Assignment 2 - Numerical Linear Algebra

Arthur Rabello Oliveira

08/05/2025

Abstract

We derive linear and polynomial regression in subsets of $\mathbb{R}$ and discuss the condition number of the associated matrices, numerical algorithms for the SVD and QR factorization are built and used on an efficiency analysis of the 3 methods to do linear or polynomial regression, stability of these algorirths is discussed.

Contents

1. Introduction

Given $D \subset \mathbb{R}^2$, a dataset, approximating this set through a *continuous* $f : \mathbb{R} \to \mathbb{R}$ is a very important problem in statistics, we will derive the 2 most important and most used methods to do this: linear and polynomial regression. Both are based on the least squares minimization problem. We will also discuss the conditioning number of the problems shown. A computational approach to regression is shown as well. We discuss how the condition number changes when the matrix is QR or SVD decomposed, and the algorithms for such decompositions are built.

2. Norms and Problems

A **norm** is a way to quantify *size* on a vector space. A norm is a class of functions that satisfy some properties, which we define below:

Definition 2.1: (Norm) Given E a vector space over a field $\mathbb{K}$, a **norm** is a function $\|\cdot\| : E \to \mathbb{R}$ that satisfies:

- $\|x\| > 0_E, \forall x \in E^*$, and $\|x\| = 0_E \Leftrightarrow x = 0_E$
- $\|x + y\| \leq \|x\| + \|y\|$
- $\|\varphi x\| = |\varphi| \|x\|, \forall \varphi \in \mathbb{K}$

Throughout this document we will use the most famous class of norms, the p-norms defined below:

Definition 2.2: (p-norm) Given $p \in \mathbb{R}$, the **p-norm** of $x \in \mathbb{C}^m$ is:

$$\|x\|_p = \left(\sum_{i=1}^{m} |x_i|^p \right)^{\frac{1}{p}} \tag{1}$$

Some famous cases are:

$$\begin{gathered} \|x\|_1 = \sum_{i=1}^{m} |x_i| \\ \|x\|_2 = \left(\sum_{i=1}^{m} |x_i|^2 \right)^{\frac{1}{2}} \\ \|x\|_\infty = \max_{1 \leq i \leq m} |x_i| \end{gathered} \tag{2}$$

Now we proceed with the main topic of this section, the condition number of a problem. It is useful to see a problem as a function $f : X \to Y$ from a *normed* vector space X of data and a space Y of solutions, f is not always a well-behaved continuous function, which is why we are interested in **well-conditioned** problems and not in **ill-conditioned** problems.

Definition 2.3: (Well-Conditioned Problem) A problem $f : X \to Y$ is *well-conditioned* at $x_0 \in X \Leftrightarrow \forall \varepsilon > 0, \exists \delta > 0 \mid \|x - x_0\| < \delta \Rightarrow f(x) - \|f(x_0)\| < \varepsilon$.

This means that small perturbations in x lead to small changes in $f(x)$, a problem is **ill-conditioned** if $f(x)$ can suffer huge changes with small changes in x.

We usually say f is well-conditioned if it is well-conditioned $\forall x \in X$, if there is at least one x_i in which the problem is ill-conditioned, then we can use that whole problem is ill-conditioned.

2.1. The Condition number of a problem

Condition numbers are a tool to quantify how well/ill conditioned a problem is:

Definition 2.1.1: (Absolute Conditioning Number) Let δx be a small pertubation of x, so $\delta f = f(x + \delta x) - f(x)$. The **absolute** conditioning number of f is:

$$\hat{\kappa} = \lim_{\delta \to 0} \sup_{\|\delta x\| \leq \delta} \frac{\|\delta f\|}{\|\delta x\|} \tag{3}$$

The limit of the supremum can be seen as the supremum of all *infinitesimal* perturbations, so this can be rewritten as:

$$\hat{\kappa} = \sup_{\delta x} \frac{\|\delta f\|}{\|\delta x\|} \tag{4}$$

If f is differentiable, we can evaluate the abs.conditioning number using its derivative, if J is the matrix whose $i \times j$ entry is the derivative $\frac{\partial f_i}{\partial x_j}$ (jacobian of f), then we know that $\delta f \approx J(x)\delta x$, with equality in the limit $\|\delta x\| \to 0$. So the absolute conditioning number of f becomes:

$$\hat{\kappa} = \|J(x)\| \tag{5}$$

2.2. The Relative Condition Number

When, instead of analyzing the whole set X of data, we are interested in *relative* changes, we use the **relative condition number:**

Definition 2.2.1: (Relative Condition Number) Given $f : X \to Y$ a problem, the *relative condition number* $\kappa(x)$ at $x \in X$ is:

$$\kappa(x) = \lim_{\delta \to 0} \sup_{\|\delta x\| \leq \delta} \left(\frac{\|\delta f\|}{\|f(x)\|} \right) \cdot \left(\frac{\|\delta x\|}{\|x\|} \right)^{-1} \tag{6}$$

Or, as we did in Definition 2.1.1, assuming that δf and δx are infinitesimal:

$$\kappa(x) = \sup_{\delta x} \left(\frac{\|\delta f\|}{\|f(x)\|} \right) \cdot \left(\frac{\|\delta x\|}{\|x\|} \right)^{-1} \tag{7}$$

If f is differentiable:

$$\kappa(x) = (\|J(x)\|) \cdot \left(\frac{\|f(x)\|}{\|x\|} \right)^{-1} \tag{8}$$

Relative condition numbers are more useful than absolute conditioning numbers because the **floating point arithmetic** used in many computers produces *relative* errors, the latter is not a highlight of this discussion.

Here are some examples of the definitions above:

Example 2.2.1.: Consider the problem of obtaining the scalar $\frac{x}{2}$ from $x \in \mathbb{R}$. The function $f(x) = \frac{x}{2}$ is differentiablle, so by eq. (8):

$$\kappa(x) = (\|J\|) \cdot \left(\frac{\|f(x)\|}{\|x\|} \right)^{-1} = \left(\frac{1}{2} \right) \cdot \left(\frac{\frac{x}{2}}{x} \right)^{-1} = 1. \tag{9}$$

This problem is well-conditioned (κ is small).

Example 2.2.2.: Consider the problem of computing the scalar $x_1 - x_2$ from $(x_1, x_2) \in \mathbb{R}^2$ (Use the ∞-norm in $\mathbb{R}^2$ for simplicity). The function associated is differentiable and the jacobian is:

$$J = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 1 & -1 \end{bmatrix} \tag{10}$$

With $\|J\|_\infty = 2$, so the condition number is:

$$\kappa = (\|J\|_\infty) \cdot \left(\frac{\|f(x)\|}{\|x\|} \right)^{-1} = \frac{2}{|x_1 - x_2| \cdot \max\{|x_1|, |x_2|\}} \tag{11}$$

This problem can be ill-conditioned if $|x_1 - x_2| \approx 0$ (κ gets huge), and well-conditioned otherwise

.

2.3. Condition Number of Matrices

We will deduce the conditioning number of a matrix from the conditioning number of *matrix-vector* multiplication:

Consider the problem of obtaining Ax given $A \in \mathbb{C}^{m \times n}$. We will calculate the relative condition number with respect to perturbations on x. Directly from Definition 2.2.1, we have:

$$\kappa = \sup_{\delta x} \frac{\|A(x + \delta x) - Ax\|}{\|Ax\|} \cdot \left(\frac{\|\delta x\|}{\|x\|} \right)^{-1} = \sup_{\delta x} \frac{\|A\delta x\|}{\|\delta x\|} \cdot \left(\frac{\|Ax\|}{\|x\|} \right)^{-1} \tag{12}$$

Since $\sup_{\forall x} \frac{\|A\delta x\|}{\|\delta x\|} = \|A\|$, we have:

$$\kappa = \|A\| \cdot \frac{\|x\|}{\|Ax\|} \tag{13}$$

This is a precise formula as a function of (A, x).

The following theorem will be useful in a near future:

Theorem 2.3.1: $\forall x \in \mathbb{C}^n, A \in \mathbb{C}^{n \times n}, \det(A) \neq 0$, the following holds:

$$\frac{\|x\|}{\|Ax\|} \leq \|A^{-1}\| \tag{14}$$

Proof: Since $\|AB\| \leq \|A\| \|B\|$, we have:

$$\|AA^{-1}x\| \leq \|Ax\| \|A^{-1}\| \Leftrightarrow \frac{\|x\|}{\|Ax\|} \leq \|A^{-1}\| \tag{15}$$

□

So using this in eq. (13), we can write:

$$\kappa \leq \|A\| \cdot \|A^{-1}\| \tag{16}$$

Or:

$$\kappa = \alpha \|A\| \cdot \left\|A^{-1}\right\| \tag{17}$$

With

$$\alpha = \frac{\|x\|}{\|Ax\|} \cdot \left(\left\|A^{-1}\right\|\right)^{-1} \tag{18}$$

From Theorem 2.3.1, we can choose x to make $\alpha = 1$, and therefore $\kappa = \|A\| \cdot \left\|A^{-1}\right\|$.

Consider now the problem of calculating $A^{-1}b$ given $A \in \mathbb{C}^{n \times n}$. This is mathematically identical to the problem we just analyzed, so the following theorem has already been proven:

Theorem 2.3.2: Let $A \in \mathbb{C}^{n \times n}$, $\det(A) \neq 0$, and consider the problem of computing b, from $Ax = b$, by perturbating x. Then the following holds:

$$\kappa = \|A\| \frac{\|x\|}{\|b\|} \leq \|A\| \cdot \left\|A^{-1}\right\| \tag{19}$$

Where κ is the condition number of the problem.

Proof: Read from eq. (12) to eq. (18). □

Finally, $\|A\| \cdot \left\|A^{-1}\right\|$ is so useful it has a name: **the condition number of A** (relative to the norm $\|\cdot\|$)

If A is singular, $\kappa(A) = \infty$. Notice that if $\|\cdot\| = \|\cdot\|_2$, then $\|A\| = \sigma_1$ and $\left\|A^{-1}\right\| = \frac{1}{\sigma_m}$, so:

$$\kappa(A) = \frac{\sigma_1}{\sigma_m} \tag{20}$$

This is the condition number of A with respect to the 2-norm, which is the most used norm in practice. The condition number of a matrix is a measure of how sensitive the solution of a system of equations is to perturbations in the data. A large condition number indicates that the matrix is ill-conditioned, meaning that small changes in the input can lead to large changes in the output.

3. Linear Regression (1a)

Given the dataset:

$$D := \left\{ t_i = \frac{i}{m} \right\}, i = 0, 1, ..., m \in \mathbb{R} \tag{21}$$

of equally spaced points , linear regression consists of finding the best *line* $f(t) = \alpha + \beta t$ that approximates the points $(t_i, b_i) \in \mathbb{R}^2$, where b_i are arbitrary.

Approximating 2 points in $\mathbb{R}^2$ by a line is trivial, now approximating more points is a task that requires linear algebra. To see this, we will analyze the following example to build intuition for the general case:

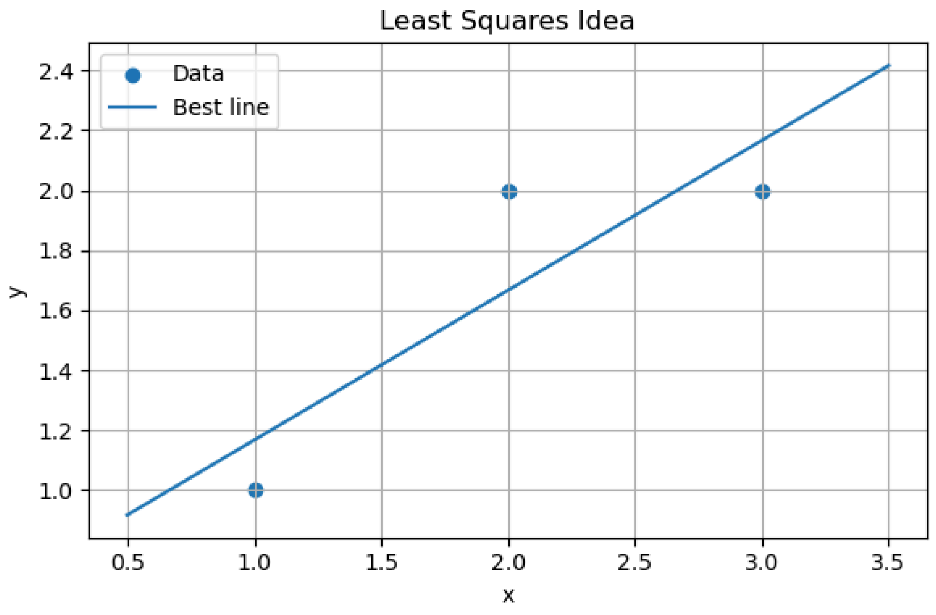

Figure 1: A good approximation for the 3 points shown

Figure 1 is a glimpse onto what we are about to produce.

Given the points $(1,1),(2,2),(3,2) \in \mathbb{R}^2$, we have $(t_1, b_1) = (1,1), (t_2, b_2) = (2,2), (t_3, b_3) = (3,2)$ we would like a *line* $f(t) = y(t) = \alpha + \beta t$ that best approximates (t_i, b_i). In other words, since we know that the line does not pass through all 3 points, we would like to find the *closest* line to **each point** of the dataset D. So the system:

$$\begin{aligned} f(1) &= \alpha + \beta = 1 \\ f(2) &= \alpha + 2\beta = 2 \\ f(3) &= \alpha + 3\beta = 2 \end{aligned} \quad (22)$$

Which is:

$$\underbrace{\begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix}}_{A} \cdot \underbrace{\begin{bmatrix} \alpha \\ \beta \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}}_{b} \quad (23)$$

Clearly has no solution. But it has a *closest solution* which we can find through **minimizing** the errors produced by this approximation.

Let $x^* \neq x$ be a solution to the system. And let the error produced by approximating the points through a line be $e = Ax - b$. Minimizing the error requires a *norm*, which is defined

$$e_1^2 + e_2^2 + e_3^2 \quad (24)$$

Is what we want to minimize, where e_i is the error (distance) from the ith point to the line:

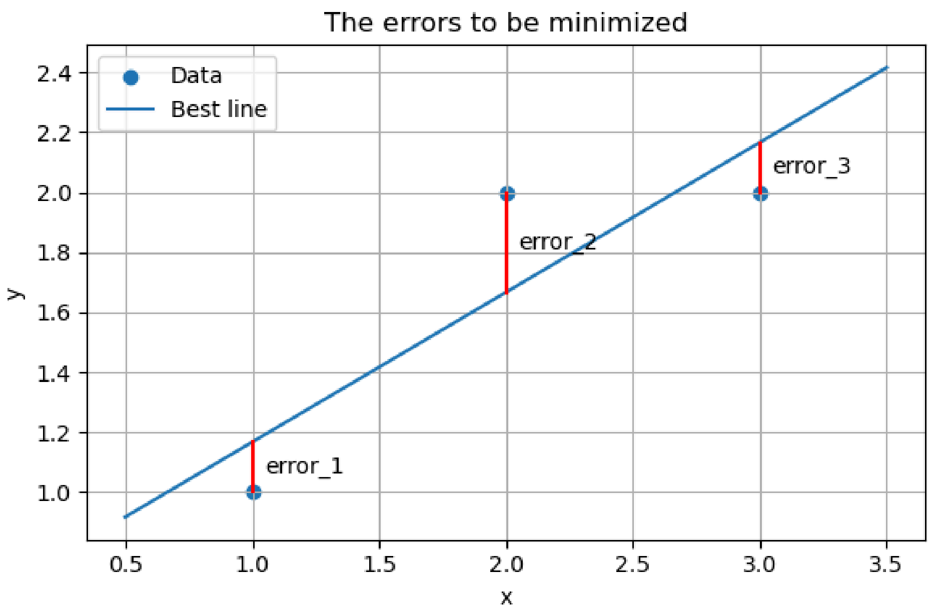

Figure 2: The errors (distances)

So we will project b into $C(A)$, giving us the closest solution, and the least squares solutions is when $\hat{x}$ minimizes $\|Ax-b\|^2$, this occurs when the residual $e = Ax - b$ is orthogonal to $C(A)$. Since $N(A^*) \perp C(A)$ and the dimensions sum up the left dimension of the matrix. so by the well-known projection formula, we have:

$$A^*A\hat{x} = A^*b$$
$$= \begin{bmatrix}1&1&1\\1&2&3\end{bmatrix}\cdot\begin{bmatrix}1&1\\1&2\\1&3\end{bmatrix}\cdot\begin{bmatrix}\alpha\\\beta\end{bmatrix} = \begin{bmatrix}3&6\\6&14\end{bmatrix}\cdot\begin{bmatrix}\alpha\\\beta\end{bmatrix} \tag{25}$$
$$= \begin{bmatrix}3&6\\6&14\end{bmatrix}\cdot\begin{bmatrix}1\\2\\3\end{bmatrix} = \begin{bmatrix}5\\11\end{bmatrix}$$

So the system to find $\hat{x} = \left[\hat{\alpha}, \hat{\beta}\right]$ becomes:

$$\begin{aligned}3\alpha + 5\beta &= 5\\ 6\alpha + 14\beta &= 11\end{aligned} \tag{26}$$

Notice that with the *errors* e_i^2 as:

$$\begin{aligned}e_1^2 &= (f(t_1) - b_1)^2 = (f(1)-1)^2 = (\alpha+\beta-1)^2\\ e_2^2 &= (f(t_2) - b_2)^2 = (f(2)-2)^2 = (\alpha+2\beta-2)^2\\ e_3^2 &= (f(t_3) - b_2)^2 = (f(3)-2)^2 = (\alpha+3\beta-2)^2\end{aligned} \tag{27}$$

This is consistent with what we see in Figure 2. Notice that eq. (26) is *precisely* what is obtained after using partial derivatives to minimize the erros sum as a function of (α, β):

$$\begin{aligned}f(\alpha,\beta) &= (\alpha+\beta-1)^2 + (\alpha+2\beta-2)^2 + (\alpha+3\beta-2)^2\\ &= 3\alpha^2 + 14\beta^2 + 12\alpha\beta - 10\alpha - 22\beta + 9,\\ \frac{\partial f}{\partial \alpha} = \frac{\partial f}{\partial \beta} = 0 &\Leftrightarrow 6\alpha + 12d - 10 = 28\beta + 12\alpha - 22 = 0 \Leftrightarrow \begin{cases}3c+6d = 11\\ 6c + 14d = 11\end{cases}\end{aligned} \tag{28}$$

This new system has a solution in $\hat{\alpha} = \frac{2}{3}, \hat{\beta} = \frac{1}{2}$, so the equation of the optimal line, obtained through *linear regression* (or least squares) is:

$$y(t) = \frac{2}{3} + \frac{1}{2}t. \tag{29}$$

If we have $n > 3$ points to approximate with linear regression, the reasoning is analogous:

Going back to D in eq. (21) , we want to find the extended system as we did in eq. (26), so let:

$$f(t) = \alpha + \beta t \tag{30}$$

Be the linear regression line, for $D = \left\{(0, b_0), \left(\frac{1}{m}, b_1\right), ..., (1, b_m)\right\}$. The system is:

$$\begin{aligned} f\left(\frac{0}{m} = 0\right) &= b_0 = \alpha, \\ f\left(\frac{1}{m}\right) &= b_1 = \alpha + \frac{\beta}{m}, \\ f\left(\frac{2}{m}\right) &= b_2 = \alpha + \frac{2}{m}\beta \\ &... \\ f\left(\frac{m}{m} = 1\right) &= b_m = \alpha + \beta \end{aligned} \tag{31}$$

Or:

$$\underbrace{\begin{bmatrix} 1 & 0 \\ 1 & \frac{1}{m} \\ \vdots & \vdots \\ 1 & 1 \end{bmatrix}}_{A} \cdot \underbrace{\begin{bmatrix} \alpha \\ \beta \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} b_0 \\ \vdots \\ b_m \end{bmatrix}}_{b} \tag{32}$$

Projecting into $C(A)$, we have:

$$\begin{aligned} A^*Ax &= A^*b \\ = \begin{bmatrix} 1 & 1 & ... & 1 \\ 0 & \frac{1}{m} & ... & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 1 & \frac{1}{m} \\ \vdots & \vdots \\ 1 & 1 \end{bmatrix} &= \begin{bmatrix} m+1 & \frac{m+1}{2} \\ \frac{m+1}{2} & \frac{(m+1)(2m+2)}{6m} \end{bmatrix} \cdot \begin{bmatrix} \hat{\alpha} \\ \hat{\beta} \end{bmatrix} \\ = \begin{bmatrix} 1 & 1 & ... & 1 \\ 0 & \frac{1}{m} & ... & 1 \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ \vdots \\ b_m \end{bmatrix} &= \begin{bmatrix} b_0 + b_2 + ... + b_m \\ \frac{1}{m}[b_1 + 2b_2 + ... + (m-1)b_{m-1} + b_m] \end{bmatrix} \end{aligned} \tag{33}$$

So the system to find the optimal vector $\begin{bmatrix} \hat{\alpha} \\ \hat{\beta} \end{bmatrix}$ is:

$$\begin{bmatrix} m+1 & \frac{m+1}{2} \\ \frac{m+1}{2} & \frac{(m+1)(2m+2)}{6m} \end{bmatrix} \cdot \begin{bmatrix} \hat{\alpha} \\ \hat{\beta} \end{bmatrix} = \begin{bmatrix} b_0 + b_1 + ... + b_m \\ \frac{1}{m}[b_1 + 2b_2 + ... + (m-1)b_{m-1} + b_m] \end{bmatrix} \tag{34}$$

Or, as a function of t_i, b_i and m:

$$\underbrace{\begin{bmatrix} m+1 & \sum_{i=1}^m t_i \\ \sum_{i=1}^m t_i & \sum_{i=1}^m t_i^2 \end{bmatrix}}_{\hat{A}} \cdot \begin{bmatrix} \hat{\alpha} \\ \hat{\beta} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^m b_i \\ \sum_{i=1}^m \frac{i}{m} \cdot b_i \end{bmatrix} \tag{35}$$

This provides the optimal vector $\hat{x}$ that minimizes the least squares error, which is the solution to the linear regression problem.

4. How the condition number of A changes (1b)

We are interested in the condition number of $\hat{A} = A^* A$ shown in eq. (34). We will analyze how the condition number of $\hat{A}$ changes with respect to perturbations on m, the number of points in the dataset. A computational approach is appropriate.

Here is a python code that numerically calculates many values of $\kappa\left(\hat{A}_m\right)$ as a function of m:

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

def cond_number(m):

    """
    This function computes the condition number of the matrix A(m) in the 2-norm. The matrix A is defined.

    Args:
        m (float): parameter for the matrix A(m)
    Returns:
        float: condition number of A(m)
    Raises:
        ZeroDivisionError: if m = 0
        np.linalg.LinAlgError: if A(m) is not invertible
    """

    A = np.array([
        [m + 1,          (m + 1) / 2],
        [(m + 1) / 2,   (m + 1)**2 / (3 * m)]
    ])
    A_inv = np.linalg.inv(A)
    return np.linalg.norm(A, 2) * np.linalg.norm(A_inv, 2)

M = float(input("Enter maximum m (M > 0): "))
N = int(input("Enter number of sample points: ")) #however the user wants to plot

m_vals = np.linspace(0, M, N)
conds  = []

for m in m_vals:
    try:
        conds.append(cond_number(m))
    except (ZeroDivisionError, np.linalg.LinAlgError):
        conds.append(np.inf) #if it is not invertible
```

```python

plt.figure()
plt.plot(m_vals, conds)
plt.xlabel('m')
plt.ylabel('Condition number $κ(A^* A)$')
plt.title('Condition number of $A^* A(m)$')
plt.grid(True)
plt.tight_layout()
plt.show()
```

Good visualizations of this are:

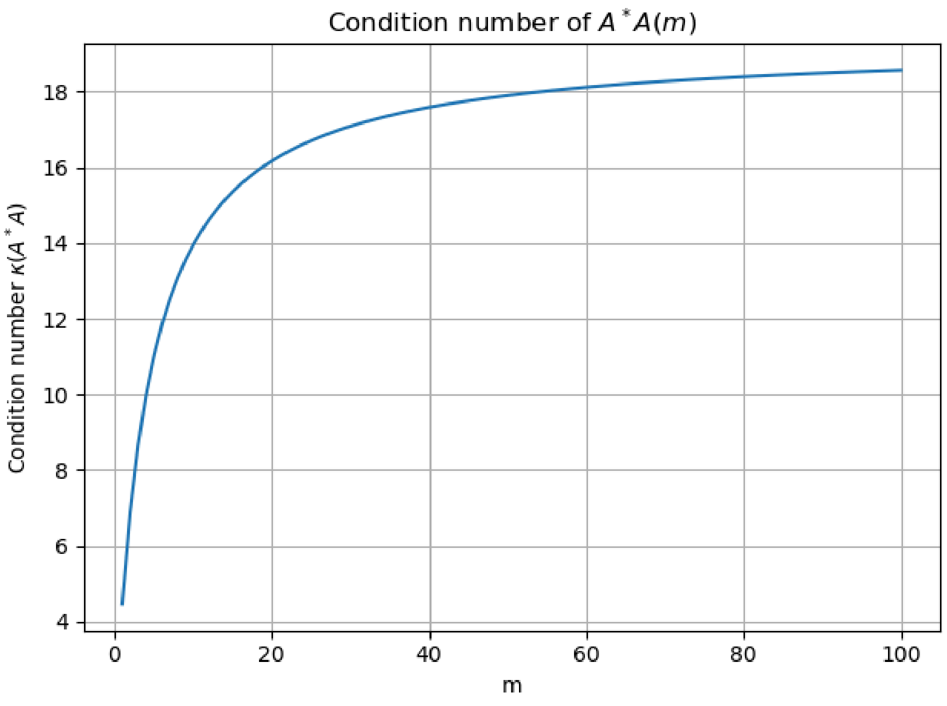

Figure 3: Condition number of $\hat{A}$ over [0, 100]

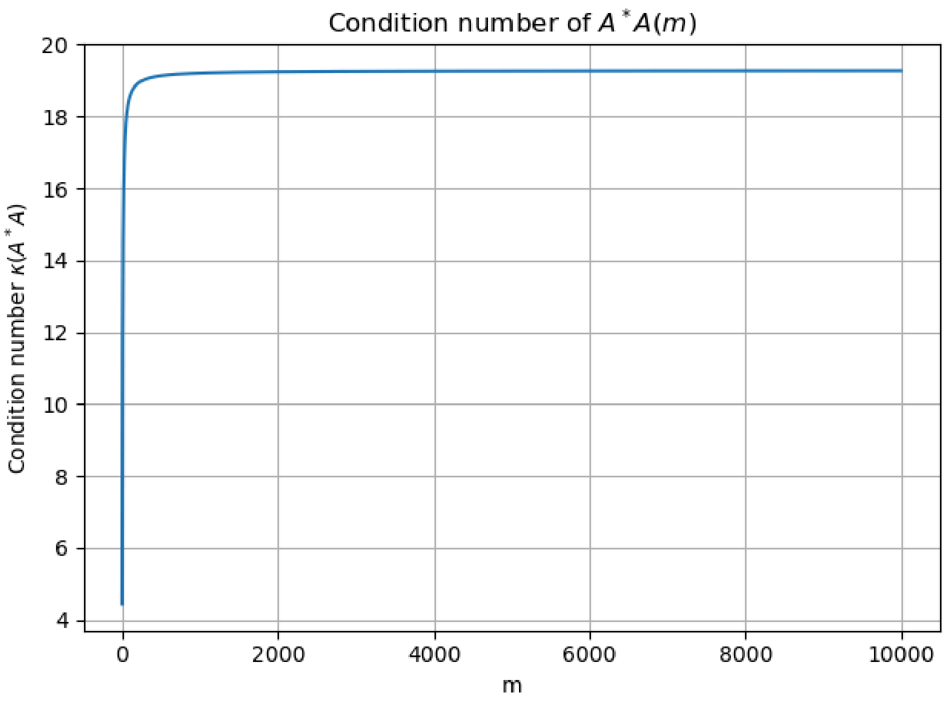

Figure 4: Condition number of $\hat{A}$ over [0, 10000]

Figure 3 and Figure 4 show us that apparently $\kappa\left(\hat{A}_m\right)$ converges to a real number. We will evaluate this hypothesis below:

Using $\|\cdot\|_2$, the condition number of $\hat{A}$ is:

$$\kappa\left(\hat{A}\right) = \left\|\hat{A}\right\|_2 \cdot \left\|\hat{A}^{-1}\right\|_2 = \frac{\sigma_1}{\sigma_m} \qquad 36$$

Singular Values are better explored in Section 10.2. Now we calculate the singular values of $\hat{A}$, which are the square roots of the eigenvalues of $\hat{A}$ (see Theorem 10.2.2). So we have:

$$\begin{aligned}
\det\left(\hat{A} - \lambda I\right) = 0 \Leftrightarrow \det\left(\begin{bmatrix} m+1-\lambda & \frac{m+1}{2} \\ \frac{m+1}{2} & \frac{(m+1)(2m+1)}{6m} - \lambda \end{bmatrix}\right) = 0 \\
\Leftrightarrow (m+1-\lambda)\left[\frac{(m+1)(2m+1)}{6m} - \lambda\right] - \left(\frac{m+1}{2}\right)^2 = 0 \\
\Leftrightarrow \lambda^2 - \frac{(m+1)(8m+1)}{6m}\lambda + \frac{(m+1)^2(m+2)}{12m} = 0 \\
\Leftrightarrow \lambda = \frac{m+1}{12m}\left[(8m+1) \pm \sqrt{52m^2 - 8m + 1}\right]
\end{aligned} \qquad 37$$

And the singular values are:

$$\begin{aligned}
\sigma_1 = \sqrt{\lambda_1} = \sqrt{\frac{m+1}{12m}\left[(8m+1) + \sqrt{52m^2 - 8m + 1}\right]}, \\
\sigma_2 = \sqrt{\lambda_2} = \sqrt{\frac{m+1}{12m}\left[(8m+1) - \sqrt{52m^2 - 8m + 1}\right]}
\end{aligned} \qquad 38$$

This gives:

$$\kappa\left(\hat{A}\right) = \frac{\sigma_1}{\sigma_m} = \frac{\sqrt{\frac{m+1}{12m}\left[(8m+1) + \sqrt{52m^2 - 8m + 1}\right]}}{\sqrt{\frac{m+1}{12m}\left[(8m+1) - \sqrt{52m^2 - 8m + 1}\right]}}$$
$$= \sqrt{\frac{(8m+1) + \sqrt{52m^2 - 8m + 1}}{(8m+1) - \sqrt{52m^2 - 8m + 1}}} \qquad 39$$

And the limit as $m \to \infty$:

$$\lim_{m\to\infty} \kappa\left(\hat{A}\right) = \lim_{m\to\infty} \sqrt{\frac{(8m+1) + \sqrt{52m^2 - 8m + 1}}{(8m+1) - \sqrt{52m^2 - 8m + 1}}} \qquad 40$$

Multiplying by the conjugate of the denominator and ignoring the square root (it is irelevant for the limit):

$$= \lim_{m\to\infty} \left[\frac{(8m+1) + \sqrt{52m^2 - 8m + 1}}{(8m+1) - \sqrt{52m^2 - 8m + 1}} \cdot \frac{(8m+1) + \sqrt{52m^2 - 8m + 1}}{(8m+1) + \sqrt{52m^2 - 8m + 1}}\right]$$
$$= \lim_{m\to\infty} \frac{\left((8m+1) + \sqrt{52m^2 - 8m + 1}\right)^2}{(8m+1)^2 - 52m^2 - 8m + 1} \qquad 41$$
$$= \lim_{m\to\infty} \frac{(8m+1)^2 + 2(8m+1)\sqrt{52m^2 - 8m + 1} + \left(52m^2 - 8m + 1\right)}{(8m+1)^2 - \left(52m^2 - 8m + 1\right)}$$
$$= \lim_{m\to\infty} \frac{64m^2 + 16m + 1 + (16m+1)\sqrt{52m^2 - 8m + 1} + 52m^2 - 8m + 1}{64m^2 + 16m + 1 - 52m^2 + 8m - 1}$$

Regretting having ignored the square root, and putting it back, we have:

$$= \lim_{m\to\infty} \sqrt{\frac{\left((8m+1) + \sqrt{52m^2 - 8m + 1}\right)^2}{12m^2 + 24m}}$$
$$= \lim_{m\to\infty} \frac{(8m+1) + \sqrt{52m^2 - 8m + 1}}{\sqrt{12m^2 + 24m}}$$
$$= \lim_{m\to\infty} \frac{8m + 1 + m\sqrt{52 - \frac{8}{m} + \frac{1}{m^2}}}{m\sqrt{12 + \frac{24}{m}}} \qquad 42$$
$$= \lim_{m\to\infty} \frac{m\left(8 + \frac{1}{m} + \sqrt{52 - \frac{8}{m} + \frac{1}{m^2}}\right)}{m\sqrt{12 + \frac{24}{m}}}$$
$$= \lim_{m\to\infty} \frac{8 + \frac{1}{m} + \sqrt{52 - \frac{8}{m} + \frac{1}{m^2}}}{\sqrt{12 + \frac{24}{m}}}$$

And finally:

$$\lim_{m\to\infty} \kappa\left(\hat{A}_m\right) = \frac{8 + \sqrt{52}}{\sqrt{12}} = \frac{4 + \sqrt{13}}{\sqrt{3}} \qquad 43$$

A very good visualization of this is:

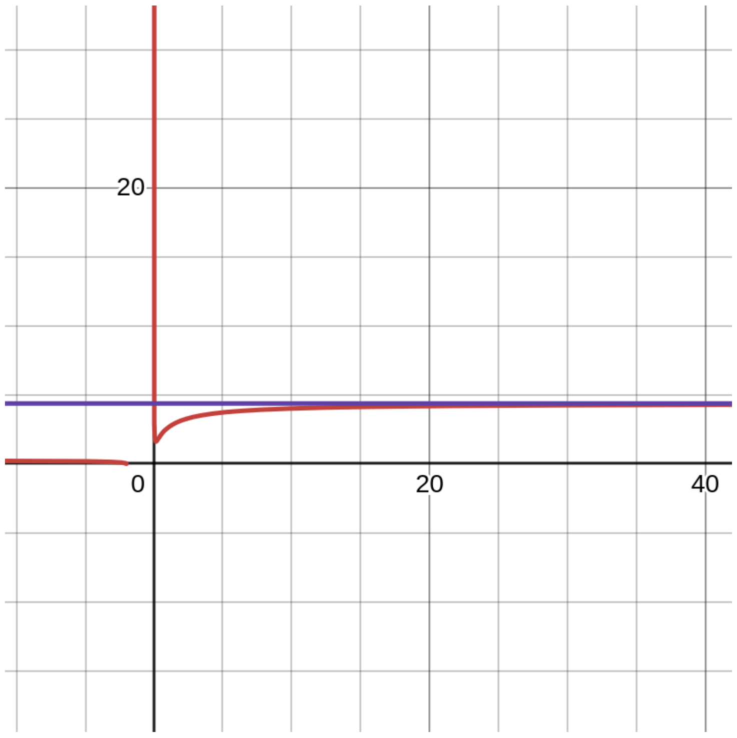

Figure 5: The purple line is the limit and the red is the function eq. (39)

Figure 5 shows the function approaching the limit. One could say that this problem is well conditioned, for $\kappa\left(\hat{A}\right)_m < \frac{4+\sqrt{13}}{\sqrt{3}}, \forall m > 0$, and $\frac{4+\sqrt{13}}{\sqrt{3}}$ is not a very big number. We will not go deep into the discussion of how well-condition this problem is, but we can say that the condition number of A is not a problem for the linear regression algorithm.

5. Polynomial Regression (1c)

In this section we will discuss what changes when we decide to use **polynomials** instead of **lines** to approximate our dataset:

$$f(t) = \alpha + \beta t \rightarrow p(t) = \varphi_0 + \varphi_1 t + \ldots + \varphi_n t^n \tag{44}$$

From a first perspective, it seems way more efficient to describe a dataset with many variables then to do so with a simple line $\alpha + \beta t$, so let's use the same dataset $S := \left\{(t_i, b_i), t_i = \frac{i}{m}\right\}, i = 0, 1, \ldots, m$. Where b_i is arbitrary. As we did in Section 3, finding the new system to be solved gives us:

$$\begin{gathered} p(t_0 = 0) = b_0 = \varphi_0, \\ p\left(t_1 = \frac{1}{m}\right) = b_1 = \varphi_0 + \varphi_1 \frac{1}{m} + \ldots + \varphi_n \left(\frac{1}{m}\right)^n \\ p\left(t_2 = \frac{2}{m}\right) = b_2 = \varphi_0 + \varphi_1 \frac{2}{m} + \varphi_2 \left(\frac{2}{m}\right)^2 + \ldots + \varphi_n \left(\frac{2}{m}\right)^n \\ \vdots \\ p(t_m = 1) = b_m = \varphi_0 + \ldots + \varphi_n \end{gathered} \tag{45}$$

Or:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & \frac{1}{m} & \left(\frac{1}{m}\right)^2 & \dots & \left(\frac{1}{m}\right)^n \\ 1 & \frac{2}{m} & \left(\frac{2}{m}\right)^2 & \dots & \left(\frac{2}{m}\right)^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}}_{A_{m+1\times n+1}} \cdot \underbrace{\begin{bmatrix} \varphi_0 \\ \varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_n \end{bmatrix}}_{\Phi_{n+1\times 1}} = \underbrace{\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}}_{b_{m+1\times 1}} \tag{46}$$

Projecting into $C(A)$:

$$\begin{aligned} A^* A \hat{\Phi} &= \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 0 & \frac{1}{m} & \frac{2}{m} & \dots & 1 \\ 0 & \left(\frac{1}{m}\right)^2 & \left(\frac{2}{m}\right)^2 & \dots & 1 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & \left(\frac{1}{m}\right)^n & \left(\frac{2}{m}\right)^n & \dots & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & \frac{1}{m} & \left(\frac{1}{m}\right)^2 & \dots & \left(\frac{1}{m}\right)^n \\ 1 & \frac{2}{m} & \left(\frac{2}{m}\right)^2 & \dots & \left(\frac{2}{m}\right)^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix} \cdot \begin{bmatrix} \widehat{\varphi_0} \\ \widehat{\varphi_1} \\ \widehat{\varphi_2} \\ \vdots \\ \widehat{\varphi_n} \end{bmatrix} \\ &= \begin{bmatrix} m+1 & \sum_{i=1}^m \frac{i}{m} & \sum_{i=1}^m \left(\frac{i}{m}\right)^2 & \dots & \sum_{i=1}^m \left(\frac{i}{m}\right)^n \\ \sum_{i=1}^m \frac{i}{m} & \sum_{i=1}^m \left(\frac{i}{m}\right)^2 & \sum_{i=1}^m \left(\frac{i}{m}\right)^3 & \dots & \sum_{i=1}^m \left(\frac{i}{m}\right)^{n+1} \\ \sum_{i=1}^m \left(\frac{i}{m}\right)^2 & \sum_{i=1}^m \left(\frac{i}{m}\right)^3 & \sum_{i=1}^m \left(\frac{i}{m}\right)^4 & \dots & \sum_{i=1}^m \left(\frac{i}{m}\right)^{n+2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \sum_{i=1}^m \left(\frac{i}{m}\right)^n & \sum_{i=1}^m \left(\frac{i}{m}\right)^{n+1} & \sum_{i=1}^m \left(\frac{i}{m}\right)^{n+2} & \dots & \sum_{i=1}^m \left(\frac{i}{m}\right)^{2n} \end{bmatrix} \cdot \begin{bmatrix} \widehat{\varphi_0} \\ \widehat{\varphi_1} \\ \widehat{\varphi_2} \\ \vdots \\ \widehat{\varphi_n} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 0 & \frac{1}{m} & \frac{2}{m} & \dots & 1 \\ 0 & \left(\frac{1}{m}\right)^2 & \left(\frac{2}{m}\right)^2 & \dots & 1 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & \left(\frac{1}{m}\right)^n & \left(\frac{2}{m}\right)^n & \dots & 1 \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^m b_i \\ \sum_{i=0}^m \frac{i b_i}{m} \\ \sum_{i=0}^m \left(\frac{i}{m}\right)^2 m \\ \vdots \\ \sum_{i=0}^m \left(\frac{i}{m}\right)^n b_i \end{bmatrix} \end{aligned} \tag{47}$$

So the system to be solved is:

$$\underbrace{\begin{bmatrix} m+1 & \sum_{i=1}^m \frac{i}{m} & \dots & \sum_{i=1}^m \left(\frac{i}{m}\right)^n \\ \sum_{i=1}^m \frac{i}{m} & \sum_{i=1}^m \left(\frac{i}{m}\right)^2 & \dots & \sum_{i=1}^m \left(\frac{i}{m}\right)^{n+1} \\ \sum_{i=1}^m \left(\frac{i}{m}\right)^2 & \sum_{i=1}^m \left(\frac{i}{m}\right)^3 & \dots & \sum_{i=1}^m \left(\frac{i}{m}\right)^{n+2} \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{i=1}^m \left(\frac{i}{m}\right)^n & \sum_{i=1}^m \left(\frac{i}{m}\right)^{n+1} & \dots & \sum_{i=1}^m \left(\frac{i}{m}\right)^{2n} \end{bmatrix}}_{\hat{A}} \cdot \begin{bmatrix} \widehat{\varphi_0} \\ \widehat{\varphi_1} \\ \widehat{\varphi_2} \\ \vdots \\ \widehat{\varphi_n} \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^m b_i \\ \sum_{i=0}^m \frac{i b_i}{m} \\ \sum_{i=0}^m \left(\frac{i}{m}\right)^2 m \\ \vdots \\ \sum_{i=0}^m \left(\frac{i}{m}\right)^n b_i \end{bmatrix} \tag{48}$$

This gives the optimal vector $\hat{\Phi}$ that solves the least squares problem. We will use computational methods to analyze this system in some of the next sections.

6. Computing the polynomial regression matrix, given (m,n) (1d)

Here is a python function that calculates the polynomial regression matrix $\hat{A} = A^* A$ from eq. (48), given the dimensions (m, n):

```python
import numpy as np

```

```python
def poly_ls(m, n):

    """
    Builds the (n+1) x (n+1) matrix A^T A for least-squares polynomial fitting.

    Args:
        m (int): number of subintervals (m >= 0)
        n (int): polynomial degree (n >= 0)
    Returns:
        np.ndarray: shape (n+1, n+1) Gram matrix
    Raises:
        ValueError: if m or n is negative or not integer
    """

    if not isinstance(m, int) or not isinstance(n, int):
        raise ValueError("m and n must be integers")
    if m < 0 or n < 0:
        raise ValueError("m and n must be non-negative")

    x = np.linspace(0, 1, m+1) #sample space

    A = np.zeros((n+1, n+1), dtype=float) #intializes 0 matrix to be filled
    np.set_printoptions(precision=3, suppress=True)
    for j in range(n+1): #THIS IS NOT A, IT IS A^* A
        for k in range(n+1):
            A[j, k] = np.sum(x**(j + k)) #fills each entry

    return A

for m, n in [(1, 1), (2, 2), (2, 3)]: #trivial examples
    M = poly_ls(m, n)
    print(f"m = {m}, n = {n}:")
    print(M, end="\n\n")
```

Some simple cases are:

$$\hat{A}(1,1) = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\hat{A}(2,2) = \begin{bmatrix} 3 & 1.5 & 1.25 \\ 1.5 & 1.25 & 1.125 \\ 1.25 & 1.125 & 1.062 \end{bmatrix} \tag{49}$$

$$\hat{A}(2,3) = \begin{bmatrix} 3 & 1.5 & 1.25 & 1.125 \\ 1.5 & 1.25 & 1.125 & 1.062 \\ 1.25 & 1.125 & 1.062 & 1.031 \\ 1.125 & 1.062 & 1.031 & 1.016 \end{bmatrix}$$

7. How Perturbations Affect The Condition Number (1e)

Still on polynomial regression, in this section we analyze what happens to $\kappa\left(\hat{A}\right)$, when $\hat{A}$ is perturbated with $m = 100$ and $n = 1, \ldots, 20$.

We will run *poly_ls(m, n)* built in Section 6 for $m = 100$ and $n = 1, \ldots, 20$. Given the sad fact that polynomial rootfinding is not a well-conditioned problem, we will then numerically calculate the condition number of $\hat{A}$ for each output of *poly_ls(m, n)*. See the code:

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

def format_scientific(x, sig=3):

    """
    Formats a number in scientific notation with a specified number of
    significant digits.

    Args:
        x (float): number to format
        sig (int): number of significant digits (default: 3)
    Returns:
        str: formatted string in scientific notation
    """

    if x == 0:
        return "0"
    exp = int(np.floor(np.log10(abs(x))))
    mant = x / 10**exp
    return f"{mant:.{sig}f} * 10^{exp}"

def compute_condition_numbers(m, max_n):

    """
    Returns a list of the condition numbers of the polynomial least-squares
    matrix A(m) for degrees n = 1 to max_n.

    Args:
        m (int): number of subintervals (m >= 0)
        max_n (int): maximum polynomial degree (max_n >= 0)
    Returns:
        list: condition numbers of A(m) for degrees n = 1 to max_n
    """

    conds = []
    for n in range(1, max_n + 1):
        A = poly_ls(m, n)
        sv = np.linalg.svd(A, compute_uv=False) #computes singular values
        conds.append(sv[0] / sv[-1]) #condition number is the ratio of the largest to smallest singular value.
    return conds

if __name__ == "__main__":
    m = 100
    max_n = 20

    cond_nums = compute_condition_numbers(m, max_n)
    n_values = np.arange(1, max_n + 1)

    print(f"Condition numbers of A (m={m}) for degree n:")
    for n, c in zip(n_values, cond_nums):
        print(f"  n = {n:2d} → κ₂(A) = {format_scientific(c)}")

    plt.figure()
    plt.semilogy(n_values, cond_nums, marker="o", linestyle="-")
    plt.xlabel("Polynomial degree $n$")
    plt.ylabel("Condition number $\\kappa(A)$")
    plt.title(f"Growth of Condition Number, $m={m}$")
    plt.grid(True, which="both", ls="--")
    plt.tight_layout()
    plt.show()
```

A good plot of the growth of the condition number is:

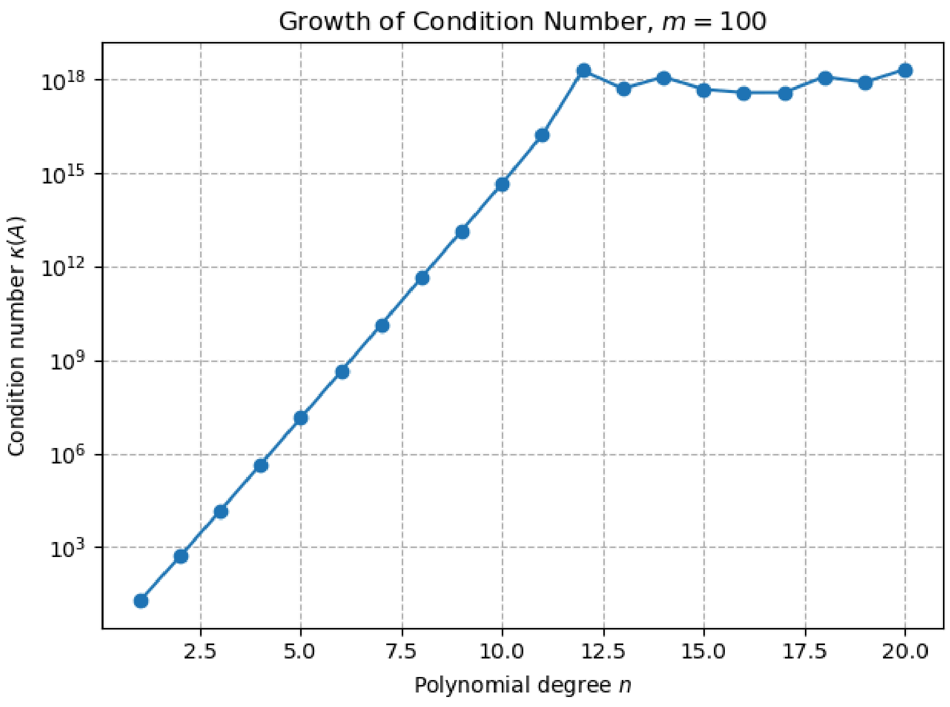

Figure 6: Growth of the condition number of A(m) for polynomial regression

Figure 6 Shows that with higher degrees of polynomials, $\kappa\left(\hat{A}_m\right)$ grows **exponentially**. This can make polynomial regression a **bad choice** for approximating datasets with many points. Which contradicts our initial assumption!

One could notice that the graph wiggles up and down at $12 \leq n \leq 20$. To understand this we must take into account that $\varepsilon_m \approx 2.22 \cdot 10^{-16}$, and at $n = 12$, $\kappa\left(\hat{A}\right)$ is already past 10^{16}. Notice as well that we are calculating on $\hat{A}$, not A, so:

$$\kappa\left(\hat{A} = A^* A\right) = \frac{\sigma_{\max(A^*A)}}{\sigma_{\min(A^*A)}} = \left(\frac{\sigma_{\max(A)}}{\sigma_{\min(A)}}\right)^2 = \kappa(A)^2 \qquad 50$$

Squaring $\kappa(A)$ pushes numbers past the IEEE 754 double precision quickly. So once the trye $\kappa\left(\hat{A} = A^* A\right)$ exceeds $\frac{1}{\varepsilon_m}$, the smallest singular value underflows to 0. So everything above that is **numerically indistinguishable**. So the line stops at $\approx 10^{16}$ and wiggles for a while.

8. Polynomial Regression with a Different Dataset

8.1. A Different Dataset

If we change $S := \left\{(t_i, b_i) \mid t_i = \frac{i}{m}, i = 0, 1, ..., m\right\}$ to $\hat{S} = \left\{(t_i, b_i) \mid t_i = \frac{i}{m} - \frac{1}{2}\right\}$, the polynomial regression becomes:

$$\begin{gathered}
p\left(t_0 = 0 - \frac{1}{2}\right) = \varphi_0 + \varphi_1\left(-\frac{1}{2}\right) + ... + \varphi_n\left(-\frac{1}{2}\right)^n = b_0 \\
p\left(t_1 = \frac{1}{m} - \frac{1}{2}\right) = \varphi_0 + \varphi_1\left(\frac{1}{m} - \frac{1}{2}\right) + \varphi_2\left(\frac{1}{m} - \frac{1}{2}\right)^2 + ... + \varphi_n\left(\frac{1}{m} - \frac{1}{2}\right)^n \\
\vdots \\
p\left(t_m = 1 - \frac{1}{2}\right) = \varphi_0 + \varphi_1\left(1 - \frac{1}{2}\right) + ... + \varphi_n\left(1 - \frac{1}{2}\right)^n
\end{gathered} \qquad 51$$

So:

$$\underbrace{\begin{bmatrix} 1 & -\frac{1}{2} & \left(-\frac{1}{2}\right)^2 & ... & \left(-\frac{1}{2}\right)^n \\ 1 & \left(\frac{1}{m} - \frac{1}{2}\right) & \left(\frac{1}{m} - \frac{1}{2}\right)^2 & ... & \left(\frac{1}{m} - \frac{1}{2}\right)^n \\ 1 & \left(\frac{2}{m} - \frac{1}{2}\right) & \left(\frac{2}{m} - \frac{1}{2}\right)^2 & ... & \left(\frac{2}{m} - \frac{1}{2}\right)^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \left(-\frac{1}{2}\right) & \left(-\frac{1}{2}\right)^2 & ... & \left(-\frac{1}{2}\right)^n \end{bmatrix}}_{A} \cdot \underbrace{\begin{bmatrix} \varphi_0 \\ \varphi_1 \\ \vdots \\ \varphi_n \end{bmatrix}}_{\Phi} = \underbrace{\begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_m \end{bmatrix}}_{b} \qquad 52$$

Projecting onto $C(A)$:

$$\underbrace{\begin{bmatrix} 1 & 1 & 1 & ... & 1 \\ -\frac{1}{2} & \left(\frac{1}{m} - \frac{1}{2}\right) & \left(\frac{2}{m} - \frac{1}{2}\right) & ... & -\frac{1}{2} \\ \left(-\frac{1}{2}\right)^2 & \left(\frac{1}{m}, -\frac{1}{2}\right)^2 & \left(\frac{2}{m} - \frac{1}{2}\right)^2 & ... & \left(-\frac{1}{2}\right)^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \left(-\frac{1}{2}\right)^n & \left(\frac{1}{m} - \frac{1}{2}\right)^n & \left(\frac{2}{m} - \frac{1}{2}\right)^n & ... & \left(-\frac{1}{2}\right)^n \end{bmatrix}}_{A^*} \cdot \underbrace{\begin{bmatrix} 1 & -\frac{1}{2} & \left(-\frac{1}{2}\right)^2 & ... & \left(-\frac{1}{2}\right)^n \\ 1 & \left(\frac{1}{m} - \frac{1}{2}\right) & \left(\frac{1}{m} - \frac{1}{2}\right)^2 & ... & \left(\frac{1}{m} - \frac{1}{2}\right)^n \\ 1 & \left(\frac{2}{m} - \frac{1}{2}\right) & \left(\frac{2}{m} - \frac{1}{2}\right)^2 & ... & \left(\frac{2}{m} - \frac{1}{2}\right)^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & -\frac{1}{2} & \left(-\frac{1}{2}\right)^2 & ... & \left(-\frac{1}{2}\right)^n \end{bmatrix}}_{A} \cdot \underbrace{\begin{bmatrix} \widehat{\varphi_0} \\ \widehat{\varphi_1} \\ \vdots \\ \widehat{\varphi_n} \end{bmatrix}}_{\hat{\Phi}} \qquad 53$$

Notice that to calculate $A^* A$ we can do:

$$(A^*A)_{ij} = \langle l_i^{A^*}, c_j^A \rangle = \langle c_i^A, c_j^A \rangle = \sum_{k=0}^{m} \left(\frac{k}{m} - \frac{1}{2} \right)^{i+j-2} \qquad 54$$

So we have:

$$\begin{bmatrix} n+1 & \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right) & \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right)^2 & \dots & \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right)^n \\ \sum_{i=0}^{n} \frac{i}{m} - \frac{1}{2} & \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right)^2 & \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right)^3 & \dots & \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right)^{n+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right)^n & \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right)^{n+1} & \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right)^{n+2} & \dots & \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right)^{2n} \end{bmatrix} \cdot \begin{bmatrix} \widehat{\varphi}_0 \\ \widehat{\varphi}_1 \\ \vdots \\ \widehat{\varphi}_n \end{bmatrix} \qquad 55$$

And doing A^*b gives:

$$= \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ -\frac{1}{2} & \left(\frac{1}{m} - \frac{1}{2}\right) & \left(\frac{2}{m} - \frac{1}{2}\right) & \dots & -\frac{1}{2} \\ \left(-\frac{1}{2}\right)^2 & \left(\frac{1}{m}, -\frac{1}{2}\right)^2 & \left(\frac{2}{m} - \frac{1}{2}\right)^2 & \dots & \left(-\frac{1}{2}\right)^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \left(-\frac{1}{2}\right)^n & \left(\frac{1}{m} - \frac{1}{2}\right)^n & \left(\frac{2}{m} - \frac{1}{2}\right)^n & \dots & \left(-\frac{1}{2}\right)^n \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{n} b_i \\ \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right) b_i \\ \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right)^2 b_i \\ \vdots \\ \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right)^n b_i \end{bmatrix} \qquad 56$$

So the system to be solved is:

$$\underbrace{\begin{bmatrix} m+1 & \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right) & \dots & \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right)^n \\ \sum_{i=0}^{n} \frac{i}{m} - \frac{1}{2} & \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right)^2 & \dots & \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right)^{n+1} \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right)^n & \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right)^{n+1} & \dots & \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right)^{2n} \end{bmatrix}}_{\hat{A}} \cdot \begin{bmatrix} \widehat{\varphi}_0 \\ \widehat{\varphi}_1 \\ \widehat{\varphi}_2 \\ \vdots \\ \widehat{\varphi}_n \end{bmatrix} = \cdot \begin{bmatrix} \sum_{i=0}^{n} b_i \\ \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right) b_i \\ \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right)^2 b_i \\ \vdots \\ \sum_{i=0}^{n} \left(\frac{i}{m} - \frac{1}{2}\right)^n b_i \end{bmatrix} 57$$

The following code calculates the new matrix $\hat{A}$ in eq. (57):

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

def poly_ls_2(m, n):

    """
    Builds the (n+1) x (n+1) matrix for least-squares polynomial fitting.

    Args:
        m (int): number of subintervals (m >= 0)
        n (int): polynomial degree (n >= 0)
    Returns:
        np.ndarray: shape (n+1, n+1) Gram matrix
    Raises:
        ValueError: if m or n is negative or not integer
    """

    if not (isinstance(m, int) and isinstance(n, int)) or m < 0 or n < 0:
        raise ValueError("m and n must be non-negative integers")

    t = np.linspace(0, 1, m + 1) - 0.5
    powers = t[:, None] ** np.arange(2 * n + 1)
    col_sums = powers.sum(axis=0)
    M = np.empty((n + 1, n + 1))
    for i in range(n + 1):
        for j in range(n + 1):
            M[i, j] = col_sums[i + j] #fills each entry

    return M

#examples:
m_1 = poly_ls_2(2, 1)
m_2 = poly_ls_2(2, 2)
m_3 = poly_ls_2(2, 3)
print("m = 2, n = 1:")
print(m_1)
print("\nm = 2, n = 2:")
print(m_2)
print("\nm = 2, n = 3:")
print(m_3)
```

The examples are:

Example 8.1.1.:

$$M(2,1) = \begin{bmatrix} 3 & 0 \\ 0 & 0.5 \end{bmatrix} \quad (58)$$

Example 8.1.2.:

$$M(2,2) = \begin{bmatrix} 3 & 0 & 0.5 \\ 0 & 0.5 & 0 \\ 0.5 & 0 & 0.125 \end{bmatrix} \quad (59)$$

Example 8.1.3.:

$$M(2,3) = \begin{bmatrix} 3 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0.125 \\ 0.5 & 0 & 0.125 & 0 \\ 0 & 0.125 & 0 & 0.031 \end{bmatrix} \quad (60)$$

8.2. How Conditioning changes (1f)

Here we will analyze how the condition number of $\hat{A}$ shown in the previous section changes with perturbations on the degree n. We will use the same method used in Section 7. $m = 100$ and $n = 1, …, 20$. The following code is used:

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

```python

def compute_condition_numbers_centered(m: int, max_n: int):

    """
    Computes the condition numbers of the polynomial least-squares matrix M(m)
    for degrees n = 1 to max_n.

    Args:
        m (int): number of subintervals (m >= 0)
        max_n (int): maximum polynomial degree (max_n >= 0)
    Returns:
        list: condition numbers of M(m) for degrees n = 1 to max_n
    """

    conds = []
    for n in range(1, max_n + 1):
        M  = poly_ls_2(m, n)
        s  = np.linalg.svd(M, compute_uv=False) #computes singular values
        conds.append(s[0] / s[-1]) #κ = σ_max / σ_min
    return conds

m, max_n = 100, 20

cond_nums = compute_condition_numbers_centered(m, max_n)
n_values  = np.arange(1, max_n + 1)

print(f"Condition numbers at (m = {m})")
for n, κ in zip(n_values, cond_nums):
    print(f"  n = {n:2d} → κ(G) = {format_scientific(κ)}")

plt.figure()
plt.semilogy(n_values, cond_nums, marker="o")
plt.xlabel("Polynomial degree $n$")
plt.ylabel(r"Condition number $\kappa_2(G)$")
plt.title(fr"Growth of $\kappa$, $m={m}$")
plt.grid(True, which="both", ls="--")
plt.tight_layout()
plt.show()
```

The expected output is:

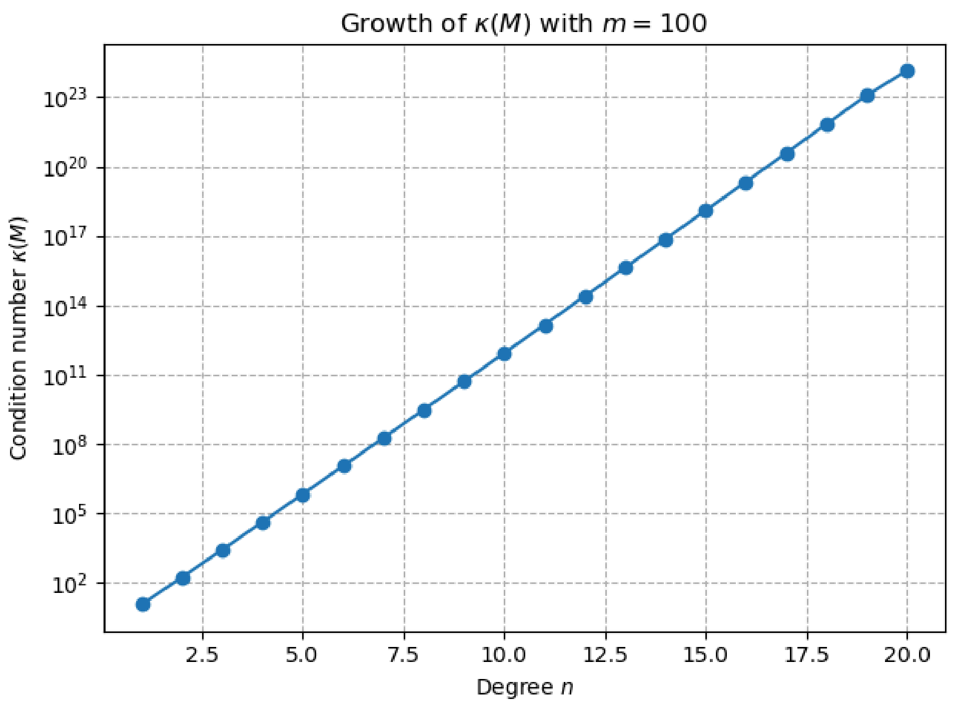

Figure 7: Growth of the condition number of $\hat{A}$ with a new dataset

The weird behavior noticed in Figure 6 is not seen in Figure 7. This is due to the fact that the dataset has been scaled to be centered at 0, reducing the dynamic range in each column and preventing the singular values from the underflow zone (at least for $20 \geq n$)

9. Comparing the Condition Number

Here we graphically compare both condition numbers seen in Figure 6 and Figure 7. The following code is used:

```python
def compute_condition_numbers(m, max_n):

    """
    Compute k(A^* A).

    Each matrix A is constructed by
    poly_ls(m, n)

    Args:
        m (int): Number of subintervals in the sample grid t_i = i/m.
        max_n (int): Maximum polynomial degree to evaluate.

    Returns:
        list[float]: The list
        [ k(A_i^* A_i)) ],
        where k(A^* A) = k(A)^2 and κ is the 2-norm condition number.
    """

    conds = []
    for n in range(1, max_n + 1):
        A = poly_ls(m, n)
        sigma = np.linalg.svd(A, compute_uv=False)
        κ2 = sigma[0] / sigma[-1]  #κ_2(A)
        conds.append(κ2 ** 2) #κ_2(A^* A) = κ_2(A)^2
    return conds


def compute_condition_numbers_centered(m, max_n):

    """
    Compute k(M) for a centred / scaled polynomial basis.
    Each matrix M is produced by poly_ls_2(m, n)

    Args:
        m (int): Number of subintervals in the sample grid t_i = i/m.
        max_n (int): Maximum polynomial degree to evaluate.

    Returns:
        list[float]: The list
        [ k(M_i) ] where
        k is the 2-norm condition number of the centred design matrix.
    """

    conds = []
    for n in range(1, max_n + 1):
        M = poly_ls_2(m, n)
        sigma = np.linalg.svd(M, compute_uv=False)
        conds.append(sigma[0] / sigma[-1])
    return conds


def plot_condition_numbers(m = 100, max_n = 20):

    """
    Plot the growth of condition numbers for raw and centred Vandermonde bases.

    Args:
        m (int, optional): Number of subintervals in the sample grid
            t_i = i/m. Defaults to 100.

        max_n (int, optional): Maximum polynomial degree to display.
            Defaults to 20.

    Returns:
        None.  The function displays a semilog plot comparing
        k(A^* A) (raw basis) and k(M) (centred / scaled basis).
    """

    n_vals = np.arange(1, max_n + 1)
    κ_raw = compute_condition_numbers(m, max_n)
    κ_centered = compute_condition_numbers_centered(m, max_n)

    plt.figure(figsize=(7, 5))
    plt.semilogy(
        n_vals,
        κ_raw,
        marker="o",
        linestyle="-",
        linewidth=1.4,
        markersize=5,
        label=r"$\kappa(A^{\top}A)$ (raw basis)",
    )
    plt.semilogy(
        n_vals,
        κ_centered,
        marker="s",
        linestyle="--",
        linewidth=1.4,
        markersize=5,
        label=r"$\kappa(M)$ (centred / scaled basis)",
    )

    plt.xlabel("Polynomial degree $n$")
    plt.ylabel("Condition number")
    plt.title(fr"Condition-number growth, $m={m}$")
    plt.grid(True, which="both", ls=":", lw=0.7)
    plt.legend()
    plt.tight_layout()
    plt.show()

plot_condition_numbers(m=100, max_n=20)
```

The expected output is the plot below:

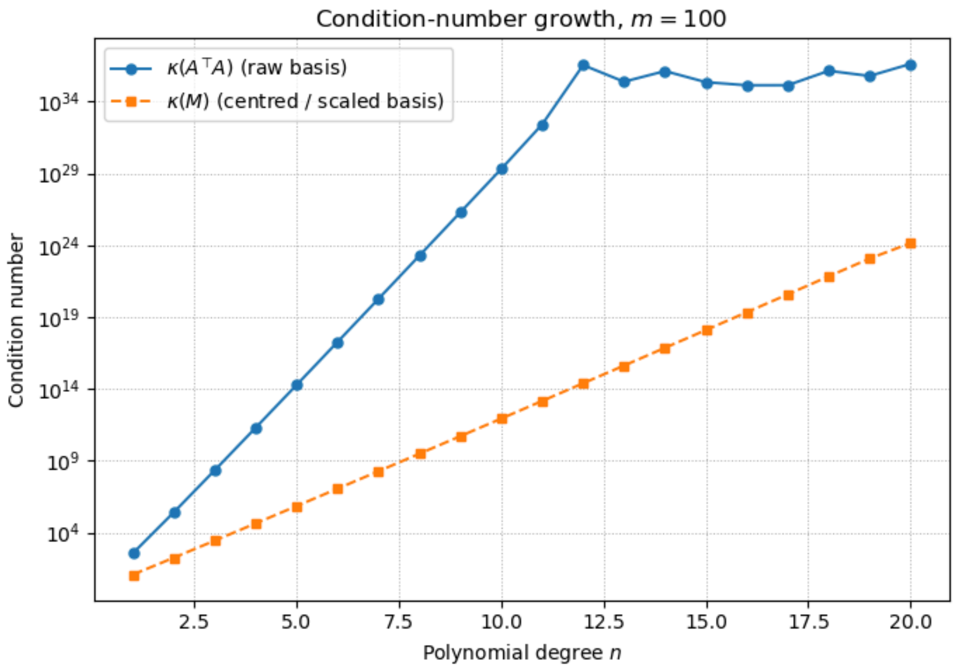

Figure 8: A comparison of both datasets

Figure 8 confirms the hypothesis that the condition number of the non-centered dataset tends to produce bigger condition numbers.

10. Least Squares with QR and SVD decompositions

We have shown the solutions to the least squares problem $Ax = b$, but this problem could be solved with factorizations of A, such as the QR and SVD, in the following sections we will show these factorizations and use them to solve the least squares problem.

10.1. QR

The QR factorization of a full-rank $A \in \mathbb{C}^{m \times n}$, $m \geq n$ an consists of finding orthonormal vectors $q_1, ..., q_n$ such that $q_1, ..., q_i$ spans $a_1, ..., q_1$, where a_i is the ith-column of A. So we want:

$$\begin{aligned} \operatorname{span}(a_1) &= \operatorname{span}(q_1) \\ \operatorname{span}(a_1, a_2) &= \operatorname{span}(q_1, q_2) \\ &\vdots \\ \operatorname{span}(a_1, ..., a_n) &= \operatorname{span}(q_1, ..., q_n) \end{aligned} \tag{61}$$

This is equivalent to:

$$A = \begin{bmatrix} q_1 & \dots & q_n \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ & r_{22} & \dots & r_{2n} \\ & & \vdots & \vdots \\ & & & r_{nn} \end{bmatrix} \tag{62}$$

Where $r_{ii} \neq 0$, because a_i will be expressed as a linear combination of q_i, and since the triangular matrix is invertible, q_i can be expressed as a linear combination of a_i. Therefore eq. (62) is:

$$\begin{aligned} a_1 &= q_1 r_{11}, \\ a_2 &= r_{12} q_1 + r_{22} q_2, \\ &\vdots \\ a_n &= r_{1n} q_1 + r_{2n} q_2 + \dots + r_{nn} q_n. \end{aligned} \quad 63$$

Or:

$$A = \hat{Q}\hat{R} \quad 64$$

Is the *reduced* QR decomposition of A.

The *full* QR decomposition of $A \in \mathbb{C}^{m \times n}$ not of full-rank is analogous to the reduced, but $|m - n|$ 0-columns are appended to $\hat{Q}$ to make it a unitary $m \times m$ matrix Q, and 0-rows are aded to $\hat{R}$ to make it a $m \times n$ still triangular matrix:

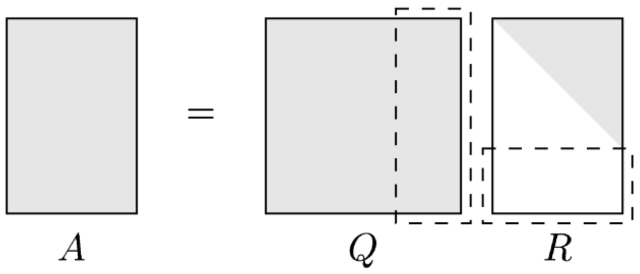

Figure 9: Full QR factorization

And the decomposition becomes:

$$A = QR \quad 65$$

Here are some examples:

Example 10.1.1.:

$$A = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 5 \end{bmatrix} \quad 66$$

This is a diagonal matrix, so its QR factorization is particularly simple:

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, R = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 5 \end{bmatrix} \quad 67$$

With diagonal matrices, Q is the identity matrix and $R = A$.

Example 10.1.2.:

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \quad 68$$

For this 3×2 matrix, we compute the reduced QR factorization:

$$\hat{Q} = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} \end{bmatrix}, \hat{R} = \begin{bmatrix} \sqrt{2} & \frac{1}{\sqrt{2}} \\ 0 & \frac{\sqrt{2}}{2} \end{bmatrix} \quad (69)$$

This is a reduced QR factorization where $\hat{Q}$ is 3×2. The full QR factorization would require extending $\hat{Q}$ to a 3×3 orthogonal matrix and adding a row of zeros to $\hat{R}$ as shown in Figure 9.

10.2. SVD

The *singular value decomposition* of a matrix is based on the fact that the image of the unit sphere under a $m \times n$ matrix is a **hyperellipse:**

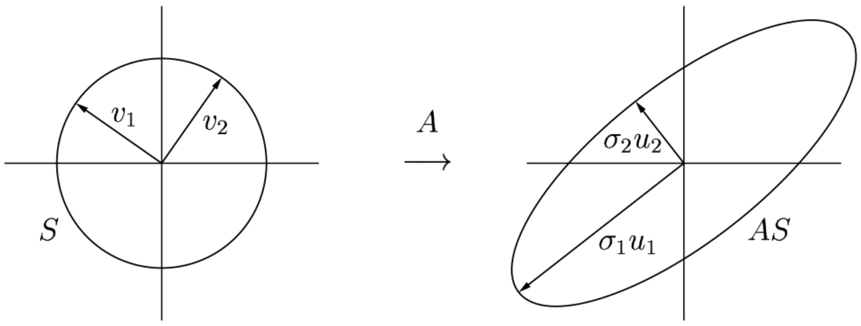

Figure 10: SVD of a 2×2 matrix

So the independent directions v_1, v_2 have been mapped to another set of orthogonal directions $\sigma_1 v_1, \sigma_2 v_2$, so with $S := \{v \in \mathbb{C}^n \mid \|v\| = 1\}$ as the unit ball, let's define:

Definition 10.2.1: (Singular Values) The n *singular values* σ_i of $A \in \mathbb{C}(m \times n)$ are the lengths of the n new axes of AS, written in non-crescent order $\sigma_1 \geq \ldots \geq \sigma_n$.

Definition 10.2.2: (Left Singular Vectors) The n **left** singular vectors of A are the unit vectors u_i laying in AS, oriented to correspond and number the singular values σ_i, respectively

Definition 10.2.3: (Right Singular Vectors) The **right** singular vectors of A are the v_i in S that are the preimages of $\sigma_i u_i \in AS$, such that $Av_i = \sigma_i u_i$

The equation $Av_i = \sigma_i u_i$ is equivalent to:

$$A \cdot \begin{bmatrix} v_1 & v_2 & \ldots & v_n \end{bmatrix} = \begin{bmatrix} \sigma_1 u_1 & \sigma_2 u_2 & \ldots & \sigma_n u_n \end{bmatrix} \quad (70)$$

Better:

$$A \cdot \begin{bmatrix} \\ v_1 & v_2 & \dots & v_n \\ \\ \end{bmatrix} = \begin{bmatrix} \\ u_1 & u_2 & \dots & u_n \\ \\ \end{bmatrix} \cdot \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \sigma_n \end{bmatrix} \tag{71}$$

Or simple $AV = U\Sigma$, but since V has orthonormal columns:

$$A = U\Sigma V^* \tag{72}$$

The SVD is a very particular factorization for matrices, as the following theorem states:

Theorem 10.2.1: (Existence of SVD) *Every* matrix $A \in \mathbb{C}^{m \times n}$ has a singular value decomposition

Proof: We prove the existane by fixing the largest image of A and using induction on the dimension of A:

Let $\sigma_1 = \|A\|_2$. There must exist unitary vectors $u_1, v_1 \in \mathbb{C}^n$ such that $Av_1 = \sigma_1 u_1$, with $\|v_1\|_2 = \|u_1\|_2 = 1$. Let $\{v_j\}$ and $\{u_j\}$ be 2 orthonormal bases of $\mathbb{C}^n$. These column vectors form the unitary matrices V_1 and U_1. We will compute:

$$\Phi = U_1^* A V_1 \tag{73}$$

Notice that the first column of Φ is $U_1^* A v_1 = \sigma_1 U_1^* v_1 = \sigma_1 e_1$, since u_1 is the first column of U_1. So Φ looks like:

$$\Phi = \begin{bmatrix} \sigma_1 & w^* \\ 0 & B \end{bmatrix} \tag{74}$$

Where w^* is the rest of the first row, the action of A onto the remaining columns v_j. B acts on the subspace orthogonal to v_1.

We want $w = 0$, we can force this by using the norm. We know that:

$$\left\| \begin{bmatrix} \sigma_1 & w^* \\ 0 & B \end{bmatrix} \cdot \begin{bmatrix} \sigma_1 \\ w \end{bmatrix} \right\|_2 = \left\| \begin{bmatrix} \sigma_1^2 + w^* w \\ Bw \end{bmatrix} \right\|_2 = \sqrt{|\sigma_1^2 + w^* w|^2 + \|Bw\|_2^2} \tag{75}$$

And:

$$\sqrt{|\sigma_1^2 + w^* w|^2 + \|Bw\|_2^2} \geq \sigma_1^2 + w^* w \tag{76}$$

We also know:

$$\|\Phi\|_2 = \sup_{\|y\|=1} \|\Phi y\|_2 \tag{77}$$

For the specific $x = [\sigma_1, w]$ scaled to the unit ball, and knowing $\|\Phi\|_2 = \sigma_1$, we have:

$$\begin{gathered} \|\Phi\|_2 \geq \frac{\|\Phi x\|_2}{\|x\|_2} \geq \frac{\sigma_1^2 + w^* w}{\sqrt{\sigma_1^2 + w^* w}} = \sqrt{\sigma_1^2 + w^* w} \Leftrightarrow \sigma_1 \geq \sqrt{\sigma_1^2 + w^* w} \\ \Leftrightarrow \sigma_1^2 \geq \sigma_1^2 + w^* w \Leftrightarrow w^* w = 0 \Leftrightarrow w = 0. \end{gathered} \tag{78}$$

If $m = 1$ or $n = 1$, we are done, If not, B has an SVD decomposition $B = U_2 \Sigma_2 V_2^*$ by the induction hypothesis, so from eq. (73) we have that the following is a SVD decomposition of A, completing the proof:

$$A = U_1 \begin{bmatrix} 1 & 0 \\ 0 & U_2 \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & V_2 \end{bmatrix}^* V_1^* \tag{79}$$

□

Is the SVD factorization of A. There are more about the SVD on computing U, Σ, V^*, as we will show below:

Theorem 10.2.2: $\forall A \in \mathbb{C}^{m \times n}$, the following holds:

- The eigenvalues of $A^* A$ are the singular values *squared* of A, and the column-eigenvectors of $A^* A$ form the matrix V.
- The eigenvalues of $A A^*$ are the singular values *squared* of A, and the column-eigenvectors of $A A^*$ form the matrix U.

Proof: Let $U \Sigma V^* = A$ be the SVD of A, then computing $A^* A$, knowing U, V are unitary matrices, we have:

$$A^* A = (U \Sigma V^*)^* (U \Sigma V^*) = V \Sigma^* U^* U \Sigma V^* = V \Sigma^* \Sigma V^* = V \Sigma^2 V^* \tag{80}$$

This is an *eigenvalue* decomposition of $A^* A$, where the eigenvalues are the entries of Σ^2, which are the singular values of A **squared**, and the eigenvectors are the columns of V.

For $A A^*$, we have:

$$A A^* = (U \Sigma V^*)(U \Sigma V^*)^* = U \Sigma V^* V \Sigma^* U^* = U \Sigma \Sigma^* U^* = U \Sigma^2 U^* \tag{81}$$

The reasoning here is analogous. So the proof is complete. □

By Theorem 10.2.2, calculating the SVD of A has been reduced to calculating the eigenvalues and eigenvectors of $A^* A$ and $A A^*$, here are some examples of singular value decompositions:

Example 10.2.1.: Consider $A = \begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix}$. Computing the SVD:

First, find $A^* A = \begin{bmatrix} 13 & 12 \\ 12 & 13 \end{bmatrix}$ and calculate its eigenvalues: $\lambda_1 = 25, \lambda_2 = 1$

The singular values are $\sigma_1 = 5, \sigma_2 = 1$.

The right singular vectors (eigenvectors of $A^* A$): $V = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$

The left singular vectors (obtained from $A v_i = \sigma_i u_i$): $U = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$

Therefore, the SVD is: $A = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \cdot \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}^*$

Example 10.2.2.: Consider a non-square matrix $A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$. For this 2×3 matrix, for the SVD we do:

$$A^*A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 1 \end{bmatrix} \tag{82}$$

The eigenvalues of A^*A are $\lambda_1 = 3, \lambda_2 = 1, \lambda_3 = 0$, so the singular values are $\sigma_1 = \sqrt{3}, \sigma_2 = 1, \sigma_3 = 0$

The right singular vectors (eigenvectors of A^*A) are:

$$V = \begin{bmatrix} \frac{1}{2} & -\frac{1}{\sqrt{2}} & \frac{1}{2} \\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \\ \frac{1}{2} & \frac{1}{\sqrt{2}} & \frac{1}{2} \end{bmatrix} \tag{83}$$

And now for AA^*:

$$AA^* = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \tag{84}$$

The eigenvalues are $\lambda_1 = 3, \lambda_2 = 1$, so the singular values are $\sigma_1 = \sqrt{3}, \sigma_2 = 1$. The eigenvectors are:

$$U = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \tag{85}$$

Therefore, the full SVD is:

$$A = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \cdot \begin{bmatrix} \sqrt{3} & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{2} & -\frac{1}{\sqrt{2}} & \frac{1}{2} \\ \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \\ \frac{1}{2} & \frac{1}{\sqrt{2}} & \frac{1}{2} \end{bmatrix}^* \tag{86}$$

10.3. Least Squares with QR and SVD

Here we will write code that solves the least squares problem usig the 2 factorizations shown in Section 10.1 and Section 10.2, as well as the ordinary approach to least squares shown in Section 3.

The following code has functions that solve the least squares problem using the QR and SVD decompositions, as well as through the normal approach shown in Section 3. A function to build the matrix A is also included. With a boolean parameter to alternate betweeen a centered $(t_i = \frac{i}{m})$ or a non-centered $\left(t_i = \frac{i}{m} - \frac{1}{2}\right)$ dataset:

```python
from numpy.linalg import qr, svd, solve, pinv, cond #easier

def ls_qr(A, b):

    """
    Solves the least squares problem Ax = b using QR decomposition.

    Args:
        A : system matrix
        b : right-hand side vector
    Returns:
        x : solution vector
        y : fitted values (Ax)
    """

    Q, R = qr(A, mode='reduced')
    x = solve(R, Q.T @ b)
    y = A @ x

    return x, y

def ls_svd(A, b):

    """
    Solves the least squares problem Ax = b using SVD decomposition.

    Args:
        A : system matrix
        b : right-hand side vector
    Returns:
        x : solution vector
        y : fitted values (Ax)
    """

    U, S, Vt = svd(A, full_matrices=False)

    S_inv = np.zeros_like(S) #calculate the pseudo-inverse using SVD, x = V * S⁻¹ * U.T * b

    tol = np.finfo(float).eps * max(A.shape) * S[0] #tolarance for singular values

    for i in range(len(S)):
        if S[i] > tol:
            S_inv[i] = 1.0 / S[i] #inverts if above tolarance

    x = (Vt.T @ np.diag(S_inv) @ U.T) @ b
    y = A @ x

    return x, y

def ls_normal(A, b):

    """
    Solves the least squares problem Ax = b using normal equations.

    Args:
        A : system matrix
        b : right-hand side vector
    Returns:
        x : solution vector
        y : fitted values (Ax)
    """

    ATA = A.T @ A
    ATb = A.T @ b

    x = solve(ATA, ATb)
    y = A @ x

    return x, y

def build_A_matrix(m, n, centralized=False):

    """
    Creates the matrix A for polynomial regression of degree n.

    Args:
        m (int): number of subintervals (m >= 0)
        n (int): polynomial degree (n >= 0)
        centralized (bool): if True, use centralized points
    Returns:
        A (np.ndarray): shape (m+1, n+1) matrix for polynomial regression
        t (np.ndarray): array of points used to create the matrix
    """

    if centralized:
        t = np.array([i/m - 1/2 for i in range(m+1)])
    else:
        t = np.array([i/m for i in range(m+1)])

    A = np.zeros((m+1, n+1))

    for i in range(n+1):
        A[:, i] = t**i  #clever

    return A, t
```

10.4. Examples (2b)

We will use the functions defined on the previous sections to do regression on the following functions:

$$\begin{aligned} f(t) &= \sin(t) \\ g(t) &= e^t \\ h(t) &= \cos(3t) \end{aligned} \qquad (87)$$

The results are shown:

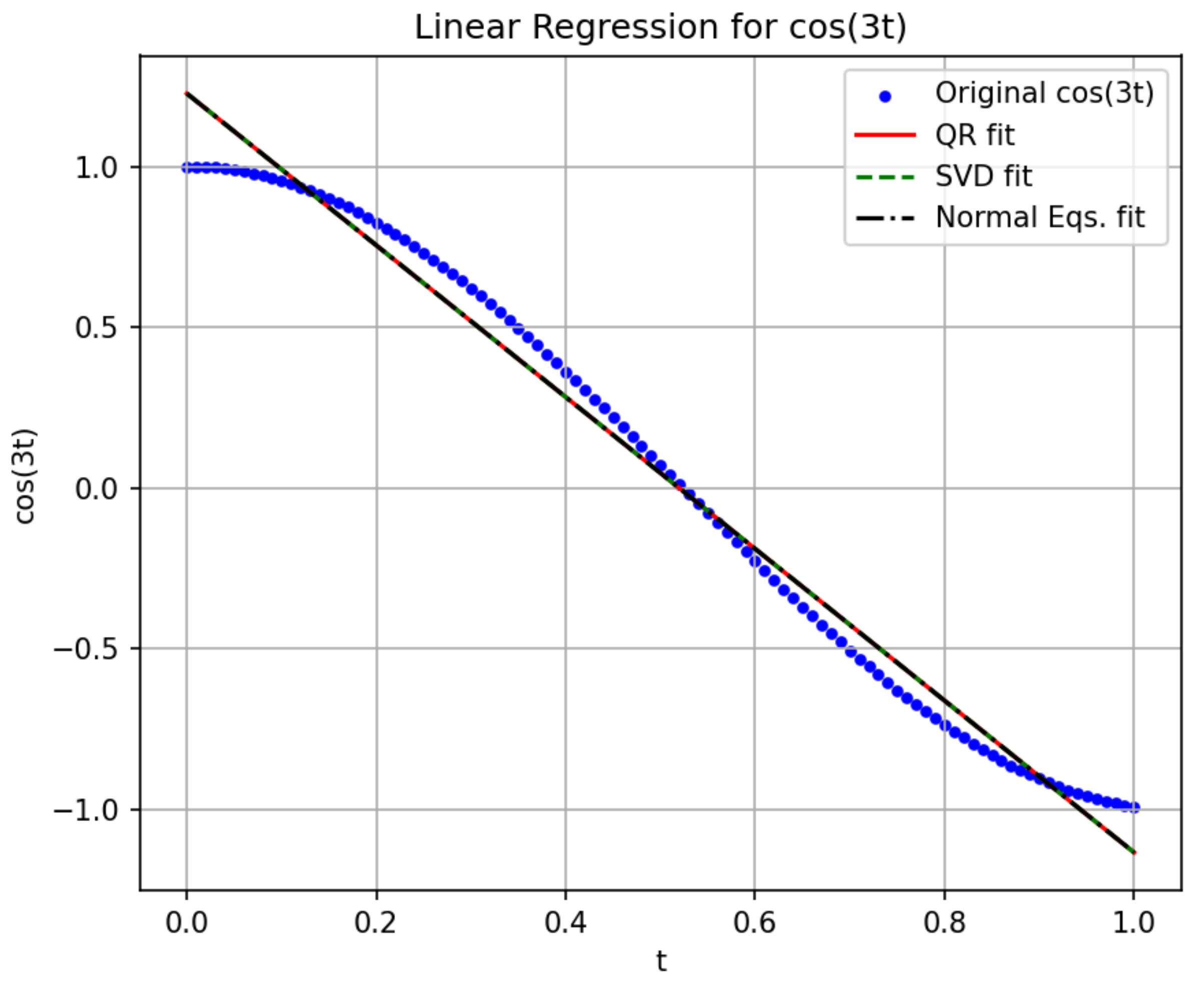

Figure 11: Linear regression on $\cos(t)$

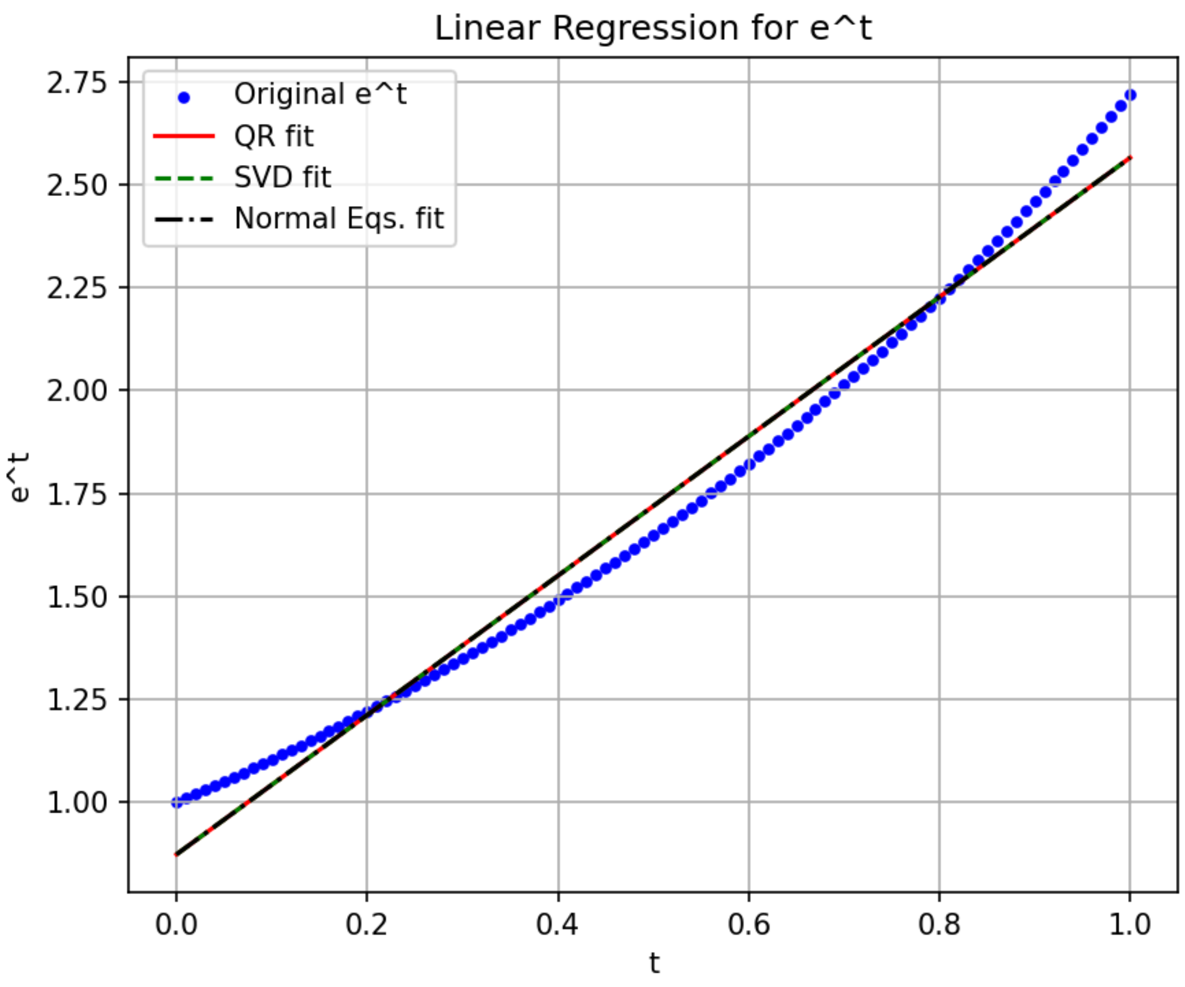

Figure 12: Linear regression on e^t

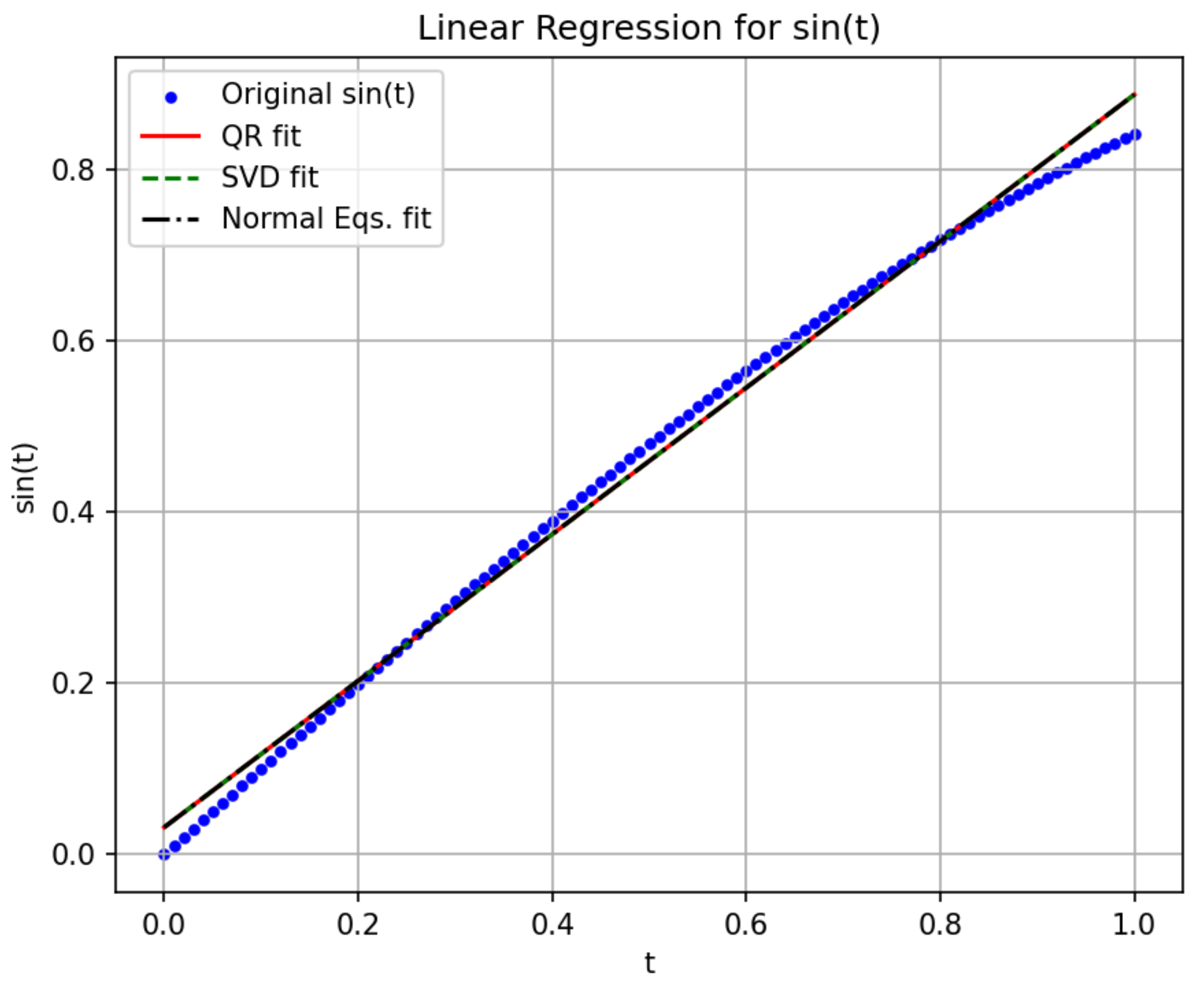

Figure 13: Linear regression on $\sin(t)$

Plots Figure 11, Figure 12, Figure 13 shows us that all 3 methods result in the same line. This is expected, for we are using 100 equally sparse points on a small portion of the image of the functions.

10.5. How good are the approximations? (2c)

In Section 7, we have seen that polynomial regression is not a very well-conditioned problem, so here we will analyze the errors produced by such method. We use what has been done in the previous section.

The following code plots the errors:

```python
m = 100
t = np.linspace(0, 1, m+1)

function_data = [(res['name'], res['values']) for res in results] #data here

max_degree = 15
all_errors = {name: {'qr': [], 'svd': [], 'normal': []} for name, _ in function_data}
all_cond_numbers = []

```

```python
for n in range(1, max_degree + 1):
    print(f"\nPolynomial degree: {n}")

    A, _ = build_A_matrix(m, n) #matrix for regression
    cond_num = cond(A)
    all_cond_numbers.append(cond_num)
    print(f"Condition number of A: {format_scientific(cond_num)}")

    for name, values in function_data: #forall funcs, solve using all methods
        try:
            x_qr, y_qr = ls_qr(A, values)
            error_qr = np.linalg.norm(y_qr - values)
            all_errors[name]['qr'].append(error_qr)
        except Exception as e:
            print(f"Error with QR for {name}, degree {n}: {e}")
            all_errors[name]['qr'].append(np.nan)

        try:
            x_svd, y_svd = ls_svd(A, values)
            error_svd = np.linalg.norm(y_svd - values)
            all_errors[name]['svd'].append(error_svd)
        except Exception as e:
            print(f"Error with SVD for {name}, degree {n}: {e}")
            all_errors[name]['svd'].append(np.nan)

        try:
            x_normal, y_normal = ls_normal(A, values)
            error_normal = np.linalg.norm(y_normal - values)
            all_errors[name]['normal'].append(error_normal)
        except Exception as e:
            print(f"Error with Normal Equations for {name}, degree {n}: {e}")
            all_errors[name]['normal'].append(np.nan)

        #show errors
        print(f"{name} - Errors: QR: {format_scientific(all_errors[name]['qr']
        [-1])}, "
              f"SVD: {format_scientific(all_errors[name]['svd'][-1])}, "
              f"Normal: {format_scientific(all_errors[name]['normal'][-1])}")

degrees = list(range(1, max_degree + 1))

for name, _ in function_data:
    plt.figure(figsize=(6, 5), dpi=150)
    plt.semilogy(degrees, all_errors[name]['qr'],      'ro-', label='QR')
    plt.semilogy(degrees, all_errors[name]['svd'],     'gs-', label='SVD')
    plt.semilogy(degrees, all_errors[name]['normal'], 'bx-', label='Normal
    Eqs.')

    plt.title(f'Error vs. Polynomial Degree for {name}')
    plt.xlabel('Polynomial degree $n$')
    plt.ylabel('Residual error (log scale)')
    plt.grid(True, which='both', ls=':')
    plt.legend()
    plt.tight_layout()
    plt.show()

plt.figure(figsize=(6, 5), dpi=150)
plt.semilogy(degrees, all_cond_numbers, 'mo-')
plt.xlabel('Polynomial degree $n$')
plt.ylabel('Condition number (log scale)')
plt.title('Condition Number vs. Polynomial Degree')
plt.grid(True, which='both', ls=':')
plt.tight_layout()
plt.show()
```

The expected output are the plots:

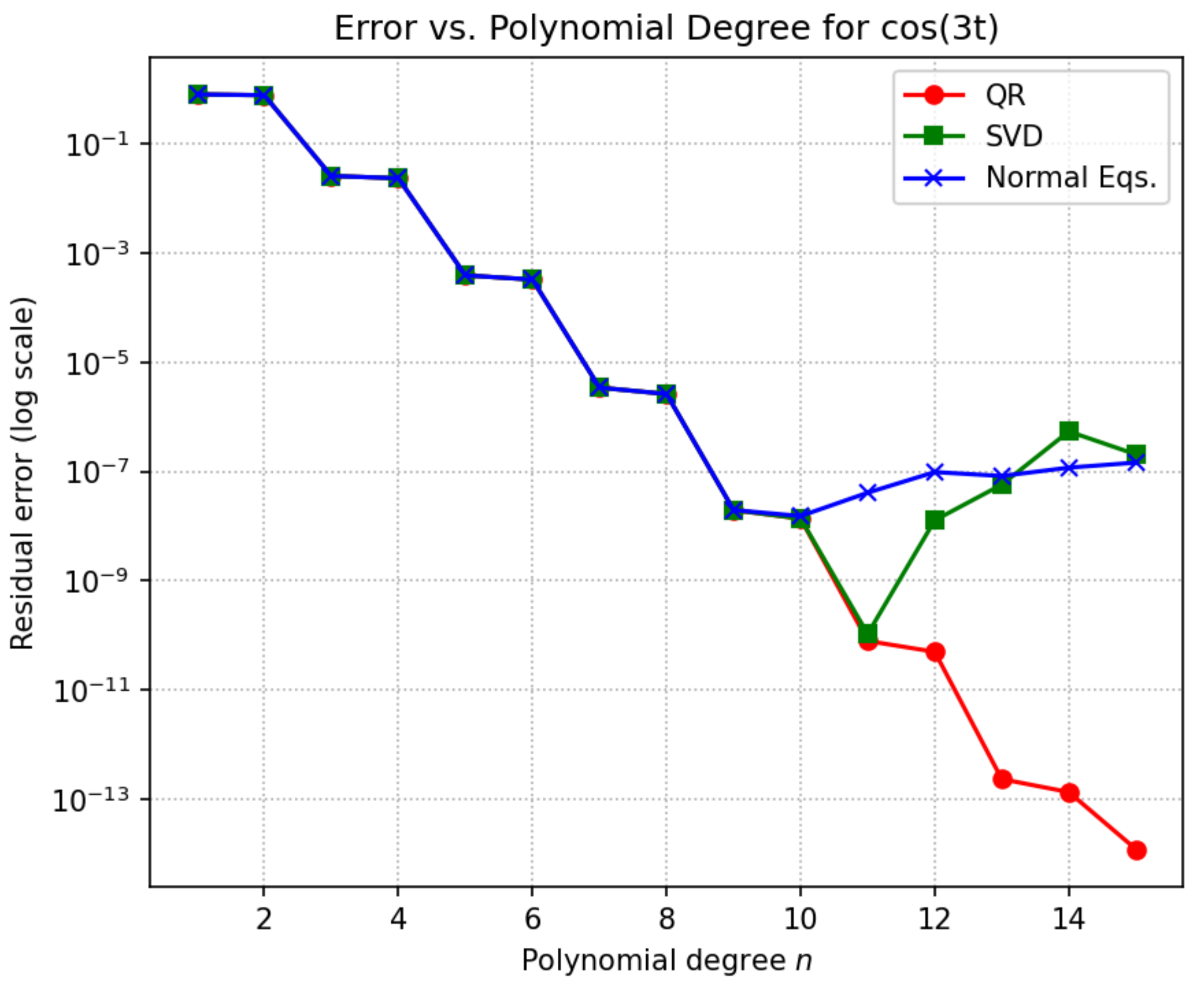

Figure 14: Error on $\cos(3t)$

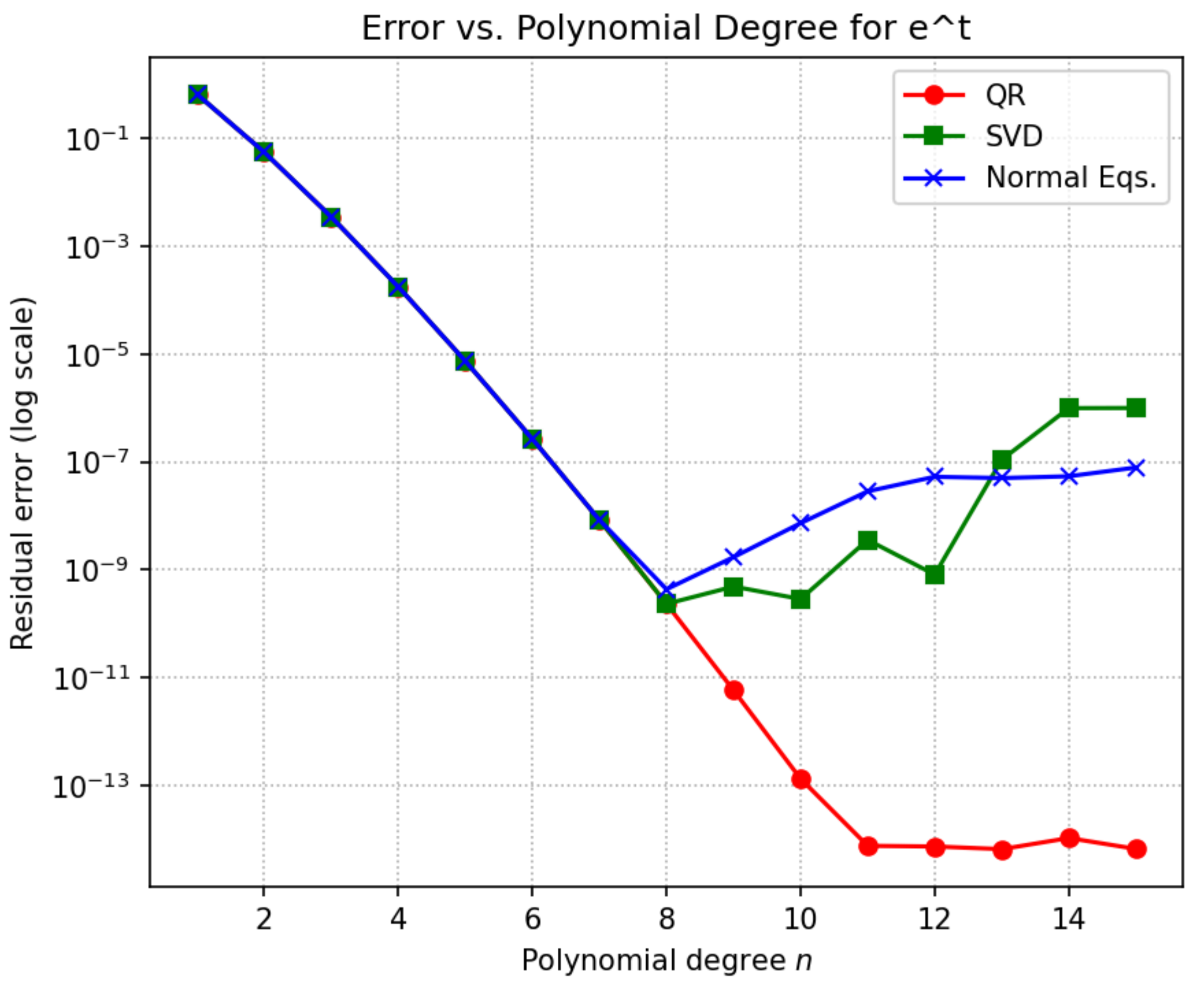

Figure 15: Error on e^t

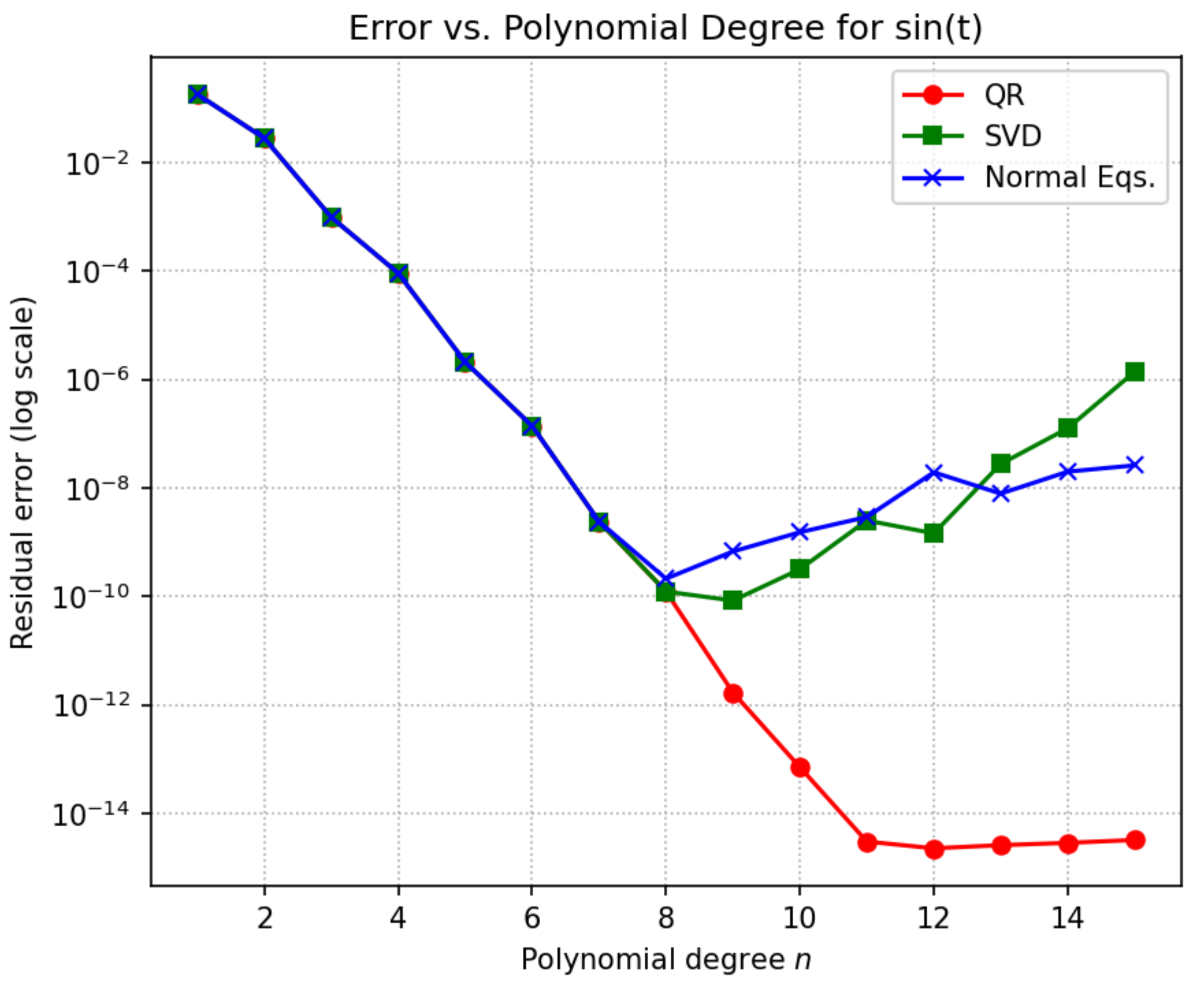

Figure 16: Error on $\sin(t)$

We know that the functions e^t and $\sin(t)$ are crescent on $[0, 1]$ and shifted up, so the linear model would require higher degree to better describe this curve.

- **Normal System:** The error falls considerably until $\approx 10^{-8}$, which is expected.
- **SVD:** After $\approx n = 8$ it starts climbing, due to the loss of significance from the ill-conditioned Vandermonde Matrix (huge condition numbers).
- **QR:** Climbs all the way down until machine precision, which is expected from the QR factorization.

Now the function $\cos(3t)$ behaves differently. Expanding it on the Taylor Series:

$$\cos(3t) = \cos(3t) \tag{88}$$