Problem 1: Can you fit a line?

Problem Statement

Consider N data points $(\mathbf{x}_i, y_i)$ for $i = 1, \ldots, N$ obtained from an experiment , where $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i \in \mathbb{R}$. Let $N > n+1$. The goal of linear regression is to find the "best" linear fit; i.e. find $\mathbf{c} \in \mathbb{R}^n$, $d \in \mathbb{R}$ s.t.:

$$y_i \approx \mathbf{c}^T \mathbf{x}_i + d, \text{ for } i = 1, \ldots, N$$

a

Suppose each measurement is corrupted by independent Gaussian noise with identical variances and 0 mean. Find $\mathbf{A} \in \mathbb{R}^{N\times(n+1)}$ in terms of $\mathbf{x}_1, \ldots, \mathbf{x}_N$ s.t. that the maximum likelihood estimate of $\hat{\mathbf{c}}$, $\hat{d}$ is

$$\begin{pmatrix} \hat{\mathbf{c}} \\ \hat{d} \end{pmatrix} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1} \mathbf{A}^T y, \text{ where } \mathbf{y} := \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$$

b

Consider the case with $n = 1$; i.e. $x_1, \ldots, x_N$ are scalars. Finding the best linear fit amounts to finding $(\hat{c}, \hat{d})$ tht minimizes:

$$J(c, d) := \sum_{i=1}^{N} (y_i - cx_i - d)^2$$

Compute a stationary point $(\hat{c}, \hat{d})$ by setting the derivative of J w.r.t c and d to zero.

c

Use the second derivative tests to conclude that your $(\hat{c}, \hat{d})$ computed in (b) is indeed a local minimizer of J. Can you conclude from this second derivative test alone that it is a global minimizer?

d

Consider the following (x, y) pairs:

- $(1.00, 1.10)$
- $(1.50, 1.62)$
- $(2.00, 1.98)$
- $(2.57, 2.37)$
- $(3.00, 3.23)$
- $(3.50, 3.69)$
- $(4.00, 3.97)$

Draw a scatter plot of these points; then, plot the best linear fit to these points using your formula in part (b).

Solution

a

We can rewrite the given equation to:

$$y_i = \mathbf{x}_i^T \mathbf{c} + d$$

Additionally, let: $\mathbf{1}_N$ be a column vector of 1's with N rows. Thus:

$$\mathbf{y} = \begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_N \end{pmatrix} \mathbf{c} + d\mathbf{1}_N$$

We can further re-arrange this to:

$$\mathbf{y} = \begin{pmatrix} \mathbf{x}_1 & 1 \\ \mathbf{x}_2 & 1 \\ \vdots & \vdots \\ \mathbf{x}_N & 1 \end{pmatrix} \begin{pmatrix} \mathbf{c} \\ d \end{pmatrix} \implies \mathbf{A} = \begin{pmatrix} \mathbf{x}_1 & 1 \\ \mathbf{x}_2 & 1 \\ \vdots & \vdots \\ \mathbf{x}_N & 1 \end{pmatrix}$$

b

$$\nabla J(c, d) = \begin{pmatrix} \frac{\partial J}{\partial c} \\ \frac{\partial J}{\partial d} \end{pmatrix} = 2 \begin{pmatrix} \sum_{i=1}^N (y_i - cx_i - d)(-x_i) \\ \sum_{i=1}^N (y_i - cx_i - d)(-1) \end{pmatrix}$$

Setting equal to zero and noting that the 2 can then just drop out (and push the negative sign around):

$$\mathbf{0} = \begin{pmatrix} \sum_{i=1}^N (cx_i + d - y_i)(x_i) \\ \sum_{i=1}^N (cx_i + d - y_i) \end{pmatrix}$$

Exanding the top:

$$0 = \sum_{i=1}^N (cx_i^2 + dx_i - y_i x_i) = c\left(\sum x_i^2\right) + d\left(\sum x_i\right) - \left(\sum x_i y_i\right)$$

And the bottom:

$$0 = \sum_{i=1}^N (cx_i + d - y_i) = c\left(\sum x_i\right) + d\left(\sum 1\right) - \left(\sum y_i\right)$$

Solve for c:

$$c = \left(\left(\sum y_i\right) - d\left(\sum 1\right)\right) / \left(\sum x_i\right)$$

Note that $\sum_{i=1}^N 1 = N$. Plug into the first partial derivative equation:

$$0 = \frac{\sum y_i - Nd}{\sum x_i} \sum x_i^2 + d\sum x_i - \sum x_i y_i$$

Solve for d:

$$\frac{\sum x_i^2 \sum y_i - Nd\sum x_i^2}{\sum x_i} + d\sum x_i - \sum x_i y_i$$

$$
\begin{gathered}
= \sum x_i^2 \sum y_i - Nd \sum x_i^2 + d \left(\sum x_i\right)^2 - \sum x_i \sum x_i y_i \\
Nd \sum x_i^2 - d \left(\sum x_i\right)^2 = \sum x_i^2 \sum y_i - \sum x_i \sum x_i y_i \\
d \left(N \sum x_i^2 - \left(\sum x_i\right)^2\right) = \sum x_i^2 \sum y_i - \sum x_i \sum x_i y_i \\
d = \frac{\sum x_i^2 \sum y_i - \sum x_i \sum x_i y_i}{N \sum x_i^2 - \left(\sum x_i\right)^2}
\end{gathered}
$$

which can then be plugged back in to get c. I'm lazy so I used Wolfram to simplify the expression and got:

$$
c = \frac{\sum x_i \sum y_i - N \sum x_i y_i}{\left(\sum x_i\right)^2 - N \sum x_i^2}
$$

c

Given the gradient, the Hessian is trivially calculable as:

$$
\nabla^2 J(c, d) = 2 \begin{pmatrix} \sum x_i^2 & \sum x_i \\ \sum x_i & N \end{pmatrix}
$$

If $\nabla^2 J \succcurlyeq 0$ everywhere then J is convex and the first order condition ($\nabla J = 0$) is sufficient for both a global and local optimality. HOWEVER, this will ultimately depend on the x's chosen, as well as the quantity N chosen. The eigenvalues are given by[1]:

$$
\lambda_{1,2} = \frac{1}{2} \left(\sum x_i^2 + N \pm \sqrt{(\sum x_i^2)^2 + 4 \sum (x_i)^2 + N^2 - 2N \sum x_i^2} \right)
$$

which must be $\lambda_{1,2} \geq 0$ for the matrix to be PSD. We know that $N \geq 1$ and $x_i \in \mathbb{R} \implies \sum x_i^2 \geq 0$, so for both eigenvalues to be non-negative the following must hold:

$$
\sum x_i^2 + N \geq \sqrt{\sum x_i^2 - 2N \sum x_i^2 + 4 \left(\sum x_i\right)^2 + N^2}
$$

Failing this condition, the matrix is not PSD and we cannot assume global optimality (although it doesn't imply it is NOT a global optimum).

[1] Thanks, Wolfram!

d

Code is made available at the end of document.

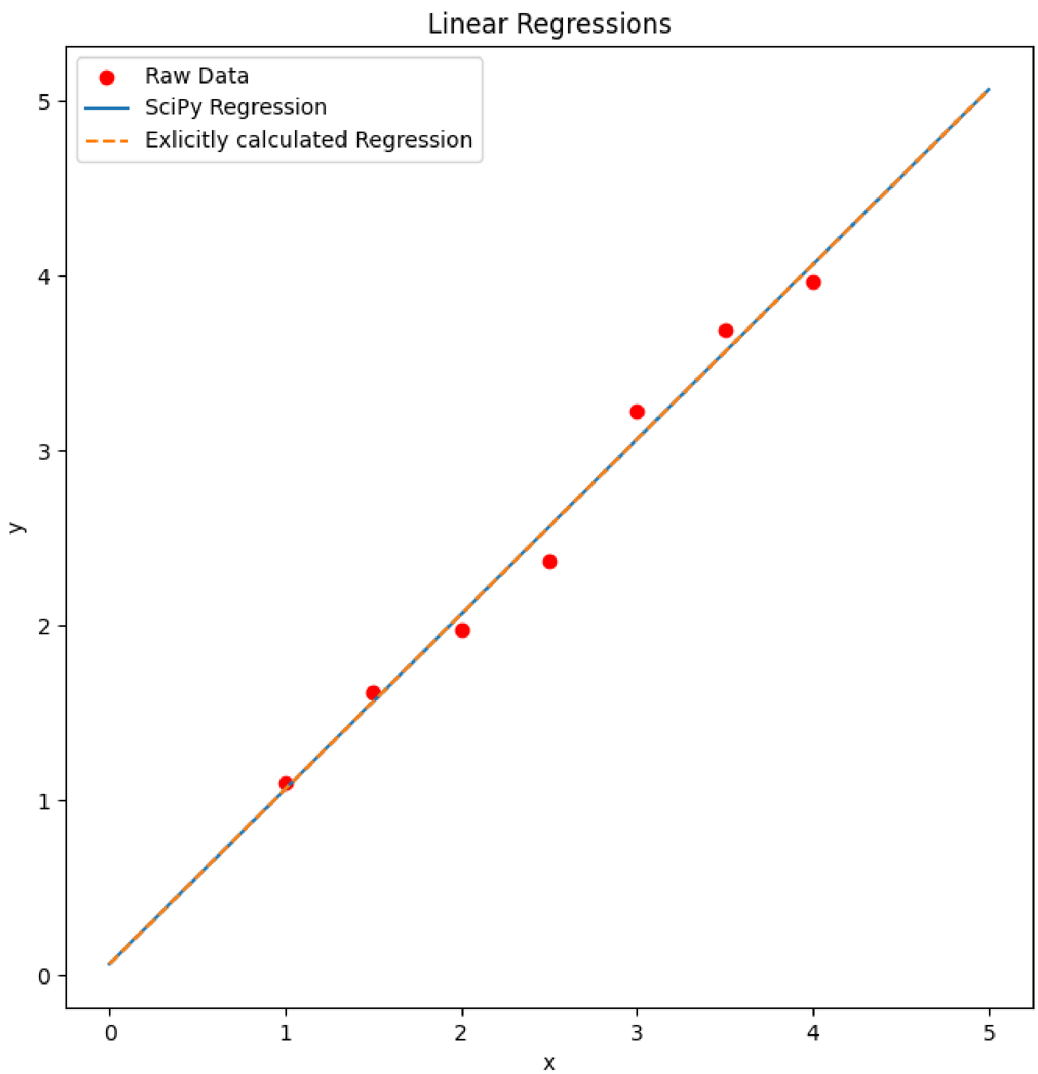

Figure 1: Linear Regression on the provided data, using both the derived expression (dashed orange line) and SciPy's built-in method for verification (solid blue line)

Problem 2: Estimate with Confidence

Problem Statement

Suppose the true parameters of a system are described by $x = (1, 1)^T$, and the measurements are given by:

$$\mathbf{y} = \underbrace{\begin{pmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \\ 4 & 5 \end{pmatrix}}_{:=A} \mathbf{x} + \mathbf{e}$$

where $e \in \mathbb{R}^4$ is sampled from a multivariate Gaussian distribution with zero mean and covariance matrix $\mathbf{R} := \text{diag}(0.1, 0.2, 0.3, 0.4)$. That is, $\mathbf{e} \sim \mathcal{N}(0, \mathbf{R})$.

a

Write code for the following experiment:

1. Generate an error vector $\mathbf{e}$ according to $\mathcal{N}(0, \mathbf{R})$. Compute $\mathbf{y}$ from it and report it.
2. In class we derived a formula to compute the maximum likelihood estimate $\hat{\mathbf{x}}$ from $\mathbf{y}$. Using $\mathbf{y}$ from the previous step, compute and report your estimate $\hat{\mathbf{x}}$.
3. The estimated error is given by $\mathbf{y} - \mathbf{A}\hat{\mathbf{x}}$. With $\mathbf{y}$ from the prior steps, compute and report:

   $$J(\hat{\mathbf{x}}) := (\mathbf{y} - \mathbf{A}\hat{\mathbf{x}})^T \mathbf{R}^{-1} (\mathbf{y} - \mathbf{A}\hat{\mathbf{x}})$$

   Recall that we claimed (without proof) that $J(\hat{\mathbf{x}}) \sim \chi_2^2$. Then, we have:

   $$\mathbb{P}\{J(\hat{\mathbf{x}}) \leq 5.9915\} = 0.95$$

   where $\mathbb{P}\{\mathcal{E}\}$ denotes the probability of an event $\mathcal{E}$. Concludewith a 95% confidence level whether your estimate error conforms to the error model we assumed.

b

Repeat the experiment in the prior section TEN-THOUSAND TIMES. Each time you run the experiment, record the value of $J(\hat{\mathbf{x}})$. Plot a histogram of $J(\hat{\mathbf{x}})$. Comment qualitatively how the histogram of $J(\hat{\mathbf{x}})$ compares to the PDF of a χ_2^2 random variable. Also, report the percentage of times you concluded that your measurement did not conform t the error model. Compare this fraction to your confidence interval in the prior portion.

Solution

a

Again, code is made available at the end of the assignment. The output of the script is:

```
First y:
[2.93953942 4.85114251 7.50790296 8.75054375]
x hat:
[1.1427999  0.90150926]
J: 1.1135043299064333, Conforms to error model (with 95% confidence): True
```

b

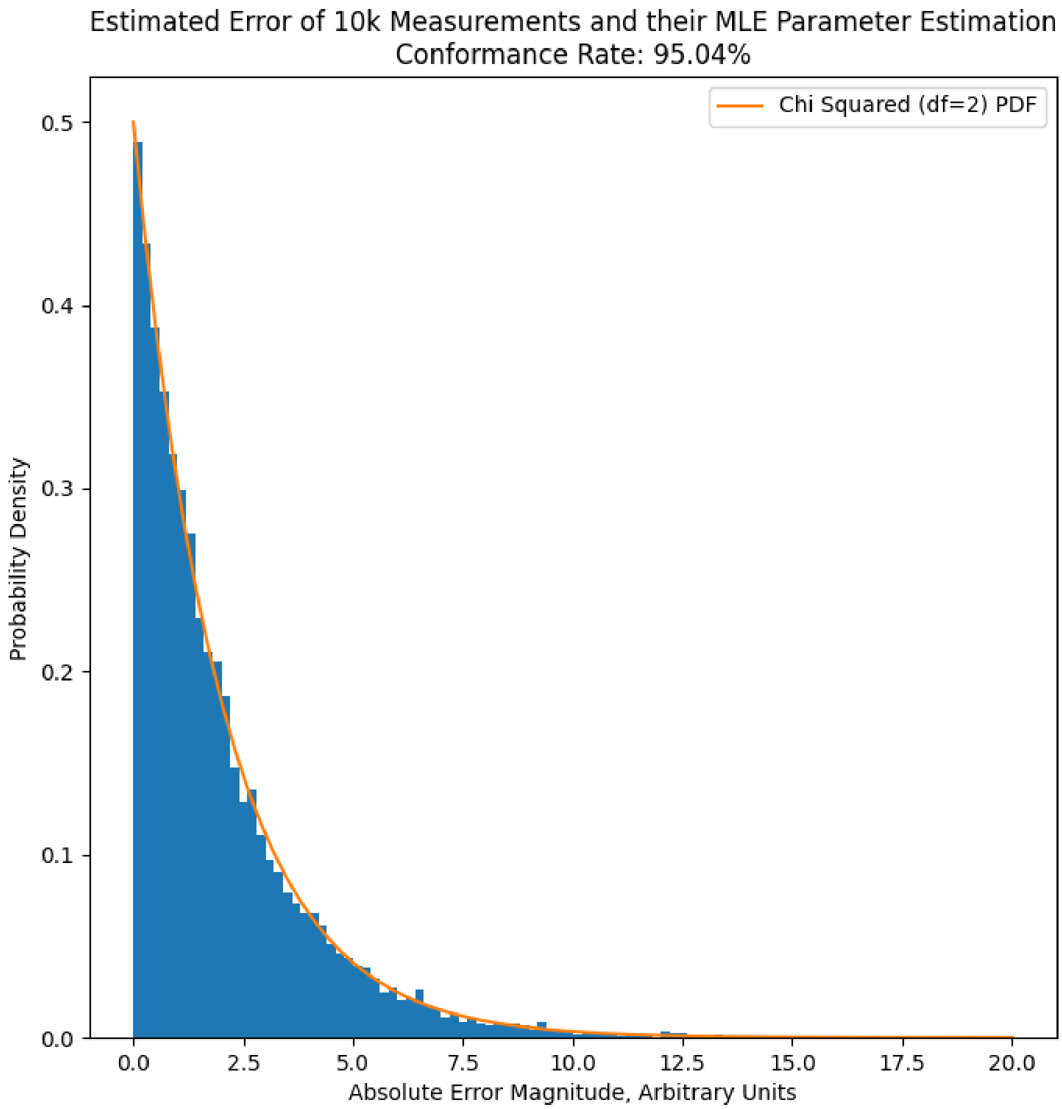

Figure 2: Density based histogram of 10k experiments with a 2-DoF Chi-Squared PDF superimposed. Qualitatively...pretty dang good!

Code

```
from typing import Callable

import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import linregress, chi2

DATA = np.array(
    [
        (1.00, 1.10),
        (1.50, 1.62),
        (2.00, 1.98),
        (2.50, 2.37),
        (3.00, 3.23),
        (3.50, 3.69),
        (4.00, 3.97),
    ]
).T


def scipy_regression(
    x: np.ndarray, y: np.ndarray
) -> Callable[[np.ndarray], np.ndarray]:
    r = linregress(x, y)
    f = lambda x: r.slope * x + r.intercept

    return f


def my_regression(x: np.ndarray, y: np.ndarray) -> Callable[[np.ndarray], np.ndarray]:
    N = y.size
    d = (np.sum(x**2) * np.sum(y) - np.sum(x) * np.sum(x * y)) / (
        N * np.sum(x**2) - np.sum(x) ** 2
    )
    c = (np.sum(x) * np.sum(y) - N * np.sum(x * y)) / (
        np.sum(x) ** 2 - N * np.sum(x**2)
    )

    return lambda x: c * x + d


def problem_one() -> None:
    x = DATA[0]
    y = DATA[1]
    f_hat = scipy_regression(x, y)
    f_mine = my_regression(x, y)
    x_base = np.linspace(0, 5, 100)
    y_hat_scipy = f_hat(x_base)
    y_hat_mine = f_mine(x_base)
    plt.figure(figsize=(8, 8))
    plt.title("Linear Regressions")
    plt.scatter(x, y, label="Raw Data", c="red")
    plt.plot(x_base, y_hat_scipy, label="SciPy Regression")
    plt.plot(
        x_base, y_hat_mine, label="Exlicitly calculated Regression", linestyle="dashed"
    )
    plt.xlabel("x")
    plt.ylabel("y")
    plt.legend()

    plt.savefig("./hw3_regression.png")


def generate_e(mean: np.ndarray, covariance_matrix: np.ndarray) -> np.ndarray:
    return np.random.multivariate_normal(mean, cov=covariance_matrix)


class NoisySystem:
    def __init__(
        self, A: np.ndarray, x: np.ndarray, mean: np.ndarray, R: np.ndarray
    ) -> None:
        self.A = A
        self.x = x
        self.mean = mean
        self.R = R

    def __call__(self) -> np.ndarray:
        return self.A @ self.x + generate_e(self.mean, self.R)

    def __repr__(self) -> str:
        return f"A:\n{self.A}\nx:\n{self.x}\nMean:\n{self.mean}\nR:\n{self.R}"


class LinearEstimator:
    def __init__(self, A: np.ndarray, R: np.ndarray) -> None:
        # I know, I know, directly calling an inverse is bad...
        self.R_inv = np.linalg.inv(R)
        self.A = A
        self.da_matrix = np.linalg.inv(A.T @ self.R_inv @ A) @ A.T @ self.R_inv

```

```python
    def __call__(self, y: np.ndarray) -> np.ndarray:
        return self.da_matrix @ y

    def estimate_error(self, y: np.ndarray, x_hat: np.ndarray) -> float:
        tmp = y - (self.A @ x_hat)
        return tmp.T @ self.R_inv @ tmp


def problem_2_experiment(system: NoisySystem, estimator: LinearEstimator) -> float:
    y = system()
    x_hat = estimator(y)
    return estimator.estimate_error(y, x_hat)


def problem_two() -> None:
    mean = np.zeros(4)
    R = np.diag([0.1, 0.2, 0.3, 0.4])

    A = np.array([[1, 2], [2, 3], [3, 4], [4, 5]])
    x_true = np.array([1, 1])
    system = NoisySystem(A, x_true, mean, R)
    estimator = LinearEstimator(A, R)

    first_y = system()
    print(f"First y:\n{first_y}")
    x_hat = estimator(first_y)
    print(f"x hat:\n{x_hat}")
    J = estimator.estimate_error(first_y, x_hat)
    print(f"J: {J}, Conforms to error model (with 95% confidence): {J <= 5.9915}")

    EXPERIMENT_COUNT = 10000
    rslts = np.zeros(EXPERIMENT_COUNT)
    for i in range(EXPERIMENT_COUNT):
        rslts[i] = problem_2_experiment(system, estimator)

    times_conformed = np.count_nonzero(rslts <= 5.9915)
    error_rate = (times_conformed / EXPERIMENT_COUNT) * 100

    df = 2
    x = np.linspace(rslts.min(), rslts.max(), 10000)

    fig, ax = plt.subplots(1, 1, figsize=(8, 8))
    ax.hist(rslts, bins=100, density=True)
    ax.set_title(
        f"Estimated Error of 10k Measurements and their MLE Parameter Estimation\nConformance Rate: {error_rate}%"
    )
    ax.set_xlabel("Absolute Error Magnitude, Arbitrary Units")
    ax.set_ylabel("Probability Density")
    ax.plot(x, chi2(df).pdf(x), label="Chi Squared (df=2) PDF")
    ax.legend()
    plt.savefig("./problem2_hist.png")


if __name__ == "__main__":
    problem_one()
    problem_two()
```