Simulation and Performance Evaluation
Homework 3

Blascovich Alessio and Di Noia Matteo

16th May 2025

Exercise 1

Sub-exercises 1, 2 and 3

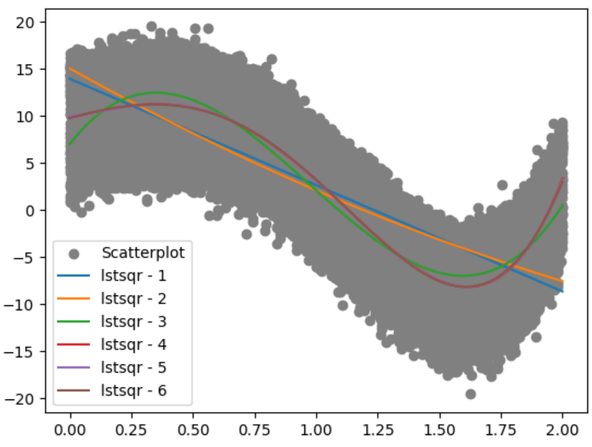

Figure 1: Measurements and "Least square" trends for various ranks

After drawing the the dataset scatter plot, it is clear that there is a trend in it. The trend is a sinusoidal-like curve with a local maximum after 0.3 seconds and a local minimum after 1.60 seconds.

We used least square method to identify the trend. To be specific we used the polynomial fit with exponents between 1 and 6. We used:

- A *Vandermonde*'s matrix called A for the timestamps;
- An array y for the measurements taken.

The trend was represented by the array $\hat{b} = (A'A)^{-1}A'y$. In our code the upper bound exponent for A has been called *rank*. For the sake of simplicity we handled all the matrix operations using the library `numpy`.

We iteratively increase the rank of the least squared function until, graphically, the curve was well fitted. Ranks before 4 gave us a poor fitting but from 4 we started seeing a well fitted curve so we decided to go once step further and use rank 5, even though 4 is probably enough.

The dataset has been de-trended by subtracting, the value of a polynomial function fitted over the `trend` array. Specifically, for each time point i, the trend value at a given time was computed by evaluating the polynomial (whose coefficients are stored in the trend array) at the corresponding time `times[i]`. The de-trended measurement is then given by:

$$\texttt{measurements[i]} - \sum_{j=\texttt{len(trend)}}^{0} \texttt{trend[j]} \cdot (\texttt{times[i]}^{j})$$

```python
for i in range(len(measurements)):
    measurements[i] = measurements[i] - np.polyval(trend, times[i])
    # np.polyval evaluates a polynomial at specific values.
```

By drawing scatter plot of new data we showed that the trend was removed.

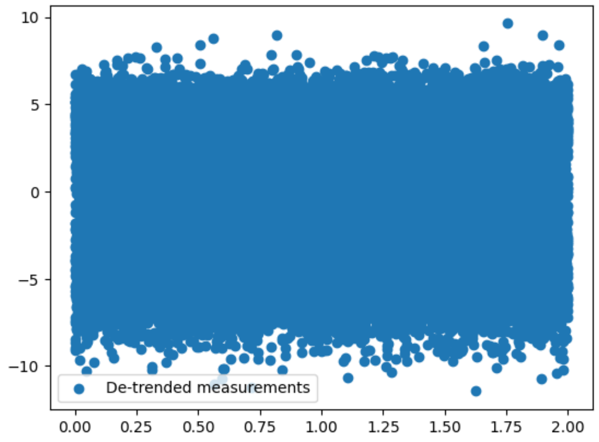

Figure 2: De-trended scatter plot

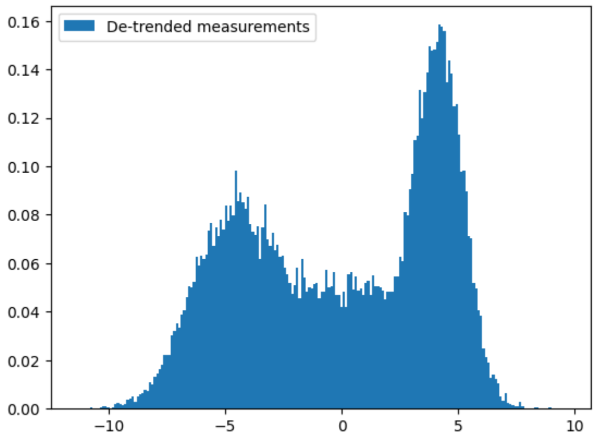

Figure 3: De-trended and normalised histogram

Sub-exercise 4

We started with three normal distributions:

- $\mu = [-5, 0, 5]$;
- $\sigma = [1, 1, 1]$;
- probability $= [1/3, 1/3, 1/3]$.

These were our initial assumptions about the Gaussians parameters. We could have done better by picking values closer to the values written on the homework text or by looking at the histogram.

We implemented the "Expectation-Maximization" following the procedures explained during the course. The algorithm proceeds iteratively through a series of iterations. During each iteration the algorithm computes the probability of a measurement to come from a specific Gaussian and then improves its assumptions about the parameters described above. In our implementation the probability of a point to be part of a Gaussian is updated as well. The resulting distribution fits quite well the dataset as shown in figure:

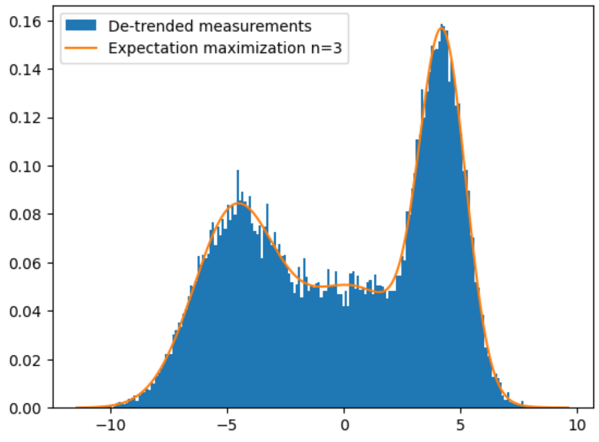

Figure 4: Distribution produced by the E-M method

The table below shows the results obtain by our algorithm compared against the given parameters. The estimators generated by our algorithm are: $\overline{\mathrm{X}}$ and S^2, while the parameters given in the assignment text are: μ and σ^2. The comparison between our estimators and given parameters highlights that the mean estimators are slightly shifted to the right while the variance estimators are fairly precise.

	$\overline{\mathrm{X}}$	μ	S^2	σ^2	probability
Gaussian # 1	-4.72	-5	3.07	3	0.345
Gaussian # 2	0.410	0	5.96	6	0.302
Gaussian # 2	4.26	4	0.980	1	0.353

Sub-exercise 5

It is possible to automatically pick the best number of Gaussians to approximate the original distribution. The algorithm estimates the parameters for, an increasing, number n of Gaussians using the "Expectation-Maximization" algorithm. For each iteration over n we produced the T-value for the "Chi-squared" test, this value either stabilises as the mixture of Gaussians fits the dataset or increases again.

We then defined the value Δ which is the minimum improvement needed to continue the search for a better Gaussian. For each sum of Gaussians, after the first, we computed the improvement as:

$$\frac{T - T'}{T'}$$

where T' is the T-value for the sum of n Gaussians, while T is the T-value for the sum of $k-1$ Gaussians. The algorithm continues its iteration over n if

$$\frac{T - T'}{T'} < \Delta$$

Once this condition was violated we checked on "Chi-squared" tables for the value of $P\{\chi^2_{k-1} \geq T'\}$, if that value was ≥ 0.05 the we could not reject (neither accept) the hypothesis about the number n. This part of the procedure can be automated as well using the function `chisquare` from the `numpy` library.

With $\Delta = 0.01 = 1\%$, we got that a possible n is 3, this can be easily seen in the following picture:

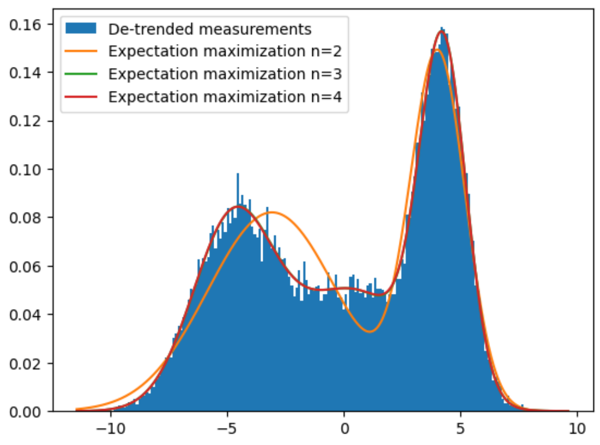

Figure 5: Graphical representation of "Chi-squared" test

Using this procedure we shown that the hypothesis with $n = 2$ must be rejected, while we could not reject (neither accept) the hypotheses $n = 3, 4$.