MLPR Tutorial Sheet 2 (Answers)

Reminders: Attempt the tutorial questions before your tutorial. Many of the best-performing students discuss the class and tutorial work with their peers during the week. You can seek clarifications and hints on the class forum. Full answers will be released after the tutorial.

This tutorial is largely just maths. While I ask for a small numerical experiment in the middle, there’s no real data, and no machine learning. However, throughout the course we will derive models and algorithms that use multivariate Gaussian distributions. And other machine learning methods share some of the same maths. I’ve put this material on the tutorial, because it’s useful stuff, and you need to work through it at your own pace. You’ll get an assignment soon that involves some data!

1. Warm-up exercise:
   If \(a\) and \(b\) are \(D \times 1\) column vectors and \(M\) is a \(D \times D\) symmetric matrix, show that
   \[a^\top M b = b^\top M a.\]
   You wouldn’t need to show this result in an exam unless you were explicitly asked to. In some working (like for the next question) you could just state that it’s true for symmetric \(M\).

   **Answer:**
   As \(a^\top M b\) is a scalar, \(a^\top M b = (a^\top M b)^\top = (M b)^\top a = b^\top M^\top a\) using the result from the background material crib-sheet \((AB)^\top = B^\top A^\top\) twice.
   Since \(M\) is symmetric, \(M^\top = M\), and hence \(a^\top M b = b^\top M a\).
   Alternatively, you could write the quadratic form as a sum of scalars, where reordering is straightforward:
   \[a^\top M b = \sum_{ij} a_i M_{ij} b_j = \sum_{ij} b_j M_{ji} a_i = \sum_{ij} b_j M_{ji} a_i = b^\top M a,\]
   where we can swap the indices on \(M\) because it is symmetric: \(M_{ij} = M_{ji}\).

2. Identifying a Gaussian:
   As part of a derivation, we may need to identify the probability density function of a vector up to a constant. For example:
   \[p(x) \propto \exp \left(-x^\top A x - x^\top c\right),\]
   where \(A\) is a symmetric invertible matrix. As this distribution is proportional to the exponential of a quadratic in \(x\), it is a Gaussian: \(p(x) = \mathcal{N}(x; \mu, \Sigma)\).
   Identify which Gaussian \(x\) comes from by identifying the mean \(\mu\) and covariance \(\Sigma\) in terms of \(A\) and \(c\). The easiest method is to compare \(p(x)\) to the standard form for the multivariate Gaussian PDF (given in class).
   The answer you should be able to show is:
   \[\Sigma = \frac{1}{2} A^{-1}, \quad \mu = -\frac{1}{2} A^{-1} c.\]
Working for answer:
The quadratic form in the multivariate Gaussian takes the form:
\[
\log \mathcal{N}(x; \mu, \Sigma) = -\frac{1}{2} (x - \mu)\Sigma^{-1}(x - \mu) + \text{const.}
\]
\[
= -\frac{1}{2} x^\top \Sigma^{-1} x + \frac{1}{2} x^\top \Sigma^{-1} \mu + \frac{1}{2} \mu^\top \Sigma^{-1} \mu + \text{const.}
\]
\[
= -\frac{1}{2} x^\top \Sigma^{-1} x + \mu^\top \Sigma^{-1} \mu + \text{const.} \quad (\Sigma \text{ and } \Sigma^{-1} \text{ are symmetric})
\]
In comparison the quadratic form given in the question is:
\[
\log p(x) = -x^\top Ax - x^\top c + \text{const.}
\]
We can read off the mean and covariance by comparing the coefficients of these forms. Comparing the quadratic term:
\[
-\frac{1}{2} \Sigma^{-1} = -A \quad \Rightarrow \quad \Sigma = \frac{1}{2} A^{-1}.
\]
Comparing the linear term:
\[
\Sigma^{-1} \mu = -c \quad \Rightarrow \quad \mu = -\Sigma c = -\frac{1}{2} A^{-1} c.
\]
Some textbooks will instruct you to ‘complete the square’ — manipulate the given equation for \( p(x) \) into the standard quadratic form that appears in a Gaussian PDF, keeping track of the constants. That approach is fine, but more work than necessary.

3. Creating a 2D multivariate Gaussian, and a simple experiment:
The first element of a vector has \( p(x_1) = \mathcal{N}(x_1; m, \sigma^2) \).
A second element is generated according to the following process:
\[
x_2 = \alpha x_1 + \nu, \quad \nu \sim \mathcal{N}(0, n^2).
\]
Here \( x_2 \) depends on \( x_1 \), but the noise term \( \nu \) is independent of \( x_1 \).
Recall that a linear combination of Gaussian values is Gaussian distributed.

a) The joint distribution of the vector \( x = [x_1 \ x_2]^\top \) is Gaussian, and so takes the form \( p(x) = \mathcal{N}(x; \mu, \Sigma) \). Identify \( \mu \) and \( \Sigma \).

b) Turning to a computer: pick a setting for each of the parameters \( m, \sigma^2, \alpha, \) and \( n \), and simulate samples from the above process. Estimate the mean and covariance from the samples. Do your estimates of the mean and covariance agree with their theoretical values?

Putting a standard error on your estimates of the means should be straightforward. You may have to use some creativity to put error bars on your estimates of the covariances.

Answer:
a) Here’s one way to answer the question. We can identify several expectations:

\[
\begin{align*}
\mathbb{E}[x_1] &= m \\
\mathbb{E}[x_1^2] &= \sigma^2 + m^2 \\
\mathbb{E}[x_2] &= \alpha \mathbb{E}[x_1] + \mathbb{E}[\nu] = am \\
\mathbb{E}[x_1 x_2] &= \mathbb{E}[\alpha x_1^2 + \nu x_1] = \alpha \mathbb{E}[x_1^2] + \mathbb{E}[\nu \mathbb{E}[x_1]] = \alpha (\sigma^2 + m^2) \\
\mathbb{E}[x_2^2] &= \mathbb{E}[\alpha^2 x_1^2] + \mathbb{E}[2\alpha x_1 \nu] + \mathbb{E}[\nu^2] = \alpha^2 (\sigma^2 + m^2) + n^2
\end{align*}
\]

And substitute these into definitions for the mean and covariance:

\[
\begin{bmatrix}
\mu \\
\Sigma
\end{bmatrix} =
\begin{bmatrix}
m \\
am m
\end{bmatrix},
\begin{bmatrix}
\sigma^2 & \alpha \sigma^2 \\
\alpha \sigma^2 & \alpha^2 \sigma^2 + n^2
\end{bmatrix}.
\]

I frequently write programs to check my maths on small problems. I wouldn’t normally bother with error bars here, where I can get agreement to several significant figures, which is unlikely to happen by chance. (See Probability and Computing by Mitzenmacher and Upfal for an example analysis of some random checking.) But it seemed a good opportunity to get you to think about standard errors.

Example code follows. I could estimate the mean with a simple standard error based on one batch of simulations. However, the covariance matrix isn’t a simple sum, so I ran multiple batches of simulations to put error bars on that. One could also consider jackknife or bootstrap estimates (see for example ‘All of Statistics’ by Larry Wasserman). Making careful statistical statements would require more work, especially as we are making multiple comparisons here. The standard errors just give a quick indication that our estimates vary from theory by roughly the right magnitude.

```matlab
% Matlab/Octave
% Pick numbers for a test case. You don’t have to pick these randomly,
% but don’t pick special numbers (like 0 or 1), or numbers that might
% be related to each other.
mm = randn();
sigma = abs(randn());
alpha = rand();
nn = abs(randn());

% The covariance is not just a simple sum. As a quick hack, I formed
% a bunch of unbiased estimates based on some independent trials, so I
% could check agreement based on simple standard errors.
num_trials = 1000;

D = 2;
mu_est = zeros(D, num_trials);
Sigma_est = zeros(D, D, num_trials);
N = 1e3; % estimates in each trial formed from N samples
xx = zeros(N, D);
for tt = 1:num_trials
    xx(:,1) = sigma*randn(N,1) + mm;
    xx(:,2) = alpha*xx(:,1) + nn*randn(N, 1);
    mu_est(:, tt) = mean(xx, 1);
    Sigma_est(:, :, tt) = cov(xx);
end

% If any of the numbers are consistently more than a couple of
% standard errors out, then I will worry. The estimates and the
% standard error estimates are coupled, so when one number is out by
% a lot, others are likely to be as well.
```
\[
\mu_{\text{theory}} = [\mu; \alpha*\mu]
\]
\[
\mu_{\text{est}} = \text{mean}(\mu_{\text{ests}}, 2)
\]
\[
\mu_{\text{eb}} = \text{std}(\mu_{\text{ests}}, [], 2) / \sqrt{\text{num\_trials}}
\]
\[
\mu_{\text{num\_se\_out}} = (\mu_{\text{est}} - \mu_{\text{theory}}) / \mu_{\text{eb}}
\]
\[
\Sigma_{\text{theory}} = [\sigma^2, \alpha*\sigma^2; \alpha*\sigma^2, \alpha^2*\sigma^2+\eta^2]
\]
\[
\Sigma_{\text{est}} = \text{mean}(\Sigma_{\text{ests}}, 3)
\]
\[
\Sigma_{\text{eb}} = \text{std}(\Sigma_{\text{ests}}, [], 3) / \sqrt{\text{num\_trials}}
\]
\[
\Sigma_{\text{num\_se\_out}} = (\Sigma_{\text{est}} - \Sigma_{\text{theory}}) / \Sigma_{\text{eb}}
\]

Or

```python
# Python
from numpy.random import rand, randn
import numpy as np

# Pick numbers for a test case. You don't have to pick these randomly,
# but don't pick special numbers (like 0 or 1), or numbers that might
# be related to each other.
mu = randn()
sigma = np.abs(randn())
alpha = rand()
eta = np.abs(randn())

# The covariance is not just a simple sum. As a quick hack, I formed
# a bunch of unbiased estimates based on some independent trials, so I
# could check agreement based on simple standard errors.
num_trials = 1000

D = 2
mu_ests = np.zeros((D, num_trials))
Sigma_ests = np.zeros((D, D, num_trials))
N = int(1e3) # estimates in each trial formed from N samples
xx = np.zeros((N, D))

for tt in range(num_trials):
    xx[:,0] = sigma*randn(N) + mu
    xx[:,1] = alpha*xx[:,0] + eta*randn(N)
    mu_ests[:, tt] = xx.mean(0)
    Sigma_ests[:,:, tt] = np.cov(xx, rowvar=False, ddof=1)

# If any of the numbers are consistently more than a couple of
# standard errors out, then I will worry. The estimates and the
# standard error estimates are coupled, so when one number is out by
# a lot, others are likely to be as well.
mu_theory = np.array([mu, alpha*mu])
mu_est = np.mean(mu_ests, 1)
mu_eb = np.std(mu_ests, 1) / np.sqrt(num_trials)
mu_num_se_out = (mu_est - mu_theory) / mu_eb

Sigma_theory = np.array([[sigma**2, alpha*(sigma**2)],
                         [alpha*(sigma**2), (alpha**2)*(sigma**2)+eta**2]])
Sigma_est = np.mean(Sigma_ests, 2)
Sigma_eb = np.std(Sigma_ests, 2) / np.sqrt(num_trials)
Sigma_num_se_out = (Sigma_est - Sigma_theory) / Sigma_eb
```
4. Sampling Gaussians, and matrix decompositions:

[This question has quite a lot of detail on computation and linear algebra, which you can skim over on first reading. You don’t actually need to understand the decompositions in detail, or know how to call them in Matlab or Python, to be able to answer some of the questions.]

In lectures we saw that we can sample from a multivariate Gaussian \( x \sim \mathcal{N}(0, \Sigma) \) by drawing a vector of standard normals, \( v \sim \mathcal{N}(0, I) \), and setting \( x = Av \), for a matrix \( A \) where \( AA^\top = \Sigma \).

The lower-triangular Cholesky decomposition will decompose a symmetric positive-definite covariance into \( \Sigma = LL^\top \).

Matlab/Octave: \( L = \text{chol}(\Sigma, \text{'lower'}) \);
Numpy: \( L = \text{np.linalg.cholesky}(\Sigma) \)

This decomposition can be used to draw samples, with \( A = L \) above.

A triangular decomposition makes computing most things we might want to know about a covariance quick and easy. Cholesky decompositions are widely used. We can quickly find the determinant:

\[
|L| = \prod d L_{dd} \quad \text{where} \quad |\Sigma| = |L|^2,
\]

or more frequently

\[
\log |L| = \sum d \log L_{dd}.
\]

We can also solve linear systems:

\[
L^{-1} b \quad \text{takes similar time to a matrix-vector multiply } L b.
\]

In Matlab/Octave replace \( \text{inv}(L) \cdot b \) with \( \text{L} \) and \( \text{inv}(\Sigma) \cdot b \) with \( L^{-\top}(L \cdot b) \). In Python use \( \text{scipy.linalg.solve_triangular} \), and \( \text{scipy.linalg.cho_solve} \).

a) Sometimes instead of decomposing the covariance matrix, we have the Cholesky decomposition of the precision matrix, \( \Sigma^{-1} = CC^\top \), where \( C \) is lower-triangular.

How would we use \( C \) to sample from \( \mathcal{N}(0, \Sigma) \)?

Answer:
If \( \Sigma^{-1} = CC^\top \), then \( \Sigma = C^{-\top} C^{-1} \). Thus we can use \( A = C^{-\top} \) in the sampling procedure given in the question. In code we don’t need to invert \( C \), we can quickly apply \( C^{-\top} \) using the triangular matrix \( C \). A quick check in case I’ve messed up the transposes (I mess up frequently):

% Matlab/Octave:
D = 3;
Sigma = cov(randn(3*D, D)) % Create a valid covariance
C = chol(inv(Sigma), 'lower');
xx = C\ randn(D, int(1e6));
Sigma_est = cov(xx)

% Seems right. I didn’t make error bars, as I’m not in any real doubt.

# Python version:
import numpy as np
import scipy.linalg as LA
D = 3;
Sigma = np.cov(np.random.randn(D, 3*D))
L = LA.cholesky(LA.inv(Sigma), lower=True)
xx = LA.solve_triangular(L, np.random.randn(D, int(1e6)),
trans=1, lower=True)
Sigma_est = np.cov(xx)

b) Real symmetric matrices, like covariance matrices, also have a decomposition of

\[
2. \text{We do not usually evaluate an expression } A^{-1} c \text{ by inverting } A \text{ and then multiplying } c \text{ by } A^{-1}. \text{There are faster and more numerically stable way to solve } A^{-1} c. \text{The method you should use depends on the properties of } A. \text{In common situations, Matlab’s } A \cdot c \text{ does something sensible and should be preferred to } \text{inv}(A) \cdot c. \text{But if you’ve cached a decomposition of } A, \text{you should probably make use of it.}
\]
the following form\(^3\): 
\[ \Sigma = Q\Lambda Q^\top, \]
where \( \Lambda \) is a diagonal matrix of eigenvalues, and the columns of \( Q \) are the eigenvectors of \( \Sigma \).

i) Describe how to sample from \( N(0, \Sigma) \) using this decomposition.

ii) \( Q \) is an orthogonal matrix, corresponding to a rigid rotation (and possibly a reflection). Describe geometrically (perhaps in 2D) how your sampling process transforms a cloud of points drawn from a standard normal.

Answer:

i) Here we split up \( \Lambda = \Lambda^{1/2}\Lambda^{1/2} \), by taking the square-root of each eigenvalue on the diagonal. Noting that a diagonal matrix is symmetric, we can write: 
\[ \Sigma = Q\Lambda^{1/2}\Lambda^{1/2}Q^\top = (Q\Lambda^{1/2})(Q\Lambda^{1/2})^\top. \]
So we can set \( A = Q\Lambda^{1/2} \) in the sampling procedure given in the question.

ii) We can compute each \( Q\Lambda^{1/2}\nu \) by first applying \( \Lambda^{1/2} \), which scales each dimension of \( \nu \) independently. The circular/spherical cloud of points is stretched by \( \sqrt{\Lambda_{dd}} \) along the \( d \)th axis (for each \( d \)) and becomes an elliptical cloud of points. (In 3D think of something like a squashed rugby ball or American football.) The tips, the axes of the ellipse, point along the coordinate axes. We then apply a rotation \( Q \). The cloud keeps its same elliptical shape, but now doesn’t point along the axes. In fact the axes of the ellipse point along the eigenvectors stored in the columns of \( Q \).

c) Yet another possible decomposition is the principal square root\(^4\): 
\[ \Sigma = \Sigma^{1/2}\Sigma^{1/2}, \]
where \( \Sigma^{1/2} \) is symmetric. None of the decompositions discussed so far are the same. In this part we’ll try to understand how they’re related.

i) Consider two different decompositions \( \Sigma = AA^\top = BB^\top \). We’ll assume the matrices are full rank so that we can write \( B = AU \). Show that \( UU^\top = I \), the identity matrix, which means that \( U \) is an orthogonal matrix.

ii) Explain geometrically why if computing \( Av \) from \( v \sim N(0, I) \) is a way to sample from \( N(0, \Sigma) \), computing \( Bv = AUv \) will be as well.

Answer:

i) \( B = AU \) means \( U = A^{-1}B \), which we substitute into 
\[ UU^\top = A^{-1}B(A^{-1}B)^\top = A^{-1}(BB^\top)A^{-\top} = A^{-1}\Sigma A^{-\top} = A^{-1}AA^\top A^{-\top} = I. \]
ii) If \( \nu \) comes from a spherically symmetric distribution, then rotating that distribution (applying an orthogonal transformation) does nothing. Individual points move, but the distribution over outcomes \( z = U \nu \) is still \( N(0, I) \). Thus whether or not we apply any orthogonal transformation \( U \) to our standard normal draws before applying \( A \) makes no difference to the distribution defined by the procedure.