6.867 Fall 2016 : Homework 0

This is a set of warm-up problems, divided into two parts. *There is no need to turn in this homework. We will post solutions.*

Problems in the first part constitute a quick diagnostic, to see if your background in linear algebra and probability is adequate to do well in the course. If these problems are not fairly easy for you, then you should probably take the relevant courses this term and take 6.867 next year.

Problems in the second part are intended to help you learn good "vectorized" programming strategies, which will help you make efficient implementations of algorithms we study in class and also to interface with existing libraries. We don't expect you to know how to do this already, but ask you to spend some time in the first week learning how to form elegant, efficient solutions to these problems. You are welcome to use MATLAB or numpy/scipy.

1 Math background problems

1.1 Just plane fun

Consider a hyperplane in n-dimensional Euclidean space, described by the $n+1$ real values w_i for $i = 0, \ldots, n$: the hyperplane consists of points $(x_1, \ldots, x_n)$ satisfying

$$w_0 + w_1 x_1 + \ldots + w_n x_n = 0 \ .$$

1. Find a unit vector normal to the hyperplane. Given a point $(v_1, \ldots, v_n)$ on the hyperplane, give the equation for the line through the point that is orthogonal to the hyperplane.

Solution: Let $\boldsymbol{w} = (w_1, \ldots, w_n)$, then $\boldsymbol{w}/\|\boldsymbol{w}\|$ is a unit vector normal to the hyperplane. To prove this, given any two points $\boldsymbol{x}$ and $\boldsymbol{x}'$ on the hyper plane, we will show that $\boldsymbol{w}$ is perpendicular to $\boldsymbol{x} - \boldsymbol{x}'$. Note

$$\boldsymbol{w}^T(\boldsymbol{x} - \boldsymbol{x}') = \sum_{i=1}^{n} w_i x_i - \sum_{i=1}^{n} w_i x_i' = (-w_0) - (-w_0) = 0$$

Therefore, all the points on the line through the point $\boldsymbol{v} = (v_1, \ldots, v_n)$ orthogonal to the hyperplane could be represented as $\boldsymbol{x} = \boldsymbol{v} + t\boldsymbol{w}$, where $t \in \mathbb{R}$ is a parameter. The equation

of this line could be written as

$$\boldsymbol{A}^T(\boldsymbol{x}-\boldsymbol{v})=\boldsymbol{0}$$

where matrix $\boldsymbol{A}=(\boldsymbol{a}_1,\ldots,\boldsymbol{a}_{n-1})\in\mathbb{R}^{n\times(n-1)}$ is a basis for the subspace associated with the hyperplane. This equation means that for any vector $\boldsymbol{x}$ on this line, $\boldsymbol{x}-\boldsymbol{v}$ must be orthogonal to all the basis vectors in the hyperplane. The choice of $\boldsymbol{A}$ is not unique. One concrete way of getting it is: let $\boldsymbol{e}_1,\ldots,\boldsymbol{e}_n$ be standard basis for $\mathbb{R}^n$. Choose $n-1$ from them by first taking all $\boldsymbol{e}_i$ such that $\boldsymbol{e}_i^T\boldsymbol{w}=0$ (note there are at most $n-1$ of them); the rest of them could be taken arbitrarily. Denote the chosen vectors by $\tilde{\boldsymbol{e}}_1,\ldots,\tilde{\boldsymbol{e}}_{n-1}$, then

$$\boldsymbol{a}_i=\tilde{\boldsymbol{e}}_i-\left(\tilde{\boldsymbol{e}}_i^T\boldsymbol{w}\right)\boldsymbol{w},\quad i=1,\ldots,n-1$$

2. Given a point $(v_1,\ldots,v_n)$, how can you determine which side of the hyperplane it is on?

Solution: Let $f(\boldsymbol{x})=w_0+\boldsymbol{w}^T\boldsymbol{x}=w_0+w_1x_1+\cdots+w_nx_n$. Then $f(\boldsymbol{x})=0$ means $\boldsymbol{x}$ is on the hyperplane; $f(\boldsymbol{x})>0$ means it is on one side of the hyperplane, while $f(\boldsymbol{x})<0$ means it is on the other side of the hyperplane.

3. What is the distance of a point $(v_1,\ldots,v_n)$ to the hyperplane?

Solution: Assume $\bar{\boldsymbol{v}}$ is the projection of $\boldsymbol{v}=(v_1,\ldots,v_n)$ on the hyperplane. Since $\boldsymbol{w}/\|\boldsymbol{w}\|$ is the unit normal vector of the hyperplane, we know that

$$\boldsymbol{v}=\bar{\boldsymbol{v}}+t\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$$

where $|t|$ is the distance of $\boldsymbol{v}$ to the hyperplane. We also know $f(\bar{\boldsymbol{v}})=0$ by the previous question, that means

$$0=f(\bar{\boldsymbol{v}})=f\left(\boldsymbol{v}-t\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}\right)=w_0+\boldsymbol{w}^T\boldsymbol{v}-t\|\boldsymbol{w}\|$$

Therefore, the distance is given by

$$|t|=\left|\frac{w_0+\boldsymbol{w}^T\boldsymbol{v}}{\|\boldsymbol{w}\|}\right|=\frac{|f(\boldsymbol{v})|}{\|\boldsymbol{w}\|}$$

4. Consider the halfspace $H = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{w}^T\boldsymbol{x} \leq b\}$. Find a nearest (in Euclidean distance) point in H to a given vector $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)$. Is this point unique?

Solution: We need to solve the problem: $\min_{\boldsymbol{x}\in H} \|\boldsymbol{x} - \boldsymbol{v}\|^2$. Using a Lagrange multiplier $\lambda \geq 0$ for the constraint $\boldsymbol{w}^T\boldsymbol{x} \leq b$ we consider

$$L(\boldsymbol{x}, \lambda) = \tfrac{1}{2}\|\boldsymbol{x} - \boldsymbol{v}\|^2 + \lambda(\boldsymbol{w}^T\boldsymbol{x} - b).$$

Now solve $\partial L/\partial \boldsymbol{x} = 0$ to obtain $\boldsymbol{x} - \boldsymbol{v} + \lambda\boldsymbol{w} = 0$, so that $\boldsymbol{x} = \boldsymbol{v} - \lambda\boldsymbol{w}$. If $\boldsymbol{v} \in H$, then we can set $\lambda = 0$. Otherwise, any solution $\boldsymbol{x}$ must lie on the boundary of the halfspace H, and satisfy $\boldsymbol{w}^T\boldsymbol{x} = b$. From $\boldsymbol{x} = \boldsymbol{v} - \lambda\boldsymbol{w}$ it follows that $\boldsymbol{x}^T\boldsymbol{w} = \boldsymbol{v}^T\boldsymbol{w} - \lambda\boldsymbol{w}^T\boldsymbol{w}$, which gives $\lambda = \frac{1}{\|\boldsymbol{w}\|^2}(\boldsymbol{v}^T\boldsymbol{w} - b)$. We can thus write the solution (it is unique) as

$$\boldsymbol{x} = \boldsymbol{v} - \frac{1}{\|\boldsymbol{w}\|^2}\max(0, \boldsymbol{v}^T\boldsymbol{w} - b)\boldsymbol{w}.$$

1.2 Multivariate Gaussian

Let X be a random variable taking values in $\mathbb{R}^n$. It is normally distributed with mean μ and covariance matrix Σ.

1. Write the probability density function (pdf) $p_X(\boldsymbol{x})$ for X.

Solution:

$$p_X(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}}\exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right)$$

2. Show how to obtain the normalization constant for the multivariate Gaussian.

Solution: We know that $p_X(\boldsymbol{x}) \propto \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right)$. To obtain the normalization constant we need to evaluate the integral

$$\int_{\mathbb{R}^n} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right) d\boldsymbol{x}.$$

Observe that wlog we can assume $\boldsymbol{\mu} = 0$. Thus, it remains to evaluate

$$\int_{\mathbb{R}^n} \exp\left(-\tfrac{1}{2}\boldsymbol{x}^T\Sigma^{-1}\boldsymbol{x}\right) d\boldsymbol{x} \tag{1}$$

Let $\boldsymbol{y} = \Sigma^{-1/2}\boldsymbol{x}$, then by Jacobian transformation (1) can be rewritten as

$$\begin{aligned}\int_{\mathbb{R}^n} \exp\left(-\tfrac{1}{2}\boldsymbol{y}^T\boldsymbol{y}\right)|\Sigma^{1/2}|d\boldsymbol{y} &= |\Sigma^{1/2}|\prod_{i=1}^{n}\int_{\mathbb{R}}\exp\left(-\tfrac{1}{2}y_i^2\right)dy_i \\ &= |\Sigma^{1/2}|\prod_{i=1}^{n}\sqrt{2\pi}\end{aligned}$$

We use the fact that $\int_{\mathbb{R}}\exp(-\frac{1}{2}x^2) = \sqrt{2\pi}$, then using the fact that $\int p(\boldsymbol{x}) = 1$, we conclude that the normalization constant is $\frac{1}{\sqrt{(2\pi)^n|\Sigma|}}$.

3. Let $Y = 2X$. What is the pdf of Y?

Solution: By the rule of probability density function for change of variable,

$$\begin{aligned}p_Y(\boldsymbol{y}) &= \left(\frac{1}{2}\right)^n \frac{1}{\sqrt{(2\pi)^n|\Sigma|}}\exp\left(-\frac{1}{2}\left(\frac{\boldsymbol{y}}{2}-\mu\right)^T\Sigma^{-1}\left(\frac{\boldsymbol{y}}{2}-\mu\right)\right) \\ &= \frac{1}{\sqrt{(2\pi)^n|4\Sigma|}}\exp\left(-\frac{1}{2}\left(\boldsymbol{y}-2\mu\right)^T\left(4\Sigma^{-1}\right)\left(\boldsymbol{y}-2\mu\right)\right)\end{aligned}$$

So Y is still normally distributed, with mean 2μ and covariance matrix 4Σ.

4. What can we say about the distribution of X if Σ is the identity matrix, I? Does this imply anything about factorization of the pdf?

Solution: When Σ is the identity matrix,

$$p_X(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^n}}\exp\left(-\frac{1}{2}\sum_{i=1}^{n}(x_i-\mu_i)^2\right) = \prod_{i=1}^{n}\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}(x_i-\mu_i)^2\right)$$

The factorization of the pdf implies that $X_1, \ldots, X_n$ are now independent random variables.

5. What can we say about the distribution of X if Σ is $5I$?

Solution: $X_1, \ldots, X_n$ are still independent. In general, let $\sigma \neq 0$ be any real number, if $\Sigma = \sigma^2 I$, then

$$p_X(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^n \sigma^{2n}}} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i - \mu_i)^2\right) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x_i - \mu_i)^2\right)$$

6. What can we say about the distribution of X if Σ is $((10, 0), (0, 1))$?

Solution: Although the covariance matrix is not (scaled) identity matrix, since the off-diagonal elements are zero, so the two components of the 2D random vector are still independent and the pdf factorizes. Specifically, we have

$$p_X(\boldsymbol{x}) = \frac{\exp\left(-\frac{1}{2\times 10}(x_1 - \mu_1)^2\right)}{\sqrt{2\pi \times 10}} \cdot \frac{\exp\left(-\frac{1}{2}(x_2 - \mu_2)^2\right)}{\sqrt{2\pi}}$$

7. Approximately what shape do equiprobability contours (i.e., sets $\{x \in R^n : p_X(x) = c\}$ for some c) of this distribution have?

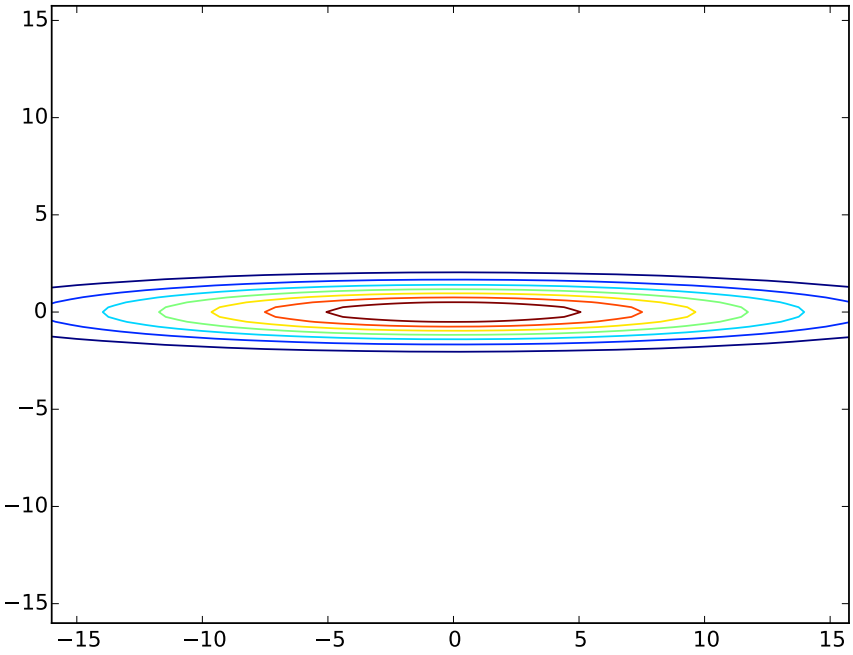
Solution: Let

$$c = p_X(\boldsymbol{x}) = \frac{\exp\left(-\frac{1}{2\times 10}(x_1 - \mu_1)^2\right)}{\sqrt{2\pi \times 10}} \cdot \frac{\exp\left(-\frac{1}{2}(x_2 - \mu_2)^2\right)}{\sqrt{2\pi}}$$

Via simple manipulation, we get the equation

$$\frac{1}{10}(x_1 - \mu_1)^2 + (x_2 - \mu_2)^2 = C$$

where $C > 0$ is a constant. This is a ellipse in $\mathbb{R}^2$ centered at μ. Specifically, the contours look like those:

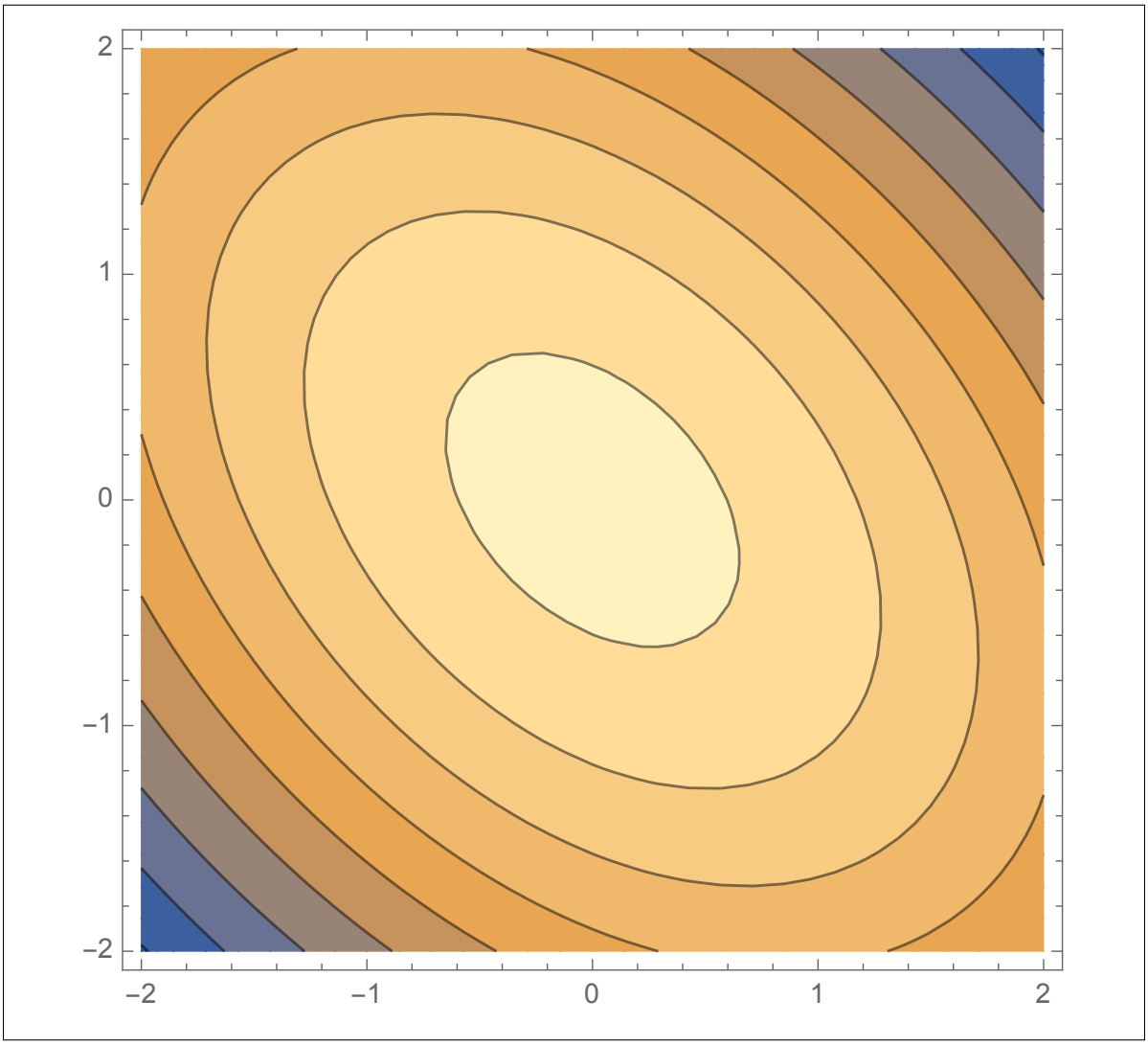


8. What can we say about this distribution if Σ is $((10, -4), (-4, 10))$?

Solution: This is a 2D normally distributed random vector. The two components are (negatively) correlated.

9. Approximately what shape do equiprobability contours of this distribution have?

Solution: Similarly, let $p_X(\boldsymbol{x}) = c$, we will get an equation of an ellipse. The contours for this specific case look like those:



10. Is $((2, 10), (10, 2))$ a valid Σ? How can you tell?

Solution: No, because it is not positive definite.

11. If $n = 2$, $\mu = (1, 2)$, and $\Sigma = ((10, 0), (0, 1))$, what is the conditional pdf $p_{X_1|X_2}(x_1 \mid 3)$?

Solution: Note there is no correlation between X_1 and X_2, so they are independent. In this case, conditioning on X_2 does not change the distribution of X_1, which is a normally distributed random variable with mean 1 and variance 10.

2 Programming problems

Try your best to avoid writing your own `for` loops!!

2.1 Regularization

Given an $n \times n$ matrix C, add a scalar a to each diagonal entry of C.

Solution:

Matlab:
Solution 1:

```
C = C + a * eye(n);
```

Solution 2:

```
idx = sub2ind([n n], 1:n, 1:n);
C(idx) = C(idx) + a;
```

Python:
Solution 1:

```
C = C + a * np.eye(n);
```

Solution 2:

```
idx = np.ravel_multi_index((range(n), range(n)), C.shape)
C.flat[idx] += a
```

While a little bit more complicated, solution 2 is much more efficient than solution 1 when n is large.

2.2 Largest Off-diagonal Element

Given an $n \times n$ matrix A, find the largest off-diagonal element.

Solution:

Matlab:
Solution 1 (this only outputs the maximum value):

```
msk = (eye(n) == 0);  %% creates a mask of off-diagonal elements.
v = max(A(msk));
```

Solution 2 (this also outputs the location of the maximum):

```
I = (1:n)' * ones(1, n);
J = I';
od_idx = find(I ~= J);   %% linear indices of all off-diagonal elements
i_od = I(od_idx);
j_od = J(od_idx);
v_od = V(od_idx);
[v, k] = max(v_od);
i = i_od(k);
j = j_od(k);
```

Python:
Solution 1 (this only outputs the maximum value):

```
msk = (np.eye(n) == 0)
v = np.max(A[msk])
```

Solution 2 (this also outputs the location of the maximum):

```
I = np.tile(np.array(range(n)).T, (n, 1)).transpose()
J = I.transpose()
idx = np.argwhere(I != J)
idx_r = [np.ravel_multi_index(x, A.shape) for x in idx]
v = np.max(A.flat(idx_r))
max_loc = np.argmax(A.flat[idx_r])
i, j = np.unravel_index(max_loc, A.shape)
```

2.3 Pairwise Computation

Given a vector x of length m, and a vector y of length n, compute $m \times n$ matrices: A and B, such that $A(i,j) = x(i) + y(j)$, and $B(i,j) = x(i) \cdot y(j)$.

Solution:

Matlab:

```
% Preparation: make sure $x$ is a column, and $y$ is a row.
if size(x, 2) > 1;  x = x.'; end
if size(y, 1) > 1;  y = y.'; end

% To compute $A$:
A = bsxfun(@plus, x, y);

% To compute $B$:
B = x * y;
```

Python:

```
# Checks are unnecessary, arrays do not have an orientation, they are all
# column arrays
A = x[:, np.newaxis] + y[np.newaxis, :]
B = x[:, np.newaxis] * y[np.newaxis, :]
```

2.4 Pairwise Euclidean Distances

Given a $d \times m$ matrix X, and a $d \times n$ matrix Y, compute an $m \times n$ matrix D, such that $D(i,j) = \|x^i - y^j\|^2$, where x^i is the i-th column of X, and y^j is the j-th column of Y.

Solution:

Matlab:
Solution 1: two-fold for-loop

```
D = zeros(m, n);   % pre-allocation is important
```

```
for j = 1 : n
    for i = 1 : m
        D(i, j) = norm(X(:,i) - Y(:,j));
    end
end
```

Solution 2: one-fold for-loop (with the help of bsxfun)

```
D = zeros(m, n);
for j = 1 : n
    Z = bsxfun(@minus, X, Y(:,j));
    D(:, j) = sum(Z.^2, 1)';
end
D = sqrt(D);
```

Solution 3: no for-loop (the best solution).

Here, we use the following decomposition:

$$\|x^i - y^j\|^2 = \sum_{k=1}^{d}(x_k^i - y_k^j)^2 = \sum_{k=1}^{d} {x_k^i}^2 + \sum_{k=1}^{d} {y_k^j}^2 - \sum_{k=1}^{d} 2x_k^i y_k^j. \tag{2}$$

There are three terms, each can be computed for all $m \times n$ pairs in batch without for-loop.

```
tx2 = sum(X.^2, 1);
ty2 = sum(Y.^2, 1);
Txy = X' * Y;
D = bsxfun(@plus, tx2', ty2) - 2 * Txy;
D = sqrt(D);
```

Python:
Solution 1: two-fold for-loop

```
import scipy as sp
D = np.zeros((m, n))
for j in range(n):
   for i in range(m):
       D[i, j] = sp.linalg.norm(X[:,i] - Y[:,j])
```

Solution 2: one-fold for-loop

```
D = np.zeros((m, n))
for j in range(n):
    Z = (X.T - Y[:,j]).T
    D[:,j] = sum(Z**2)
D = np.sqrt(D)
```

Solution 3: no for-loop (the best solution).

Here, we use the following decomposition:

$$\|x^i - y^j\|^2 = \sum_{k=1}^{d}(x_k^i - y_k^j)^2 = \sum_{k=1}^{d} {x_k^i}^2 + \sum_{k=1}^{d} {y_k^j}^2 - \sum_{k=1}^{d} 2x_k^i y_k^j. \tag{3}$$

There are three terms, each can be computed for all $m \times n$ pairs in batch without for-loop.

```
tx2 = np.sum(X**2, 0)
ty2 = np.sum(Y**2, 0)
Txy = np.dot(X.T, Y)
D = tx2[:, np.newaxis] + ty2[np.newaxis,:] - 2*Txy
D = np.sqrt(D)
```

Simple benchmark shows that solution 3 is $10x$ to $30x$ faster than solution 1 for moderate size problems.

2.5 Compute Mahalanobis Distances

The Mahalanobis distance is a measure of the distance between a point P and a distribution D, introduced by P. C. Mahalanobis in 1936. It is a multi-dimensional generalization of the idea of measuring how many standard deviations away P is from the mean of D. This distance is zero if P is at the mean of D, and grows as P moves away from the mean: Along each principal component axis, it measures the number of standard deviations from P to the mean of D. If each of these axes is rescaled to have unit variance, then Mahalanobis distance corresponds to standard Euclidean distance in the transformed space. Mahalanobis distance is thus unitless and scale-invariant, and takes into account the correlations of the data set (from `http://en.wikipedia.org/wiki/Mahalanobis_distance`). Given a center vector c, a covariance matrix S, and a set of n vectors as columns in matrix X, compute the distances of each column in X to c, using the following

formula:

$$D(i) = (x^i - c)^T S^{-1} (x^i - c). \tag{4}$$

Here, D is a row vector of length n.

Solution:

Matlab:
Solution 1: naive solution as baseline

```
D = zeros(1, n);
for i = 1 : n
    z = X(:,i) - c;
    D(i) = z' * inv(S) * z;
end
```

Not good: inversing S for n times.

Solution 2: do pre-computation

```
D = zeros(1, n);
invS = inv(S);   % do pre-computation when possible
for i = 1 : n
    z = X(:,i) - c;
    D(i) = z' * invS * z;
end
```

Solution 3: vectorization

```
Z = bsxfun(@minus, X, c);
D = dot(Z, inv(S) * Z, 1);  % 'dot' function performs dot products
                            % for each column.
```

Solution 4: directly solving linear equations is more efficient than doing inverse.

```
Z = bsxfun(@minus, X, c);
D = dot(Z, S \ Z, 1);
```

Python:
Solution 1: naive solution as baseline

```
D = np.zeros(n);
for i in range(n):
    z = X[:,i] - c;
    D[i] = np.dot(np.dot(z.T, np.linalg.inv(S)), z)
```

Not good: inversing S for n times.

Solution 2: do pre-computation

```
D = np.zeros(n);
invS = np.linalg.inv(S)
for i in range(n):
    z = X[:,i] - c;
    D[i] = np.dot(np.dot(z.T, invS), z)
```

Solution 3: vectorization

```
Z = X - c[:, np.newaxis]
invS = np.linalg.inv(S)
D = np.sum(Z.conj() * (np.dot(invS,Z)) , axis=0)
```

Solution 4: directly solving linear equations is more efficient than doing inverse.

```
Z = X - c[:, np.newaxis]
D = np.sum(Z.conj() * (np.linalg.solve(S, Z)), axis=0)
```

2.6 2-D Gaussian

Generate 1000 random points from a 2-D Gaussian distribution with mean $\mu = [4, 2]$ and covariance

$$\Sigma = \begin{pmatrix} 1 & 1.5 \\ 1.5 & 3 \end{pmatrix} \tag{5}$$

Plot the points so obtained, and estimate their mean and covariance from the data. Find the eigenvectors of the covariance matrix and plot them centered at the sample mean.

Solution:

Matlab:

```
R = mvnrnd([4, 2], [1, 1.5; 1.5, 3], 1000);
plot(R(:, 1), R(:, 2), '.');
muhat = mean(R);
R_0 = bsxfun(@minus, R, muhat);
SIGMA_hat = R_0'*R_0;
[Q, L] = eig(SIGMA_hat);
arrowline([muhat(1), muhat(1) + Q(1,1)], [muhat(2), muhat(2) + Q(2,1)]);
arrowline([muhat(1), muhat(1) + Q(1,2)], [muhat(2), muhat(2) + Q(2,2)])
hold on;
plot(R(:, 1), R(:, 2), '.r');
axis equal;
```

Python:

```
import matplotlib.pyplot as plt

R = np.random.multivariate_normal([4, 2], [[1, 1.5], [1.5, 3]], 1000)
plt.plot(*(zip(*R)), marker='.', ls='')
plt.show()

muhat = np.mean(R, 0)
R_0 = R - muhat[np.newaxis, :]
SIGMA_hat = np.dot(R_0.T, R_0)
L, Q = np.linalg.eig(SIGMA_hat)
plt.arrow(muhat[0],muhat[1], Q[0,0], Q[1,0], shape='full',
    lw=3, length_includes_head=True, head_width=.01)
plt.arrow(muhat[0],muhat[1], Q[0,1], Q[1,1], shape='full',
    lw=3, length_includes_head=True, head_width=.01)
plt.plot(*(zip(*R)), marker='.', ls='')
plt.axis([1, 6, -2, 6])
plt.show()
```