Homework 9-2: Getting comfortable with PCA

Table of Contents

(a-d) Initialize and generate data

```
D=100;  % number of dimensions
N=500; % number of datapoints
sigma = 1; % noise magnitude
axisLim = 20; % For plotting

% Generate data

rng(0); % for reproducibility
% The variances along the three dominant directions
trueSignalVars = [20 5 0.5];
% The three dominant directions -> select them randomly (and
 normalize).
% When the number of dimensions is large, any two random vectors
% are almost orthogonal!
vs = randn(D,3);
norms = sqrt(sum(vs.^2));
vs = vs./norms;

v1 = vs(:,1);
v2 = vs(:,2);
v3 = vs(:,3);

% Generate the data that looks like an ellipsoid with the main
 principal
% directions we chose.
coef = sqrt(trueSignalVars);
X = coef(1)*randn(N,1)*v1'+...
    coef(2)*randn(N,1)*v2'+...
    coef(3)*randn(N,1)*v3'+...
```

```
    sigma*randn(N,D);
```

(b) In large-dimensional spaces, random vectors are almost orthogonal

```
fprintf('Scalar products between vectors v1..v3:\n')
disp(vs'*vs)
```

```
Scalar products between vectors v1..v3:
    1.0000    0.0671    0.1187
    0.0671    1.0000   -0.1734
    0.1187   -0.1734    1.0000
```

Why do we expect random vectors to be almost orthogonal, and how much is "almost"? First, recall a useful implication of central limit theorem: if we have a long sum of *D* random terms, some positive, some negative, so each term zero on average, but each individually of order 1 -- what's the expected magnitude of the sum? The answer is $\sqrt{D}$. Indeed: this is just like a sum of *D* independent draws from a Gaussian of width 1, and when summing independent random variables, their variances add. And if the variance of the sum is *D*, then the typical value is of order $\sqrt{D}$.

Now, back to scalar products of random vectors. If $\vec{a},\ \vec{b}$ are unit vectors, then:

$$\sum_{i=1}^{D} a_i^2 = 1 \quad \Rightarrow \quad |a_i| \simeq 1/\sqrt{D}$$

and similarly for $\vec{b}$. For the scalar product we then have:

$$(\vec{a}\cdot\vec{b}) = a_1 b_1 + a_2 b_2 + \ldots + a_D b_D.$$

This is a sum of D random terms (which can be of either sign), each with average zero, and each of magnitude $\left(1/\sqrt{D}\right)^2 = 1/D$. We conclude that the expected value of the scalar product is:

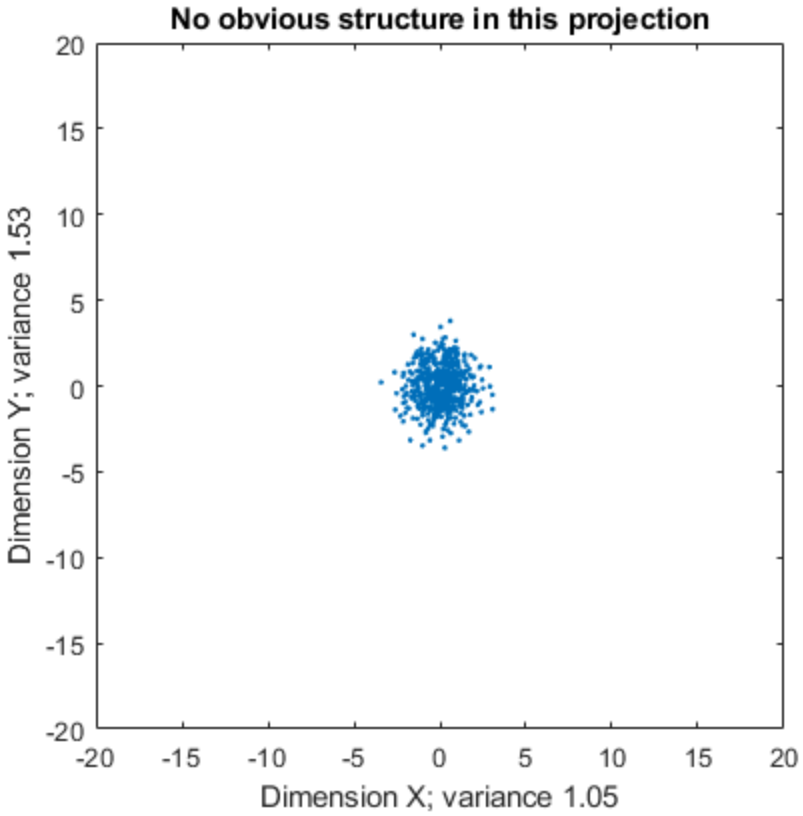
$$(\vec{a}\cdot\vec{b}) \simeq \sqrt{D}\ \frac{1}{D} = \frac{1}{\sqrt{D}}$$

In this respect the dimension 100 considered here is not actually that big, and we expect a scalar product of ~0.1. But as *D* increases, expected scalar product goes to zero.

(e) Plot in the XY plane

```
xs = X(:,1);
ys = X(:,2);
clf;
plot(xs, ys,'.')
axis(axisLim*[-1 1 -1 1]);
```

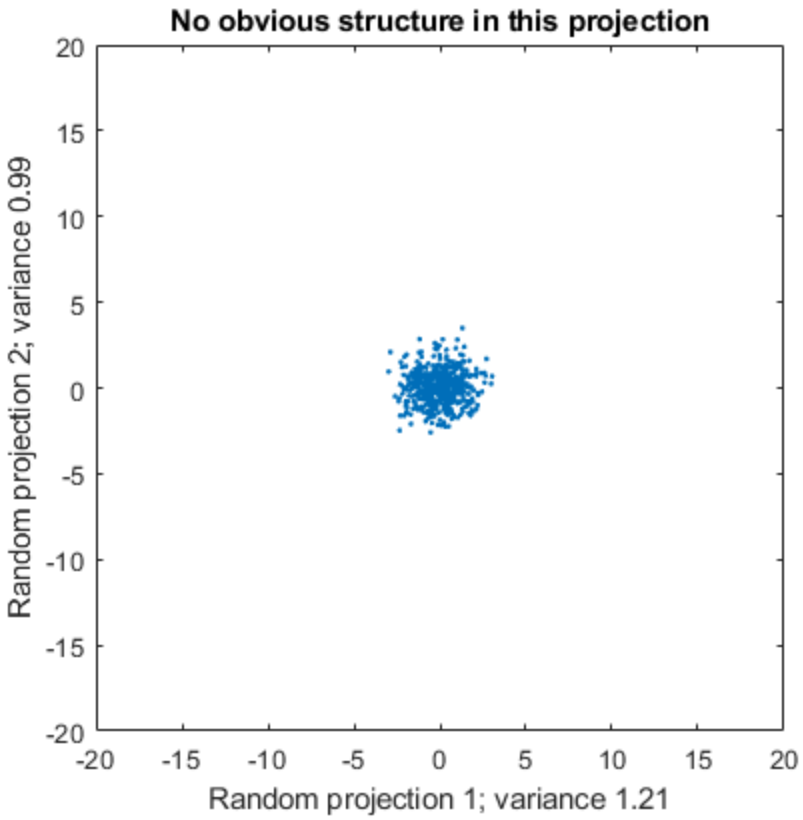
```
axis square
xlabel(sprintf('Dimension X; variance %.2f',var(xs)))
ylabel(sprintf('Dimension Y; variance %.2f',var(ys)))
title('No obvious structure in this projection')
```



(f,g) A random projection does no better:

The projection of *x* onto a unit vector *u* is given by the scalar product (x.u) (and if *u* weren't a unit vector, we would have had to divide this scalar product by u).

```
randDirs = randn(2,D);
norms = sqrt(sum(randDirs.^2,2));
randDirs = randDirs./norms;

xs = X*randDirs(1,:)';
ys = X*randDirs(2,:)';

clf;
plot(xs, ys,'.')
axis(axisLim*[-1 1 -1 1]);
axis square
xlabel(sprintf('Random projection 1; variance %.2f',var(xs)))
ylabel(sprintf('Random projection 2; variance %.2f',var(ys)))
title('No obvious structure in this projection')
```



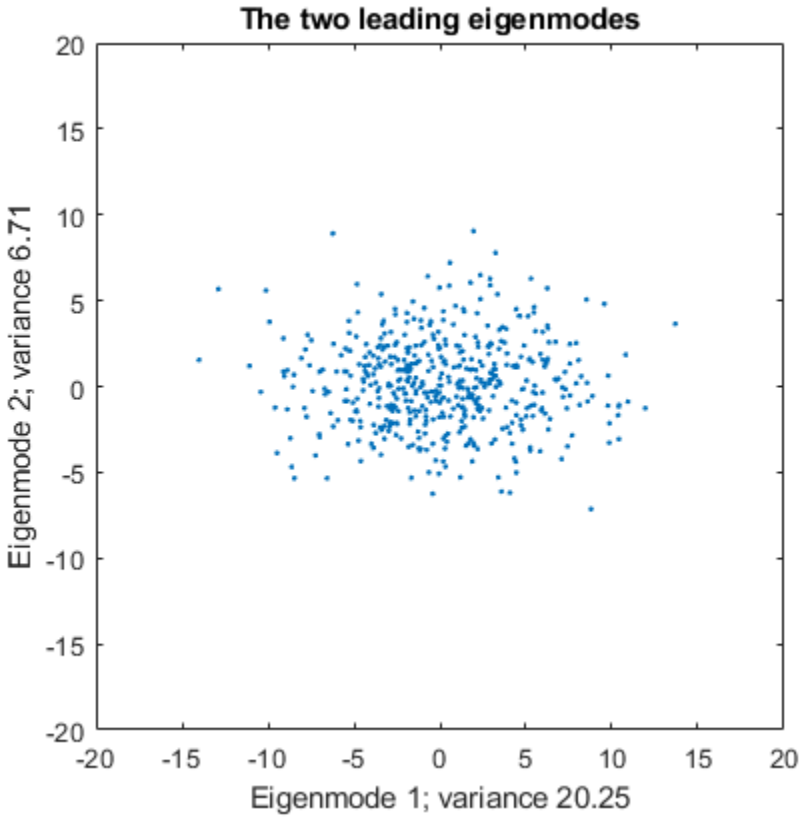
(h, i) Let's do PCA!

```
C = X'*X/N; % The correlation matrix
% Digonalize the correlation matrix, and determine the three leading
 modes
[v, lambdas] = eig(C); % The eigenvectors are the COLUMNS of V
lambdas = diag(lambdas); % The vector of eigenvalues

% Sort the eigenvalues and find the top 3
[lambdas, ord] = sort(lambdas,'descend');
v = v(:,ord);
% let's make sure the eigenmodes are normalized (this step is actually
% unnecessary, because MatLab computes _normalized_ eigenvectors)
normV = sqrt(sum(v.^2));
v = v./normV;

fprintf('The top 3 eigenvalues (expecting ~%.1f, %.1f, %.1f):\n',
 trueSignalVars+sigma^2);
disp(lambdas(1:3));
```

```
The top 3 eigenvalues (expecting ~21.0, 6.0, 1.5):
   20.2275
    6.7466
    2.0020
```

We expect these values to be close to $\simeq \lambda_{a,b,c} + \sigma^2$, because we programmed in a signal of the respective variance, and on top of that we have noise of variance σ^2 -- and once again, variances of independent variables add.

```
xInNewBasis = X*v; % Data projected onto the eigenmodes = converting
 into a new basis

eigMode123 = v(:,1:3);

% Projections of data on these directions:
p123 = X*eigMode123;

xs = p123(:,1);
ys = p123(:,2);
clf;
plot(xs, ys,'.')
axis(axisLim*[-1 1 -1 1]);
axis square
xlabel(sprintf('Eigenmode 1; variance %.2f',var(xs)))
ylabel(sprintf('Eigenmode 2; variance %.2f',var(ys)))
title('The two leading eigenmodes')
```



Some sanity checks:

```
fprintf('Total variance of data computed in two ways (should be same):
\n');
totalVar = sum(X(:).^2)
totalVar = sum(xInNewBasis(:).^2)

fprintf('First two dimensions capture only %.2g%% of this total
 variance.\n', 100*sum(sum(X(:,1:2).^2))/totalVar);
fprintf('In contrast, two eigenmodes capture %.2g%%!\n',
 100*sum(sum(xInNewBasis(:,1:2).^2))/totalVar);
```

```
Total variance of data computed in two ways (should be same):

totalVar =

   6.1947e+04


totalVar =

   6.1947e+04

First two dimensions capture only 2.1% of this total variance.
In contrast, two eigenmodes capture 22%!
```

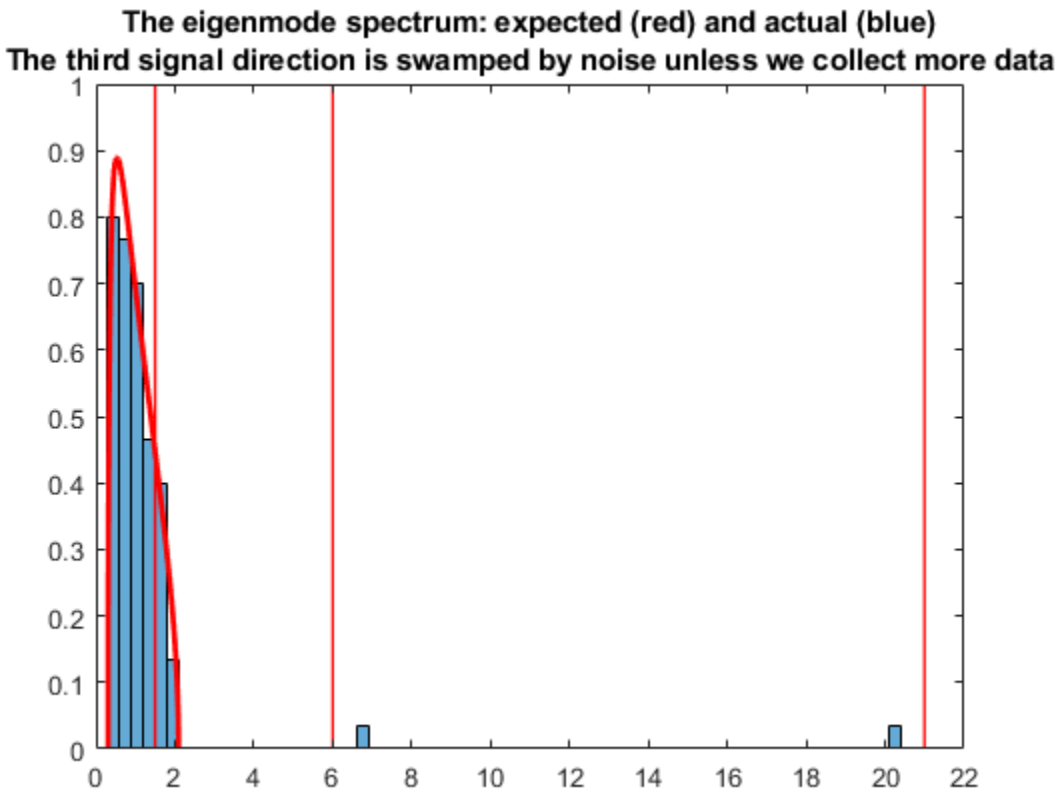
(j) How well do the recovered directions correspond to the modes we "programmed in"?

```
fprintf('Projections of recovered directions on the "true" signal:
\n');
disp(diag(eigMode123'*vs))

% For unit vectors, the scalar product is the angle between them.
 Scalar
% products 1 and -1 mean angle 0 or pi -- both mean perfect alignment
 (an
% eigenvector is only defined up to a sign anyway! -- If u is an
% eigenvector, then so is -u). We see that the first two directions
 are
% recovered near perfectly. But the third signal direction is
 receovered
% very poorly. Should we be surprised by this, or was this expected?
```

```
Projections of recovered directions on the "true" signal:
   -0.9912
    0.9723
    0.4034
```

(k) Compute the Marchenko-Pastur distribution

```
r=D/N;
lambdaMin = sigma^2*(1-sqrt(r))^2;
lambdaMax = sigma^2*(1+sqrt(r))^2;
xs = linspace(lambdaMin, lambdaMax, 1000);
dxStep = xs(2)-xs(1);
mp = 1/(2*pi*sigma^2)*sqrt((lambdaMax-xs).*(xs-lambdaMin))./(r*xs);

% Compare the eigenvalues
clf
histogram(lambdas,'BinWidth', 0.3, 'Normalization','pdf');
hold on;
plot(xs, mp, 'r-', 'LineWidth', 2);
axis([0 max([trueSignalVars+sigma^2, lambdas'])+1, 0, 1])
axLim = axis;
for i=1:length(trueSignalVars)
    plot((trueSignalVars(i)+sigma^2)*[1 1],[0
 axLim(4)],'r-','LineWidth', 1);
end
title({'The eigenmode spectrum: expected (red) and actual (blue)',...
    'The third signal direction is swamped by noise unless we collect
 more data'})
```



(l) How many datapoints to see a weak signal?

To be able to recover a weak signal like $\lambda_c = 0.5$, we must have this stand out from the the Marchenko-Pastur "bulk". Requiring that

$$\lambda_+ = \sigma^2 \left(1 + \sqrt{\frac{D}{N}}\right)^2 < \sigma^2 + 1 \quad \Rightarrow \quad \sqrt{\frac{D}{N}} < \sqrt{1 + \frac{\lambda_c}{\sigma^2}} - 1$$

In the case at hand, this gives us N ~ 2000. But to get a better feeling for how the required number of datapoints depends on various parameters, we note that if signal is weak, we can Taylor-expand the square root. This yields:

$$\sqrt{\frac{N}{D}} > \frac{2\sigma^2}{\lambda_c}$$

Published with MATLAB® R2018a