W. Viles, w.viles@northeastern.edu

hw06 - least squares applications

You are free to use either R or Python for the programming exercises. Please refrain from using libraries. Please write your code using only linear algebra concepts as opposed to using libraries. I would anticipate that this entire assignment can be completed only with the use of numpy for those who will use Python.

We begin this assignment with an example where a least squares approximation is not required and then we generalize the problem so that it is required.

Polynomial interpolation

We begin this section with a theorem.

Theorem 1. *Let $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ be a collection of n points in $\mathbb{R}^2$. There exists a **unique** $(n-1)$-order polynomial*

$$f(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_{n-1} x^{n-1},$$

where $b_0, b_1, \ldots, b_{n-1} \in \mathbb{R}$ are scalars, such that $p(x_k) = y_k$ for all $k = 1, 2, \ldots, n$.

Before we prove this theorem, consider a few low-order cases: If we are provided with two points $(x_1, y_1), (x_2, y_2)$ in $\mathbb{R}^2$, then we are able to find a unique line that passes through these two points. Moreover, if we are provided with three points $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ in $\mathbb{R}^2$, then we are able to find a unique parabola that passes through these three points, and so forth.

Proof. Let $\mathbf{b} = (b_0, b_1, \ldots, b_{n-1})'$ be the vector of coefficients and define the matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ as consisting of columns

$$\mathbf{a}_k = \begin{pmatrix} x_1^k \\ x_2^k \\ \vdots \\ x_n^k \end{pmatrix} \in \mathbb{R}^n$$

for $k = 0, 1, \ldots, n-1$. Since the columns of $\mathbf{A} = \begin{bmatrix} \mathbf{a}_0 & \mathbf{a}_1 & \ldots & \mathbf{a}_{n-1} \end{bmatrix}$ are linearly independent then there exists a unique solution $\hat{\mathbf{b}} \in \mathbb{R}^n$ such that

$$\mathbf{y} = \mathbf{A}\hat{\mathbf{b}},$$

where $\mathbf{y} = \begin{pmatrix} y_1 & y_2 & \ldots & y_n \end{pmatrix}'$, namely, $\hat{\mathbf{b}} = \mathbf{A}^{-1}\mathbf{y}$. □

1.1 Write a program that simulates $n \in \mathbb{Z}_+$ data points (x_i, y_i), for $i = 1, 2, \ldots, n$. I would recommend using the standard normal probability distribution to simulate data.

1.2 Write a program that accepts as input the vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ consisting of

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \tag{1}$$

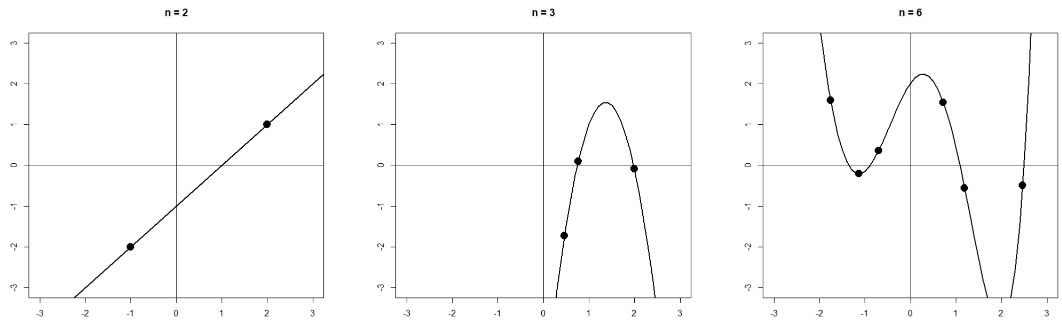
and computes the vector of coefficients $\hat{\mathbf{b}} = \mathbf{A}^{-1}\mathbf{y}$.

1.3 Write a program that computes

$$\hat{f}(x) = \hat{b}_0 + \hat{b}_1 x + \ldots + \hat{b}_{n-1} x^{n-1}.$$

for any $x \in \mathbb{R}$. I would recommend implementing Horner's rule.

1.4 Write a script that can produce plots that look somewhat like the following examples.



I encourage you to consider a wide range of n-values in order to see how the constraints, i.e., the n data points, cause the polynomial to exhibit erratic behavior.

Least squares with polynomial basis

Once again, let $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ be a collection of n points in $\mathbb{R}^2$ for some $n \in \mathbb{Z}_+$. Suppose now, however, that one does not wish to perfectly fit the data points (which occurs in nearly 100% of cases) but would prefer instead to find a low-order polynomial approximation to the data points.

Let $0 \le m < n$ be an integer and define the $(m-1)$-order polynomial $g(x)$ as

$$\begin{aligned} g(x) &= b_0 + b_1 x + \ldots + b_{m-1} x^{m-1} \\ &= \sum_{j=0}^{m-1} b_j x^j. \end{aligned}$$

We seek $g(x)$ such that $\sum_{i=1}^{n} (y_i - g(x_i))^2$ is minimized. Let $\mathbf{y} \in \mathbb{R}^n$ be defined as in Equation (1) and define the matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$ as consisting of columns

$$\mathbf{a}_k = \begin{pmatrix} x_1^k \\ x_2^k \\ \vdots \\ x_n^k \end{pmatrix} \in \mathbb{R}^n$$

for $k = 0, 1, \ldots, m-1$. That is, $\mathbf{A} = \begin{bmatrix} \mathbf{a}_0 & \mathbf{a}_1 & \ldots & \mathbf{a}_{m-1} \end{bmatrix}$. We seek $\mathbf{b} \in \mathbb{R}^m$ such that the quantity $\|\mathbf{y} - \mathbf{A}\mathbf{b}\|^2$ is minimized.

2.1 Write the normal equations for this least squares problem and then determine an expression for $\hat{\mathbf{b}} \in \mathbb{R}^m$, the coefficients of the polynomial

$$\hat{g}(x) = \hat{b}_0 + \hat{b}_1 x + \ldots + \hat{b}_{m-1} x^{m-1}$$

that minimizes $\|\mathbf{y} - \mathbf{A}\mathbf{b}\|^2 = \sum_{i=1}^{n} (y_i - g(x_i))^2$.

2.2 Write a program that accepts as input $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, as defined in Equation (1), and computes the vector of coefficients $\hat{\mathbf{b}}$.

2.3 Use your program from **1.1** to simulate $n \in \mathbb{Z}_+$ data points $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and store these values. These same values must be used in each of the following subproblems.

2.3a Write a program that computes the squared error $\sum_{i=1}^{n} (y_i - g(x_i))^2$ for any set of coefficients in the polynomial.

2.3b For the data points that you recently simulated and stored, fit an $(m-1)$-order polynomial to the data points, for $m = 1, 2, \ldots, n$ and compute the squared error loss $\sum_{i=1}^{n}(y_i - g(x_i))^2$ for each value of m. Produce a plot of these values with squared error loss along the vertical axis and $m = 0, 1, \ldots, n-1$ along the horizontal axis.

2.4 We will now simulate data, so that we have knowledge of the true relationship between $x, y \in \mathbb{R}$.

2.4a Program this function.

Input: .
n: number of data points (x_i, y_i)
m: one more than the order of the polynomial $g(x)$
$\sigma > 0$: the standard deviation of the error terms
Assumption: $m < n$
Output: .
$\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$: the simulated data
$\mathbf{b} \in \mathbb{R}^m$: the vector of coefficients used to simulate the data

---

$\mathbf{x} \in \mathbb{R}^n \sim N(\mathbf{0}, \mathbf{I})$; // x consists of n independent standard normal random variables.
$\mathbf{b} \in \mathbb{R}^m \sim N(\mathbf{0}, \mathbf{I})$; // b consists of m independent standard normal random variables.
$\boldsymbol{\epsilon} \in \mathbb{R}^n \sim N(\mathbf{0}, \sigma^2\mathbf{I})$; // $\boldsymbol{\epsilon}$ consists of n independent normal random variables with mean 0 and standard deviation σ.
Initialize $\mathbf{y} \in \mathbb{R}^n$
for $i = 1, 2, \ldots, n$ **do**
  $y_i \leftarrow \text{Horner}(\mathbf{b}, x_i)$; // Horner's rule from Exercise 1.3
end
return $\mathbf{x}, \mathbf{y}, \mathbf{b}$

Please see https://en.wikipedia.org/wiki/Horner%27s_method for more information on Horner's rule.

2.4b For $m = 0, 1, \ldots, n-1$, find the best fit polynomial to the data simulated by the program in Exercise 2.4a.

2.4c Use your program from 2.3*a* and compute the squared error loss by the least squares approximation, i.e., low-order polynomial fit, to the data for polynomials of order $m = 0, 1, \ldots, n-1$. Produce a plot of these values with squared error loss along the vertical axis and values of m along the horizontal axis.

2.5 Now that you are able to simulate $n \in \mathbb{R}^n$ data points in $\mathbb{R}^2$ from a polynomial of a given order, and that you are able to estimate the coefficients of a polynomial of any order $m = 0, 1, \ldots, n-1$, and that you are able to compute the error of approximation through the squared error loss function, please experiment with situations in which

- m^*: the order of the polynomial that generated the data
- m: the order of the polynomial that you will fit to the data via least squares approximation
- n: sample size
- σ: standard deviation of the random errors.

Note that it must be that $n > m$ and that $n > m^*$ in order for your programs to run properly. Produce some interesting plots that may suggest that the squared error loss $\sum_{i=1}^{n}(y_i - g(x_i))$ is generally minimized around $m \approx m^*$.