UNIVERSITÄT BONN

Exercise 1

Anand Karna: **50393435**
Orhan Ugur Aydin: **3209243**

What We'll Cover: The power and pitfalls of Linear Least Squares.

- Task 1.1: The Problem: Numerical Instability
- Task 1.2: The Limit: Fitting Non-Linear Functions
- Task 1.3: Application 1: Estimating Fractal Dimensions
- Task 1.4: Application 2: Fitting Exponential Models
- Task 1.5: Optimization on Barycentric Coordinate System

Task 1.1: Numerical Instability
Goal: Solve for a "tricky" matrix. $w = \operatorname{argmin} ||X^\top w - y||^2$

Methods:

1. **Naïve Algebra:** $w = (XX^\top)^{-1} X y$
2. **QR Decomposition:** $w = R_1^{-1} Q_1^\top y$
3. **numpy.linalg.lstsq**

Finding:

The "naïve" method involves inverting $XX^\top$. This matrix can be ill-conditioned or even singular, leading to massive errors. The lstsq and QR methods are numerically stable and find the correct solution $w = [1, 1]^\top$

Task 1.2: Fitting Non-Linear Boolean Functions Goal: Fit a model to a complex Boolean rule

Part 1: Simple Linear Model

- **Model:** $y \approx w_1x_1 + w_2x_2 + w_3x_3$
- **Result:** A **very bad fit**. A simple plane cannot capture the complex, XOR-like logic.

Part 2: Fourier Feature Map Model

- **Model:** $y \approx w_1 + w_2x_1 + ... + w_8x_1x_2x_3$
- **Result:** A **perfect fit!**

Key Takeaway: We need **feature engineering** to let linear models fit non-linear data.

Task 1.2: The Catch - Curse of Dimensionality

Problem:

Our simple 3-input problem ($n = 3$) required $2^3 = 8$ features for a perfect fit.

What if we have $n = 30$ inputs?

- We would need $2^{30} \approx 1,073,741,824$ features.

This exponential explosion of features is the **Curse of Dimensionality**.

Task 1.3: Application - Finding Fractal Dimension Goal

Method: Box Counting

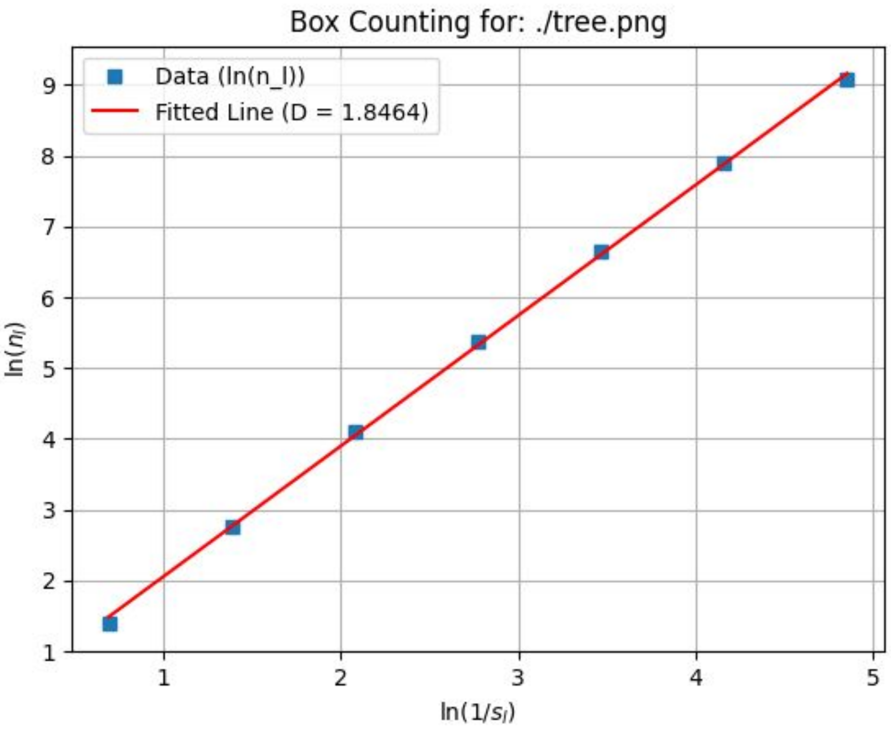
1. Cover the image with boxes of size **s**.
2. Count the number of boxes, **N(s)**, that touch the object.
3. Theory says: $N(s) \propto (1/s)^D$

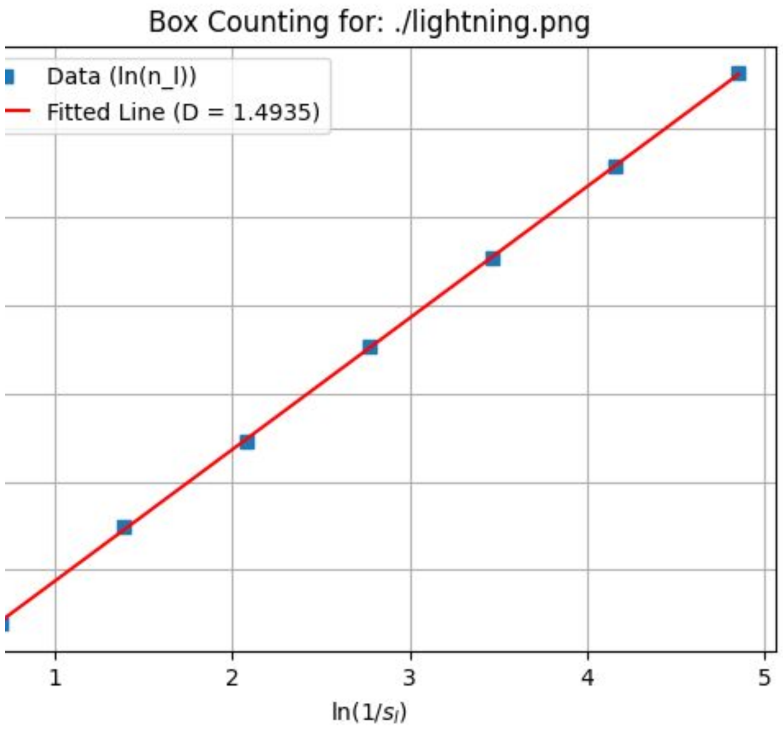
The Math Trick:

Take the logarithm of both sides: $\ln(N(s)) = D \cdot \ln(1/s) + c$

This is just the equation of a line: $y = mx + b$. We use least squares to find the slope m, which is our fractal dimension D .

Task 1.3 Results





Task 1.4: Fitting Exponential Data Goal: Fit the model $y = A \cdot e^{Bx}$

The Math Trick:Again, we take the logarithm: $\ln(y) = \ln(A) + Bx$

Let $y'=\ln(y)\$and\$a=\ln(A)$. This gives us a new linear model: $y' = a + Bx$

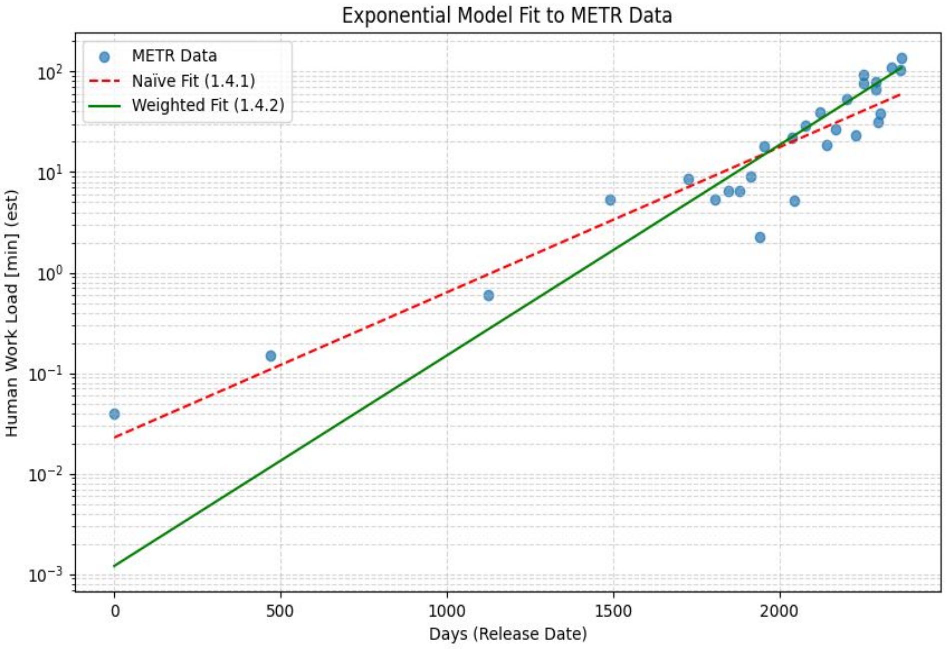
Task 1.4: Naïve vs. Weighted LSQ

Naïve LSQ (Task 1.4.1)

- **Minimizes:** $\sum(\ln(y_j) - y'_j)^2$ (Error in *log-space*)
- **Result: Poor fit.** This method cares too much about fitting the small **y** values and misses the explosive growth at the end.

Weighted LSQ (Task 1.4.2)

- **Minimizes:** $\sum y_j \cdot (\ln(y_j) - y'_j)^2$ (Gives *more weight* to large **y** values)
- **Result: Excellent fit!** This method correctly captures the exponential trend.



Task 1.5: Optimization on Barycentric System

- Representation of any point inside a triangle in planar space as a **convex combination** of its vertices, can be found in **barycentric coordinates,** by assigning weights to points in the space
- Solved for these coordinates by **minimizing** a **least squares optimization** while enforcing a constrains to those weights.
- For an **external point q**, the unconstrained solution yields negative weights, showing it lies **outside** the triangle.
- For a given point p, which lies outside of the triangle, its projection p’, inside the triangle, is the closest point to the point p.

Key Takeaways

Least Squares is a powerful, fundamental tool.

But... you must use **numerically stable** methods (`lstsq`, QR) and **regularization** (Ridge) to avoid instability. (Tasks 1.1, 1.5)

Linear models are limited. Use **feature engineering** to fit non-linear data, but beware the **Curse of Dimensionality**. (Task 1.2)

Many non-linear problems can be **linearized** using `log()` transforms. (Tasks 1.3, 1.4)

The *way* you define your "error" (e.g., naïve vs. weighted LSQ) can drastically change your model's fit. (Task 1.4)