STATISTICAL SIGNAL AND DATA PROCESSING THROUGH APPLICATIONS
Lecturer: Dr. Andrea Ridolfi
Assistant: Sepand Kashani
Spring Semester 2022

Solutions 6

2022.04.01

Solution 1. Annihilating filter vs. MUSIC

- (a) and (b): In Python we have the following code:

```
import matplotlib.pyplot as plt
import numpy as np
import numpy.random as npr
import scipy.linalg as spl


def gen_samples(
    a: np.ndarray,
    f: np.ndarray,
    N_realization: int,
    N_sample: int,
    random_phase: bool,
) -> np.ndarray:  # (N_realization, N_sample)
    X = np.zeros((N_realization, N_sample), dtype=np.complex128)
    for i in range(N_realization):
        n = np.arange(N_sample).reshape((-1, 1))
        x_ = 2 * np.pi * f * n
        if random_phase:
            rng = npr.default_rng()
            phase = rng.uniform(0, 2 * np.pi, size=len(f))
            x_ += phase
        X[i] = np.exp(1j * x_).sum(axis=-1)
    return X


def noisy(signal: np.ndarray, var: float) -> np.ndarray:
    rng = npr.default_rng()
    params = dict(loc=0, scale=0.5 * np.sqrt(var), size=signal.shape)
    noise = rng.normal(**params) + 1j * rng.normal(**params)
    y = signal + noise
    return y


def AF_method(Y: np.ndarray, K: int):
    # Get best filter which annihilates all realizations
    N_realization, N_sample = Y.shape
```

```
    M1 = np.zeros((N_realization, K + 1, K), dtype=np.complex128)
    b1 = np.zeros((N_realization, K + 1), dtype=np.complex128)
    for i, y in enumerate(Y):
        y = y[: 2 * K + 1]
        M1[i] = spl.toeplitz(c=y[K - 1 : -1], r=y[:K][::-1])
        b1[i] = -y[K:]
    h, *_ = spl.lstsq(
        np.concatenate(M1, axis=0),
        np.concatenate(b1, axis=0),
    )
    h = np.r_[1, h]
    w_ = np.fmod(np.angle(np.roots(h)) + 2 * np.pi, 2 * np.pi)
    f = np.sort(w_ / (2 * np.pi))

    # Get amplitudes (per realization)
    a = np.zeros((N_realization, K), dtype=np.double)
    M2 = np.exp(1j * np.arange(N_sample).reshape((-1, 1)) * f)
    for i, y in enumerate(Y):
        b2 = y
        a_, *_ = spl.lstsq(M2, b2)
        a[i] = np.abs(a_)
    return np.mean(a, axis=0), f


def MUSIC_method(y: np.ndarray, K: int, var: float):
    # Estimate correlation matrix
    N_realization, N_sample = y.shape
    y_ = y.reshape((N_realization, N_sample, 1))
    R = np.mean(y_ * y_.transpose((0, 2, 1)).conj(), axis=0)

    # Get noise spectrum
    D, V = spl.eigh(R)
    Dn, Vn = D[:-K], V[:, :-K]

    # Find directions of minimum energy when projecting signal into noise space
    f = np.linspace(0, 1, 1000)
    E = np.exp(1j * 2 * np.pi * f.reshape((-1, 1)) * np.arange(N_sample))
    residual = spl.norm(E.conj() @ Vn, axis=-1)
    i = np.argsort(residual)[:K]
    f = np.sort(f[i])

    # Estimate amplitudes.
    # Straightforward application of E @ (R - s^2 I) @ E.H leads to numerical issues due to
    # We therefore transform equations a bit to overcome these issues.
    D_ = np.clip(D - var, 0, None)
    E = np.exp(1j * 2 * np.pi * f.reshape((-1, 1)) * np.arange(N_sample))
    A_ = E @ (V * np.sqrt(D_))
    a = np.sqrt(np.sum(A_ * A_.conj(), axis=-1).real)
    return a, f
```

```
if __name__ == "__main__":
    a_gt = np.r_[1, 2, 3]
    f_gt = np.r_[0.2, 0.3, 0.4]
    K = len(f_gt)

    a_AF, f_AF = dict(), dict()  # noise_var -> seq[float]
    a_MU, f_MU = dict(), dict()  # noise_var -> seq[float]
    noise_var = np.r_[1, 4]
    for n in noise_var:
        X = gen_samples(
            a=a_gt,
            f=f_gt,
            N_realization=20,
            N_sample=250,
            random_phase=True,
        )
        Y = noisy(X, var=n)

        a_AF[n], f_AF[n] = AF_method(Y, K)
        a_MU[n], f_MU[n] = MUSIC_method(Y, K, var=n)

    fig, ax = plt.subplots()
    ax.plot(f_gt, a_gt, ".", label=f"Ground Truth")
    for n in noise_var:
        ax.plot(f_AF[n], a_AF[n], "o", label=f"AF method, sigma^2 = {n:.02f}")
        ax.plot(f_MU[n], a_MU[n], "x", label=f"MUSIC method, sigma^2 = {n:.02f}")
    ax.set_xlabel("f")
    ax.set_ylabel("a")
    ax.legend()
    fig.show()
```

- (c) The annihilating filter method can be used in the same way as for the previous case. The MUSIC method can be used as well. The only difference would be when estimating the moduli of the amplitudes from the lecture notes as the covariance matrix A of $\mathbf{X}^{K1}[n]$ changes.

Solution 2.

(a) The goal is to estimate the PSD $S_X(e^{j\omega})$ from samples $X[n]$ of the piezo-electric sensor. Since the PSD is assumed smooth, it can be modeled as the spectrum of an AR process of yet-to-be-determined order M. We therefore seek an analysis filter $P(z) = \sum_{k=0}^{M} p_k z^{-k}$ such that $P(z)X[n] = W[n]$, where $W[n]$ denotes some i.i.d. noise process of unknown variance σ^2. Provided $P(z)$ and σ^2 can be estimated, then the smooth spectrum will be given by $S_X(e^{j\omega}) = \sigma^2/|P(e^{j\omega})|^2$.

Using the Yule-Walker equations

$$\sum_{k=0}^{M} p_k R_X[n-k] = \sigma^2 \delta[n],$$

one can solve the following linear system to estimate $P(z)$ and σ^2: [1] [2]

$$\begin{bmatrix} R_X[0] & \cdots & R_X[-M] & -1 \\ R_X[1] & \cdots & R_X[1-M] & 0 \\ \vdots & \ddots & \vdots & \vdots \\ R_X[M+1] & \cdots & R_X[1] & 0 \end{bmatrix} \begin{bmatrix} p_0 \\ \vdots \\ p_M \\ \sigma^2 \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{1}$$

System (1) should be solved for increasing orders M until the loss function $\hat{M} = \arg\min_{M \geq 0} \|P(z)X[n]\|^2$ is minimized.

(b) If the smooth-spectrum assumption is invalid, then estimating $S_X(e^{j\omega})$ via a parametric AR-process will require a large value M.

(c) The periodogram is a non-parametric method. Its main drawback is its limited spectral resolution and high variance. Since mechanical stress in the mast is probably due to resonance effects, a high-precision estimate of the spectrum is desired, hence our choice of a parametric approach.

Solution 3.

- (a) First, it can be observed that since $X[n]$ is a constant ($X[n] = A$) and $W[n]$ is iid with distribution $\mathcal{N}(0,1)$, the process $Y[n]$ is also iid, with distribution $\mathcal{N}(A,1)$. In other words, knowing the value of A, the joint distribution of N samples $Y[0], \ldots, Y[N-1]$ is given by

$$\begin{aligned} p(Y[1], \ldots, Y[n]|A) &\overset{iid}{=} \prod_{i=0}^{N-1} p(Y[i]|A) \\ &= \prod_{i=0}^{N-1} \frac{1}{\sqrt{2\pi}} e^{-\frac{(Y[i]-A)^2}{2}}. \end{aligned}$$

Furthermore, according to the definition, the likelihood function associated to the probability density function of the observed data is given by

$$\begin{aligned} \mathcal{L}(A) &\triangleq p(Y[1], \ldots, Y[n]|A) \\ &= \prod_{i=0}^{N-1} \frac{1}{\sqrt{2\pi}} e^{-\frac{(Y[i]-A)^2}{2}}, \end{aligned}$$

and its logarithm, the *log-likelihood*, is

$$\begin{aligned} \mathcal{L}^*(A) &= \ln \mathcal{L}(A) \\ &= -\frac{N}{2} \ln 2\pi - \sum_{i=0}^{N-1} \frac{(Y[i]-A)^2}{2}. \end{aligned}$$

[1] (1) can be re-written in various forms. The advantage of (1) is that its nullspace fully determines the solution space.

[2] Recall that $R_X[k] = X[n] \star X[-n]$.

Maximizing the likelihood function is equivalent to maximising the log-likelihood, which can be done by setting to zero the partial derivatives of $\mathcal{L}^*(A)$ with respect to A. It gives:

$$\begin{aligned} \frac{\partial \mathcal{L}^*(A)}{\partial A} = 0 \quad &\Leftrightarrow \quad 2\sum_{i=0}^{N-1} \frac{Y[i]-A}{2} = 0 \\ &\Rightarrow \quad \hat{A} = \frac{1}{N}\sum_{i=0}^{N-1} Y[i]\,; \end{aligned}$$

- (b) For the data provided, the mean is 5.71.

- (c) and (d) Our likelihood from the first part of the exercise have been calculated assuming that A is a parameter, so we have $p(Y[i]|A)$. We are now interested in posterior distribution, that is:

$$p(A|Y[i]).$$

We can calculate it using Bayes rule:

$$p(A|Y[i])p(Y[i]) = p(Y[i], A) = p(Y[i]|A)p(A)$$

In a more elaborate way, we can write it as:

$$\text{posterior} \times \text{evidence} = \text{likelihood} \times \text{prior}.$$

Since evidence is probability of data, it's constant for the given data and we can ignore it in the optimisation. We are then interested in maximising:

$$\begin{aligned} p(Y[1], \ldots, Y[n]|A)p(A) &= \left(\prod_{i=0}^{N-1} p(Y[i]|A)\right) p(A) \\ &= \left(\prod_{i=0}^{N-1} \frac{1}{\sqrt{2\pi}} e^{-\frac{(Y[i]-A)^2}{2}}\right) \frac{1}{\sqrt{2\pi\sigma_A^2}} e^{-\frac{A^2}{2\sigma_A^2}} \end{aligned}$$

similarly like with likelihood optimisation, it's easier to optimise the logarithm:

$$-\frac{N}{2}\ln 2\pi - \frac{1}{2}\ln 2\pi\sigma_A^2 - \sum_{i=0}^{N-1} \frac{(Y[i]-A)^2}{2} - \frac{A^2}{2\sigma_A^2},$$

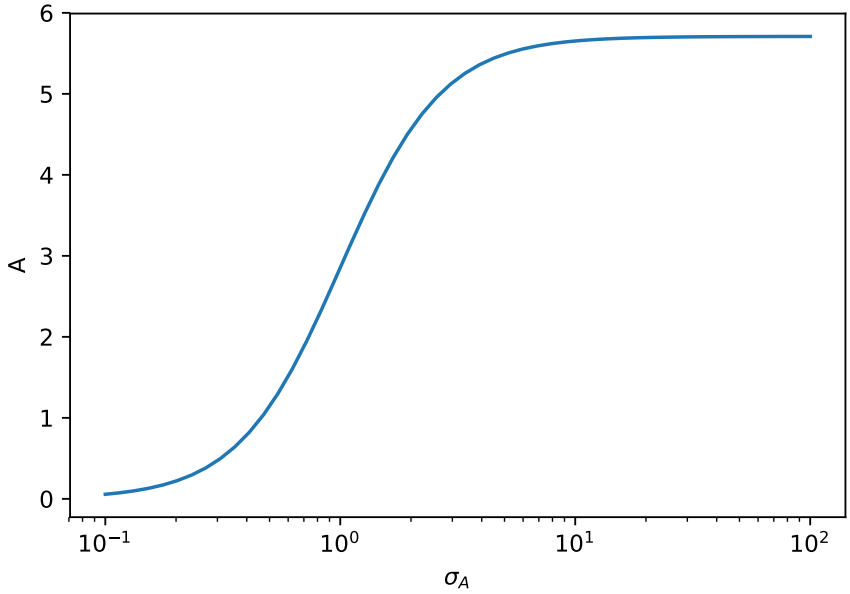
we want it's derivative to be equal to zero:

$$-\sum_{i=0}^{N-1}(Y[i]-A) + \frac{A}{\sigma_A^2} = 0 \tag{2}$$

And maximum a posterior estimator is:

$$\hat{A} = \frac{\sum_{i=0}^{N-1} Y[i]}{\left(N + \frac{1}{\sigma_A^2}\right)} \tag{3}$$

- (e) Value of the estimator $\hat{A}$ depending on the variance of the prior σ_A^2. Small variance of the prior means we have a strong belief that A should be close to zero, and this belief is reflected in the value of the estimator, see the plot below.


6
5
4
3
2
1
0
A
10^{-1}
10^{0}
10^{1}
10^{2}
σ_A