
    x = max(1e-6, (l + 0.5) * 0.5)
    step = pi * 0.9
    if x_max is None:
        x_max = (l + n_zeros + 10) * pi
    f_prev = spherical_jn(l, x)
    while len(zeros) < n_zeros and x < x_max:
        x_next = x + step
        f_next = spherical_jn(l, x_next)
        if f_prev * f_next < 0:
            root = brentq(lambda t:spherical_jn(l, t),x, x_next)
            zeros.append(root)
            x = root + 1e-6
            f_prev = spherical_jn(l, x)
        else:
            x = x_next
            f_prev = f_next
    return np.array(zeros)

def cylinder_eigenvalues(R=1.0, H=1.0, lam_max=5000.0):
    eigs = []
    m_max = int(np.sqrt(lam_max) * R) + 15
    n_max = int(np.sqrt(lam_max) * H / pi) + 15
    for m in range(0, m_max + 1):
        jzeros = jn_zeros(m, n_max + 30)
        for j in jzeros:
            for n in range(1, n_max + 1):
                lam = (j / R)**2 + (n * pi / H)**2
                if lam <= lam_max:
                    mult = 1 if m == 0 else 2
                    eigs.extend([lam] * mult)
    return np.array(sorted(eigs))
```

```
def sphere_eigenvalues(R=1.0, lam_max=5000.0):
    eigs = []
    l_max = int(np.sqrt(lam_max) * R) + 20
    for l in range(0, l_max + 1):
        n_zeros = int(np.sqrt(lam_max) * R / pi) + 30
        zeros = spherical_bessel_zeros(l, n_zeros)
        for a_ln in zeros:
            lam = (a_ln / R)**2
            if lam <= lam_max:
                mult = 2 * l + 1
                eigs.extend([lam] * mult)
            else:
                break
    return np.array(sorted(eigs))

def weyl_3d(vol, lam_grid):
    return (vol / (6 * pi**2)) * lam_grid**1.5

R_cyl, H_cyl = 1.0, 2.0
R_sph = 1.0
lam_max = 5000.0
lam_grid = np.linspace(0, lam_max, 800)

cyl_vals = cylinder_eigenvalues(R=R_cyl, H=H_cyl,lam_max=lam_max)
sph_vals = sphere_eigenvalues(R=R_sph, lam_max=lam_max)

N_cyl = np.searchsorted(np.sort(cyl_vals),lam_grid, side='right')
N_sph = np.searchsorted(np.sort(sph_vals),lam_grid, side='right')

vol_cyl = pi * R_cyl**2 * H_cyl
vol_sph = 4 / 3 * pi * R_sph**3
W_cyl = weyl_3d(vol_cyl, lam_grid)
W_sph = weyl_3d(vol_sph, lam_grid)
```

References

[1] Arendt, W., Nittka, R., Peter, W., & Steiner, F. (2009). *Weyl's Law: Spectral Properties of the Laplacian in Mathematics and Physics*.

[2] Strauss, W. A. (2008). *Partial differential equations: an introduction*. Brown University.

[3] Evans, L. C. (2010). *Partial Differential Equations* (2nd ed.). UC Berkeley.

[4] Zelditch, S. (2017). Eigenfunctions of the Laplacian of Riemannian manifolds. Northwestern University.

[5] Laugesen, R. S. (2021). *Spectral Theory of Partial Differential Equations*. University of Illinois.

[6] Müller, W. (2007). Weyl's law in the theory of automorphic forms. Germany.

[7] Ivrii, V. (2017). 100 years of Weyl's law.

[8] Musser, S. (2016). Weyl's Law on Riemannian Manifolds.

[9] Courant, R., & Hilbert, D. (1953). *Methods of Mathematical Physics* (1st English ed., Vol. 1). Interscience Publishers, New York.