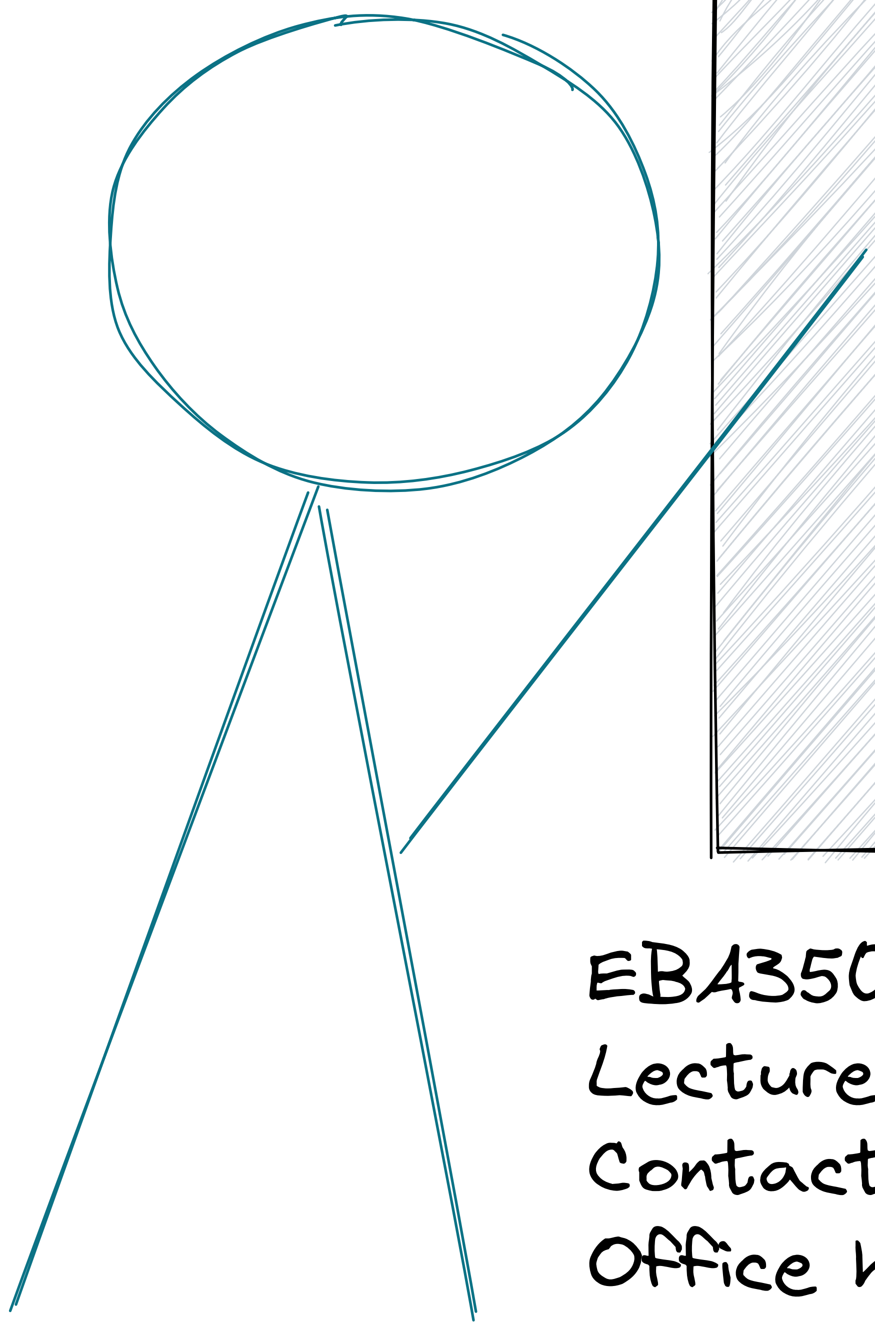
Ouverture, an introduction to Numpy, and some Scipy



EBA3500: Data analysis with programming

Lecturer: Jonas Moss
Contact: jonas.moss@bi.no

Office hours: Tuesdays 11:00-12:00

23/8 2022

About me:

Ph.D. in statistics, University of Oslo

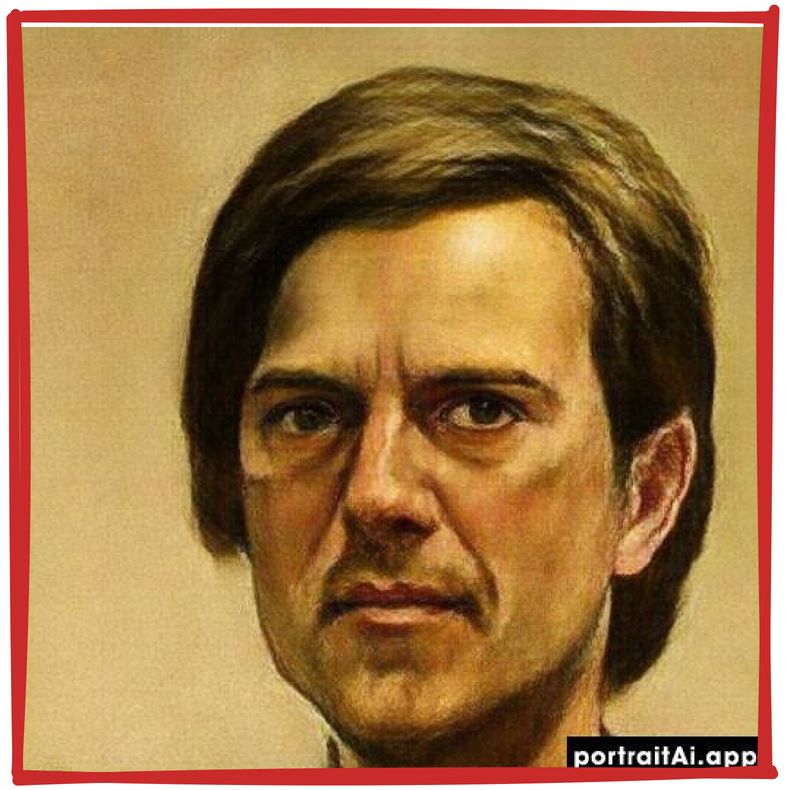
* Three kids, two cats, one wife.
* Passionate about effective altruism and existential risk.
* Fan of Super Nintendo music and professional StarCraft II.

Research:

* Meta-science. How does science work? How reliable are its subfields?
* Fixing the problems of science, e.g. publication bias and p-hacking.
* Psychometrics: How do you measure agreement? What about personality?
* Informal forecasting.

Favourite business-adjacent books:

* The Scout Mindset
* Good Strategy Bad Strategy
* Zero to One
* Superforecasting



100% authentic portrait!

About the course

* This is not a lax course! Spend time on it!
* I'll be uploading readings and exercises each week.
* The textbook won't be used much.
* Reach me at jonas.moss@bi.no or during office hours.

The course isn't for this guy!

Find your own resources!

* You should probably use more resources than the lecture notes, the book, and the exercises.
* There is a document online with more details.
* In particular, you might need to learn some Python on your own.

Got any comments about the course?
Or questions about Python and statistics?

Feel free to use the course's Padlet!
https://padlet.com/jonasmoss/EBA3500

(I've added some anonymous notes there to show how it can be used.)

Lecture notes, tips, exercises, readings and so on:

🌐https://hackmd.io/@JonasMoss/eba3500

The lecture notes are part of the curriculum!
They also contain suggested additional readings and exercises.

You can add comments to the documents.
(Unless I've skrewed up... If so, tell me.)
That's an experimental feature. Please don't misuse it.

This helps me:

* Understand what you already know,
* Improve the materials for next year's students.

Tentative plan

1. Ouverture and introduction to Numpy and Scipy
2. Recap and simulation
3. Simple linear regression
4. Multiple linear regression
5. Categorical covariates
6. Covariate transformations
7. Model selection and model fit
8. Binary regression (i)
9. Binary regression (ii)
10. Multinomial regression
12. Inference in multiple linear regression
13. Inference for general regression models

Required tools.

Proper statistics / machine learning / data science / etc.

Introduction to Numpy

Important: Make a habit of using the official documentation!

A Python package for numerical computing.

All statistical / ML packages in Python are based on Numpy or use its syntax
(PyTorch and Keras use their own architecture.)

Bookmark the documentation: https://numpy.org/doc/stable/

Curriculum: https://numpy.org/doc/stable/user/absolute_beginners.html

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

Numpy arrays

Don't know `zip`? It "zips" together two lists into a single list of tuples. E.g. zip([1,2],[3,4])=[(1,3),(2,4)]. Very important function you absolutely need to know!

```python
x = [1, 2, 3] # Define Python lists
y = [5, 9, 4]
z = [x + y for x, y in zip(x, y)] # Add the lists elementwise.
z # [6, 11, 7]
```

```python
import numpy as np # Standard terminology
x = np.array([1, 2, 3]) # Define an np.array
y = np.array([5, 9, 4])
z = x + y # Add the lists elementwise automatically!
z.dtype # dtype('int32')
```

Elementwise operations such as these are called vectorized operations.

Numpy has more basic data types than Python. This allows finer control of memory.

Vectorization

```
x - y # array([-4, -7, -1])
x * y # array([ 5, 18, 12])
x / y # array([0.2       , 0.22222222, 0.75      ])
x ** y # array([  1, 512,  81], dtype=int32)
```

Why?

1. Faster to write.
2. Easier to understand.
3. Faster to compute!

The vectorized operations are done in C(++) instead of Python. C is a compiled language that is much, much faster than Python.

You want to use Numpy as much as possible!

Try to avoid Python loops. They are slow!

Arrays and matrices

```
>>> a = np.array([[0.45053314, 0.17296777, 0.34376245, 0.5510652],
...               [0.54627315, 0.05093587, 0.40067661, 0.55645993],
...               [0.12697628, 0.82485143, 0.26590556, 0.56917101]])
```

This is a 3x4 matrix. The dimensions are called "axes".

* Two-dimensional arrays are matrices.
* Remember that matrices have dimensions RxC, rows first, then columns.

```
>>> array_example = np.array([[[0, 1, 2, 3],
...                            [4, 5, 6, 7]],
...
...                           [[0, 1, 2, 3],
...                            [4, 5, 6, 7]],
...
...                           [[0 ,1 ,2, 3],
...                            [4, 5, 6, 7]]])
```

This is a three-dimensional array.

```
>>> array_example.ndim
3
```

dimension

```
>>> array_example.size
24
```

number of elements

```
>>> array_example.shape
(3, 2, 4)
```

number of elements in each axis.

Basic indexing

Remember that indexing starts at zero!

```
>>> array_example = np.array([[[0, 1, 2, 3],
...                             [4, 5, 6, 7]],
...
...                            [[0, 1, 2, 3],
...                             [4, 5, 6, 7]],
...
...                            [[0 ,1 ,2, 3],
...                             [4, 5, 6, 7]]])
```

Remember that ":" selects everything?

The first argument is the first axis.

```
>>> array_example[0]
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

```
>>> array_example[1]
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

The second argument is the second axis...

```
>>> array_example[:, 0]
array([[0, 1, 2, 3],
       [0, 1, 2, 3],
       [0, 1, 2, 3]])
```

```
>>> array_example[:, 1]
array([[4, 5, 6, 7],
       [4, 5, 6, 7],
       [4, 5, 6, 7]])
```

What is `array_example[:, :, 0]` ?

This sort of indexing selects subarrays for you!

```
>>> array_example = np.array([[[0, 1, 2, 3],
...                            [4, 5, 6, 7]],
...
...                           [[0, 1, 2, 3],
...                            [4, 5, 6, 7]],
...
...                           [[0 ,1 ,2, 3],
...                            [4, 5, 6, 7]]])
```

What is `array_example[:, :, 0]` ?

```
>>> array_example[:, :, 0]
array([[0, 4],
       [0, 4],
       [0, 4]])
```

Useful operations

```
>>> a = np.array([[0.45053314, 0.17296777, 0.34376245, 0.5510652],
...               [0.54627315, 0.05093587, 0.40067661, 0.55645993],
...               [0.12697628, 0.82485143, 0.26590556, 0.56917101]])
```

This is a 3x4 matrix. The dimensions are called "axes".

We can calculate the minimum of all elements!

```
>>> a.min()
0.05093587
```

min, max, argmax, argmin, trace, sum, cumsum, mean, var, std, prod, all, any

Some of the methods we will use. (Read about them in the docs.)

Convenience functions work on the array!

```
>>> a.min(axis=0)
array([0.12697628, 0.05093587, 0.26590556, 0.5510652 ])
```

This are the column-wise minima! (axis = 0 makes the 0th axis disappear.)

Always use these axis-based Numpy operations if you can!

Generating arrays

From list:

```
>>> data = np.array([[1, 2], [3, 4], [5, 6]])
>>> data
array([[1, 2],
       [3, 4],
       [5, 6]])
```

Generate evenly spaced numbers:

```
>>> np.linspace(0, 1, 10)
array([0.        , 0.11111111, 0.22222222, 0.33333333, 0.44444444,
       0.55555556, 0.66666667, 0.77777778, 0.88888889, 1.        ])
...
```

Numpy variant of range:

```
>>> np.arange(0, 10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

with steps to!

```
>>> np.arange(0, 10, 2)
array([0, 2, 4, 6, 8])
```

Identity matrix.

```
>>> np.eye(3)
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

Array of ones:

```
>>> np.ones((4, 3, 2))
array([[[1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.]]])
```

Diagonal matrix from vector.

```
>>> np.diag([1, 2, 3])
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

Array of zeros:

```
>>> np.zeros((2, 3, 2))
array([[[0., 0.],
        [0., 0.],
        [0., 0.]],

       [[0., 0.],
        [0., 0.],
        [0., 0.]]])
```

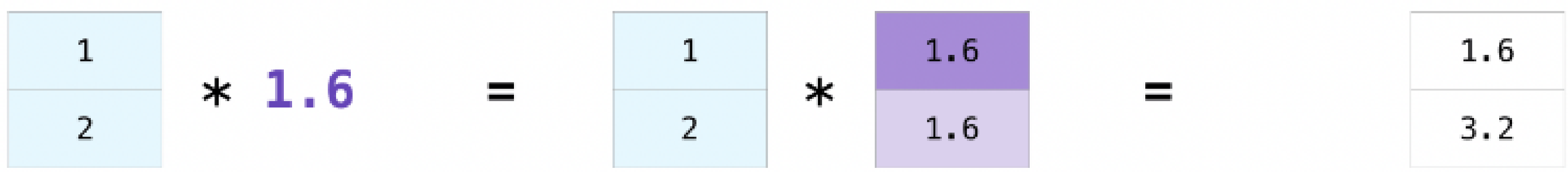
Numpy often does conversios between array dimensions for you!

... but sometimes you must do stuff on your own.

Broadcasting

There are times when you might want to carry out an operation between an array and a single number (also called *an operation between a vector and a scalar*) or between arrays of two different sizes. For example, your array (we'll call it "data") might contain information about distance in miles but you want to convert the information to kilometers. You can perform this operation with:

```
>>> data = np.array([1.0, 2.0])
>>> data * 1.6
array([1.6, 3.2])
```




```
>>> a = np.arange(6)
>>> print(a)
[0 1 2 3 4 5]
```

```
>>> b = a.reshape(3, 2)
>>> print(b)
[[0 1]
 [2 3]
 [4 5]]
```

Matrix and vector operations

Dot product

```
>>> x = np.array([1, 2, 3])
>>> y = np.array([3, 2, 1])
>>> x.dot(y)
10
>>> np.dot(x, y)
10
```

Matrix product

```
>>> X = np.array([[1, 2, 3],
...               [2, 1, 2],
...               [3, 2, 1]])
>>>
>>> Y = np.array([[1, 0, 0],
...               [1, 1, 0],
...               [1, 1, 1]])
>>>
>>> X @ Y #np.matmul(X, Y)
array([[6, 5, 3],
       [5, 3, 2],
       [6, 3, 1]])
```

Solve matrix equations

```
>>> np.linalg.solve(X, x)
array([1., 0., 0.])
```

Get the matrix inverse

```
>>> np.linalg.inv(X)
array([[-0.375,  0.5  ,  0.125],
       [ 0.5  , -1.   ,  0.5  ],
       [ 0.125,  0.5  , -0.375]])
```

Transpose a matrix

```
>>> Y.T
array([[1, 1, 1],
       [0, 1, 1],
       [0, 0, 1]])
```

Elementwise/Hadamard product

```
>>> X * Y
array([[1, 0, 0],
       [2, 1, 0],
       [3, 2, 1]])
```

Prefer solving instead of finding inverse!

Calculate the determinant and trace (sum of diagonal elements)

```
>>> np.linalg.det(Y)
1.0
>>> np.trace(Y)
3
```

Eigenvectors and eigenvalues

```
>>> np.linalg.eig(X)
(array([ 5.70156212, -2.        , -0.70156212]),
 array([[-6.05912800e-01, -7.07106781e-01,  3.64
512933e-01],
       [-5.15499134e-01,  1.47635207e-16, -8.568
90100e-01],
       [-6.05912800e-01,  7.07106781e-01,  3.645
12933e-01]]))
```

First array contains eigenvalues, second the eigenvectors.

SciPy

Fundamental algorithms for scientific computing in Python

Scipy is used for a lot, but we will only use the scipy.stats module.

In particular, we use the continuous and discrete distributions a lot!

Continuous Statistical Distributions

Overview

All distributions will have location (L) and Scale (S) parameters along with any shape parameters needed, the names for the shape parameters will vary. Standard form for the distributions will be given where $L = 0.0$ and $S = 1.0$. The nonstandard forms can be obtained for the various functions using (note U is a standard uniform random variate).

Function Name	Standard Function	Transformation
Cumulative Distribution Function (CDF)	$F(x)$	$F(x; L, S) = F\left(\frac{(x-L)}{S}\right)$
Probability Density Function (PDF)	$f(x) = F'(x)$	$f(x; L, S) = \frac{1}{S} f\left(\frac{(x-L)}{S}\right)$
Percent	$G(q) = F^{-1}(q)$	$G(q; L, S) = L + SG(q)$

Scipy has implemented more than 100 continuous distributions.

You can use their methods to calculate the pdf, the cdf, the quantile function, mean, variance, and a lot more.

(Not every implementation supports every method.)

This is the Scipy object for the normal distribution.

Examples

```
>>> from scipy.stats import norm
>>> import matplotlib.pyplot as plt
>>> fig, ax = plt.subplots(1, 1)
```

Calculate the first four moments:

```
>>> mean, var, skew, kurt = norm.stats(moments='mvsk')
```

Display the probability density function (pdf):

```
>>> x = np.linspace(norm.ppf(0.01),
...                 norm.ppf(0.99), 100)
>>> ax.plot(x, norm.pdf(x),
...        'r-', lw=5, alpha=0.6, label='norm pdf')
```

Don't know skewness and kurtosis? Read on wikipedia!

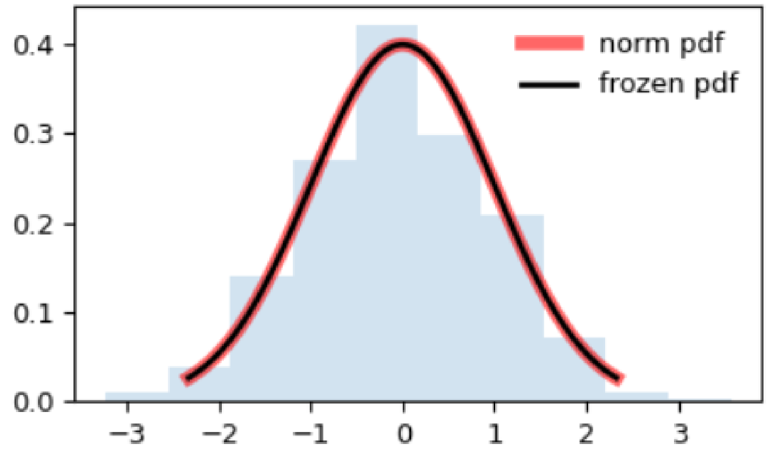
One one many things you can do.

Generate random numbers:

```
>>> r = norm.rvs(size=1000)
```

And compare the histogram:

```
>>> ax.hist(r, density=True, histtype='stepfilled', alpha=0.2)
>>> ax.legend(loc='best', frameon=False)
>>> plt.show()
```



You can generate randoms too! This is also possible solely within Numpy, which we prefer, when possible, in this course. (Starting next lecture.)

Important distributions with one underscore;
very important ones with two underscores.

Continuous Distributions in `scipy.stats`

- Alpha Distribution
- Anglit Distribution
- Arcsine Distribution
- Beta Distribution
- Beta Prime Distribution
- Bradford Distribution
- Burr Distribution
- Burr12 Distribution
- Cauchy Distribution
- Skewed Cauchy Distribution
- Chi Distribution
- Chi-squared Distribution
- Cosine Distribution
- Double Gamma Distribution
- Double Weibull Distribution
- Erlang Distribution
- Exponential Distribution
- Exponentiated Weibull Distribution
- Exponential Power Distribution
- Fatigue Life (Birnbaum-Saunders) Distribution
- Fisk (Log Logistic) Distribution
- Folded Cauchy Distribution
- Folded Normal Distribution
- Fratio (or F) Distribution
- Gamma Distribution
- Generalized Logistic Distribution
- Generalized Pareto Distribution
- Generalized Exponential Distribution
- Generalized Extreme Value Distribution
- Generalized Gamma Distribution
- Generalized Half-Logistic Distribution
- Generalized Hyperbolic Distribution
- Generalized Inverse Gaussian Distribution
- Generalized Normal Distribution
- Gibrat Distribution
- Gompertz (Truncated Gumbel) Distribution
- Gumbel (LogWeibull, Fisher-Tippetts, Type I Extreme Value) Distribution
- Gumbel Left-skewed (for minimum order statistic) Distribution
- HalfCauchy Distribution
- HalfNormal Distribution
- Half-Logistic Distribution
- Hyperbolic Secant Distribution
- Gauss Hypergeometric Distribution
- Inverted Gamma Distribution
- Inverse Normal (Inverse Gaussian) Distribution
- Inverted Weibull Distribution
- Johnson SB Distribution
- Johnson SU Distribution
- KSone Distribution
- KStwo Distribution
- KStwobign Distribution
- Laplace (Double Exponential, Bilateral Exponential) Distribution
- Asymmetric Laplace Distribution
- Left-skewed Lévy Distribution
- Lévy Distribution
- Logistic (Sech-squared) Distribution
- Log Double Exponential (Log-Laplace) Distribution
- Log Gamma Distribution
- Log Normal (Cobb-Douglass) Distribution
- Log-Uniform Distribution
- Maxwell Distribution
- Mielke's Beta-Kappa Distribution
- Nakagami Distribution
- Noncentral chi-squared Distribution
- Noncentral F Distribution
- Noncentral t Distribution
- Normal Distribution
- Normal Inverse Gaussian Distribution
- Pareto Distribution
- Power Log Normal Distribution
- Power Normal Distribution
- Power-function Distribution
- R-distribution Distribution
- Rayleigh Distribution
- Rice Distribution
- Reciprocal Inverse Gaussian Distribution
- Semicircular Distribution
- Studentized Range Distribution
- Student t Distribution
- Trapezoidal Distribution
- Triangular Distribution
- Truncated Exponential Distribution
- Truncated Normal Distribution
- Truncated Weibull Minimum Extreme Value Distribution
- Tukey-Lambda Distribution
- Uniform Distribution
- Von Mises Distribution
- Wald Distribution
- Weibull Maximum Extreme Value Distribution
- Weibull Minimum Extreme Value Distribution
- Wrapped Cauchy Distribution

Why do we care?
* Double-check your results.
* Test how methods respond to different assumptions.
* Especially the tails of the distribution are important.

Try to get a repartoire of distributions under your belt that you can quickly employ.

How long do we have to wait for the law of large numbers to kick in?

The weak law of large numbers

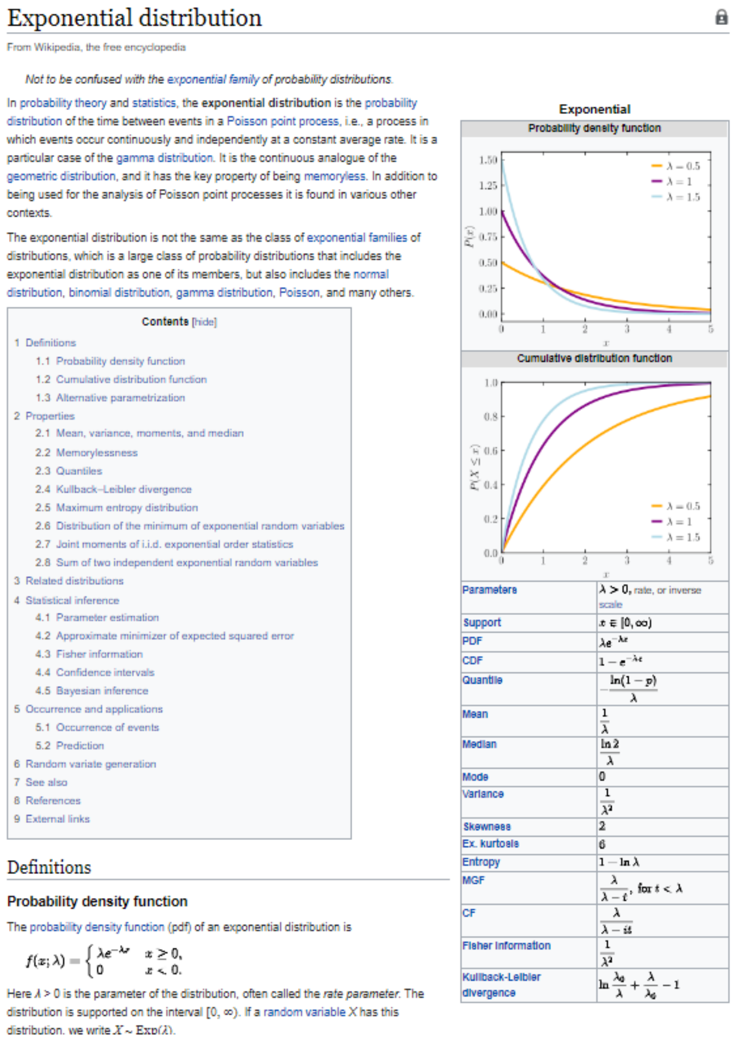
If X_i are iid with finite expected value $E(X_1)$, its samplemean converges in probability to$\overline{X}_n$

$$\overline{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i \xrightarrow{p} E(X_1)$$

1. How fast can we expect this to work?
2. Is it equally fast for every density? If so, when can we expect it to be really slow?

Exercise!

Read about distributions on wikipedia, the scipy documentation and other places!



Exponential distribution

From Wikipedia, the free encyclopedia

Not to be confused with the exponential family of probability distributions.

In probability theory and statistics, the exponential distribution is the probability distribution of the time between events in a Poisson point process, i.e., a process in which events occur continuously and independently at a constant average rate. It is a particular case of the gamma distribution. It is the continuous analogue of the geometric distribution, and it has the key property of being memoryless. In addition to being used for the analysis of Poisson point processes it is found in various other contexts.

The exponential distribution is not the same as the class of exponential families of distributions, which is a large class of probability distributions that includes the exponential distribution as one of its members, but also includes the normal distribution, binomial distribution, gamma distribution, Poisson, and many others.

Contents [hide]

Definitions

Probability density function

The probability density function (pdf) of an exponential distribution is

$$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x} & x \ge 0, \\ 0 & x < 0. \end{cases}$$

Here $\lambda > 0$ is the parameter of the distribution, often called the *rate parameter*. The distribution is supported on the interval $[0, \infty)$. If a random variable X has this distribution, we write $X \sim \mathrm{Exp}(\lambda)$.

Parameters	$\lambda > 0$, rate, or inverse scale
Support	$x \in [0, \infty)$
PDF	$\lambda e^{-\lambda x}$
CDF	$1 - e^{-\lambda x}$
Quantile	$-\frac{\ln(1-p)}{\lambda}$
Mean	$\frac{1}{\lambda}$
Median	$\frac{\ln 2}{\lambda}$
Mode	0
Variance	$\frac{1}{\lambda^2}$
Skewness	2
Ex. kurtosis	6
Entropy	$1 - \ln \lambda$
MGF	$\frac{\lambda}{\lambda - t}$, for $t < \lambda$
CF	$\frac{\lambda}{\lambda - it}$
Fisher information	$\frac{1}{\lambda^2}$
Kullback-Leibler divergence	$\ln \frac{\lambda_0}{\lambda} + \frac{\lambda}{\lambda_0} - 1$

Most pages contain all this nice information.

Also look out for interpretations and how fat tails they have.

What is are these distributions?

```
>>> from scipy.stats import norm
>>> print(norm)
<scipy.stats._continuous_distns.norm_gen object at 0x000001F874A77910>
```

They are instances of the Python class scipy.stats.rv_continuous.

Methods:
These can be called, such as norm.pdf, i.e., they are functions.

Attributes:
Not called, such as x.dtype, i.e., they are data.

Classes and objects are part of object-oriented Python. I would urge you to go watch a video or something, just learn a little about it on your own. (Tell me if you find good resources, etc.)

What is this `norm`? What can I do with it? Use `help` to view the docstring.

```
>>> help(norm)
Help on norm_gen in module scipy.stats._continuous_distns:

<scipy.stats._continuous_distns.norm_gen object>
    A normal continuous random variable.

    The location (``loc``) keyword specifies the mean.
    The scale (``scale``) keyword specifies the standard deviation.

    As an instance of the `rv_continuous` class, `norm` object inherits from it
    a collection of generic methods (see below for the full list),
    and completes them with details specific for this particular distribution.

    Methods
    -------
    rvs(loc=0, scale=1, size=1, random_state=None)
```

What methods / attributes does it have?
Use `dir` to view them!

```
>>> dir(norm)
['__call__', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '_
_getattribute__', '__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '_
_module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__setstate__', '__s
izeof__', '__str__', '__subclasshook__', '__weakref__', '_argcheck', '_argcheck_rvs', '_attach_argparser_method
s', '_attach_methods', '_cdf', '_cdf_single', '_cdfvec', '_construct_argparser', '_construct_default_doc', '_co
nstruct_doc', '_ctor_param', '_entropy', '_fit_loc_scale_support', '_fitstart', '_get_support', '_isf', '_logcd
f', '_logpdf', '_logsf', '_mom0_sc', '_mom1_sc', '_mom_integ0', '_mom_integ1', '_moment_error', '_munp', '_nnlf
', '_nnlf_and_penalty', '_open_support_mask', '_parse_arg_template', '_parse_args', '_parse_args_rvs', '_parse_
args_stats', '_pdf', '_penalized_nnlf', '_ppf', '_ppf_single', '_ppf_to_solve', '_ppfvec', '_random_state', '_r
educe_func', '_rvs', '_rvs_size_warned', '_rvs_uses_size_attribute', '_sf', '_stats', '_stats_has_moments', '_s
upport_mask', '_unpack_loc_scale', '_updated_ctor_param', 'a', 'b', 'badvalue', 'cdf', 'entropy', 'expect', 'ex
tradoc', 'fit', 'fit_loc_scale', 'freeze', 'generic_moment', 'interval', 'isf', 'logcdf', 'logpdf', 'logsf', 'm
ean', 'median', 'moment', 'moment_type', 'name', 'nnlf', 'numargs', 'pdf', 'ppf', 'random_state', 'rvs', 'sf',
'shapes', 'stats', 'std', 'support', 'var', 'vecentropy', 'xtol']
```

Looks like junk, but you'll probably use `dir` a lot because you forget the names of methods!

The end