EE263 Homework 9 Solutions

Fall 2025

15.2170. Positive semidefinite (PSD) matrices.

a) Show that if A and B are PSD and $\alpha \in \mathbb{R}$, $\alpha \geq 0$, then so are αA and $A + B$.

b) Show that any (symmetric) submatrix of a PSD matrix is PSD. (To form a symmetric submatrix, choose any subset of $\{1, \ldots, n\}$ and then throw away all other columns and rows.)

c) Show that if $A \geq 0$, $A_{ii} \geq 0$.

d) Show that if A is (symmetric) PSD, then $|A_{ij}| \leq \sqrt{A_{ii}A_{jj}}$. In particular, if $A_{ii} = 0$, then the entire ith row and column of A are zero.

Solution.

a) To show that $\alpha A \geq 0$ we verify that $x^\mathsf{T}(\alpha A)x \geq 0$ for all x. But $x^\mathsf{T}(\alpha A)x = \alpha(x^\mathsf{T}Ax)$ and since $x^\mathsf{T}Ax \geq 0$ ($A \geq 0$) and $\alpha \geq 0$, we immediately get $x^\mathsf{T}(\alpha A)x \geq 0$. Again, to show that $A + B \geq 0$ we show that $x^\mathsf{T}(A + B)x \geq 0$ for all x. This is easy because $x^\mathsf{T}(A + B)x = x^\mathsf{T}Ax + x^\mathsf{T}Bx$ and $A, B \geq 0$ imply that $x^\mathsf{T}Ax, x^\mathsf{T}Bx \geq 0$ and therefore $x^\mathsf{T}(A + B)x \geq 0$.

b) Suppose that $A = A^\mathsf{T} \geq 0$. Any symmetric submatrix of A can be written as $Z^\mathsf{T}AZ$ for some suitable matrix Z. For example, if $A \in \mathbb{R}^{3\times 3}$ and we want to pick the submatrix formed by the first and third columns and rows we simply take

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

so that

$$Z^\mathsf{T}AZ = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{12} & A_{22} & A_{23} \\ A_{13} & A_{23} & A_{33} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{13} \\ A_{13} & A_{33} \end{bmatrix}.$$

The idea here is to pick the columns of Z as the unit vectors corresponding to the column/row numbers we want to keep. In this example, we wanted to keep the first and third columns/rows so we took $Z = [e_1\ e_3]$. In general, consider the $m \times m$ symmetric submatrix of A which consists of elements of A that are only on the columns and rows $i_1, \ldots, i_m$ of A. Then it is easy to verify that

$$\text{(submatrix formed from columns/rows } i_1, \ldots, i_m) = Z^\mathsf{T}AZ, \quad Z = [e_{i_1}\ \cdots\ e_{i_m}],$$

where e_{i_j} is the i_jth unit vector in $\mathbb{R}^n$. Using the result of problem (**??**), $A \geq 0$ implies that $Z^\mathsf{T}AZ \geq 0$ and therefore any symmetric submatrix of A is also positive semidefinite.

c) This is easy. We can simply use the result of the previous part ($A_{ii} \in \mathbb{R}$ is a 1×1 symmetric submatrix of A), or more directly, use the fact that $A \geq 0$ implies $e_i^\mathsf{T} A e_i \geq 0$ and note that $e_i^\mathsf{T} A e_i$ is nothing but A_{ii}.

d) Choose any 2×2 symmetric submatrix of A, say

$$\tilde{A} = \begin{bmatrix} A_{ii} & A_{ij} \\ A_{ij} & A_{jj} \end{bmatrix}.$$

According to problem (b) this (symmetric) submatrix is positive semidefinite and therefore its eigenvalues are nonnegative. Hence, the determinant of the submatrix (which is equal to the product of the eigenvalues) is also nonnegative. In other words

$$\det \tilde{A} = \begin{bmatrix} A_{ii} & A_{ij} \\ A_{ij} & A_{jj} \end{bmatrix} = A_{ii}A_{jj} - A_{ij}^2 \geq 0$$

and immediately we get $A_{ij}^2 \leq A_{ii}A_{jj}$ or $|A_{ij}| \leq \sqrt{A_{ii}A_{jj}}$. In particular, if $A_{ii} = 0$ then $|A_{ij}| \leq 0$ or $A_{ij} = 0$ (for any j) and the entire ith row (and hence ith column since A is symmetric) should be zero.

15.2320. Approximate left inverse with norm constraints. Suppose $A \in \mathbb{R}^{m \times n}$ is full rank with $m \geq n$. We seek a matrix $F \in \mathbb{R}^{n \times m}$ that minimizes $\|I - FA\|$ subject to the constraint $\|F\| \leq \alpha$, where $\alpha > 0$ is given. Note that $\|I - FA\|$ gives a measure of how much F fails to be a left inverse of A. Give an explicit description of an optimal F. Your description can involve standard matrix operations and decompositions (eigenvector/eigenvalue, QR, SVD, ...).

Solution. Let $A = \sum_{i=1}^n \sigma_i u_i v_i^\mathsf{T}$ be the SVD of A, and let B be *any* matrix with $\sigma_{\max}(B) \leq \alpha$. Then note that

$$u_n^\mathsf{T}(I - AB) = u_n^\mathsf{T} - \sigma_n v_n^\mathsf{T} B.$$

Therefore

$$\|u_n^\mathsf{T}(I - AB)\| \geq \|u_n\| - \|\sigma_n v_n^\mathsf{T} B\| \geq 1 - \alpha\sigma_n(A),$$

since $\|v_n^\mathsf{T} B\| \leq \sigma_{\max}(B)$. So we conclude that $\sigma_{\max}(I - AB) \geq 1 - \alpha\sigma_n(A)$. Therefore, if we can find a matrix B_0 such that $\sigma_{\max}(I - AB_0) = 1 - \alpha\sigma_n(A)$, then this matrix is optimal. Here is one choice of B_0 that works:

$$B_0 = \sum_{i=1}^n \gamma_i v_i u_i^\mathsf{T}, \quad \gamma_i = \min\left\{\alpha, \frac{1}{\sigma_i}\right\}.$$

Roughly speaking, we invert the singular values of A as much as the constraint allows. The singular values of B_0 are the (re-ordered) γ_i's; so $\sigma_{\max}(B) \leq \alpha$ as required. Now let's find $\sigma_{\max}(I - AB)$.

$$I - AB_0 = \sum_{i=1}^n (1 - \sigma_i\gamma_i) u_i u_i^\mathsf{T}$$

and so

$$\sigma_{\max}(I - AB_0) = \max_i |1 - \sigma_i\gamma_i| = 1 - \sigma_n(A)\alpha.$$

with our choice of γ_i's. We note that B_0 is *not* the only optimal B.

16.2710. The EE263 search engine. In this problem we examine how linear algebra and low-rank approximations can be used to find matches to a search query in a set of documents. Let's assume we have four documents: **A**, **B**, **C**, and **D**. We want to search these documents for three terms: *piano*, *violin*, and *drum*. We know that:
in **A**, the word *piano* appears 4 times, *violin* 3 times, and *drum* 1 time;
in **B**, the word *piano* appears 6 times, *violin* 1 time, and *drum* 0 times;
in **C**, the word *piano* appears 7 time, *violin* 4 times, and *drum* 39 times; and
in **D**, the word *piano* appears 0 times, *violin* 0 times, and *drum* 5 times.
We can tabulate this as follows:

	A	**B**	**C**	**D**
piano	4	6	7	0
violin	3	1	4	0
drum	1	0	39	5

This information is used to form a *term-by-document* matrix A, where A_{ij} specifies the frequency of the ith term in the jth document, *i.e.*,

$$A = \begin{bmatrix} 4 & 6 & 7 & 0 \\ 3 & 1 & 4 & 0 \\ 1 & 0 & 39 & 5 \end{bmatrix}.$$

Now let q be a *query vector*, with a non-zero entry for each term. The query vector expresses a criterion by which to select a document. Typically, q will have 1 in the entries corresponding to the words we want to search for, and 0 in all other entries (but other weighting schemes are possible.) A simple measure of how relevant document j is to the query is given by the inner product of the jth column of A with q:

$$a_j^T q.$$

However, this criterion is biased towards large documents. For instance, a query for *piano* ($q = [1\ 0\ 0]^T$) by this criterion would return document **C** as most relevant, even though document **B** (and even **A**) is probably much more relevant. For this reason, we use the inner product normalized by the norm of the vectors,

$$\frac{a_j^T q}{\|a_j\|\,\|q\|}.$$

Note that our criterion for measuring how well a document matches the query is now the cosine of the angle between the document and query vectors. Since all entries are non-negative, the cosine is in $[0, 1]$ (and the angle is in $[-\pi/2, \pi/2]$.) Define $\tilde{A}$ and $\tilde{q}$ as normalized versions of A and q (A is normalized column-wise, *i.e.*, each column is divided by its norm.) Then,

$$c = \tilde{A}^T \tilde{q}$$

is a column vector that gives a measure of the relevance of each document to the query. And now, the question. In the file `term_by_doc.json` you are given m search terms, n documents, and the corresponding term-by-document matrix $A \in \mathbb{R}^{m \times n}$. (They were obtained randomly from Stanford's *Portfolio* collection of internal documents from the 1990s.) The variables `term`

and `document` are lists of strings. The string `term[i]` contains the ith word. Each document is specified by its former URL, *i.e.*, the jth document used to be at the URL `document[j]`; the documents are no longer available online, but you might be able to find some of them on some internet archive (like the Wayback Machine) if you're curious. (You don't need to in order to solve the problem.) The matrix entry `A[i,j]` specifies how many times term i appears in document j.

When you specify documents in your results, please just specify the indices, that is, give us `j` rather than `document[j]`.

a) Compute $\tilde{A}$, the normalized term-by-document matrix. Compute and plot the singular values of $\tilde{A}$.

b) Perform a query for the word *students* ($i = 53$) on $\tilde{A}$. What are the 5 top results?

c) We will now consider low-rank approximations of $\tilde{A}$, that is

$$\hat{A}_r = \min_{\hat{A},\, \mathbf{rank}(\hat{A}) \le r} \|\tilde{A} - \hat{A}\|.$$

Compute $\hat{A}_{32}$, $\hat{A}_{16}$, $\hat{A}_8$, and $\hat{A}_4$. Perform a query for the word *students* on these matrices. Comment on the results.

d) Are there advantages of using low-rank approximations over using the full-rank matrix? (You can assume that a very large number of searches will be performed before the term-by-document matrix is updated.)

Note: Variations and extensions of this idea are actually used in commercial search engines (although the details are closely guarded secrets . . .) Issues in real search engines include the fact that m and n are enormous and change with time. These methods are very interesting because they can recover documents that don't include the term searched for. For example, a search for *automobile* could retrieve a document with no mention of *automobile*, but many references to cars (can you give a justification for this?) For this reason, this approach is sometimes called *latent semantic indexing*.

Julia hints: You may find the command `sortperm` useful. It returns the index permutation that would sort its argument into an ascending order, or if the `rev=true` argument is supplied, into a descending order. Here's some sample code that prints the indices of the two largest elements of a vector `c`:

```
p = sortperm(c, rev=true)
@show p[1:2]                # indices of two largest elements
c_sorted = c[p]             # equivalent of sort(c, rev=true)
@show c[p[1:2]]             # two largest elements of c
```

Solution. The singular values of the normalized matrix are shown in the figure. The full-rank search yields the documents:

```
[106, 105, 107, 115, 111]
```

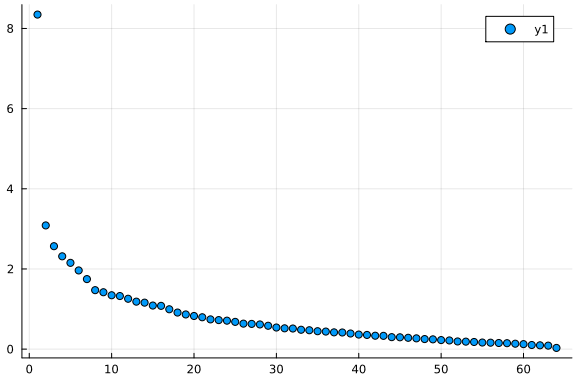


Figure 1: Singular values of $\tilde{A}$

The corresponding weights are $c = [\,0.60 \quad 0.55 \quad 0.50 \quad 0.49 \quad 0.41\,]^\mathsf{T}$. Given the SVD of $\tilde{A} = U\Sigma V^\mathsf{T}$, the nearest low-rank approximation is easily computed:

$$\hat{A}_r = \min_{\hat{A},\,\mathbf{rank}(\hat{A})\leq r} \|\tilde{A} - \hat{A}\| = \sum_{k=1}^{r} \sigma_k u_k v_k^\mathsf{T} = U_1 W_1^\mathsf{T},$$

where σ_k are the singular values, u_k are the columns of U, v_k are the columns of V. We have also written the low rank approximation as the product of two smaller full-rank matrices, $U_1 = [\,u_1 \;\cdots\; u_r\,]$, and $W_1 = [\,\sigma_1 v_1 \;\cdots\; \sigma_r v_r\,]$. The search results are as follows (the table includes the document numbers only, *i.e.*, the last part of the URL):

	rank 32	rank 16	rank 8	rank 4
1st	106	106	115	115
2nd	105	107	106	105
3rd	107	105	120	107
4th	115	115	107	66
5th	111	111	111	63

And the corresponding weights are:

	rank 32	rank 16	rank 8	rank 4
1st	0.6445	0.5474	0.3751	0.2212
2nd	0.4952	0.4519	0.3666	0.1961
3rd	0.4726	0.4312	0.3462	0.1912
4th	0.4299	0.3910	0.3322	0.1894
5th	0.3846	0.3745	0.2998	0.1824

Note the "loss of resolution" for the lower rank searches, as the weights become less differentiated. Down to rank 16 the low-rank search is essentially equivalent to the full-rank search. For rank 8, the top 5 results are still very similar, although their ordering is somewhat changed. For rank 4 the results start differing significantly, although there are still some top 5 documents in common with the full-rank search. However, if you look at the content of the documents

(something that almost no-one seems to have done), you'll see that the ones returned by the low-rank searches appear to be more relevant! Of course, this is a subjective appreciation... But we'll get back to this in a while, when discussing the advantages of the low-rank search. The Julia code for this problem is as follows:

```
include("../../../../../julia_code/readclassjson.jl")
using  LinearAlgebra, Plots

function solution()
 data = readclassjson("../data/term_by_doc.json")
 println(keys(data))

 A = data["A"]
 document = data["document"]
 term = data["term"]
 m = data["m"]
 n = data["n"]

 t = [53]  # index of term(s) to search for (t=[53 64] also works)
 q = zeros(m,1)  # query vector

 q[t] .= 1.0  # put 1 in the entries indexed by the vector term

 println("query:")
 for k=1:length(t)
  println(term[t[k]]) # print words in query
 end

 An = zeros(size(A)) # An = "A normalized"
 for j=1:n
  An[:,j] = A[:,j]/norm(A[:,j]) # normalize columns
 end

 q = q / norm(q) # normalize query

 # Your TAs recommend this syntax over [U,S,Vt] = svd(A)
 # because it eliminates the risk of confusing V and Vt.
 F = svd(An)
 U, S, V = F.U, F.S, F.V

 scatter(collect(1:size(S)[1]), S) # plot of singular values
 savefig("../graphics/singular_values.pdf")

 # Iterate over rank-k approximations, starting with the full rank matrix
 for r in [length(S) 32 16 8 4]
  U1=U[:,1:r];
```

```
  S1=S[1:r];
  V1=V[:,1:r];
  c1=(V1*diagm(S1))*(U1'*q);

  p1 = sortperm(c1, rev=true, dims=1)
  @show p1[1:5]
  @show c1[p1[1:5]]

  println("Rank $r top 5:") # print top 5 docs

  for k=1:5
   println(document[p1[k]])
  end
  println(c1[1:5]') # print weights of the top 5 docs
 end
end

solution()
```

Now, for the advantages of low-rank search. The most obvious one is query speed, which can be achieved by computing the query using the factored form of $\hat{A}_r$:

$$c = W_1(U_1^\mathsf{T} q)$$

(with U_1 and W_1 as defined above.) The full-rank query (and the low-rank without using the factored form) requires about nm operations. The factored low-rank query requires about $r(m+n)$ operations. Of course, computing the SVD for a large matrix requires a large numbers of operations, so the number of searches performed between updates of the matrix must be large for this to pay off. (Note that there are techiques for doing an approximate update of the low-rank approximation when a new document is added without recomputing the whole SVD.) Also, note that the SVD can be done off-line. Even if it's overall more expensive (*i.e.*, requires more operations) to do the SVD plus the low-rank searches than to do all searches on the full-rank matrix, the user will be happy to have faster searches. Meanwhile another computer can be used to work in parallel on the next updating of the low-rank approximation. The only part of this problem that most people didn't get was the "latent semantic indexing" property, which can be interpreted as a form of "de-noising." More than query speed, this is actually the driving motivation for the use of low rank approximations in database searches. The best way to explain this is by example, so here's a good one (from a student's exam solution!) Consider the following term-by-document matrix:

	Doc 1	Doc 2	Doc 3	Doc 4
car	30	15	1	2
automobile	0	15	0	1
penguin	0	0	20	1
coffee	0	0	3	14

The SVD of the normalized $\tilde{A}$ yields the singular values:

$$\sigma = [\,1.33\ 1.08\ 0.88\ 0.54\,].$$

Note that the fourth is smaller, but not by that much. Also from the SVD, we get:

$$U = \begin{bmatrix} 0.89 & -0.23 & 0.09 & -0.38 \\ 0.37 & -0.11 & -0.01 & 0.92 \\ 0.13 & 0.71 & 0.69 & 0.03 \\ 0.23 & 0.66 & -0.72 & -0.02 \end{bmatrix}.$$

We can give a "semantic" interpretation of each dyad. The first vector weights the terms *car* and *automobile*. The second vector weights the terms *penguin* and *coffee*. The third vector puts differential weights on *penguin* and *coffee*. The fourth vector puts differential weights on *car* and *automobile*. If we remove the fourth dyad, the result is that the differentiation between *car* and *automobile* is removed! Here's the resulting low-rank approximation of $\tilde{A}$:

$$\hat{A}_3 = \begin{bmatrix} 0.85 & 0.85 & 0.05 & 0.14 \\ 0.35 & 0.35 & -0.01 & 0.08 \\ 0.01 & -0.01 & 0.99 & 0.07 \\ -0.01 & 0.01 & 0.15 & 0.98 \end{bmatrix}.$$

We see that *car* and *automobile* are now weighted equally in documents 1 and 2. A search for *automobile* (*i.e.*, $q = [\,0\ 1\ 0\ 0\,]^\mathsf{T}$) on the original $\tilde{A}$ yields zero relevance for the first document:

$$c = \tilde{A}^\mathsf{T} q = [\,0.00 \;\; 0.71 \;\; 0.00 \;\; 0.07\,]^\mathsf{T}.$$

The same search on the low-rank approximation $\hat{A}_3$ produces the much more sensible result:

$$c = \hat{A}_3^\mathsf{T} q = [\,0.35 \;\; 0.35 \;\; -0.01 \;\; 0.08\,]^\mathsf{T}.$$

Documents 1 and 2 are now considered to have equal relevance for the word *automobile*! The conclusion is that even though the search results of a low-rank approximation may be "less accurate" (in the sense that they differ from those of a search with the original matrix), they may actually be better from the user's point of view.

17.1340. Navigation. Suppose a ship is located at position $x \in \mathbb{R}^2$. We measure distances to beacons located at

$$q_1 = \begin{bmatrix} 500 \\ 0 \end{bmatrix} \qquad q_2 = \begin{bmatrix} 0 \\ 500 \end{bmatrix} \qquad q_3 = \begin{bmatrix} 500 \\ -200 \end{bmatrix}$$

As usual we model this as $y = Ax + w$, where $y \in \mathbb{R}^3$ is a vector of measurements (with appropriate constants to make the relationship linear) such that $y_i = \frac{1}{\|q_i\|} q_i^\top x$ and $w \in \mathbb{R}^3$ is Gaussian noise, $w \sim \mathcal{N}(0, \Sigma_w)$. We also have prior information $x \sim \mathcal{N}(\mu_x, \Sigma_x)$. Here

$$\mu_x = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \qquad \Sigma_x = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \Sigma_w = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

a) The MMSE estimate has the form

$$\phi_{\text{mmse}}(y) = \mu_x + L_{\text{mmse}}(y - A\mu_x)$$

Find the estimator gain L_{mmse} and compute it using Julia.

b) Perform the experiment in Julia; i.e., simulate x and w with the above distributions and compute y. Using this y, plot the 90% confidence ellipsoid for x conditioned on your measurement, centered on your estimate.

Hint: In Julia, you can sample from a multivariate normal distribution with the code

```
using Distributions
mu = zeros(2)
sigma = I(2)
dist = MvNormal(mu, sigma)
sample = rand(dist)
```

c) Now collect data, consisting of 500 samples of $y = Ax + w$. For each measurement y, compute the corresponding estimate $\phi_{\text{mmse}}(y)$, and plot the error in $\mathbb{R}^2$ given by $x - \phi_{\text{mmse}}(y)$. Also plot the 90% confidence ellipsoid for the error (centered at the origin.)

d) Suppose instead we were to use the least squares estimator

$$\phi_{\text{ls}}(y) = A^{\dagger} y$$

What is the mean square error

$$\mathrm{E}\|\phi_{\text{ls}}(y) - x\|^2$$

achieved by this estimator?

e) Using the same data as above, plot the error in $\mathbb{R}^2$ given by $x - \phi_{\text{ls}}(y)$. Also plot the 90% confidence ellipsoid for the error, centered at the origin.

f) Is it true that the ellipsoid of part (c) is always contained in the ellipsoid of part (e)? Prove or give a counterexample.

Solution.

a) The MMSE estimator is

$$\phi_{\text{mmse}}(y) = \mu_x + L_{\text{mmse}}(y - A\mu_x)$$

where

$$L_{mmse} = \Sigma_{xy}\Sigma_y^{-1}$$

We have $y = Ax + w$ where A has the normalized beacon vectors as each row.

$$\begin{aligned} A &= \begin{bmatrix} q_1^T/\|q_1\| \\ q_2^T/\|q_2\| \\ q_3^T/\|q_3\| \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0.9285 & -0.3714 \end{bmatrix} \end{aligned}$$

We now have

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} I & 0 \\ A & I \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix}$$

and the joint covariance matrix is

$$\begin{bmatrix} \Sigma_x & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_y \end{bmatrix} = \begin{bmatrix} \Sigma_x & \Sigma_x A^T \\ A\Sigma_x & A\Sigma_x A^T + \Sigma_w \end{bmatrix}$$

Hence,

$$L_{\text{mmse}} = \Sigma_{xy}\Sigma_y^{-1} = \begin{bmatrix} 0.8470 & 0.0137 & 0.0736 \\ 0.1366 & 0.4699 & -0.1618 \end{bmatrix}$$

b) We have

$$\mathrm{E}(x|y) = \phi_{\text{mmse}}(y)$$

and define

$$\Sigma_{x|y} = \mathrm{cov}(x|y)$$

then

$$\Sigma_{x|y} = \Sigma_x - \Sigma_x A^T (A\Sigma_x A^T + \Sigma_w)^{-1} A\Sigma_x$$

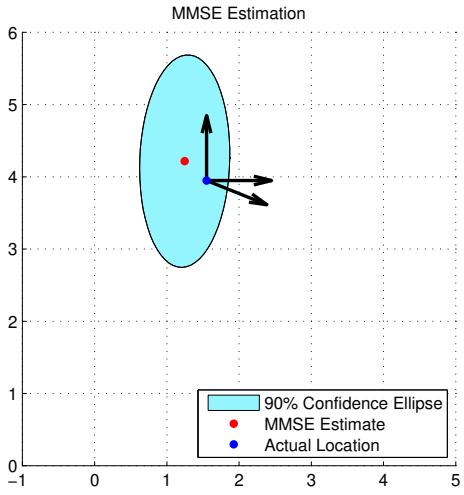
The 90% confidence ellipse for x conditioned on y is

$$\left\{ x \in \mathbb{R}^2 \mid (x - \phi_{\text{mmse}}(y))^T \Sigma_{x|y}^{-1} (x - \phi_{\text{mmse}}(y)) \leq \alpha \right\}$$

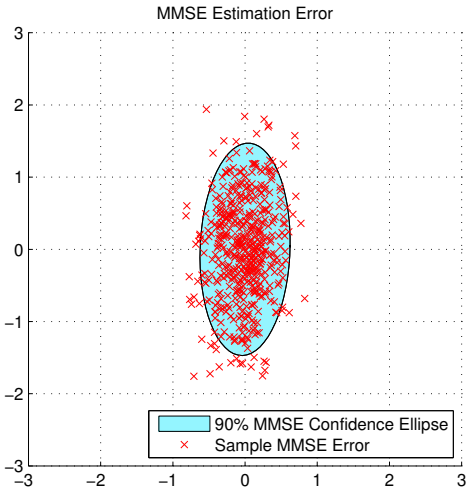
where

$$\alpha = F_{\chi_2^2}^{-1}(0.9) = 4.6052$$

The following figure shows the 90% confidence ellipse centered on the MMSE estimate for a particular simulation.



c) Define the random variable $z = x - \phi_{mmse}(y)$. It has a Gaussian distribution with zero mean and covariance $\Sigma_{x|y}$. Thus, the 90% confidence ellipse for the error z looks the same as the ellipse in part b, translated to be centered on the origin, as seen in the following figure.



d) Define the random variable $z_{\text{ls}} = \phi_{\text{ls}}(y) - x$. We have

$$\begin{aligned} z_{\text{ls}} &= A^{\dagger}y - x \\ &= A^{\dagger}(Ax + w) - x \\ &= A^{\dagger}w \end{aligned}$$

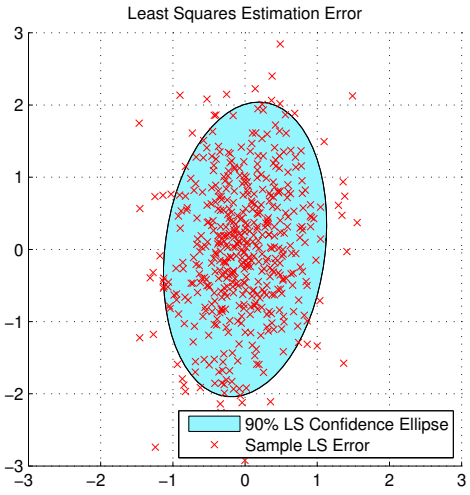
Therefore z_{ls} is Gaussian with zero mean. Let $\Sigma_{\text{ls}} = \text{cov}(z_{\text{ls}})$, then

$$\Sigma_{\text{ls}} = A^{\dagger}\Sigma_w A^{\dagger T}$$

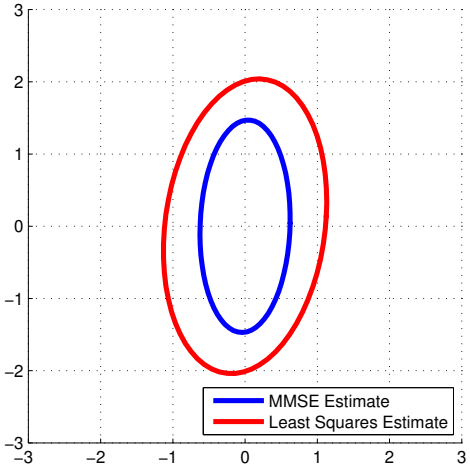
So, the mean square error achieved by this estimator is

$$\text{E}\|\phi_{\text{ls}}(y) - x\|^2 = \text{trace}(\Sigma_{\text{ls}}) = 1.1819$$

e) As in part (c), the 90% confidence ellipse corresponds to the covariance of z_{ls}, as shown in the following figure.



f) We can visually verify this in the current example by plotting the two ellipses, as shown below.



In order to prove that the ellipse from part (c) is always contained within the ellipse from part (e), we need to show that

$$\Sigma_{x|y}^{-1} \geq \Sigma_{\text{ls}}^{-1}$$

We have

$$\begin{aligned}\Sigma_{x|y}^{-1} - \Sigma_{\text{ls}}^{-1} &= (\Sigma_x - \Sigma_x A^T (A\Sigma_x A^T + \Sigma_w)^{-1} A\Sigma_x)^{-1} - (A^\dagger \Sigma_w A^\dagger)^{-1} \\ &= \Sigma_x^{-1} + A^T \Sigma_w^{-1} A - A^T A (A^T \Sigma_w A)^{-1} A^T A\end{aligned}$$

by the matrix inversion lemma, and hence

$$\Sigma_{x|y}^{-1} - \Sigma_{\text{ls}}^{-1} = \Sigma_x^{-1} + A^T (\Sigma_w^{-1} - A(A^T \Sigma_w A)^{-1} A^T) A$$

To analyze the second term in the above expression, define the matrix

$$M = \begin{bmatrix} A^T \Sigma_w A & A^T \\ A & \Sigma_w^{-1} \end{bmatrix}$$

Then M satisfies

$$M = B^T \begin{bmatrix} \Sigma_w & 0 \\ 0 & 0 \end{bmatrix} B$$

where

$$B = \begin{bmatrix} A & \Sigma_w^{-1} \\ 0 & I \end{bmatrix}$$

and hence we have that $M \geq 0$. Since $M_{11} > 0$, we have by the completion of squares argument that the schur complement of M must be positive semidefinite also. That is,

$$M_{22} - M_{21} M_{11}^{-1} M_{12} \geq 0$$

and this is just

$$\Sigma_w^{-1} - A(A^T \Sigma_w A)^{-1} A^T \geq 0$$

As a result,

$$\Sigma_{x|y}^{-1} - \Sigma_{\text{ls}}^{-1} \geq 0$$

as desired.

17.1360. MMSE with correlated noise. Suppose

$$y = Ax + w$$

and x and w are Gaussian, with

$$\mathrm{E} \begin{bmatrix} x \\ w \end{bmatrix} = \begin{bmatrix} \mu_x \\ \mu_w \end{bmatrix} \qquad \mathrm{cov} \begin{bmatrix} x \\ w \end{bmatrix} = \begin{bmatrix} \Sigma_x & \Sigma_{xw} \\ \Sigma_{wx} & \Sigma_w \end{bmatrix}$$

In particular, x and w are *correlated*. Find the minimum mean-square-error estimate of x given a measurement of y.

Solution. We can write

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} I & 0 \\ A & I \end{bmatrix} + \begin{bmatrix} x \\ w \end{bmatrix}$$

Hence $\mu_y = A\mu_x + \mu_w$ and

$$\begin{aligned} \begin{bmatrix} \Sigma_x & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_y \end{bmatrix} &= \begin{bmatrix} I & 0 \\ A & I \end{bmatrix} \begin{bmatrix} \Sigma_x & \Sigma_{xw} \\ \Sigma_{wx} & \Sigma_w \end{bmatrix} \begin{bmatrix} I & 0 \\ A & I \end{bmatrix}^T \\ &= \begin{bmatrix} \Sigma_x & \Sigma_x A^T + \Sigma_{xw} \\ A\Sigma_x + \Sigma_{wx} & A\Sigma_x A^T + \Sigma_{wx} A^T + A\Sigma_{xw} + \Sigma_w \end{bmatrix} \end{aligned}$$

The MMSE estimator is given by

$$\hat{x}_{\text{mmse}} = \mu_x + \Sigma_{xy} \Sigma_y^{-1} (y_{\text{meas}} - \mu_y)$$

17.1365. Orthogonality. Suppose x, y are jointly Gaussian random variables. Let $\phi(y_{\text{meas}})$ be the MMSE estimate of x given $y = y_{\text{meas}}$. Let z be the estimation error

$$z = \phi(y) - x$$

a) Show that

$$\mathrm{E}\, zy^T = 0$$

b) This is called the *orthogonality principle* in estimation. It says that the estimation error is *uncorrelated* with the measurement y. Explain why you would expect this to be the case.

Solution.

a) We will use the following results in this part.

$$\begin{aligned}
\hat{x} &= \mu_x + L(y - \mu_y) \\
\mathbf{E}(yy^T) &= A\Sigma_x A^T + \Sigma_w + \mu_y \mu_y^T \\
\mathbf{E}(xy^T) &= \Sigma_x A^T + \mu_x \mu_y^T
\end{aligned}$$

where $L = \Sigma_x A^T (A\Sigma_x A^T + \Sigma_w)^{-1}$. We have (using above results)

$$\begin{aligned}
\mathbf{E}\big((\hat{x} - x)(y - \mu_y)^T\big) &= \mathbf{E}\big(\mu_x - L\mu_y + Ly - x)(y - \mu_y)^T\big) \\
&= \mu_x \mu_y^T - \mu_x \mu_y^T - L\mu_y \mu_y^T + L\mu_y \mu_y^T \\
&\quad + L\mathbf{E}(yy^T) - L\mu_y \mu_y^T - \mathbf{E}(xy^T) + \mu_x \mu_y^T \\
&= 0
\end{aligned}$$

An intuitive explanation for this is as follows. If the error was correlated with the measurement, we could use MMSE estimation to estimate the error, and hence get reduce its posterior covariance. This would give a lower mean square error than that of the MMSE estimator, which is not possible.