```
import numpy as np
import scipy.stats
import matplotlib.pyplot as plt
```

Estimators and data exploration

Statistics and estimators

A statistic is a function of a sample of RVs.

An *estimator* is a statistic that estimates a parameter of the population distribution that the samples are drawn from.

An estimator for population parameter θ is usually written with a hat: $\hat{\theta}(X_1, \ldots, X_n)$, for sample $X_1, \ldots, X_n$.

For example:

- Sample mean:
  - $\hat{\mu} = \bar{X} = \frac{1}{n} \sum_{i=1}^{n} X_i$
- Sample variance:
  - $\hat{\sigma}^2 = s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \bar{X})^2$

Bias

An estimator is unbiased if its expectations agrees with the population parameter:

$$\mathrm{E}[\hat{\theta}] = \theta$$

For example, the sample mean:

$$\mathrm{E}[\bar{X}] = \mathrm{E}[\frac{1}{n} \sum_i X_i] = \frac{1}{n} \sum_i \int x p_{X_i}(x) \mathrm{d}x$$

If the X_i are all from the same distribution $p_{X_i}(x) = p_X(x)$ then

$$\mathrm{E}[\bar{X}] = \frac{1}{n}\sum_i \int x p_X(x) \mathrm{d}x = \int x p_X(x) \mathrm{d}x = \mu$$

So the sample mean is an unbiased estimator of the population mean.

Variance

Another property of an estimator is its variance. For the sample mean:

$$\begin{aligned}\mathrm{Var}[\bar{X}] &= \mathrm{E}[\bar{X}^2] - \mathrm{E}[\bar{X}]^2 = \frac{1}{n^2}\sum_{ij}\mathrm{E}[X_i X_j] - \mu^2 \\ &= \frac{1}{n^2}\sum_{i=j}\mathrm{E}[X_i X_j] + \frac{1}{n^2}\sum_{i\neq j}\mathrm{E}[X_i X_j] - \mu^2\end{aligned}$$

If the X_i are iid, then $\mathrm{E}[X_i X_j] = \mathrm{E}[X_i]\,\mathrm{E}[X_j] = \mu^2$ and

$$\begin{aligned}\mathrm{Var}[\bar{X}] &= \frac{1}{n^2}\sum_{i=j}\mathrm{E}[X_i X_j] + \frac{n^2-n}{n^2}\mu^2 - \mu^2 \\ &= \frac{1}{n}\left(\mathrm{E}[X_1^2] - \mu^2\right) = \frac{1}{n}\sigma^2\end{aligned}$$

The variance of the mean therefore decreases as $\frac{1}{n}$ and the standard deviation as $\frac{1}{\sqrt{n}}$.

Note that for dependent $X_1, \ldots, X_n$ this is not the case (generally, not just for Gaussian distributed X_i):

$$\begin{aligned}\mathrm{Var}[\bar{X}] &= \mathrm{Var}[\frac{1}{n}\sum X_i] = \frac{1}{n^2}\sum_i \mathrm{Var}[X_i] + \frac{1}{n^2}\sum_{i\neq j}\mathrm{Cov}[X_i, X_j] \\ &= \frac{1}{n}\mathrm{Var}[X_1] + \frac{2}{n^2}\sum_{i>j}\mathrm{Cov}[X_i, X_j]\end{aligned} \tag{1}$$

Sampling distributions

In a Bayesian analysis, we need to know the likelihood of the data: the conditional probability of the observed data, given the parameters.

If we compress the data using a statistic, we therefore need to know the distribution of the statistic.

Even if the underlying data are normally distributed and the statistic is a simple function, the resulting distribution can become quite complicated.

We can often appeal to the central limit theorem and assume that our statistic is approximately Gaussian distributed but this assumption should be carefully checked, especially when the

averaging happens over low number of data.

Sample mean:

- For n iid $\vec{X}_i \sim \mathcal{N}(\vec{\mu}, \Sigma)$, the sample mean is also Gaussian distributed: $\hat{\vec{\mu}} \sim \mathcal{N}(\vec{\mu}, \frac{1}{n}\Sigma)$.

Sample variance

$$\hat{\sigma}^2 = s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$$

The sample variance is distributed as $\frac{\sigma^2}{\sigma^2} \sim \chi^2_{n-1}$, or written differently
$\hat{\sigma}^2 \sim \sigma^2\chi^2_{n-1} = \mathrm{Gamma}(\frac{n-1}{2}, 2\sigma^2)$, where Gamma is the Gamma distribution.

Example: angular power spectra

A scalar field on the sphere can be decomposed into spherical harmonic coefficients $a_{\ell m}$.

For a Gaussian random field these are Gaussian distributed $\Re(a_{\ell m}), \Im(a_{\ell m}) \sim \mathcal{N}(0, \frac{C_\ell}{2})$.

The estimator for the angular power spectrum is

$$\hat{C}_\ell = \frac{1}{2\ell+1}\sum_{m=-\ell}^{\ell} a^*_{\ell m} a_{\ell m}$$

From this follows that

$$\hat{C}_\ell \sim \mathrm{Gamma}\left(\frac{2\ell+1}{2}, \frac{2C_\ell}{2\ell+1}\right)$$

Covariance

The multivariate generalisation of the sample variance is the sample covariance:

$$\hat{\Sigma} = \frac{1}{n-1}\sum_{i}^{n}(\vec{X}_i - \bar{\vec{X}})(\vec{X}_i - \bar{\vec{X}})^T$$

The sample covariance is Wishart distributed.

$$\hat{\Sigma} \sim \mathcal{W}_p(\frac{1}{\nu}\Sigma, \nu)\ ,$$

where p is the size of the data vector and the number of degrees of freedom ν is $\nu = n - 1$.

The Wishart distribution is a matrix distribution, that is it assigns probabilities to matrices. It is a multivariate generalisation of the Gamma distribution.

Example: Bias of the inverse of the sample covariance

We are often in the situation where we estimate the covariance from a sample but what we actually care about is the the inverse to compute likelihoods.

If $\hat{\Sigma}$ is Wishart distributed, then $\hat{\Sigma}^{-1}$ is inverse Wishart distributed:

$$\hat{\Sigma}^{-1} \sim \mathcal{W}_p^{-1}(\nu\Sigma^{-1}, \nu)$$

Note that $\hat{\Sigma}^{-1}$ is a biased estimator of the precision matrix Σ^{-1}:

$$\mathrm{E}[\hat{\Sigma}^{-1}] = \frac{\nu}{\nu - p - 1}\Sigma^{-1} = \frac{n-1}{n-p-2}\Sigma^{-1}$$

An unbiased estimate of the inverse of the covariance is therefore $\frac{n-p-2}{n}\hat{\Sigma}^{-1}$.

The factor $\frac{n-1}{n-p-2}$ is sometimes known as the Hartlap factor in cosmology but it is just a consequence of the inverse Wishart distribution.

A cleaner treatment is to marginalise the likelihood over the distribution of the (unknown) true covariance (see Selletin et al. 2016)

Correlation

If we know little about the data, looking for correlations is often a good start. The Pearson correlation coefficient is defined as

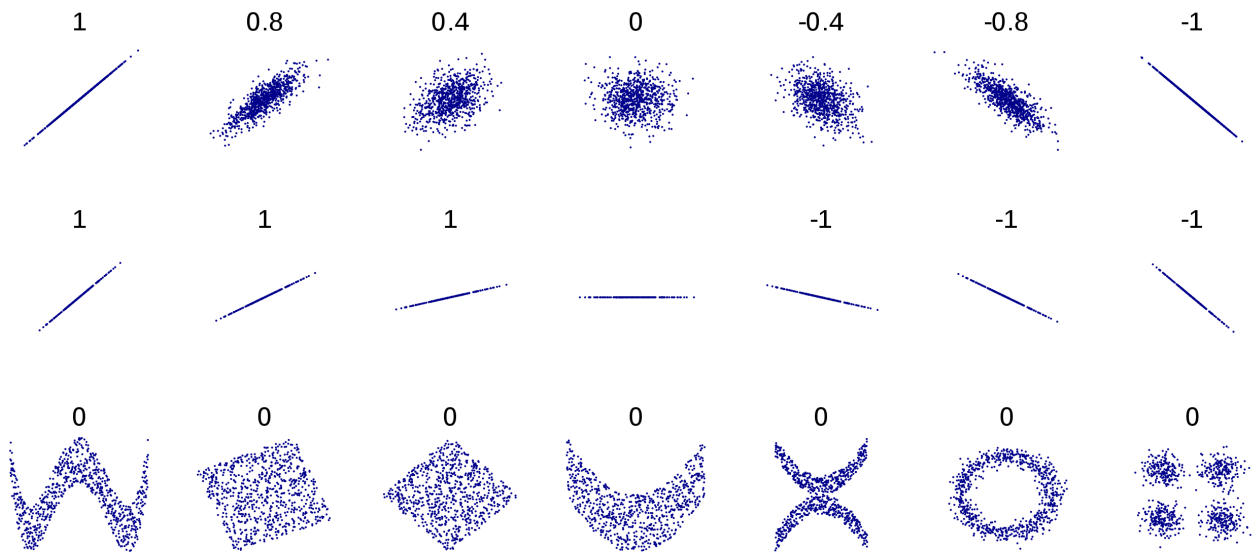
$$\rho_{X,Y} = \frac{\mathrm{Cov}[X,Y]}{\sigma_X \sigma_Y} \,.$$

It takes values between -1 and 1, with 0 being no correlation.

An estimator for the correlation is:

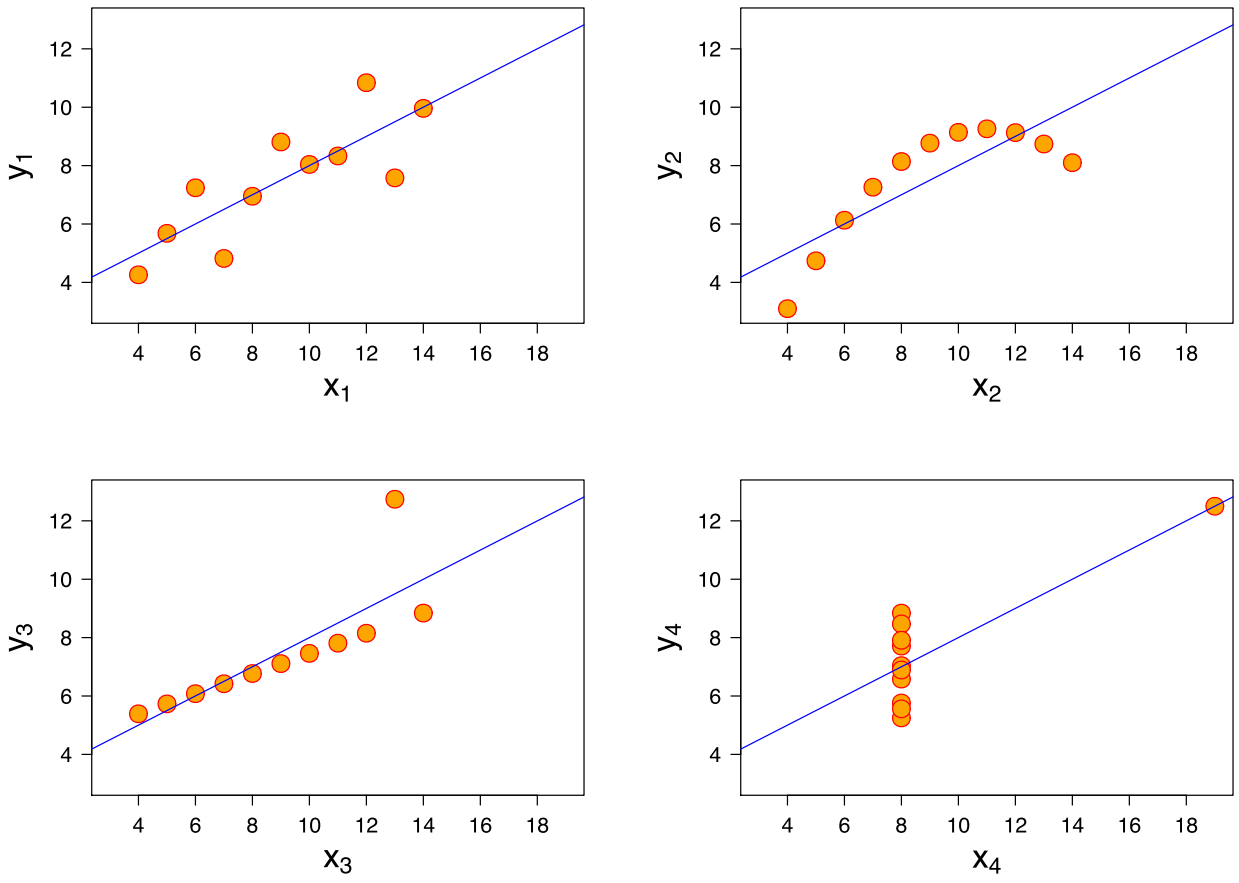
$$\hat{r} = \frac{\sum_i (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_i (X_i - \bar{X})^2}\sqrt{\sum_i (Y_i - \bar{Y})^2}}$$

Except for jointly Gaussian distributed RVs, lack of correlation does not mean lack of dependence!



by Denis Boigelot

If there is a non-linear relationship between the RVs, the correlation can also be a very insufficient statistic, as illustrated by Anscombe's quartet: all entries have the same mean, variance, and correlation



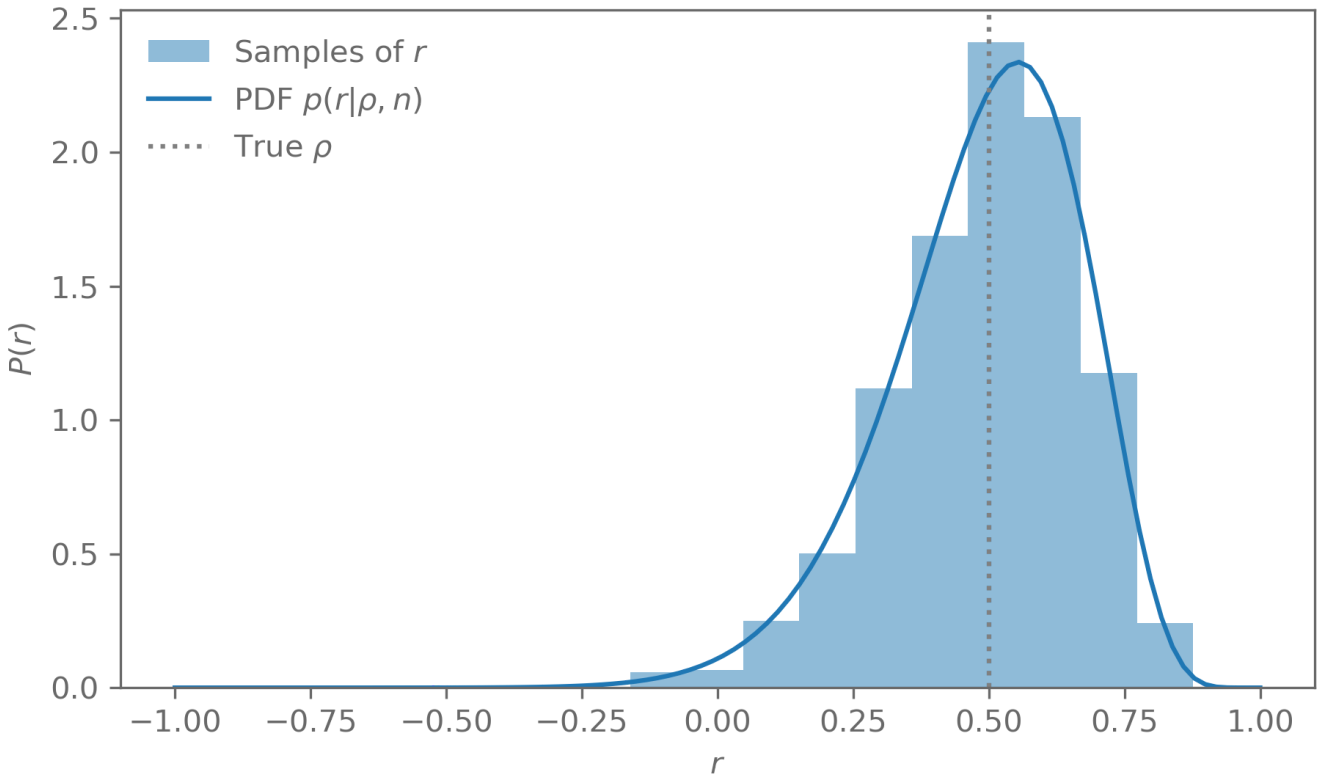
The distribution is slightly complicated, involving hypergeometric functions. But we can implement

it:

```
def pearson_r(x, y, axis=0):
    """Compute the Pearson correlation coefficient."""
    x_ = x - np.mean(x, axis=axis, keepdims=True)
    y_ = y - np.mean(y, axis=axis, keepdims=True)

    r = np.sum(x_*y_, axis=axis) / (np.sqrt(np.sum(x_**2, axis=axis))*np.sqrt(np.sum(y
    return r

def pearson_r_pdf(r, n, rho):
    """Compute the PDF of the Pearson correlation coefficient."""
    # https://mathworld.wolfram.com/CorrelationCoefficientBivariateNormalDistribution.
    hypergeom = scipy.special.hyp2f1(0.5, 0.5, (2*n-1)/2, (rho*r+1)/2)
    Gamma = scipy.special.gamma
    return (
        (n-2)*Gamma(n-1)*(1-rho**2)**((n-1)/2)*(1-r**2)**((n-4)/2)
        / (np.sqrt(2*np.pi)*Gamma(n-0.5)*(1-rho*r)**(n-3/2))
        * hypergeom
    )
```

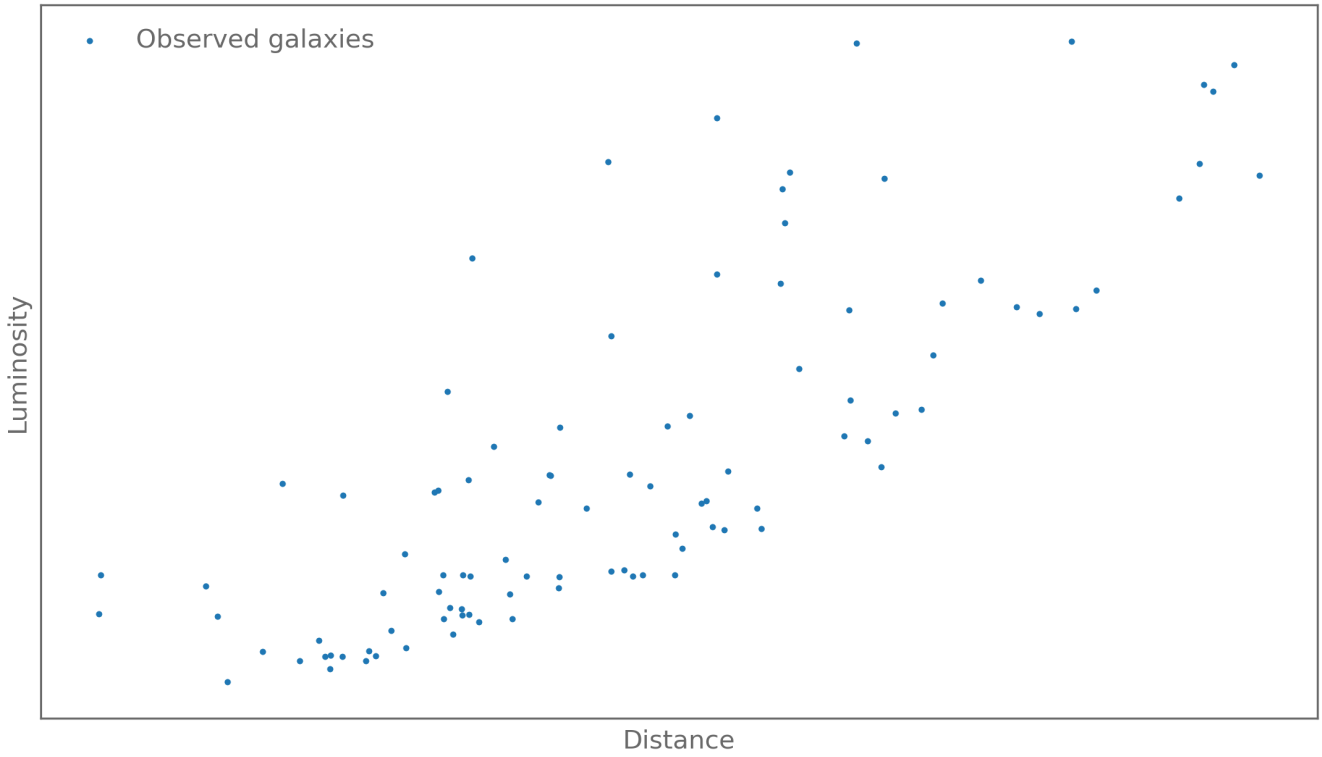
To check if the PDF is implemented correctly, let us simulate a large number of iid bivariate normal RVs and compare the histogram of the estimated Pearson correlation coefficient to the analytic PDF.

```
# Define the covariance matrix for our RVs
rho = 0.5
cov = np.array(
    [[1, rho],
     [rho, 1]]
)
# We use 20 sample to estimate the correlation coefficient from
n = 20

s = scipy.stats.multivariate_normal(cov=cov).rvs((1000, n))
# s.shape = (1000, n, 2), r.shape = (1000,)
r = pearson_r(x=s[...,0], y=s[..., 1], axis=1)
```

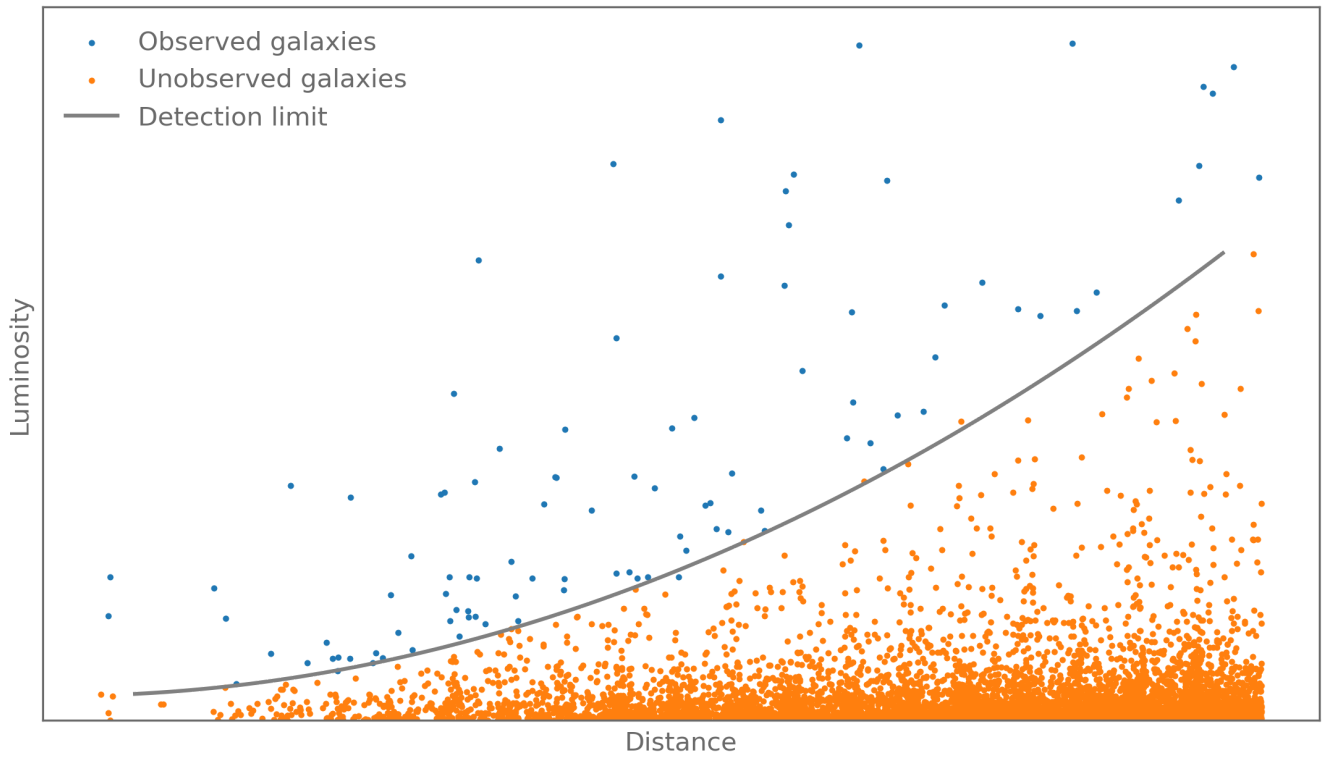


Malmquist bias

Imagine you measure the luminosities and distances of a sample of galaxies. The plot might look something like this:



Is there a correlation of galaxy luminosity with distance?



Be wary of selection effects!

Power-law distributions are common in the physical sciences but they can behave in unintuitive ways.

Exercise

Show that the sample variance $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$ is an unbiased estimator of the population variance.

Where does the factor of $\frac{1}{n-1}$ come from? What is the unbiased estimator of the population variance if the population mean μ is known?

Make a toy data set by sampling from a bivariate Gaussian. Compute the posterior of the correlation coefficient ρ, given the observed $\hat{r}$.

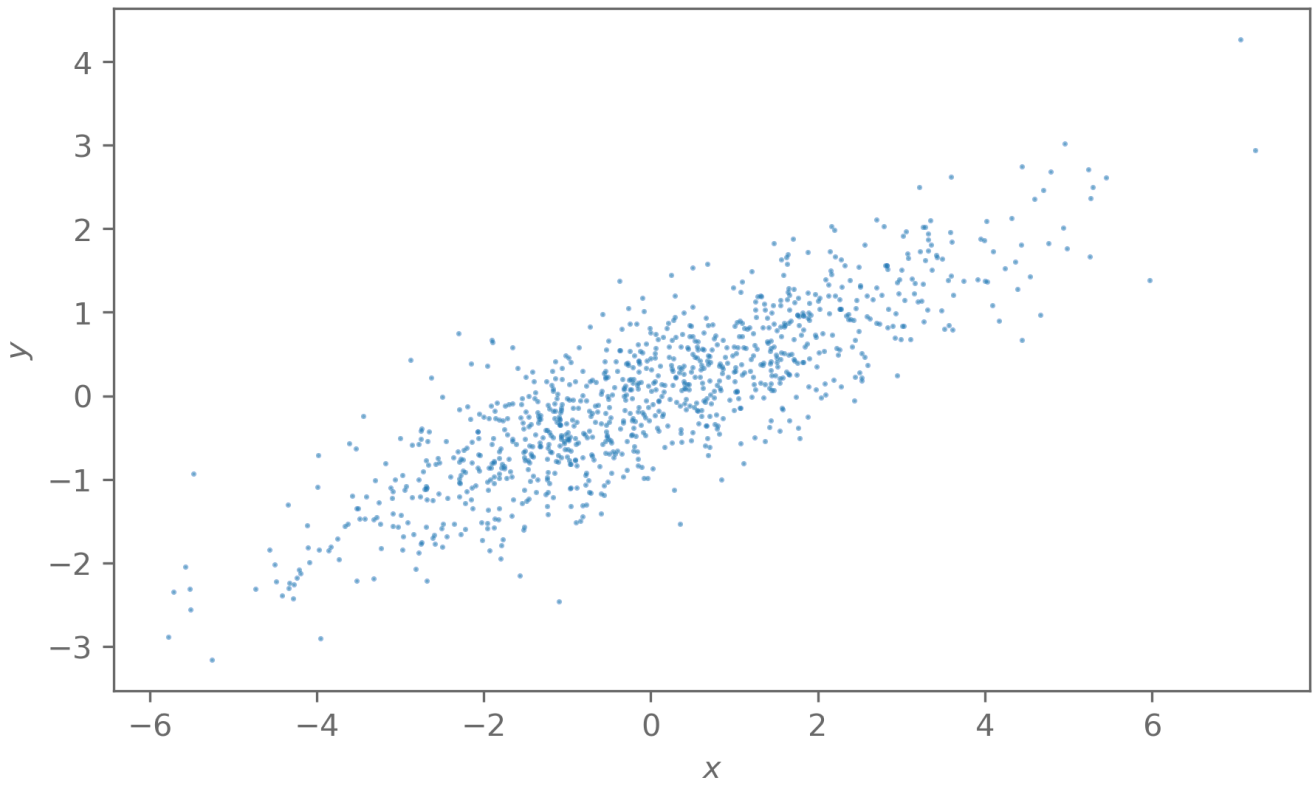
Data exploration

PCA

Principal component analysis is useful when the data has many dimensions and we want to know which of the dimensions are important, in the sense that we can reproduce the salient features of the data with a smaller number of dimensions.

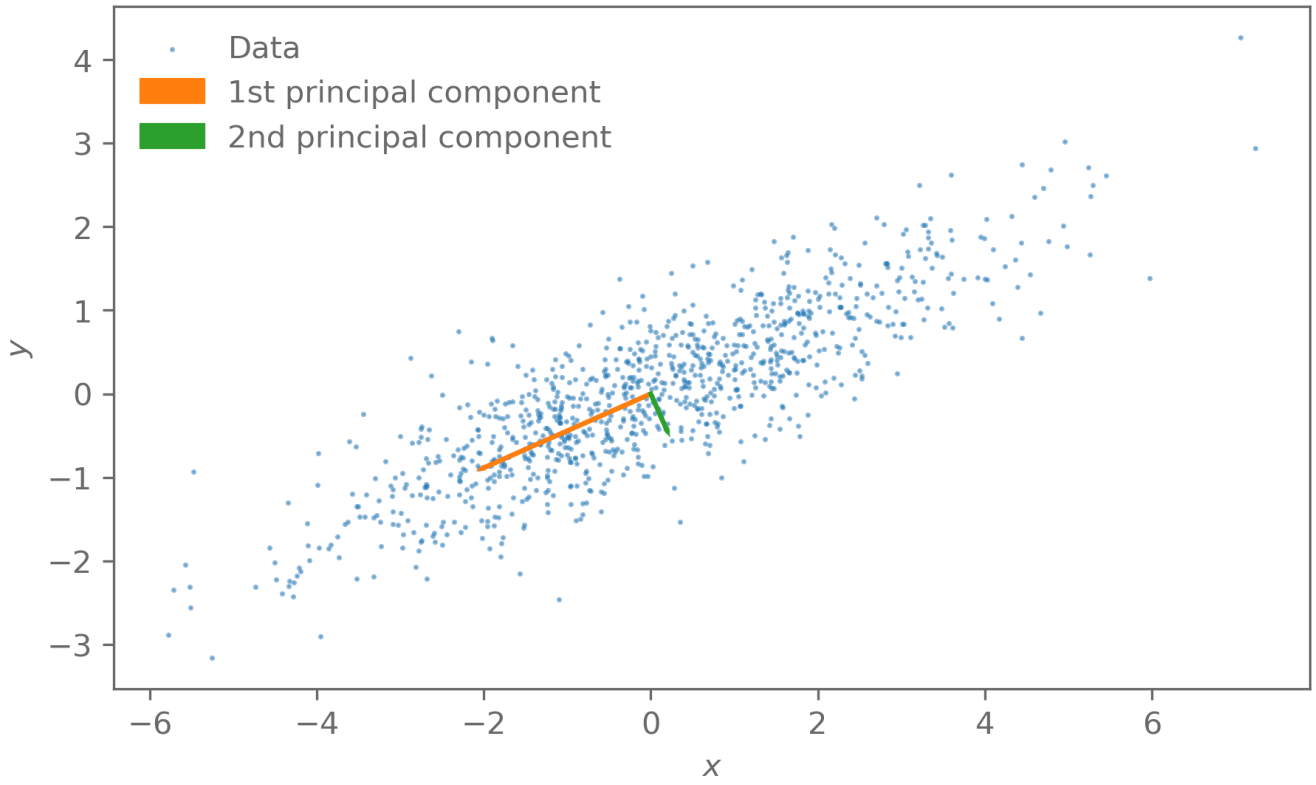
We start with n observations $\vec{X}_i$, $i = 1, \ldots, n$ with p features, that we arrange in a $n \times p$ matrix X_{ij}.

1. Remove the mean of the observations from each row $Y_{ij} = X_{ij} - \frac{1}{n} \sum_k X_{kj}$
2. Take the covariance $C = Y^T Y$
3. Find the eigenvalues λ_i and eigenvectors $\vec{E}_j$ of the covariance C
4. Sort the eigenvalues by descending order, sort the eigenvectors the same way, and arrange the eigenvectors as columns in a matrix E
5. Project the data onto the principal components (the eigenvectors) to get the principal component coefficients $T = YE$
6. Optionally project back into data-space, using only a small number of coefficients




```
def PCA(data):
    # Remove the mean
    data_mean = data.mean(axis=1)[:, None]
    X = data - data_mean
    # Get covariance and its eigenvectors
    cov_X = np.cov(X)
    l, E = np.linalg.eigh(cov_X)

    # Sort by the eigenvalues
    sort_idx = np.argsort(l)[::-1]
```

```
    l = l[sort_idx]
    E = E[:,sort_idx]

    def project_onto_pc(d):
        if d.ndim == 1:
            d = d[:, np.newaxis]
        return E.T @ (d - data_mean)

    def reconstruct_from_pc(c):
        if c.ndim == 1:
            c = c[:, np.newaxis]
        return E[:, :c.shape[0]] @ c + data_mean

    return l, E, project_onto_pc, reconstruct_from_pc

l, E, project_onto_pc, reconstruct_from_pc = PCA(samples.T)
```



For Gaussian data, the principal components capture all the information about the distribution.

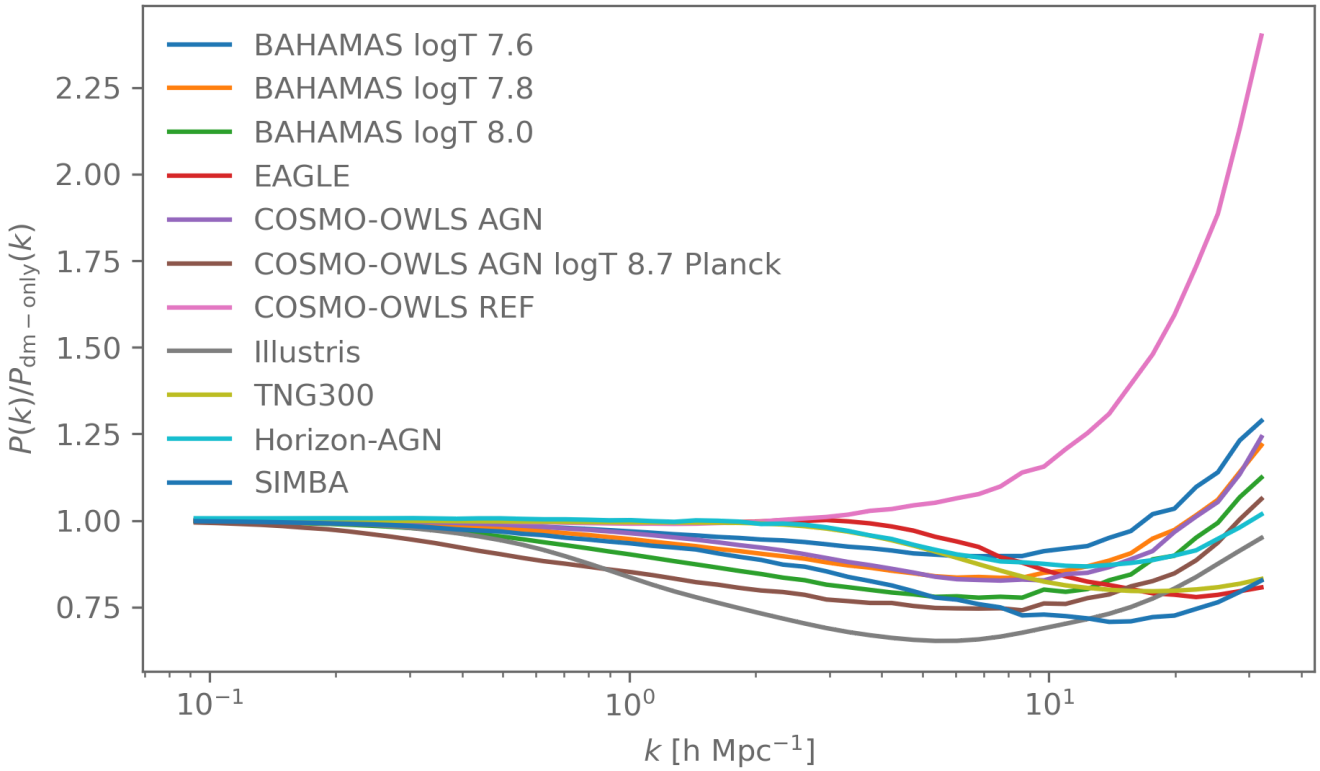
For non-Gaussian data that is not the case but PCA might still be useful for finding informative subspaces.

Example: matter power spectra

In `data/Pk_lib/power_spectrum_suppression.txt` are a list of ratios of matter power spectra from hydrodynamical simulations and the same simulations but assuming all matter is dark matter.

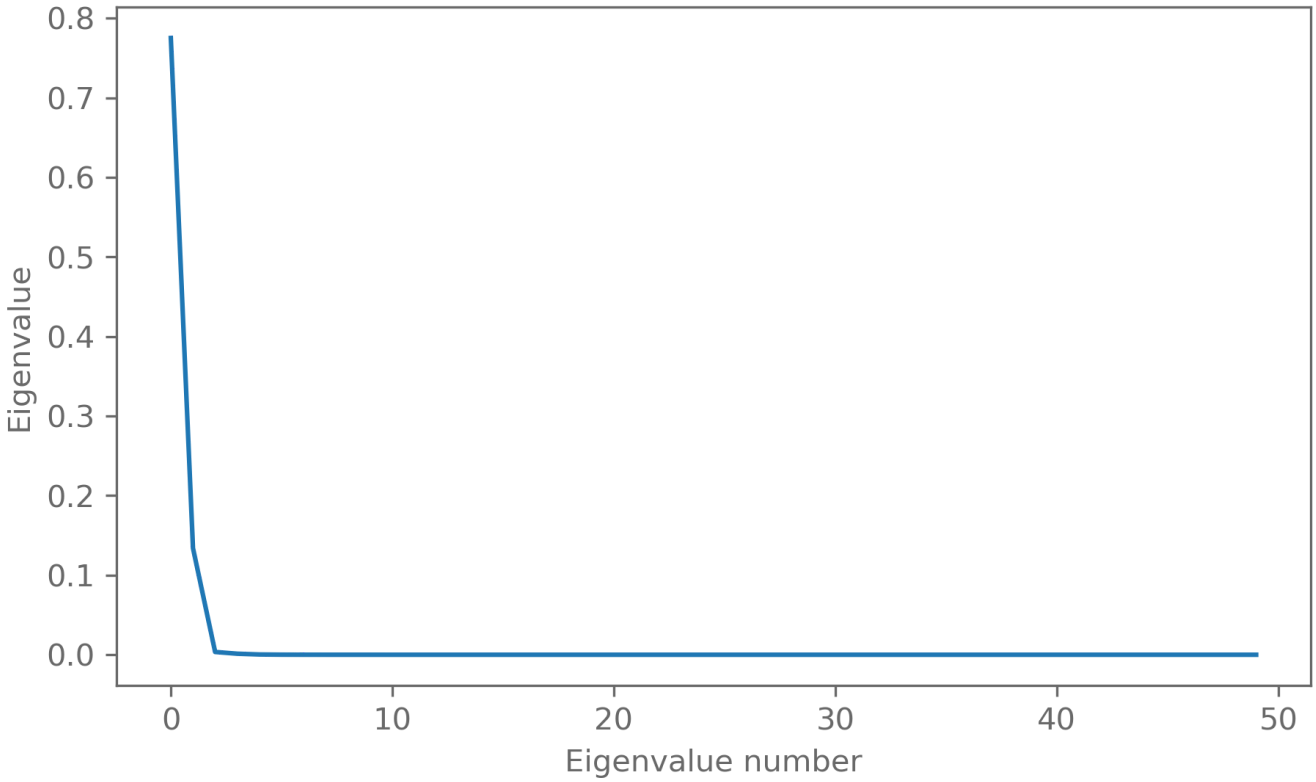
```
d = np.loadtxt("data/Pk_lib/power_spectrum_suppression.txt")
```

```
k = d[:, 0]
power_spectra = d[:, 1:]

with open("data/Pk_lib/power_spectrum_suppression.txt", "r") as f:
    names = [s.strip() for s in f.readline().lstrip("#").split(",")][1:]
```

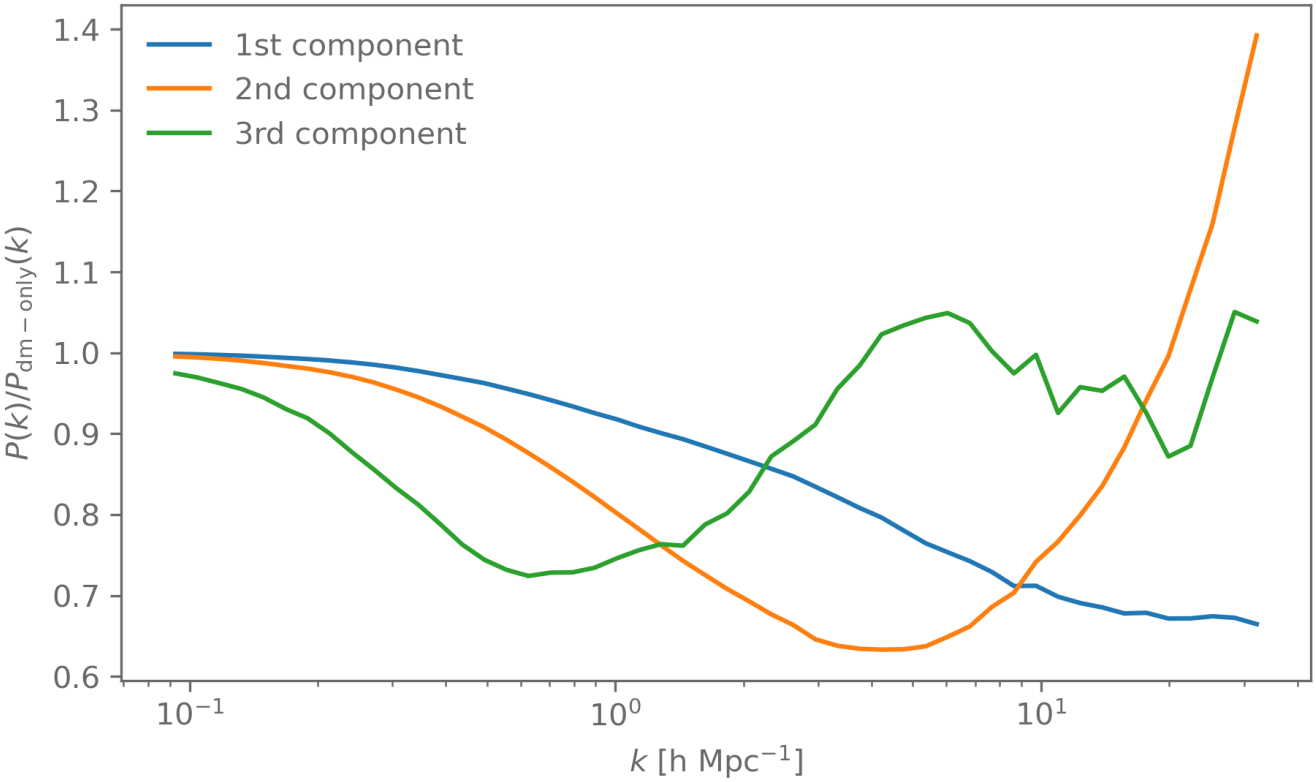



```
l, E, project_onto_pc, reconstruct_from_pc = PCA(power_spectra)
```

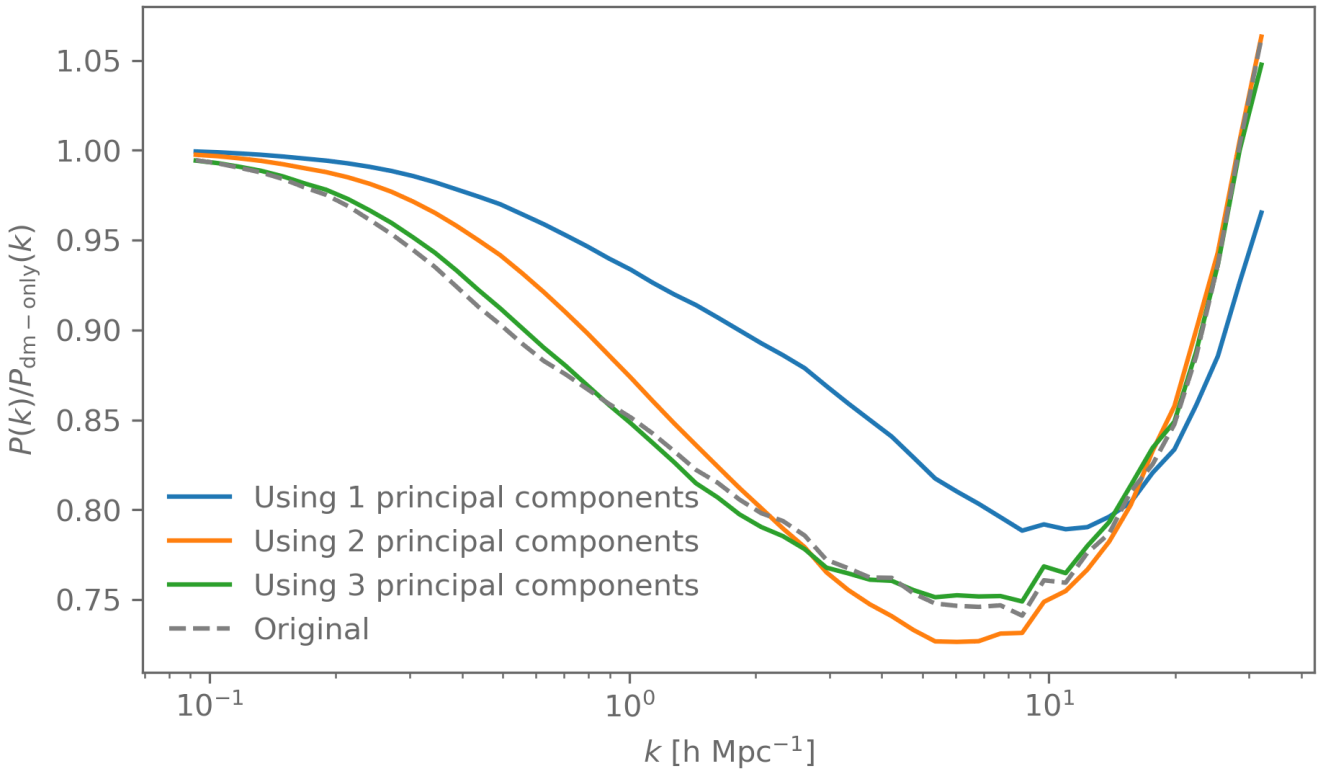
Check the eigenvalues:



What are the principal components that the data are projected onto (show in data space here)?



Project back into data space with only a few components.



Bootstrap

We are often in the situation where we have data but do not know the distribution of the statistics of the data.

Here the bootstrap comes in handy, as it allows us to approximate the sampling distribution of our statistic from the sample:

Assume we have n iid data X_i, and a statistic ϕ.

We can calculate the value of the statistic on the sample $f = \phi(X_1, \ldots, X_n)$.

What we want to know is the sampling distribution of ϕ, i.e., what values f would take if we were to sample new X_i from the true distribution of X.

1. Sample with replacement n times from the n samples X_i. Let us call these Y_i
2. Compute the statistic using the resampled data: $f^* = \phi(Y_1, \ldots, Y_n)$
3. Repeat steps 1) and 2) N times (N being large), recording the values of f_j^*, $j = 1, \ldots, N$

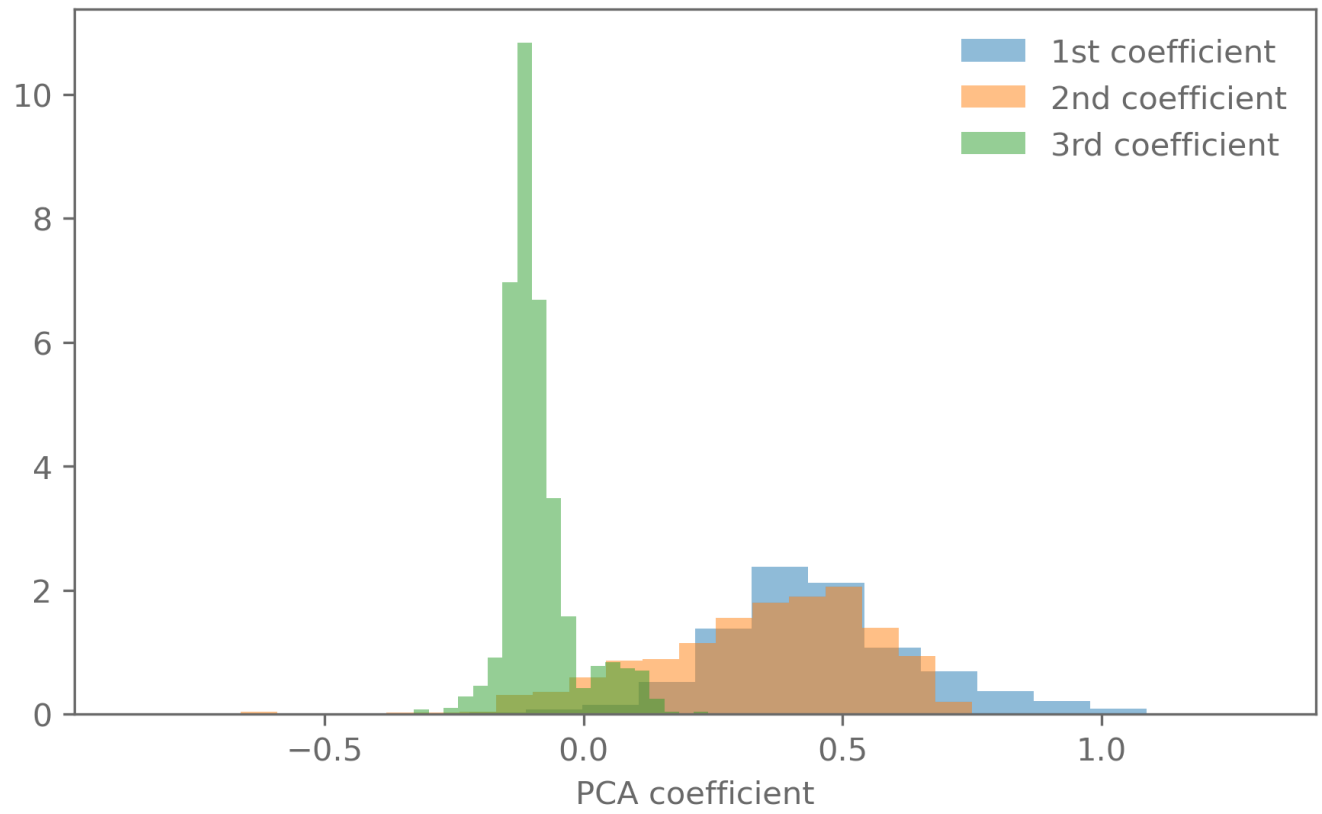
The distribution of f_j^* approximates the true sampling distribution of f.

This is a pretty amazing result. Because we seem to get something (the sampling distribution) from nothing (just the one realisation of the samples), this is called the bootstrap.

In its basic form it assumes independent data, extensions to dependent data do exist and usually revolve around resampling blocks of the data that are approximately independent.

```
def bootstrap(data, statistic, n_trial):
    n_data = data.shape[1]
    s = []
    for i in range(n_trial):
        idx = np.random.choice(n_data, size=n_data, replace=True)
        data_resampled = data[:, idx]
        s.append(statistic(data_resampled))

    return np.array(s)
```

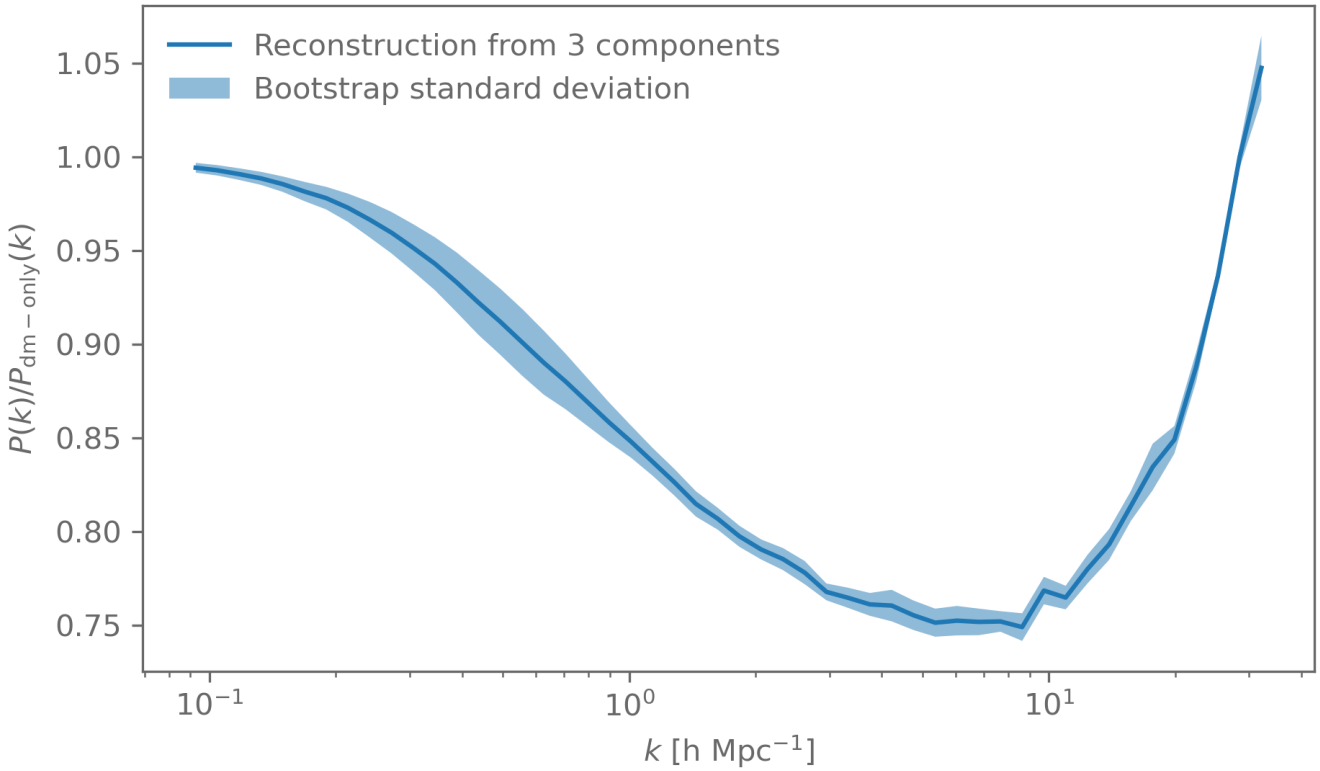
```
def PCA_coefficients(data, data_point):
    l, E, project_onto_pc, reconstruct_from_pc = PCA(data)
    coefficients = project_onto_pc(data_point)
    return coefficients

coefficient_samples = bootstrap(
    power_spectra,
    statistic=lambda data: PCA_coefficients(
        data, data_point=data_point),
    n_trial=1000
)
```



Estimate the uncertainty of a reconstructed power spectrum.

```
def PCA_and_reconstruct(data, data_point, n_component):
```

```
    l, E, project_onto_pc, reconstruct_from_pc = PCA(data)
    coefficients = project_onto_pc(data_point)
    return reconstruct_from_pc(coefficients[:n_component])

data_point = power_spectra[:, 5]
l, E, project_onto_pc, reconstruct_from_pc = PCA(power_spectra)
coefficients = project_onto_pc(data_point)
reconstructed = reconstruct_from_pc(coefficients[:3])

reconstructed_samples = bootstrap(
    power_spectra,
    statistic=lambda data: PCA_and_reconstruct(
        data, data_point=data_point, n_component=3),
    n_trial=200
)
```



Exercise

Check that the boostrap works for a case where we do know the sampling distribution. For example the estimators discussed at the beginning of this lecture.