```
>>> import numpy
>>> import scipy

% ipython notebook --pylab=inline
```

# Scientific Programming I

SciPy.org

Berian James <berian@berkeley.edu>

Astronomy Department, UC Berkeley

# Scientific Tools for Python

- SciPy is the scientific toolbox for Python, aimed at mathematics, science and engineering applications.
- It is built on NumPy, i.e., NumPy arrays are the most practical data type; they are generic, efficient and straight-forward to handle.
- SciPy is open-source software, compiled on top of NumPy
- SciPy is also a conference for scientific Python discussion: recent meeting was July 16 - 21, in Austin, TX; http://conference.scipy.org/scipy2012/
- See also Josh's plenary address at SciPy
  http://profjsb.github.com/ScienceWithPython/
  as well as the talk and tutorial by Fernando Perez

# SciPy resources

- http://scipy-lectures.github.com/
  SciPy lecture notes, fairly complete and usefully formatted

- http://www.tau.ac.il/~kineret/amit/scipy_tutorial/
  Older lecture notes (2004) by Travis Oliphant (Enthought/Continuum); incomplete but very detailed and informative.

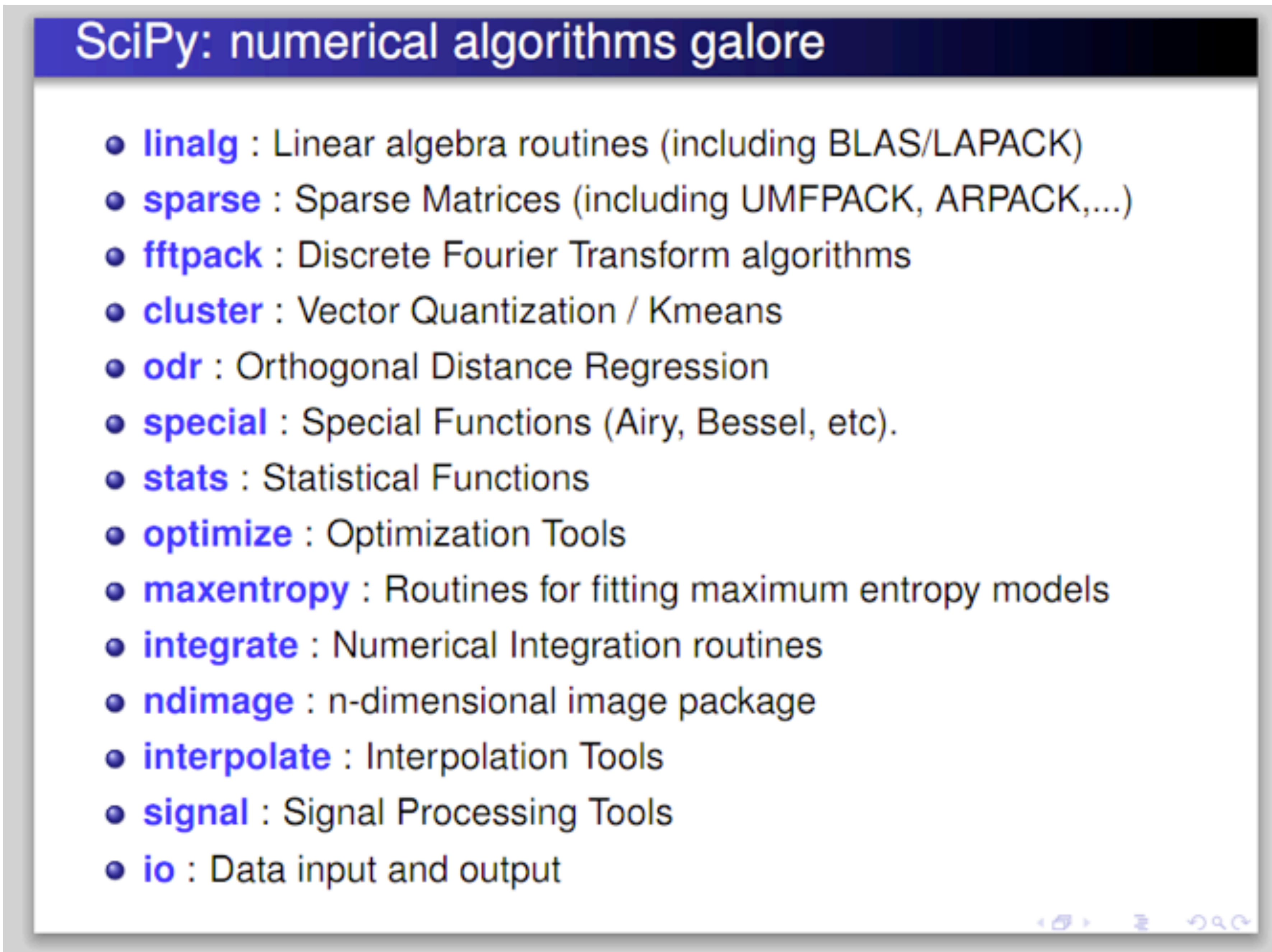
- http://scipy-central.org/
  Collection of code snippets and modules, cookbooks, miscellany

- http://docs.scipy.org/doc/scipy/reference/
  SciPy reference guide, tutorial

- http://www.scipy.org/NumPy_for_Matlab_Users

# SciPy packages

- This lecture explores SciPy and symbolic computation, including: `linalg`, `fftpack`, `optimize`, `integrate` and `interpolate`



# I. Overview of SciPy & symbolic computation

# Getting data in and out of SciPy

- Remember Josh's lecture from yesterday morning;
  also http://www.scipy.org/Cookbook/InputOutput

- Python provides powerful read/write routines for ascii files and some binary types (C/Fortran)

- Arbitrary input and output
  `np.loadtxt()/savetxt(), np.genfromtxt()/recfromcsv(), np.save()/load()`

  Certain proprietary (but common) binary formats:
  `scipy.io.matlab, scipy.io.idl`

# Special binaries (Matlab, IDL, HDF5): `scipy.io`

- Support for Matlab, IDL, HDF5 (though the PyTables module)
- Includes support for advanced data structures in these languages
- E.g., Matlab data:

```
>>> from scipy import io
>>> struct = io.loadmat('file.mat', struct_as_record=True)
>>> io.savemat('file.mat', struct)
```

# Building and referencing your own arrays quickly

- (Row) vector of numbers: `np/sp.r_` and `np/sp.linspace`

```
>>> np.r_[1.:11.] # N.b. (1,2,...,10)
>>> np.linspace(a,b,n)
```

- n-d grid of coordinates: `np/sp.mgrid`

```
>>> x,y = np.mgrid(1:5,1:5) # A 4x4 array
>>> r = np.sqrt(x**2 + y**2)
```

- n-d array: `np/sp.c_` and `np.tile`

```
>>> x = np.linspace(0,10,11);
>>> np.c_[x,x]
```

# Symbolic mathematics with Python

- http://sympy.org/
  SymPy home page

- http://docs.sympy.org
  Reference, tutorial

- Think of SymPy as Mathematica for Python, including integration, geometry, linear algebra, statistics, ODE solving and tensor algebra

```
>>> import sympy
```

# Interfacing with other languages

- E.g., http://www.scipy.org/PerformancePython
  An interesting and useful comparison of possibilities
- Cython (<- Pyrex)
  The most comprehensive option; requires a lecture of its own
- f2py
  Interface with Fortran, great for number-crunching
- PyPy (<- Psyco)
  Truly amazing, but does not support NumPy :’(
- scipy.weave
  Very cool to use, perhaps becoming less common(?)

# Blending languages: f2py & weave

- You will need: Python, a Fortran compiler (e.g. g95, gfortran) and f2py
- See also: http://www.scipy.org/Cookbook/Weave (needs C/C++ compiler)
- Let's try this out in the notebook!

# II. SciPy packages and data analysis challenges

# Overview of SciPy challenges in this lecture

- Generating random variables with the same distribution as an input data set
  Using: sp.integrate, sp.interpolate, np.random
  Presupposes: Some statistics background, but I will provide a primer

- Calculating a Fourier transform of 1D data, with error bars
  Using: sp.fftpack, np.random
  Presupposes: Knowledge of Fourier transformation

- Fitting a model to (perhaps covariant) data
  Using: sp.optimize, sp.linalg
  Presupposes: A bit more statistics background, which, again, I will describe

  scipy.*package_name*.[tab]
  scipy.*package_name*.*function_name*?
  [http://docs.scipy.org/doc/scipy/reference/](http://docs.scipy.org/doc/scipy/reference/) *SciPy reference guide, tutorial*

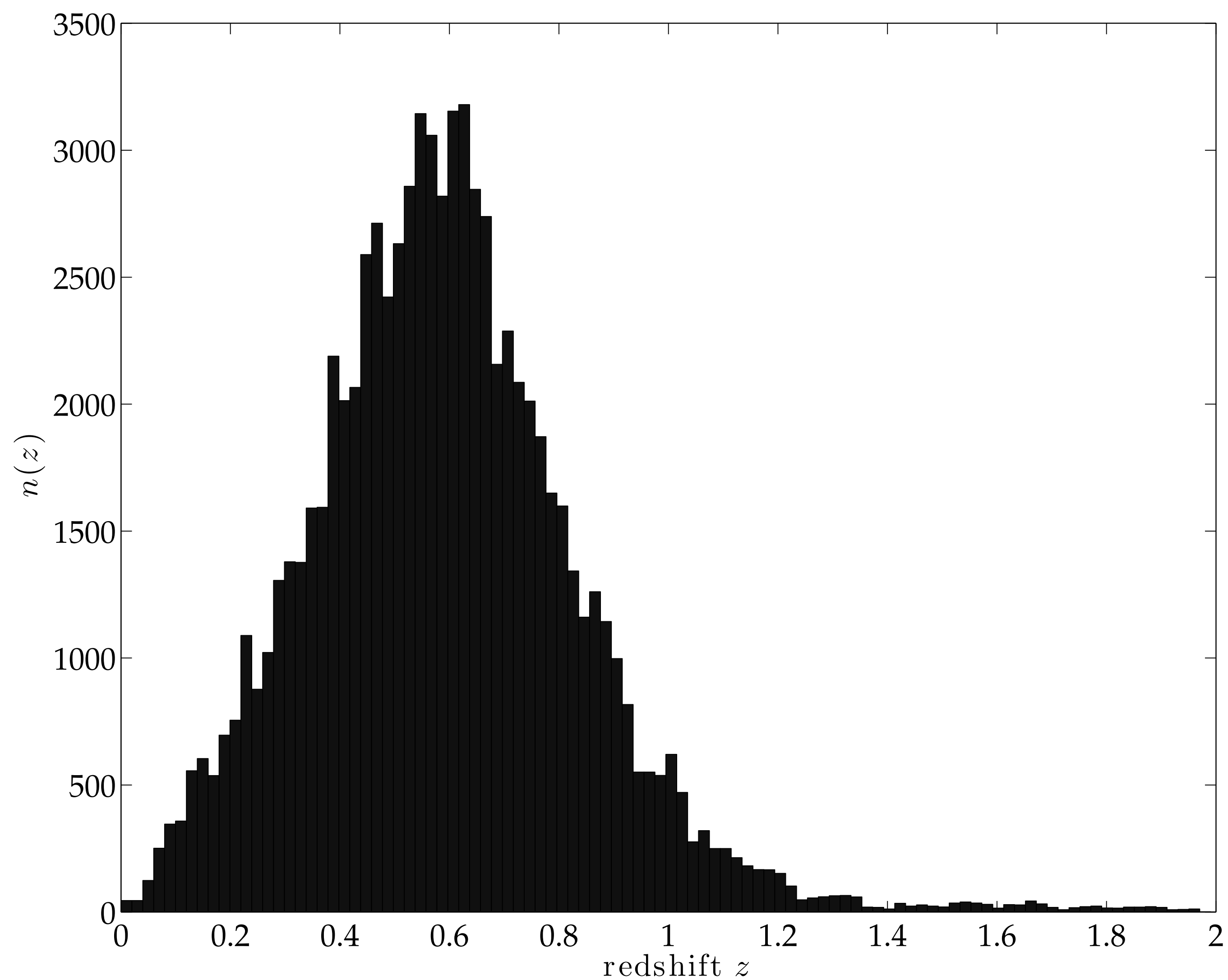
IIa. Integration and interpolation

```
>>> import scipy.integrate
>>> import scipy.interpolate
```

# Problem: generating random variables from data

Data: $[0.674, 1.053, 0.453 \ldots]$

PDF: $\hat{p}(z) \propto \texttt{hist}(\mathrm{Data})$

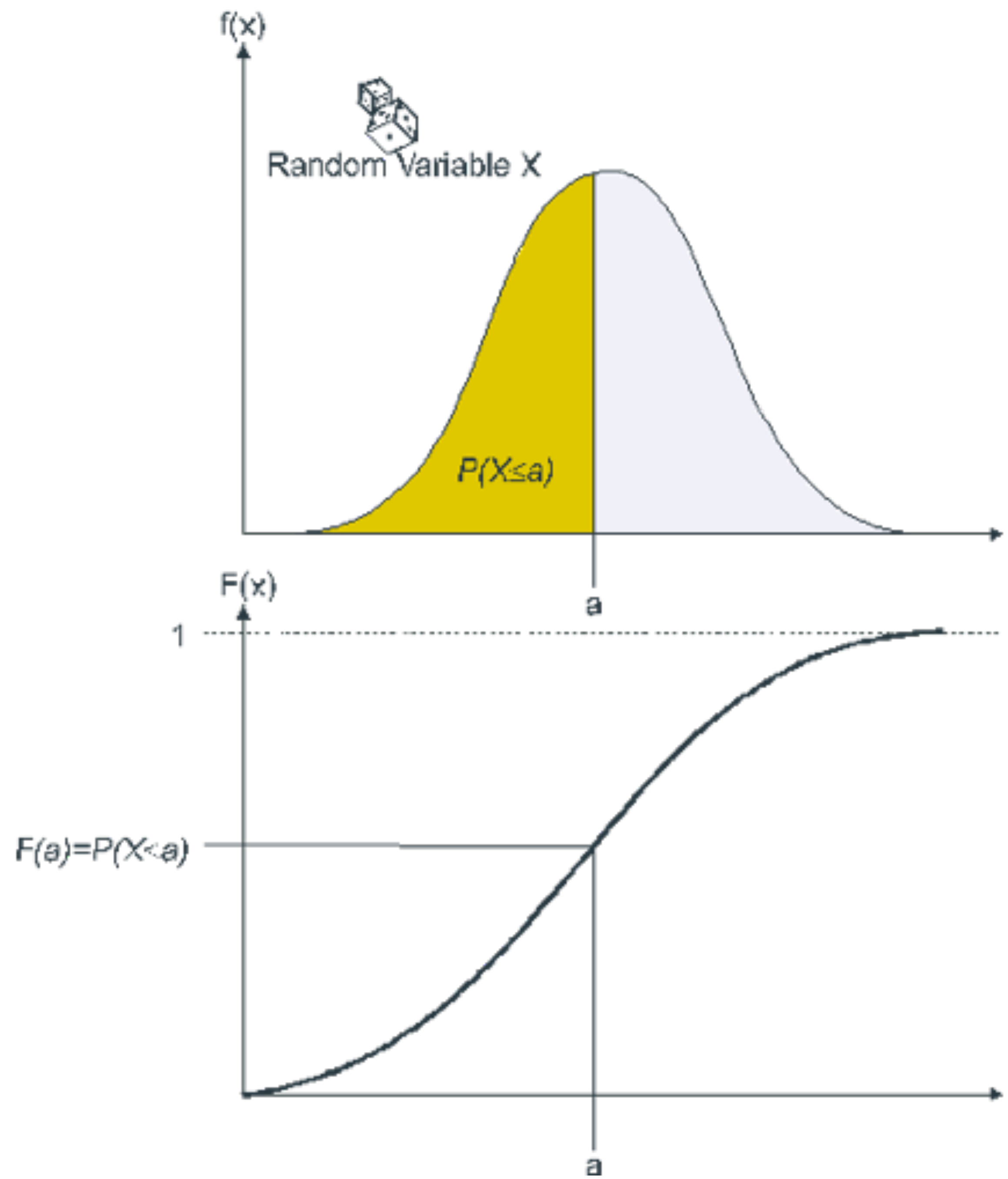


# Problem: generating random variables from data

Data: $[0.674, 1.053, 0.453\ldots]$

PDF: $\hat{p}(z) \propto \texttt{hist}(\mathrm{Data})$

CDF : $\hat{P}(z) = \int_0^z p(z')dz'$

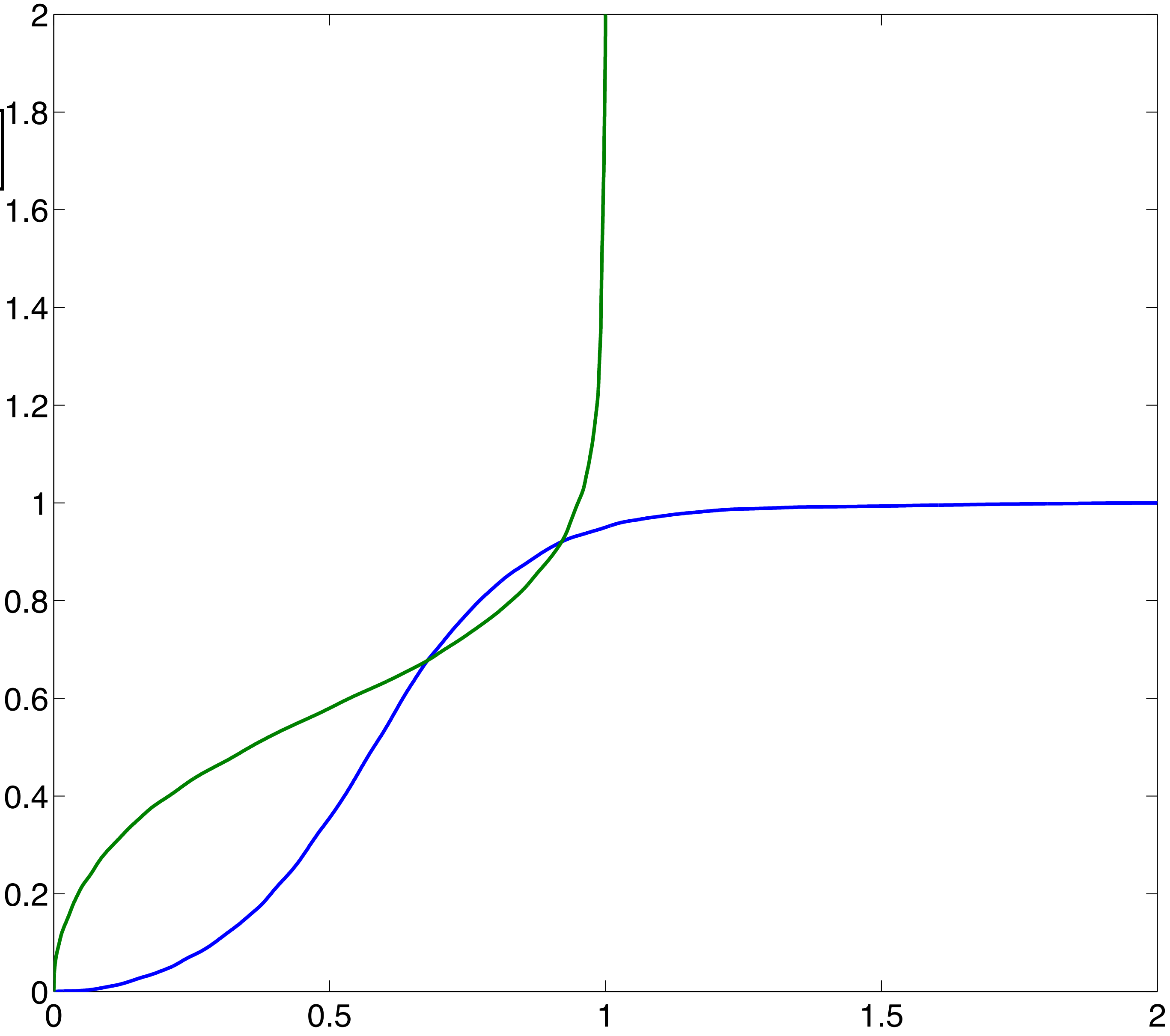


# Problem: generating random variables from data

Data: $[0.674, 1.053, 0.453 \ldots]$

PDF: $\hat{p}(z) \propto \texttt{hist}(\text{Data})$

CDF : $\hat{P}(z) = \int_0^z p(z')dz'$

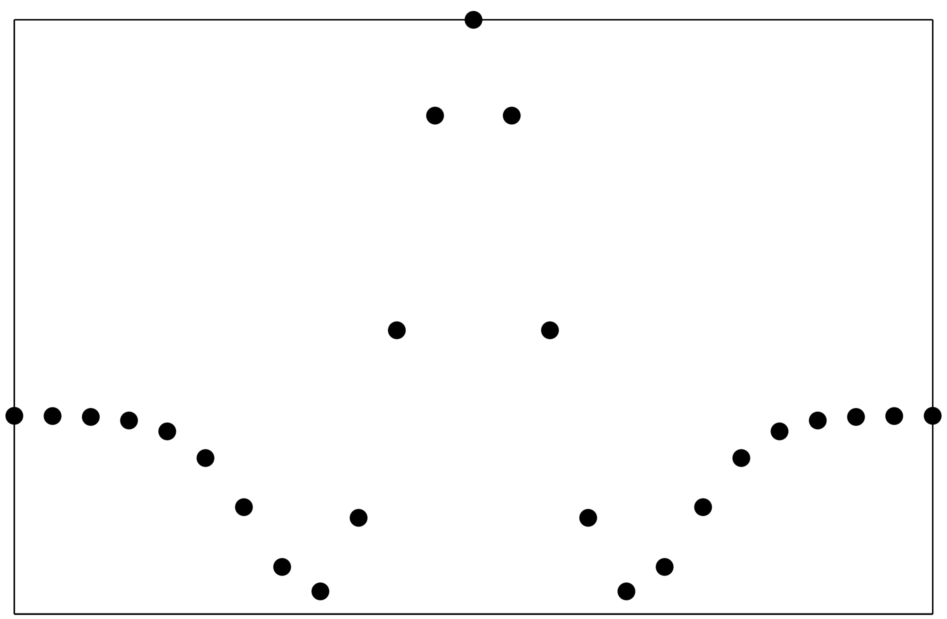



If $U \sim \text{Uniform}[0, 1)$, then $\hat{P}^{-1}(U) \sim \text{Data}$

IIb. An uncertain Fourier transform

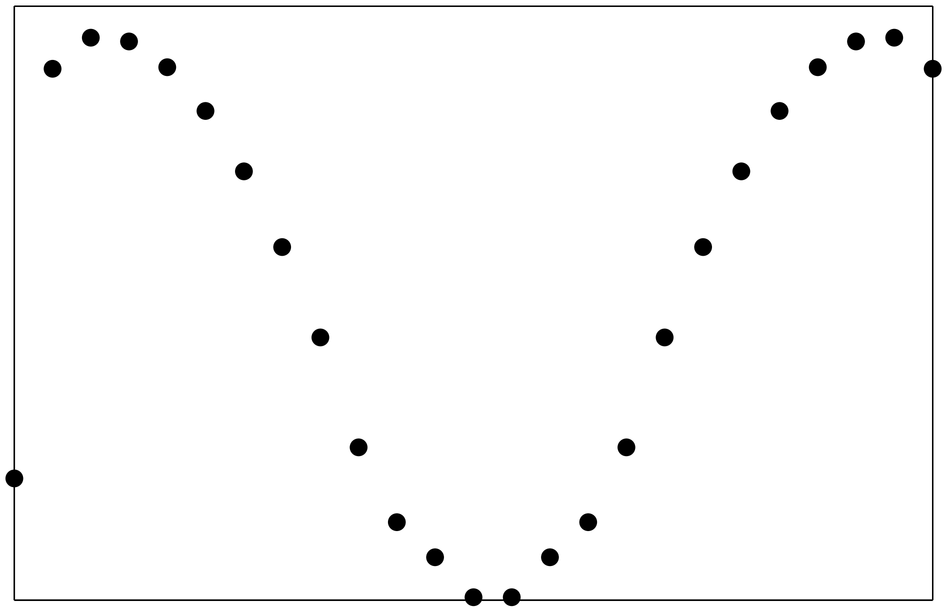
```
>>> import numpy.random
>>> import scipy.fftpack
```

# Problem: FFT with error bars

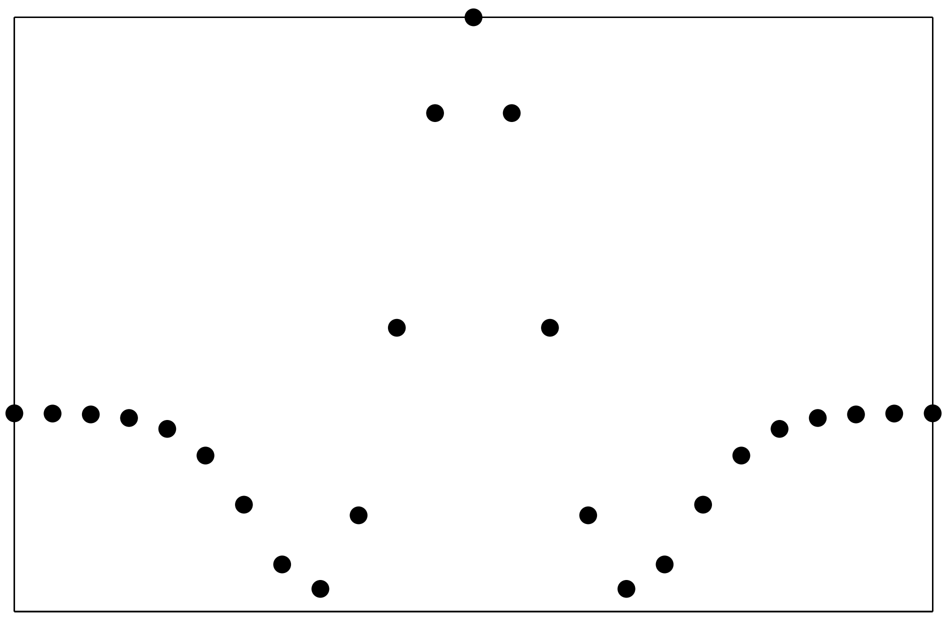


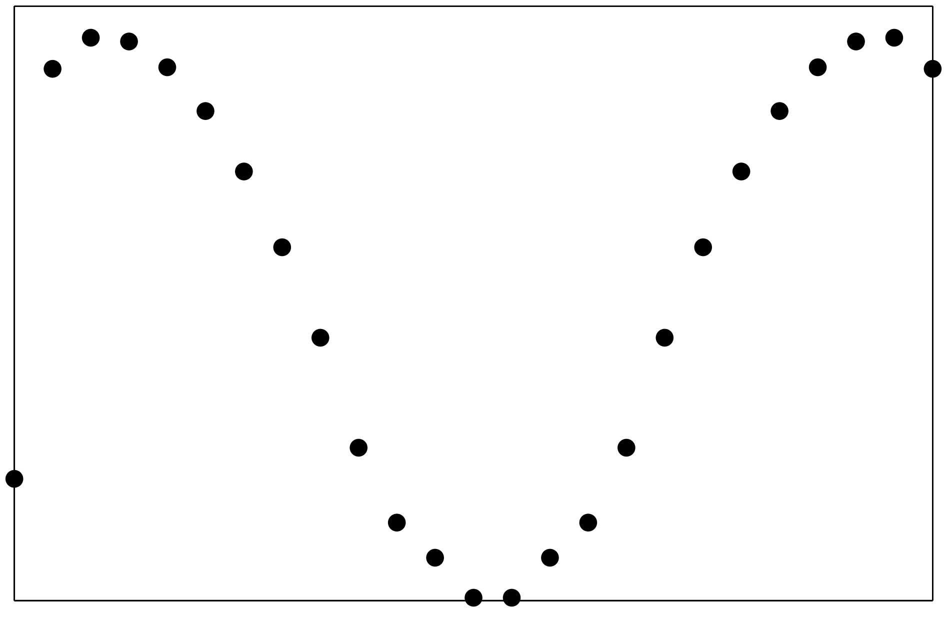
$$F(\tau) = \mathcal{F}(f(t))$$

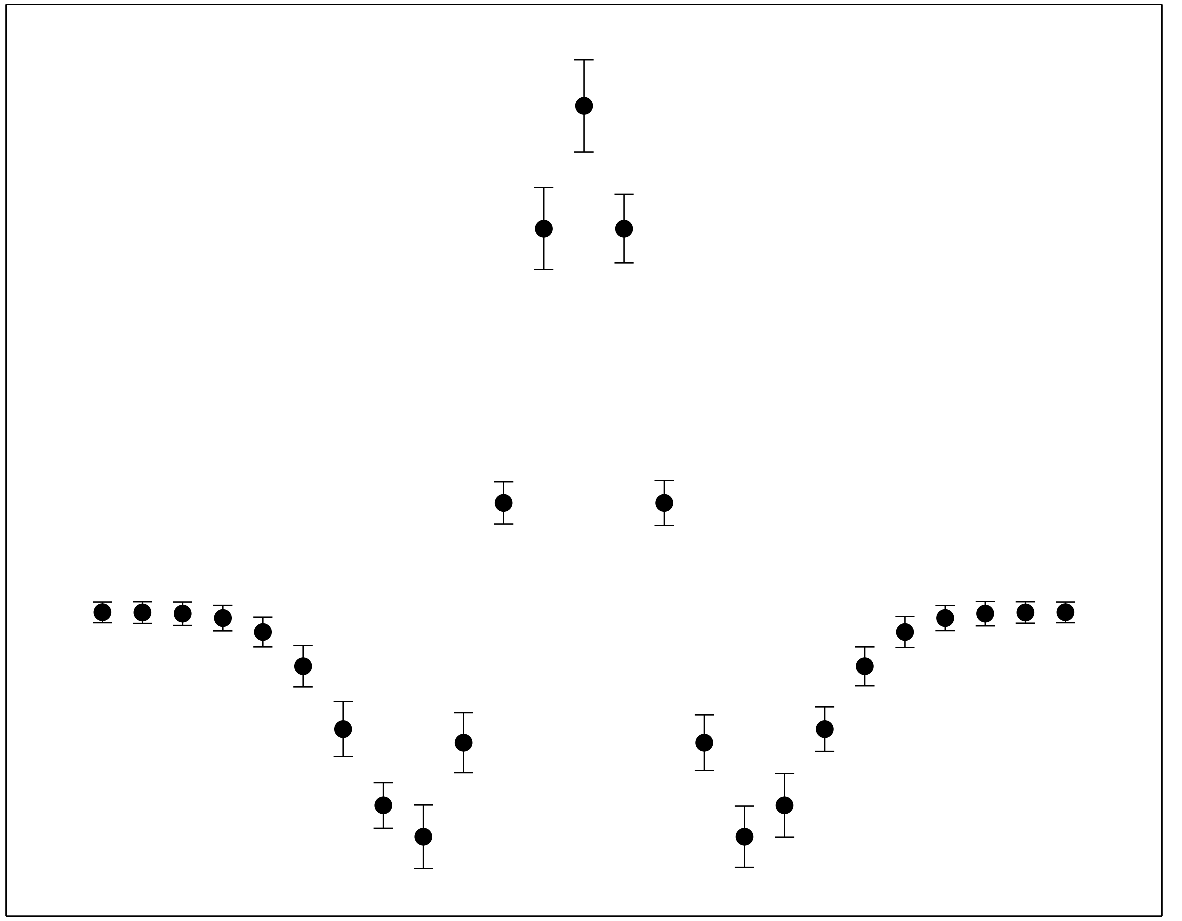
$$P(\tau) = F(\tau) \times F^*(\tau)$$



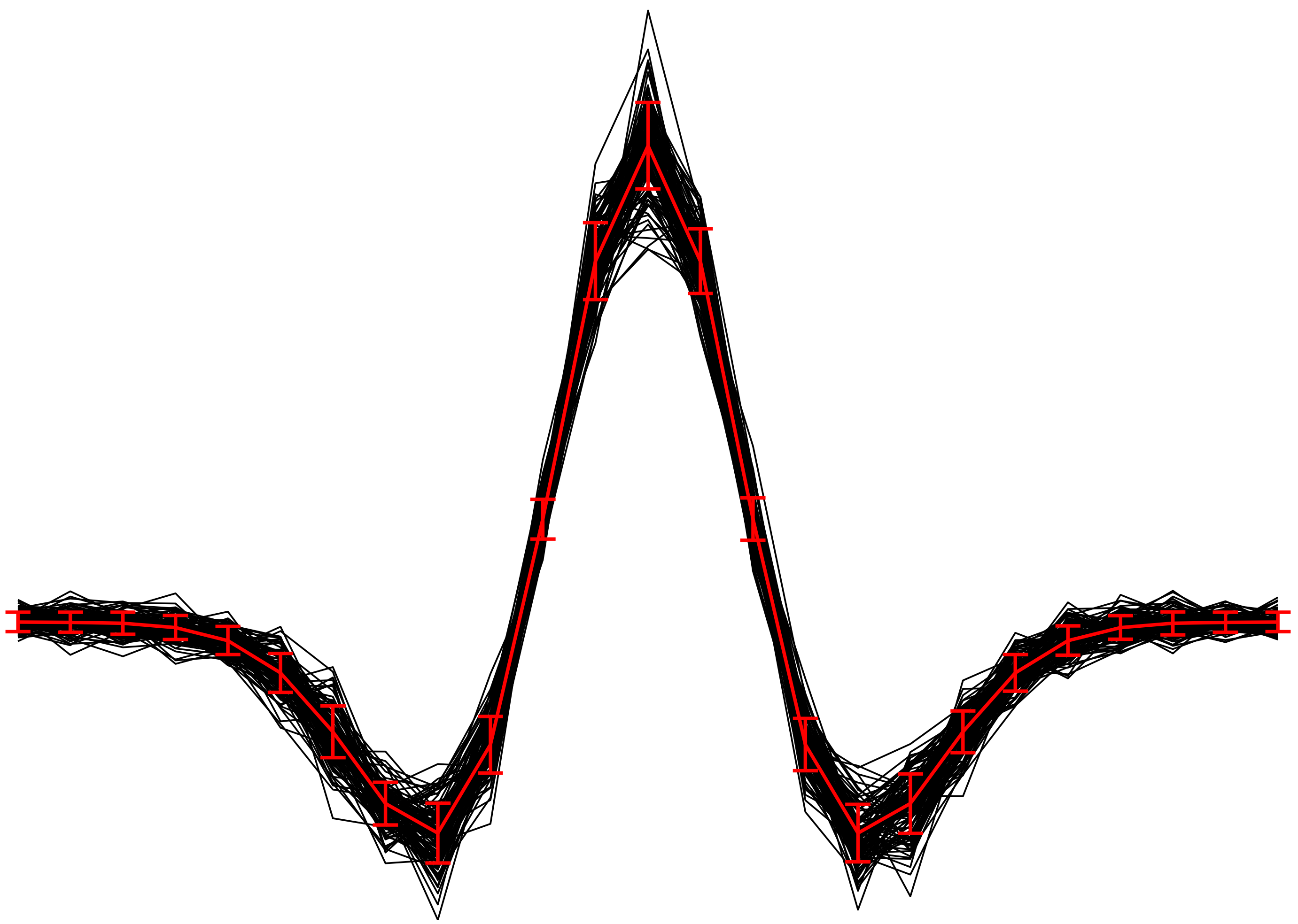
# Problem: FFT with error bars







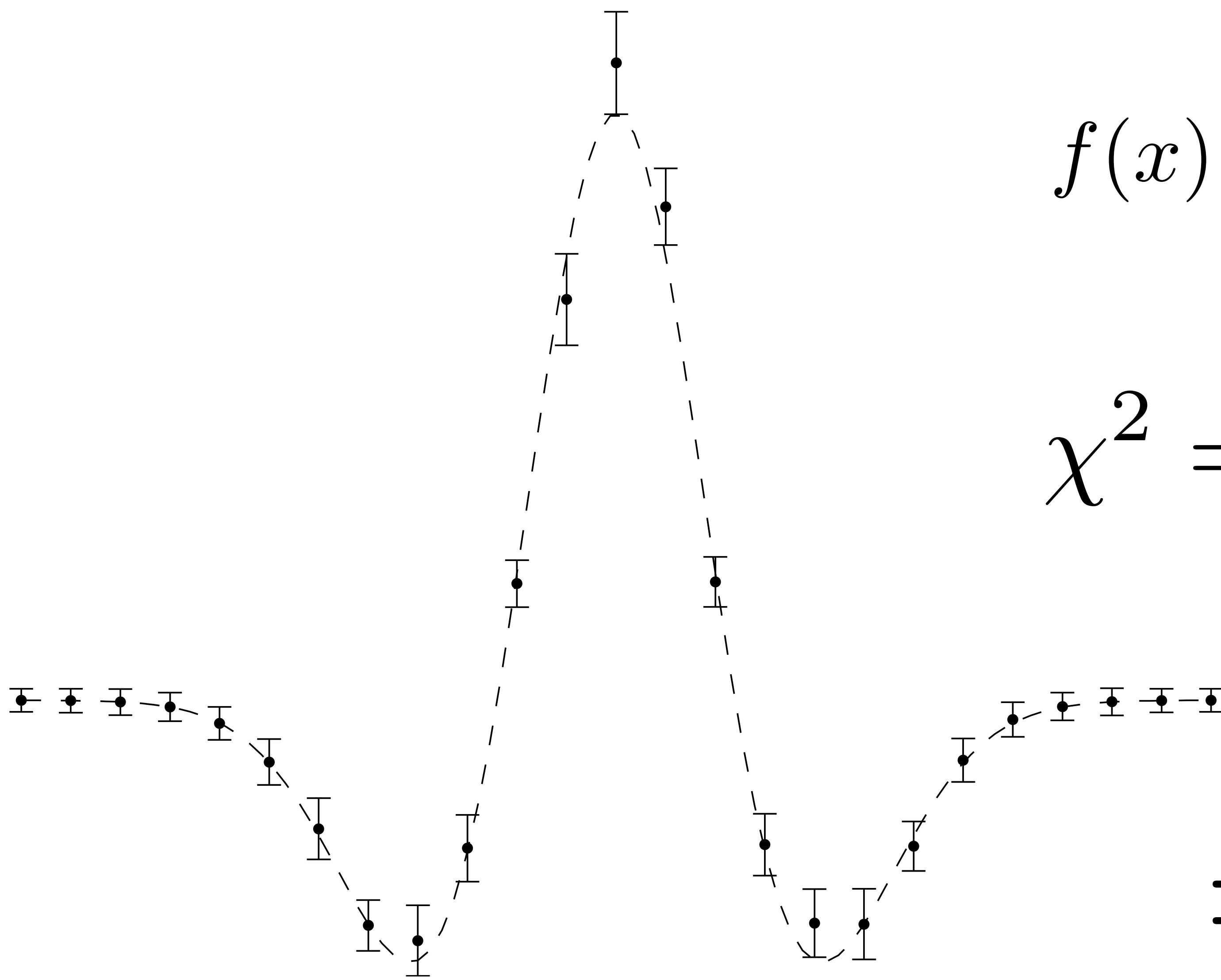
# (Unsatisfactory) solution: brute sampling



IIc. Fitting (likelihood estimation) with covariant data

```
>>> import scipy.linalg as la
>>> import scipy.optimize as opt
```

# Problem: non-linear model fitting



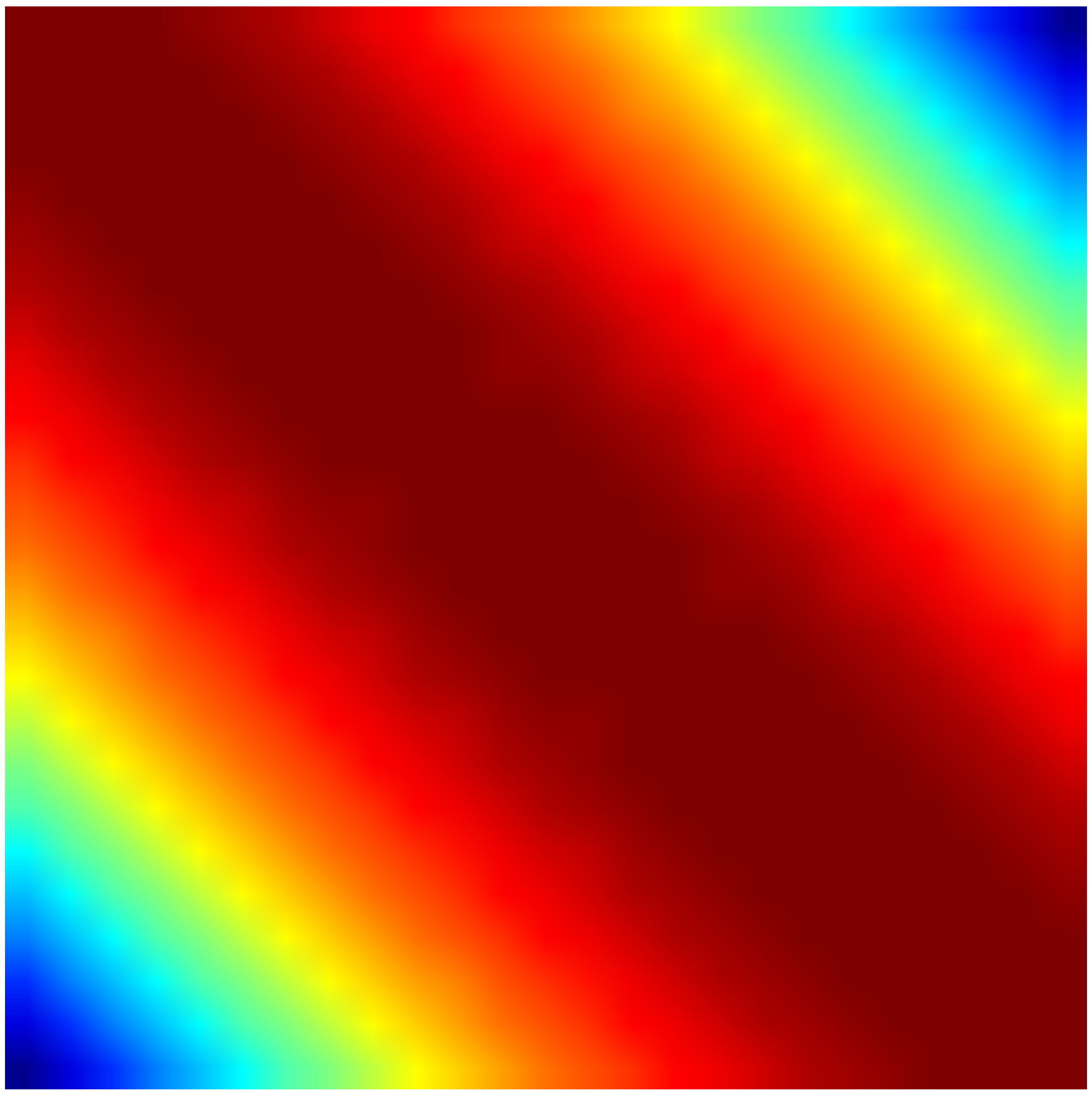
$$f(x) = P \exp\left(-\frac{x^2}{2}\right)(1 - x^2)$$

$$\chi^2 = \sum_i \left(\frac{y_i - f(x_i)}{\sigma_i}\right)^2$$

```
fmin(chi2,[x0])
```

# Extension: Fitting covariant data points

# Problem: non-linear model fitting with covariance



$$f(x) = P \exp\left(-\frac{x^2}{2}\right)(1-x^2)$$

$$\Delta = y_i - f(x_i)$$

$$\chi^2 = \Delta \mathbf{C}^{-1} \Delta'$$

```
fmin(chi2,[x0])
```

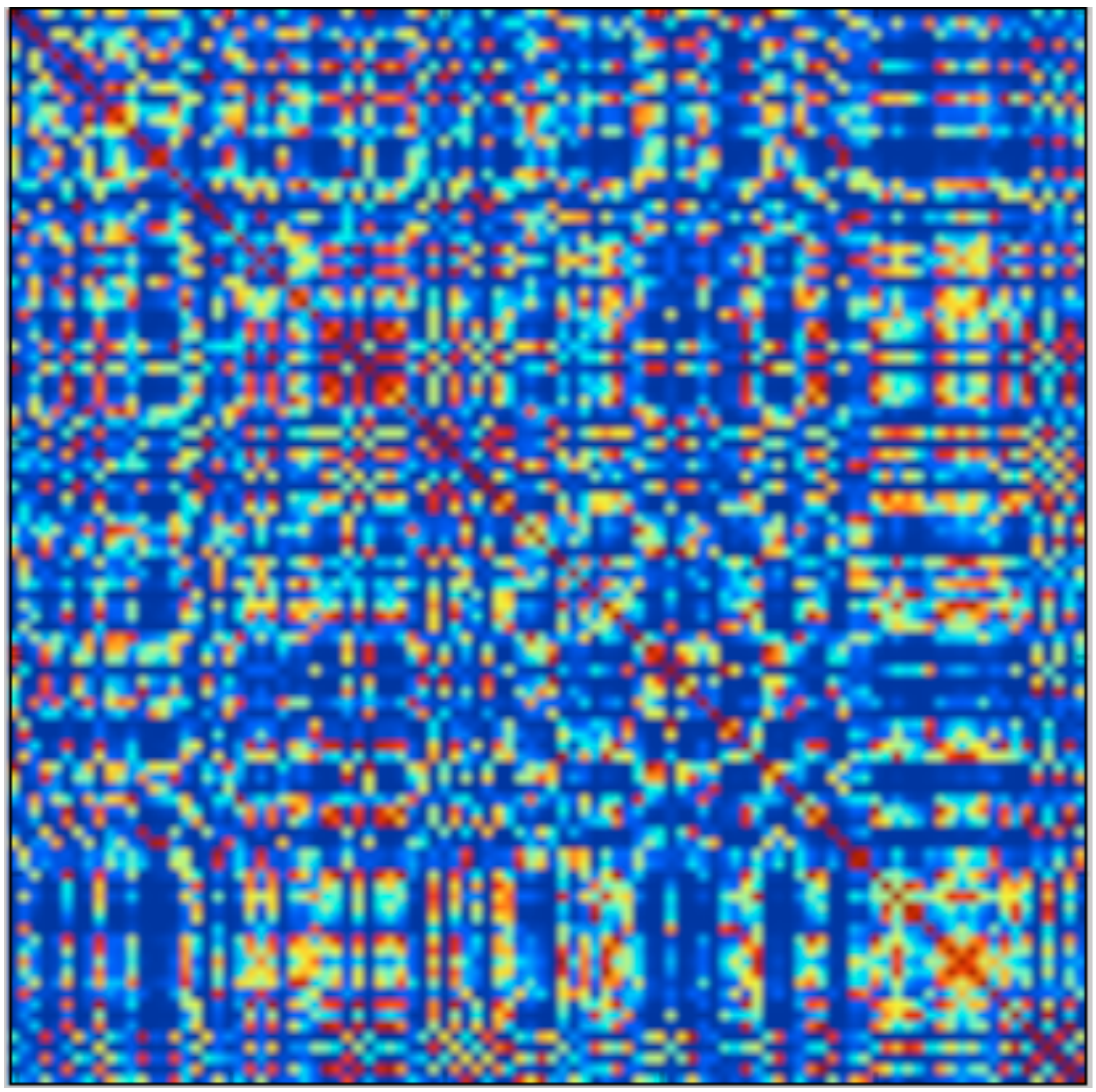
# Eigendecomposition (for covariant data points)

- For a symmetric matrix, eigendecomposition can be used to deal with tricky inversions

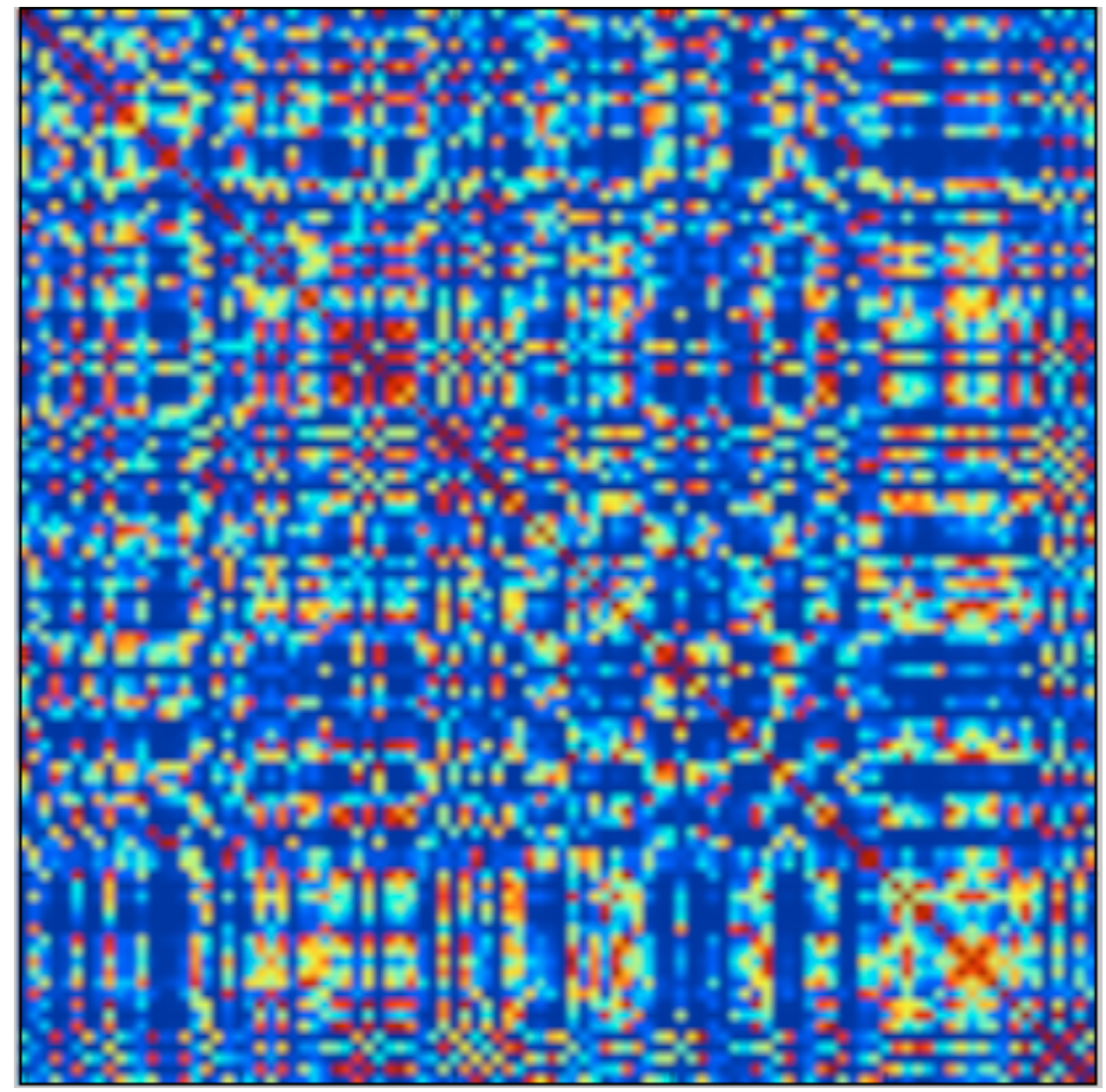
- One method is to locate very small eigenvalues, and set their inverse to zero.

$$C = VEV' \Rightarrow C^{-1} = VE^{-1}V'$$

# Pick the invertible covariance matrix

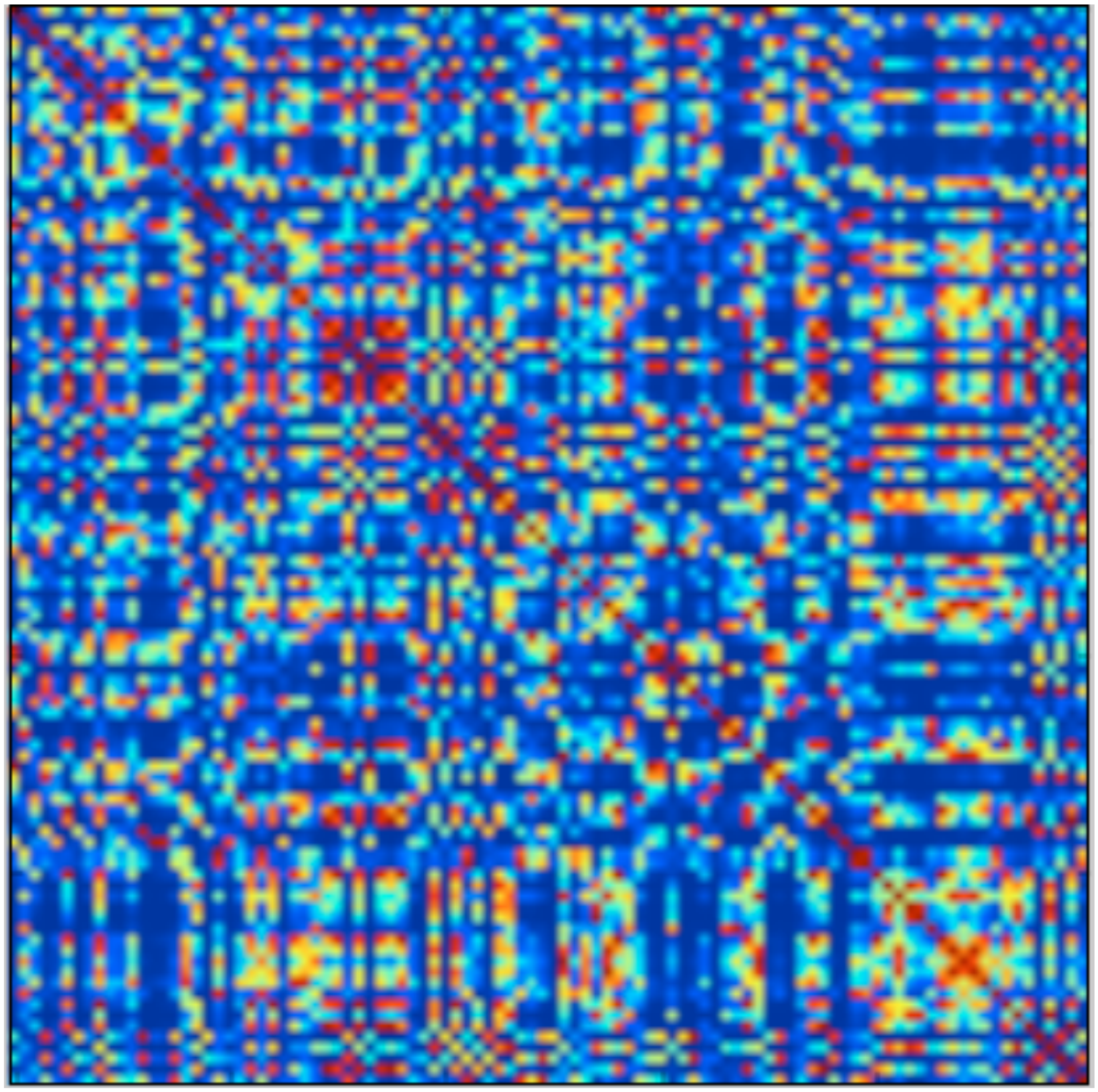


```
>>> U, luflag = cho_factor(C)
```

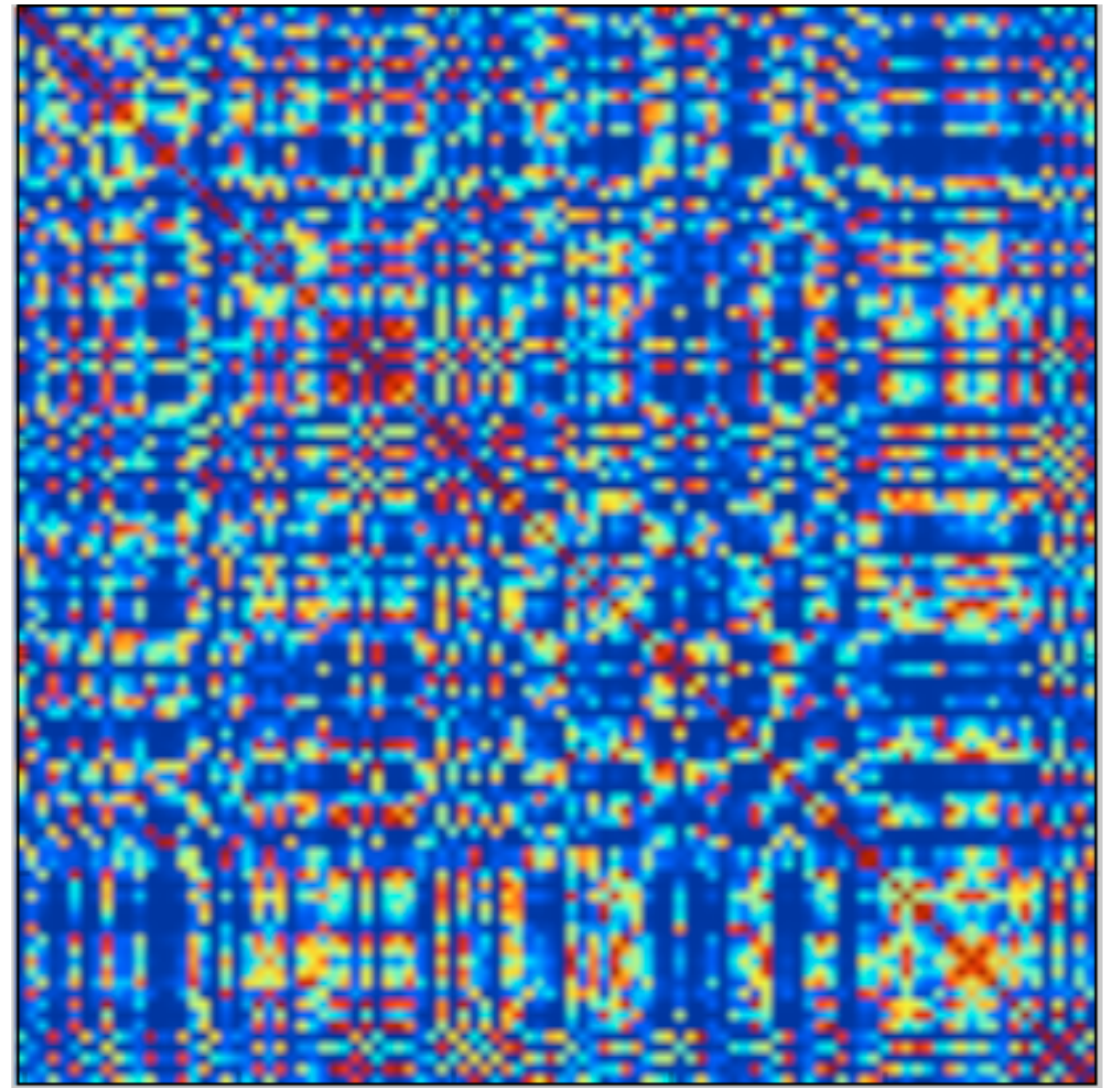


```
>>> U, luflag = cho_factor(C)
```

# Pick the invertible covariance matrix



```
>>> U, luflag = cho_factor(C)
>>>
```



```
>>> U, luflag = cho_factor(C)
> LinAlgError: 74-th leading
minor not positive definite
```

# Computing the 'nearest' covariance matrix

```
from scipy.linalg import cho_factor, cho_solve, eigh

# Do computation using Cholesky decomposition
try:

    U, luflag = cho_factor(C)
```

*Higham N. (1988) Linear Algebra and Appl., 103:103-118*

# Computing the 'nearest' covariance matrix

```
from scipy.linalg import cho_factor, cho_solve, eigh

# Do computation using Cholesky decomposition
try:

    U, luflag = cho_factor(C)

except LinAlgError:

    # Matrix is not positive semi-definite, so replace it with the
    #  positive semi-definite matrix that is nearest in the Frobenius norm

    E, EV = eigh(C) # Get eigenvalues and eigenvectors
    E[E<0] = 1e-12  # Replace negative eigenvalues with small number > 0
    U, luflag = cho_factor(EV.dot(np.diag(Ep)).dot(EV.T))
```

*Higham N. (1988) Linear Algebra and Appl., 103:103-118*

# Computing the 'nearest' covariance matrix

```
from scipy.linalg import cho_factor, cho_solve, eigh

# Do computation using Cholesky decomposition
try:

    U, luflag = cho_factor(C)


except LinAlgError:

    # Matrix is not positive semi-definite, so replace it with the
    #  positive semi-definite matrix that is nearest in the Frobenius norm

    E, EV = eigh(C) # Get eigenvalues and eigenvectors
    E[E<0] = 1e-12  # Replace negative eigenvalues with small number > 0
    U, luflag = cho_factor(EV.dot(np.diag(Ep)).dot(EV.T))


finally:

    x2 = cho_solve((U, luflag), dxy)
    L1 = dxy.dot(x2)
```

*Higham N. (1988) Linear Algebra and Appl., 103:103-118*