**Least Squares Estimation, Filtering, and Prediction**

- Principle of least squares
- Normal equations
- Weighted least squares
- Statistical properties
- FIR filters
- Windowing
- Combined forward-backward linear prediction
- Narrowband Interference Cancellation

**Motivation**

- If the second-order statistics are known, the optimum estimator is given by the normal equations
- In many applications, they aren’t known
- Alternative approach is to estimate the coefficients from observed data
- Two possible approaches
  - Estimate required moments from available data and build an approximate MMSE estimator
  - Build an estimator that minimizes some error functional calculated from the available data

---

**MMSE versus Least Squares**

- Recall that MMSE estimators are optimal in expectation across the ensemble of all stochastic processes with the same second order statistics
- Least squares estimators minimize the error on a given block of data
  - In signal processing applications, the block of data is a finite-length period of time
- No guarantees about optimality on other data sets or other stochastic processes
- If the process is ergodic and stationary, the LSE estimator approaches the MMSE estimator as the size of the data set grows
  - This is the first time in this class we’ve discussed estimation from data
  - First time we need to consider ergodicity

---

**Principle of Least Squares**

- Will only discuss the sum of squares as the performance criterion
  - Recall our earlier discussion about alternatives
  - Essentially, picking sum of squares will permit us to obtain a closed-form optimal solution
- Requires a data set where both the inputs and desired responses are known
- Recall the range of possible applications
  - Plant modeling for control (system identification)
  - Inverse modeling/deconvolution
  - Interference cancellation
  - Prediction
Recalling the Book’s Notation

- \( y(n) \in \mathbb{C}^{1 \times 1} \) is the target or desired response
- \( x_k(n) \) represent the inputs
- These may be of several types
  - Multiple sensors, no lags: \( x(n) = [x_1(n), x_2(n), \ldots, x_M(n)]^T \)
  - Lag window: \( x(n) = [x(n), x(n-1), \ldots, x(n-M+1)]^T \)
  - Combined
- Data sets consists of values over the time span \( 0 \leq n \leq N-1 \)
- Boldface is now used for vectors and matrices
- The coefficients are represented as \( c(n) \)

Change in Notation

In a trade of elegance and simplicity for inconsistency, I’m going to break with some of the book’s notational conventions

- Notes: \( \hat{y}(n) \triangleq \sum_{k=1}^{M} c_k(n)x_k(n) = c^T(n)x(n) \)

- Book: \( \hat{y}(n) \triangleq \sum_{k=1}^{M} c_k^*(n)x_k(n) = c^H(n)x(n) \)

- In the case that \( c \) is real, they are consistent
- Rationale
  - The inner product notation leads to unnecessary complications in the notation
  - Most books use the same notation that the notes use
  - Leads to a symmetry: \( c^T(n)x(n) = x^T(n)c(n) \)

Definitions

Estimate:
\[
\hat{y}(n) \triangleq \sum_{k=1}^{M} c_k(n)x_k(n) = c^T(n)x(n)
\]

Estimation error:
\[
e(n) \triangleq y(n) - \hat{y}(n) = y(n) - c(n)x(n)
\]

Sum of squared errors:
\[
E_e \triangleq \sum_{n=0}^{N-1} |e(n)|^2
\]

- Coefficient vector \( c(n) \) is typically held constant over the data window, \( 0 \leq n \leq N-1 \)
- Contrast with adaptive filter approach
- The coefficients \( c \) that minimize \( E_e \) are called the linear LSE estimator

Matrix Formulation

\[
\begin{bmatrix}
  e(0) \\
e(1) \\
  \vdots \\
e(N-1)
\end{bmatrix}
= \begin{bmatrix}
y(0) \\
y(1) \\
\vdots \\
y(N-1)
\end{bmatrix} - \begin{bmatrix}
x_1(0) & x_2(0) & \cdots & x_M(0) \\
x_1(1) & x_2(1) & \vdots & x_M(1) \\
\vdots & \vdots & \ddots & \vdots \\
x_1(N-1) & x_2(N-1) & \cdots & x_M(N-1)
\end{bmatrix}
\begin{bmatrix}
c_1 \\
c_2 \\
\vdots \\
c_M
\end{bmatrix}
\]

\[
e = y - Xc
\]

where
\[
e \triangleq [c(0) \ e(1) \ \ldots \ \ e(N-1)]^T \quad \text{error data vector} \ (N \times 1)
\]
\[
y \triangleq [y(0) \ y(1) \ \ldots \ y(N-1)]^T \quad \text{desired response vector} \ (N \times 1)
\]
\[
X \triangleq [x^T(0) \ x^T(1) \ \ldots \ x^T(N-1)]^T \quad \text{input data matrix} \ (N \times M)
\]
\[
c \triangleq [c_1 \ c_2 \ \ldots \ c_M]^T \quad \text{combiner parameter vector} \ (M \times 1)
\]

Note: my definitions differ from the book by a conjugate factor *
Block Processing

- LSE estimators can be used in block processing mode
  - Take a segment of \( N \) input-output observations, say \( n_1 \leq n \leq n_1 + N - 1 \)
  - Estimate the coefficients
  - Increment the temporal location of the block to \( n_1 + N_0 \)
- The blocks overlap by \( N - N_0 \) samples
- Reminiscent of Welch’s method of PSD estimation
- Useful for parametric time-frequency analysis
- In most other nonstationary applications, adaptive filters are usually used instead

Normal Equations

\[
E_e = ||e||^2 = e^H e = (y - Xc)^H(y - Xc) = (y^H - c^H X^H)(y - Xc) = y^H y - c^H X^H y - y^H X c + c^H X^H X c
\]

- \( E_e \) is a nonlinear function of \( y, X, \) and \( c \)
- \( E_e \) is a quadratic function of each of these components
- \( E_e \) is the sum of squared errors
  - Energy of the error signal over the interval \( 0 \leq n \leq N - 1 \)
- If we take the average squared errors (ASE), we have an estimate of the mean square error = the estimated power of the error
\[
\hat{P}_e = \frac{1}{N} E_e = \frac{1}{N} \sum_{n=0}^{N-1} |e(n)|^2
\]
Recall the estimate from ECE 538/638

With the FFT

\[ n(n) \text{ comes from a stationary time series, we can speed this up} \]

This should look familiar...

If we assume \( \hat{R} > 0 \), we can complete the square (fourth time!)

\[ \hat{P}_c(e) = \hat{P}_y + c^H \hat{R}(c - \hat{R}^{-1} \hat{d}) - \hat{d}^H \hat{R}^{-1} \hat{d} + \hat{R}^H \hat{R}^{-1} \hat{d} \]

\[ = \hat{P}_y + (c^H \hat{R}^H \hat{R}^{-1} + \hat{d}^H \hat{R}^{-1} \hat{d} - \hat{d}^H \hat{R}^{-1} \hat{d} \]

\[ = \hat{P}_y - \hat{d}^H \hat{R}^{-1} \hat{d} + (c - \hat{R}^{-1} \hat{d})^H \hat{R} \hat{R}^{-1} (\hat{R} - \hat{d}) \]

\[ = \hat{P}_{ls} + (\hat{R} - \hat{d})^H \hat{R}^{-1} (\hat{R} - \hat{d}) \]

If we replaced the sample average \( \frac{1}{N} \sum_{n=0}^{N-1} (\cdot) \) with expectation, \( E[\cdot] \), each of these terms would be the mean ensemble value rather than the sample average estimate.

### Normal Equation Components

- Let us define
  
  \[ \hat{P}_c = \frac{1}{N} \sum_{n=0}^{N-1} |e(n)|^2 \]

  \[ \hat{P}_y = \frac{1}{N} \sum_{n=0}^{N-1} |y(n)|^2 \]

- Average correlation:
  \[ \hat{R} = \frac{1}{N} X^H X = \frac{1}{N} \sum_{n=0}^{N-1} x(n)x^T(n) \]

- Average cross-correlation:
  \[ \hat{d} = \frac{1}{N} X^H y = \frac{1}{N} \sum_{n=0}^{N-1} x(n)y(n) \]

- If we replaced the sample average \( \frac{1}{N} \sum_{n=0}^{N-1} (\cdot) \) with expectation, \( E[\cdot] \), each of these terms would be the mean ensemble value rather than the sample average estimate.

### Least Squares Estimate

\[ \hat{P}_c(e) = \hat{P}_y - \hat{d}^H \hat{R}^{-1} \hat{d} + (\hat{R} - \hat{d})^H \hat{R}^{-1} (\hat{R} - \hat{d}) \]

So the least squares estimate and minimum average squared error are given by

\[ c_{ls} = \hat{R}^{-1} \hat{d} \]

\[ = (X^H X)^{-1} X^H y \]

\[ \hat{P}_{ls} = \frac{1}{N} \sum_{n=0}^{N-1} |e(n)|^2 \]

\[ = \hat{P}_y - \hat{d}^H c_{ls} \]

- Both the LSE and MSE criteria are quadratic functions of the coefficient vector \( c \)

### Relating LSE and MMSE Estimate

Then

\[ \hat{P}_c = \frac{1}{N} \left( y^H y - c^H X^H y - y^H X c + c^H X^H X c \right) \]

\[ = \hat{P}_y - c^H \hat{d} - \hat{d}^H c + c^H \hat{R} c \]

### Computational Issues

- Typically \( M \ll N \)
- The most computationally expensive operation is calculating the input correlation matrix!
- If \( x(n) \) comes from a stationary time series, we can speed this up with the FFT
  - Recall the estimate from ECE 538/638
  - More later
**Projection Theorem: Estimator**

- The projection theorem holds
  - Therefore the projection of \( y \) onto the column space of \( X \) minimizes the length of the residual vector

\[
\|e\|^2 = \sum_{n=0}^{N-1} |y(n) - cx(n)|^2
\]

- The error vector is also orthogonal to the observations

\[
\langle X, y - Xc_{ls} \rangle = X^H y - X^H Xc_{ls} = N\bar{d} - N \bar{R}c_{ls} = 0
\]

- Thus we have another way of obtaining the normal equations

\[
\bar{R}c_{ls} = \bar{d}
\]

**Projection Theorem: LSE**

- Further, by orthogonality we have

\[
\|y\|^2 = \|e\|^2 + \|\bar{y}\|^2
\]

\[
\|e\|^2 = \|y\|^2 - \|\bar{y}\|^2
\]

\[
P_{ls} = P_y - c_{ls}^H X^H X c_{ls}
\]

\[
P_{ls} = P_y - \bar{d}^H \bar{R}^{-1} \bar{R} c_{ls}
\]

\[
P_{ls} = P_y - \bar{d}^H c_{ls}
\]

**Geometric Derivation**

\[e = y - Xc\]

- I was disparaging of the geometric interpretation for the MSE case
- It makes a lot more sense for the LSE case
- We are trying to estimate an \( N \) dimensional vector as a linear combination of the \( M \) columns of \( X \)
- The orthogonal projection of \( y \) onto the \( M \) dimensional subspace minimizes the error

**Inner Product Definition**

- In this case an inner product can be defined as

\[
\langle \bar{x}_i, \bar{x}_j \rangle \triangleq \bar{x}_i^H \bar{x}_j = \sum_{n=0}^{N-1} x_i(n)x_j^*(n)
\]

\[
\|\bar{x}_i\|^2 \triangleq \bar{x}_i^H \bar{x}_i = \sum_{n=0}^{N-1} |x_i(n)|^2
\]

- Can easily show that this has all of the required properties of an inner product
Weighted Least Squares Comments

- Weighted least squares can be applied with any weighting matrix \( W^2 \) that is positive definite
  - \( W^2 \) does not have to be diagonal
  - All positive definite matrices have a square root \( W^2 = W^H W \)
  - The square root is not unique
- Useful for
  - Windowing parametric estimators
  - Accounting for correlated observation noise

Statistical Models

\[
y = Xc_o + v
\]
\[
c_{ls} = (X^H X)^{-1} X^H y = (X^H X)^{-1} X^H X c_o + (X^H X)^{-1} X^H v = c_o + (X^H X)^{-1} X^H v
\]

- Most of the statistical properties require a statistical model of how the data was generated
- This idea is similar to the state space model used by the Kalman filter
- The linear statistical model used here is more general (no assumption of state dynamics)
- Some of the properties don’t hold when the model is not accurate

Uniqueness

- As with the MMSE case we have
  - \( c_{ls} \) is unique if \( X \) has full column rank and \( M \leq N \)
  - Otherwise there are infinite, equivalent solutions with the same LSE
- Regardless the LSE estimate \( \hat{y}(n) = c^T(n)x(n) \) is the same
- Full column rank is equivalent to the requirement that \( \hat{R} \) be positive definite

Weighted Least Squares

- Suppose some errors are more significant than others
- A closely related problem is weighted least squares
\[
E_c = \sum_{n=0}^{N-1} w_n^2 |y(n) - c^T x(n)|^2
= (y - Xc)^H W^2 (y - Xc)
= (Wy - WXc)^H(Wy - WXc)
\]
If we define
\[
\hat{y} \triangleq Wy \quad \hat{X} \triangleq WX
\]
we have the original LSE problem
\[
E_c = \sum_{n=0}^{N-1} |\hat{y}(n) - c^T \hat{x}(n)|^2 = (\hat{y} - \hat{X}c)^H(\hat{y} - \hat{X}c)
\]
Deterministic Case: Estimator Properties

\[ y = Xc_o + v \]

- \( y \) is a vector of observed measurements
- \( X \) is well conditioned
  - Deterministic: \( X \) has full column rank
  - Stochastic: \( E[X^H X] \) is positive definite
- \( c_o \) is considered the "true" parameter vector
  - The notational overlap with MSE estimation notes is intentional
- \( v \) is a vector of random noise or "errors"
  - Note my notation differs from the text, which used \( e_o \)
  - All of the elements of \( X \) are independent with \( v \)
  - \( v \) has zero mean: \( E[v] = 0 \)
  - The elements of \( v \) are uncorrelated: \( E[vv^H] = \sigma_v^2 I \)

\[ c_{ls} = c_o + (X^H X)^{-1} X^H v \]

- The estimator \( c_{ls} \) is unbiased
  \[ E[c_{ls}] = c_o \]
- The estimator covariance matrix is
  \[ \Lambda_{ls} = E[(c_{ls} - c_o)(c_{ls} - c_o)^H] = \sigma_v^2 (X^H X)^{-1} = \frac{\sigma_v^2}{N} \hat{R}^{-1} \]
  - The diagonal elements of \( \Lambda_{ls} \) contain the variance of the elements of \( c_{ls} \)
  - If \( v \) is Gaussian, we can construct exact confidence intervals!
  - Difference in definition of \( \hat{R} \) make the dependence on \( N \) explicit
  - Covariance is inversely proportional to the number of observations

\[ P \triangleq X(X^H X)^{-1} X^H \]

\[ c_{ls} = c_o + (X^H X)^{-1} X^H v \]

\[ e = y - \hat{y} = y - Xc_{ls} = y - X(X^H X)^{-1} X^H y = Xc_o + v - X(X^H X)^{-1} X(Xc_o + v) = v - Pv = (I - P)v \]
Deterministic Case: Error Variance

\[
E_e = e^H e \\
= v^H (I - P)^H (I - P) v \\
= v^H (I - P)^H (I - P) v \\
= v^H (I - P) v - v^H (P - PP) v \\
= v^H (I - P) v \\
= \text{trace}[v^H (I - P) v] \\
= \text{trace}[(I - P) v v^H] \\
E[E_e] = \text{trace}[(I - P) \sigma_v^2 I] \\
= \sigma_v^2 \text{trace}[I - P] \\
\]

Properties of the trace[·] operator

\[
\text{trace}[AB] = \text{trace}[BA] \\
\text{trace}[A + B] = \text{trace}[A] + \text{trace}[B] \\
\]

Deterministic Case: Other, Unproved Properties

\[
y = X c_o + v \\
c_{ls} = c_o + (X^H X)^{-1} X^H v \\
\]

- The weighted LSE estimate is also unbiased
- If the error covariance is not diagonal, \( R_v \neq \sigma_v^2 I \), the optimal estimator is obtained by setting \( W^2 = R_v^{-1} \)
- In both cases, the LSE estimator is the best linear unbiased estimator (BLUE)
  - Of all the unbiased estimators, this one has the smallest variance
- If \( v \) is normally distributed, the LSE estimator is also the maximum likelihood estimator

Stochastic Case: Properties

\[
y = X c_o + v \\
c_{ls} = c_o + (X^H X)^{-1} X^H v \\
\]

- Model assumptions
  - \( X \) and \( v \) are statistically independent
  - Merely uncorrelated is insufficient in this case
- The estimator is still unbiased in this case
- The covariance of \( c_{ls} \) is given by
  \[
  \Lambda_{ls} \triangleq E \left[ (c_{ls} - c_o)(c_{ls} - c_o)^H \right] = \sigma_v^2 E[(X^H X)^{-1}] \\
  \]
- Proofs are in the text
### Least Squares Filters

\[ e(n) = y(n) - \sum_{k=0}^{M-1} h(k)x(n - k) = y(n) - e^T x(n) \]

\[ x(n) \triangleq [x(n) \quad x(n-1) \quad \ldots \quad x(n-M+1)]^T \]

- Several things change when we consider the lagged window case
  - The observations are stochastic, not deterministic
  - The estimated correlation matrix \( \hat{R} \) has a relationship to the estimated correlation of the stochastic process
  - Edge effects of the signals mean our input vectors are not always complete

### Computing the Correlation Matrix

\[ \hat{r}_{i+1,j+1} = x^*(N_i - i)x(N_i - j) - x^*(N_f + 1 - i)x(N_f + 1 - j) + \hat{r}_{ij} \]

- The data matrix \( X \) is toeplzit
- This gives a structure to \( X^H X \) that can be used to increase computational efficiency
- However, \( \hat{R} \) is not necessarily toeplzit
  - Depends on the data matrix

### Derivation of Correlation Matrix Recursion

\[ \hat{r}_{ij} = \sum_{n=N_i}^{N_f} x^*(n+1-i)x(n+1-j) \]

\[ \hat{r}_{i+1,j+1} = \sum_{n=N_i}^{N_f} x^*(n+1-i+1)x(n+1-j+1) \]

\[ = \sum_{n=N_i-1}^{N_f} x^*(n+1-i)x(n+1-j) \]

\[ = x^*(N_i - 1 + 1 - i)x(N_i - 1 + 1 - j) \]

\[ - x^*(N_f + 1 - i)x(N_f + 1 - j) + \sum_{n=N_i}^{N_f} x^*(n+1-i)x(n+1-j) \]

\[ = x^*(N_i - i)x(N_i - j) - x^*(N_f + 1 - i)x(N_f + 1 - j) + \hat{r}_{ij} \]
Windowing Continued

• **Short/No Windowing**: \( N_i = M - 1 \) and \( N_f = N - 1 \)
  - Use only available data
  - No artificial data
  - No distortions
  - Unbiased, highest variance
  - Sometimes called the autocorrelation method

• **Tall/Full windowing**: \( N_i = 0 \) and \( N_f = N + M - 2 \)
  - \( \hat{R} \) becomes toeplitz (efficient order recursions)
  - Sometimes called the covariance method
  - Equivalent to calculating \( \hat{R} \) with the biased signal correlation estimate (ECE 5/638)
  - Tip: make sure data is zero mean or detrended

Unbiased Autocorrelation Estimate?

• Full windowing is equivalent to using the biased correlation estimate discussed last term

• Conceptually could also use the unbiased estimate

• Not discussed in text

• Properties
  - \( \hat{R} \) is toeplitz (by construction)
  - \( \hat{R} \) may not be positive definite
  - Estimated AR process models may be unstable — but does it matter?
  - Uses all of the data to calculate every element of \( \hat{R} \)

Correlation Matrix Recursions

\[
\hat{r}_{i+1,j+1} = x^*(N_i - i)x(N_i - j) - x^*(N_f + 1 - i)x(N_f + 1 - j) + \hat{r}_{ij}
\]

- Once the first row of \( \hat{R} \) is calculated, the recursion above can be used to fill out the rest of the matrix
- Reduces the computation from \( \mathcal{O}(NM^2) \) to \( \mathcal{O}(NM) \)
- Can reduce even further to \( \mathcal{O}(N \log N) \) via the FFT if the first row is equivalent to the signal correlation estimates discussed last term

Windowing

\[
E_e = \sum_{n=N_i}^{N_f} |e(n)|^2 = e^H e
\]

There are four ways to select the range for LSE estimation

• **Prewindowing**: \( N_i = 0 \) and \( N_f = N - 1 \)
  - Essentially treats \( x(-1), \ldots, x(-M + 1) \) equal to zero
  - Used in adaptive filters mostly for sake of simplicity

• **Postwindowing**: \( N_i = M - 1 \) and \( N_f = N + M - 2 \)
  - No one uses this
  - Included for completeness only
Forward and Backward Linear Prediction Continued

\[ \hat{x}_f(n) = -\sum_{k=1}^{M} a_k(n)x(n-k) = -a^T(n)x(n-1) \]

\[ \hat{x}_b(n) = -\sum_{k=0}^{M-1} b_k(n)x(n-k) = -b^T(n)x(n) \]

\[ e_f = x(n) + \sum_{k=1}^{M} a_k(n)x(n-k) = x(n) + a^T(n)x(n-1) \]

\[ e_b = \sum_{k=0}^{M-1} b_k(n)x(n-k) + x(n-M) = b^T(n)x(n) + x(n-M) \]

- Recall that for stationary stochastic processes, the optimum MMSE forward and backward linear predictors have conjugate symmetry
  \[ a_o = Jb_o^* \]

- This symmetry stems from the Toeplitz structure of the autocorrelation matrix
- If the estimated autocorrelation matrix doesn't have it, perhaps we could improve performance by minimize the forward and backward sum of squared errors

\[
\begin{bmatrix}
\begin{bmatrix}
  x(0) & 0 & \ldots & 0 \\
  x(1) & x(0) & \ldots & 0 \\
  \vdots & \vdots & \ddots & \vdots \\
  x(M) & x(M-1) & \ldots & x(0) \\
  \vdots & \vdots & \ddots & \vdots \\
  x(N-1) & x(N-2) & \ldots & x(N-M-1) \\
  0 & x(N-1) & \ldots & x(N-M) \\
  \vdots & \vdots & \ddots & \vdots \\
  0 & 0 & \ldots & x(N-1)
\end{bmatrix}
\end{bmatrix}
\begin{bmatrix}
  1 \\
  a_{fb}
\end{bmatrix}
= X_{fb}
+ \begin{bmatrix}
  X \\
  X^*J
\end{bmatrix}
\begin{bmatrix}
  1 \\
  a_{fb}
\end{bmatrix}
\]

\[
\begin{bmatrix}
  e_f \\
  e_b^*
\end{bmatrix}
\triangleq
\begin{bmatrix}
  1 \\
  a_{fb}
\end{bmatrix}
\begin{bmatrix}
  X \\
  X^*J
\end{bmatrix}
\begin{bmatrix}
  1 \\
  a_{fb}
\end{bmatrix}
\]
Forward and Backward Linear Prediction Continued

\[ \hat{R}_{fb} = X^H X + J X^T X^* J \]

- Makes the autocorrelation matrix more symmetric
  \[ \hat{R}_{fb} = JR_{fb} J \]
- If no windowing is used, is sometimes called the modified covariance method
- With full windowing
  \[ a_{fb} = (a + Jb^*) / 2 \]
- Works really well for AR signal modeling and parametric spectral estimation (more later)

Summary of Least Squares Filter Estimation

<table>
<thead>
<tr>
<th>Technique</th>
<th>PD</th>
<th>Unbiased</th>
<th>FFT</th>
<th>Toeplitz</th>
<th>WLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-Windowing</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Post-Windowing</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Full-Windowing</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Short-Windowing</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Unbiased</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Forward/Backward</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

*: Can be, depending on windowing technique used. **Persymmetric, \( R = JR J \).  
- Forward/backward limited to one-step prediction applications and stationary segments.
  - Works best when these conditions are met
- Otherwise short-windowing or unbiased is probably best, depending on importance of \( \hat{R} \) being positive definite
Narrowband Interference

- Many types of signals are corrupted by **narrowband interference**
- Typically electrical noise from electrical lines, radio frequency interference, and electrical equipment
- Sensors can pick up this noise through many different means
  - Closed loops (magnetic coupling)
  - Capacitive (electrostatic) coupling
  - Electromagnetic radiation (wires = antennas)
  - Power line interference
  - Voltage drops on ground lines or power lines

Narrowband Interference Properties

\[ x(n) = s(n) + y(n) + v(n) \]

where

- \( s(n) \) = signal of interest
- \( y(n) \) = narrowband interference
- \( v(n) \) = white noise

- In many applications signal is broadband and unpredictable
- Cannot be separated from white noise with a linear filter (spectral overlap)
- Narrowband interference is predictable
- Consists of sharp peaks in the frequency domain
- Can be eliminated with notch filters, but must know the frequencies

Narrowband Interference Cancellation Concept

\[ x(n) = s(n) + y(n) + v(n) \]

Suppose all three components of \( x(n) \) are mutually uncorrelated

\[ r_x(\ell) \triangleq E[x(n)x^*(n-\ell)] = E[(s(n) + y(n) + v(n))(s^*(n-\ell) + y^*(n-\ell) + v^*(n-\ell))] = r_s(\ell) + r_y(\ell) + \sigma^2_v \delta(\ell) \]

- Suppose \( r_s(\ell) \approx 0 \) for \( \ell > D \) and \( r_y(\ell) \neq 0 \) for \( \ell > D \)
- This implies \( s(n) \) is broadband and \( r_y(\ell) \) is "narrowband"
- This means if we try to predict \( x(n) D \) steps ahead, only part of the \( y(n) \) component will be (partially) predictable

\[ \hat{x}_{n-D}(n) = \hat{y}_{n-D}(n) \]

- The difference will then contain less of the interference

Example 1: Microelectrode Narrowband Interference

- Microelectrode recordings often contain narrowband interference
- The signal of interest are spikes that can be modeled as a point process
- In most cases, the spikes are not predictable with linear filters
- The duration of the spikes is typically 1 ms
- Use a narrowband interference canceller to eliminate the narrowband interference
$$h_{pe} = [1; \text{zeros}(d-1,1); -c]; \quad \text{Impulse response of the prediction error filter}$$

$$e = y - y_h; \quad \text{Residuals}$$

% Figures
BlackmanTukey(y,fs,1); FigureSet(1,'Slides'); ylim([0 0.005]); title('Input PSD'); AxisSet(8);

print('NBISignalPSD','-depsc');

BlackmanTukey(y-y_h,fs,1); FigureSet(1,'Slides'); ylim([0 0.005]); title('Output PSD'); AxisSet(8); print('NBIOutputPSD','-depsc');

figure;

k = 1:nx; t = (k-0.5)/fs; h = plot(t,y,'b',t,y_h,'g'); set(h,'LineWidth',0.2); xlim([0 0.3]); xlabel('Time (s)'); ylabel('Signal'); legend('Observed','Estimated'); set(get(gca,'Title'),'Interpreter','LaTeX'); title(sprintf('NMSE=\%5.1f\%\%%',100*sum((y-y_h).^2)/sum((y-my).^2))); box off; AxisSet(8); print('NBISignalEstimate','-depsc');

FigureSet(2,'Slides');

[h,f] = freqz(c,1,2^11,fs);
h = semilogy(f,abs(h).^2,'r'); set(h,'LineWidth',0.4); xlim([0 fs/2]); xlabel('Frequency (Hz)'); set(get(gca,'YLabel'),'Interpreter','LaTeX'); ylabel('Magnitude Response $|H(e^{j\omega})|^2$'); title('Prediction Filter'); box off; AxisSet(8); print('NBIPredictionFrequencyResponse','-depsc');

figure;

[h,f] = freqz(hpe,1,2^11,fs);
semilogy([0 fs/2],[1 1],'k:'); hold on; h = semilogy(f,abs(h).^2,'r'); set(h,'LineWidth',0.4); hold off; xlim([0 fs/2]); xlabel('Frequency (Hz)'); set(get(gca,'YLabel'),'Interpreter','LaTeX'); ylabel('Magnitude Response $|H(e^{j\omega})|^2$'); title('Prediction Error Filter'); box off; AxisSet(8); print('NBIPredictionErrorFrequencyResponse','-depsc');

clear all;
close all;

% Parameters
T = 5; \quad \text{Signal duration (s)}
td = 2e-3; \quad \text{Prediction delay (s)}
tf = 50e-3; \quad \text{Filter window duration (s)}

% Load the Data

% Preprocessing
mx = real(feat); \quad \text{Duration of extracted segment}
d = round(td*fs); \quad \text{Delay in units of samples}
f0 = round(tf*fs); \quad \text{Extract the target output}
y = y(d+(1:nx)); \quad \text{Extract a segment of the signal}

my = mean(y); \quad \text{Mean of target signal}

% Estimate the Coefficients

% Post Processing

% Preprocessing
mx = real(feat); \quad \text{Duration of extracted segment}
d = round(td*fs); \quad \text{Delay in units of samples}
f0 = round(tf*fs); \quad \text{Extract the target output}
y = y(d+(1:nx)); \quad \text{Extract a segment of the signal}

my = mean(y); \quad \text{Mean of target signal}

% Estimate the Coefficients

% Post Processing
Example 1: Least Squares Filter MATLAB Code

```matlab
function [c,yh] = LeastSquaresFilter(x,y,fsa,foa,wta,pfa); %LeastSquaresFilter: Least squares estimate of FIR filter coefficients % 
% [c,yh] = NonparametricSpectrogram(x,y,fs,wl,fr,nf,ns,pf); % % x Input signal. % y Target signal. % fo Filter order. Default = 10. % wt Window type: 0=full (default), 1=none, 2=biased autocorrelation estimate % pf Plot flag: 0=none (default), 1=screen, 2=return figure. % c Vector of coefficients. % yh Estimate of y using the estimator. % % Calculates the least squares estimate of the coefficient vector % c using the input data. Efficiently calculates the % autocorrelation matrix using the recursive approach described % in Mancikas. % % Example: Estimate the coefficients for doing narrowband % interference of a microelectrode recording. % % load MER.mat; % d = round(2e-3*fs); % y = x(d+(1:50e3)); % x = x(1:50e3); % [c,yh] = LeastSquaresFilter(x,y,fs,500,1,1); % % D. G. Manolakis, Y. K. Ingle, S. M. Kogon, "Statistical and % adaptive signal processing," Artech House, 2005. % % Version 1.00 JM
```

**Error Checking**

```matlab
if nargin<2,
    help LeastSquaresFilter;
    return;
end;
if isempty(x) | isempty(y),
    error('Signal is empty.');
end;
if length(x)~=length(y),
    error('Input signals are different lengths.');
end;
if var(x)==0 | var(y)==0,
    error('Signal is constant.');
end;
```

**Process Function Arguments**

```matlab
%====================================================================
% Create the Estimated Autocorrelation (R) and Cross-Correlation (d) %====================================================================
R = zeros(fo,fo); d = zeros(fo,1); switch wt,
    case {0,2},
        for c1=1:fo,
            d(c1) = cc(c1);% Default = unbiased estimate for c2=c1:fo,
                R(c1,c2) = ac(c2-c1+1);
                R(c2,c1) = R(c1,c2);
        end;
    case 1,
        %----------------------------------------------------------------% Calculate the First Row%----------------------------------------------------------------for c1=1:fo,
            d(c1) = sum(y(c1).*x(c1-(c1-1):nx-(c1-1)));% Default = unbiased estimate
            R(1,c1) = sum(x(c1).*x(c1-(c1-1):nx-(c1-1)));
        end;
        %----------------------------------------------------------------% Fill out the Remainder of the Matrix%----------------------------------------------------------------for c1=2:fo,
```

```matlab
%====================================================================
% Preprocessing %====================================================================
x = x(:); % Make into a column vector
%====================================================================
% Estimate the ADF %====================================================================
if wt==0, % Biased estimate
    ac = ac./nx;
    cc = cc./nx;
    else % Unbiased estimate
        k = (0:fo-1).';
        ac = ac./(nx-k);
        cc = cc./(nx-k);
    end;
end;
```

```matlab
%====================================================================
% Calculate Basic Signal Statistics %====================================================================
x = length(x); % No. samples in x
mx = mean(x); % Input signal mean
my = mean(y); % Target signal mean
```

```matlab
%====================================================================
% Error Checking %====================================================================
if nargin<2,
    help LeastSquaresFilter;
    return;
end;
if isempty(x) | isempty(y),
    error('Signal is empty.');
end;
if length(x)~=length(y),
    error('Input signals are different lengths.');
end;
if var(x)==0 | var(y)==0,
    error('Signal is constant.');
end;
```

```matlab
%====================================================================
% Calculate Basic Signal Statistics %====================================================================
x = length(x); % No. samples in x
mx = mean(x); % Input signal mean
my = mean(y); % Target signal mean
```

```matlab
%====================================================================
% Process Function Arguments %====================================================================
```

```matlab
%====================================================================
% Preprocessing %====================================================================
x = x(:); % Make into a column vector
%====================================================================
% Estimate the ADF %====================================================================
if wt==0, % Biased estimate
    ac = ac./nx;
    cc = cc./nx;
    else % Unbiased estimate
        k = (0:fo-1).';
        ac = ac./(nx-k);
        cc = cc./(nx-k);
    end;
end;
```

```matlab
%====================================================================
% Create the Estimated Autocorrelation (R) and Cross-Correlation (d) %====================================================================
R = zeros(fo,fo); d = zeros(fo,1); switch wt,
    case {0,2},
        for c1=1:fo,
            d(c1) = sum(y(c1).*x(c1-(c1-1):nx-(c1-1)));% Default = unbiased estimate
            R(1,c1) = sum(x(c1).*x(c1-(c1-1):nx-(c1-1)));
        end;
        %----------------------------------------------------------------% Fill out the Remainder of the Matrix%----------------------------------------------------------------for c1=2:fo,
```
for \(c2 = c1:fo\),
\[
R(c1,c2) = R(c1-1,c2-1) + x(fo-(c1-1)) \cdot x(fo-(c2-1)) - x(nx-(c1-2)) \cdot x(nx-(c2-2));
\]
\[
R(c2,c1) = R(c1,c2);
\]
end;
end;
% X = zeros(nx-fo+1,fo); % Verification code
% for c1=1:fo,
% X(:,c1) = x(fo-(c1-1):nx-(c1-1));
% end;
% R2 = X'*X;
% d2 = X'*y(fo:nx);
% disp([max(max(abs(R-R2))) max(abs(d-d2))]);
% end;
%====================================================================
% Calculate the Coefficients
%====================================================================
\[
c = pinv(R)*d;
\]
\[
yh = filter(c,1,x);
\]
%====================================================================
% Plot Results
%====================================================================
if pf>=1,
if pf~=2,
figure;
end;
FigureSet(1);
k = 1:nx;
th = (k-0.5)/fs;
h = plot(t,y,'r',t,yh,'g');
set(h,'LineWidth',1.2);
xlim([0 nx/fs]);
xlabel('Time (s)');
ylabel('Signal');
legend('Observed', 'Estimated');
set(get(gca,'Title'),'Interpreter','LaTeX');
title(sprintf('NMSE=%5.1f\%%',100*sum((y-yh).^2)/sum((y-my).^2)));
box off;
AxisSet;
end;
if pf~=2,
figure;
end;
FigureSet(2);
[h,f] = freqz(c,1,2^12,fs);
h = semilogy(f,abs(h).^2,'r');
set(h,'LineWidth',1.2);
xlim([0 fs/2]);
xlabel('Frequency (Hz)');
ylabel('Magnitude Response $|H(\exp{j\omega})|^2$');
box off;
AxisSet;
end;
%====================================================================
% Process Return Arguments
%====================================================================
if nargout==0,
clear('c','yh');
end;