Euclid summer school: Parameter estimation for a polynomial model

August 21, 2023

1 Analytical solution

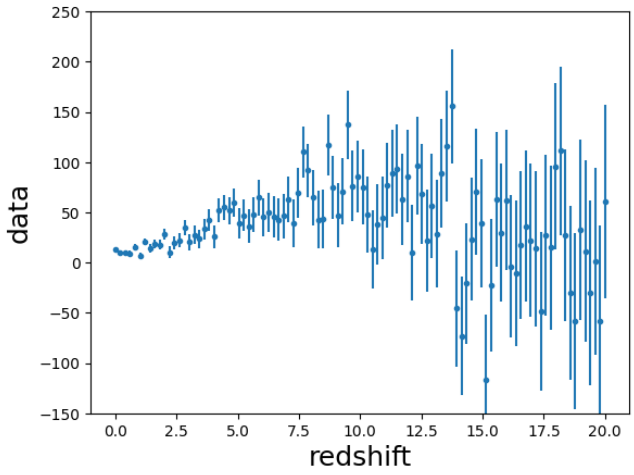

Figure 1: The measured data points as a function of redshift

We have access to N measurements of a physical phenomenon that evolves as a function of redshift. Our data model for this phenomenon is an order M polynomial

$$d(z_i) = d_i = \sum_{k=0}^{M} a_k (z_i - z_c)^k + n(z_i) \tag{1}$$

a_k are the model parameters that we would like to estimate, z_i are the N redshifts at which the measurements have been done and $n(z_i) = n_i$ is the noise on the measurements which follows a multivariate gaussian distribution $\mathcal{N}(0, \Sigma)$.
We will assume for now that M=3 and $z_c = 0$.

1) Give an analytic formula for the likelihood of the data given the model parameters $\mathcal{L}(\{d_i\}|\{a_k\})$ (think about the probability distribution followed by $d_i - \sum_{k=0}^{M} a_k (z_i - z_c)^k$

2) Download the data file: data_example.txt, in here, the column are redshift, data, error, and reproduce Figure 1.

3) For convenience, we will write the data model in vectorial form $\boldsymbol{d} = \mathcal{P}\boldsymbol{a} + \boldsymbol{n}$ where

$$\mathcal{P} = \begin{pmatrix} 1 & (z_0 - z_c) & (z_0 - z_c)^2 & ... & (z_0 - z_c)^M \\ 1 & (z_1 - z_c) & (z_1 - z_c)^2 & ... & (z_1 - z_c)^M \\ ... & ... & ... & ... & ... \\ 1 & (z_{N-1} - z_c) & (z_{N-1} - z_c)^2 & ... & (z_{N-1} - z_c)^M \end{pmatrix}, \quad \boldsymbol{a} = \begin{pmatrix} a_0 \\ a_1 \\ ... \\ a_M \end{pmatrix} \tag{2}$$

$\mathcal{P}$ is a (N $\times$ (M+1)) matrix, write a function that build the matrix $\mathcal{P}$ using the redshift column of the data file.

4) The variance of the noise on the measurement is given by $\Sigma_{ii} = \sigma_0^2 + \sigma_1^2(1+z_i)^3$, with $\sigma_0 = \sqrt{3}$ and $\sigma_1 = 1$ the correlation between the different measurements is null but for adjacent redshifts, in that case it takes the constant value of 0.1

$$\rho_{ij} = \Sigma_{ij}/\sqrt{\Sigma_{ii}\Sigma_{jj}} = 0.1 \quad \text{only if} \quad |i-j| = 1 \tag{3}$$

Write a function that construct the covariance matrix, check that the square root of its diagonal do agree with the error column of the data file, then evaluate the $\chi^2 = (\boldsymbol{d} - \mathcal{P}\boldsymbol{a})^T \Sigma^{-1} (\boldsymbol{d} - \mathcal{P}\boldsymbol{a})$ for the set of parameters $a_0 = 8$, $a_1 = 3$, $a_2 = 0.4$, $a_3 = -0.02$, you should get $\chi^2 \sim 122.9827$, is this set of parameter a good fit to the data ?

5) in our model, the different parameters a_k enter linearly in the model, we can analytically derive the Maximum Likelihood (ML) solution, write down an expression for $\ln(\mathcal{L}(\boldsymbol{d}|\boldsymbol{a}))$, and show that its extremum is obtained when

$$\hat{\boldsymbol{a}} = (\mathcal{P}^T \Sigma^{-1} \mathcal{P})^{-1} \mathcal{P}^T \Sigma^{-1} \boldsymbol{d} \tag{4}$$

tip: ask for $\nabla_{\boldsymbol{a}} \ln\mathcal{L}(\boldsymbol{d}|\boldsymbol{a}) = 0|_{\boldsymbol{a}=\hat{\boldsymbol{a}}}$ This solution is called the Maximum Likelihood estimate of $\boldsymbol{a}$ and is denoted $\hat{\boldsymbol{a}}$.

6) Show (analytically) that the expectation value of the ML estimator and its covariance are given by

$$\langle \hat{\boldsymbol{a}} \rangle = \boldsymbol{a}, \quad \text{Cov}(\hat{\boldsymbol{a}}) \quad = \quad (\mathcal{P}^T \Sigma^{-1} \mathcal{P})^{-1} \tag{5}$$

7) Write a function that evaluate the maximum solution corresponding to the provided data file, print the value of the $\hat{a}_k$ parameters and their associated uncertainties.

8) Plot the best fit model with respect to the data point, estimate the $\chi^2 = (\boldsymbol{d} - \mathcal{P}\hat{\boldsymbol{a}})^T \Sigma^{-1} (\boldsymbol{d} - \mathcal{P}\hat{\boldsymbol{a}})$ and p-value of the best fit model.

9) Compute the correlation matrix associated with $\text{Cov}(\hat{\boldsymbol{a}})$ and plot it, you should see that the parameters are pretty correlated.

2 Monte Carlo verification of the estimator

A Monte Carlo method designs a numerical method using random processes to estimate numerical quantities.

We have written an analytic formula for the maximum likelihood solution to our parameter estimation problem, here we would like to check numerically that the estimator is indeed unbiased and that the analytic covariance formula is indeed correct
We will draw simulations of the data, there is two steps: 1. Perform a Choleski decomposition of the noise covariance matrix $\Sigma = LL^T$, 2. Generate vectors of random gaussian number, $\boldsymbol{u}$, of size N with mean 0 and covariance δ_{ij}. A noise simulation will be given by: $\boldsymbol{n}^{simu} = L\boldsymbol{u}$

1) Choose a set of parameters $\boldsymbol{a}$ and generate a bunch of simulation of the data

$$\boldsymbol{d}_{\text{ sim 1}} \quad = \quad \mathcal{P}\boldsymbol{a} + \boldsymbol{n}^{\text{sim1}} \tag{6}$$

$$\ldots.$$

$$\ldots.$$

$$\boldsymbol{d}_{\text{ sim 1000}} \quad = \quad \mathcal{P}\boldsymbol{a} + \boldsymbol{n}^{\text{sim1000}} \tag{7}$$

for each simulation estimate the maximum solution for $\boldsymbol{a}$, create a list of estimated $\{\hat{\boldsymbol{a}}_{\text{sim1}}, \hat{\boldsymbol{a}}_{\text{sim2}}, ..., \hat{\boldsymbol{a}}_{\text{sim1000}}\}$

2) Compute the numerical mean and covariances of the list of estimated maximum likelihood parameters, does the mean match your input parameters ? does the numerical covariance matches your analytical formula ?

3) For each simulation compute the associated χ^2 e.g, $\chi^2_{\text{sim1}} = (\boldsymbol{d}_{\text{sim1}} - \mathcal{P}\hat{\boldsymbol{a}}_{\textbf{sim1}})^T \Sigma^{-1} (\boldsymbol{d}_{\text{sim1}} - \mathcal{P}\hat{\boldsymbol{a}}_{\textbf{sim1}})$ and p-value.

4) Display the χ^2 distribution of the simulations (plot an histogram with the values of $\{\chi^2_{\text{sim1}}, ..., \chi^2_{\text{sim1000}}\}$ and compare with the expected distribution of χ^2 with N-M DoF, do the same with the p-value, what is the expected distribution of p-value ?

3 Monte Carlo Markov chains

We have been able to solve this problem analytically, this is due to the fact that the parameters enter linearly in the model, for a more general case we will have to use different methods. A standard method is Monte Carlo Markov chains, the idea is to design an algorithm that draw samples from the posterior distribution of the parameters. In this section we will code our own MCMC algorithm and redo the parameter estimation of our model.

A popular MCMC algorithm is the Metropolis Hasting algorithm, schematically

(I_1) Define a function $f(\boldsymbol{a}) = \mathcal{L}(\boldsymbol{d}|\boldsymbol{a})P(\boldsymbol{a})$

(I_2) Choose a starting point for your parameters $\boldsymbol{a}_0$

(I_3) Choose a proposal $g(\boldsymbol{a}'|\boldsymbol{a}_0)$, it is the function that will help us move in the parameter space, a usual choice is a multivariate distribution centered on $\boldsymbol{a}_0$ with covariance $\Sigma = \frac{2.4^2}{D}\mathrm{Cov(a)}$

$$g(\boldsymbol{a}'|\boldsymbol{a}_0) = \frac{1}{(2\pi)^{D/2}\sqrt{|\Sigma|}}\exp -\frac{1}{2}\left[(\boldsymbol{a}' - \boldsymbol{a}_0)\right]^T \Sigma^{-1}\left[(\boldsymbol{a}' - \boldsymbol{a}_0)\right] \tag{8}$$

(I_4) Randomly draw a candidate $\boldsymbol{a}'$ from the proposal

(I_5) Compute the acceptance rate $\alpha = \frac{f(\boldsymbol{a}')}{f(\boldsymbol{a_0})}$

(I_6) Draw a uniform random number between 0 and 1

$$u \leq \alpha \quad \text{accept the candidate} \quad \boldsymbol{a_1} = \boldsymbol{a}' \tag{9}$$
$$u > \alpha \quad \text{reject the candidate} \quad \boldsymbol{a_1} = \boldsymbol{a_0} \tag{10}$$

(I_7) Iterate

The idea of the algorithm is the following, we explore the parameter space, if we move toward a state with higher probability, the move is always accepted. If we move towards a less probable state, the move can be either accepted or rejected, the rejection depends on the ratio of probability between the two points in parameter space. The random walk preferentially explores the parameter space where the posterior is high but occasionally moves in region with lower probability, this helps avoiding being trapped in local maxima.

1) Write down a version of the Metropolis Hasting algorithm, and draw 10^4 samples of $\boldsymbol{a}$.

2) Plot the corresponding chains. At the beginning of the chains we will notice a "burn in" phase, a phase where parameters value varies by a lot, this should be discarded for later analysis.

3) Plot an histogram of the chains, this represents the posterior distribution of our estimated parameters, compare this with the analytical estimate.

4) In practice you won't have to use your own MCMC sampler since many of them are already publicly available, a popular one in cosmology is cobaya an example of how to use cobaya is given below:

```
def cobaya_mcmc(z, data, inv_data_cov, z_c, min_list, max_list,
          chain_name, Rminus1_stop=0.003, Rminus1_cl_stop=0.05):

    from cobaya.run import run

    def log_prob(a0, a1, a2, a3):
        params = [a0, a1, a2, a3]
        model = fp1.generate_model(z, params, z_c)
        res = data - model
        return -0.5 * res @ inv_data_cov @ res

    info = {
        "likelihood": {"my_like": log_prob},
        "params": {
            "a0": {"prior": {"min": min_list[0], "max": max_list[0]}, "latex": "a_{0}"},
            "a1": {"prior": {"min": min_list[1], "max": max_list[1]}, "latex": "a_{1}"},
```

```
            "a2": {"prior": {"min": min_list[2], "max": max_list[2]}, "latex": "a_{2}"},
            "a3": {"prior": {"min": min_list[3], "max": max_list[3]}, "latex": "a_{3}"},
        },
        "sampler": {
            "mcmc": {
                "max_tries": 10 ** 8,
                "Rminus1_stop": Rminus1_stop,
                "Rminus1_cl_stop": Rminus1_cl_stop,
            }
        },
        "output": f"{chain_name}",
        "force": True,
        "debug": False,
    }


    updated_info, sampler = run(info)
```

you can then plot the result using getDist

```

def plot_cobaya_chain(chain_name, params):
    from getdist.mcsamples import loadMCSamples
    import getdist.plots as gdplt

    samples = loadMCSamples( f"{chain_name}", settings = {"ignore_rows": 0.5})
    gdplot = gdplt.get_subplot_plotter()
    gdplot.triangle_plot(samples, params, filled = True, title_limit=1)
    plt.savefig(f"{chain_name}.png", dpi = 300)
    plt.clf()
    plt.close()
```