480 - 20080528 - stats 2

Stats -- day 2

I talked about distributions in scipy.stats last time. Today I'll talk about some of numpy and scipy's other statistical capabilities. On Friday I'll talk about using R via Sage.

Distributions

Last time I mentioned that scipy has nearly 100 distinct distributions, and for each distribution one can compute lots of information. For example:

```
import scipy.stats
```

```
help(scipy.stats)
```

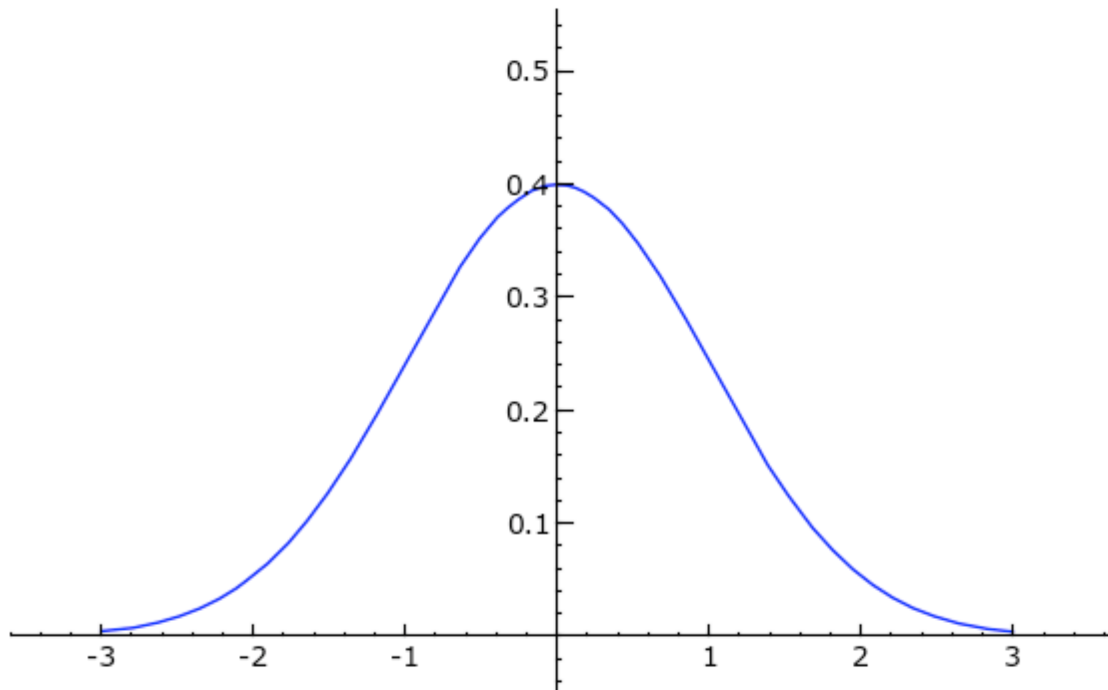
Click to open help window

```
n = scipy.stats.norm()
```

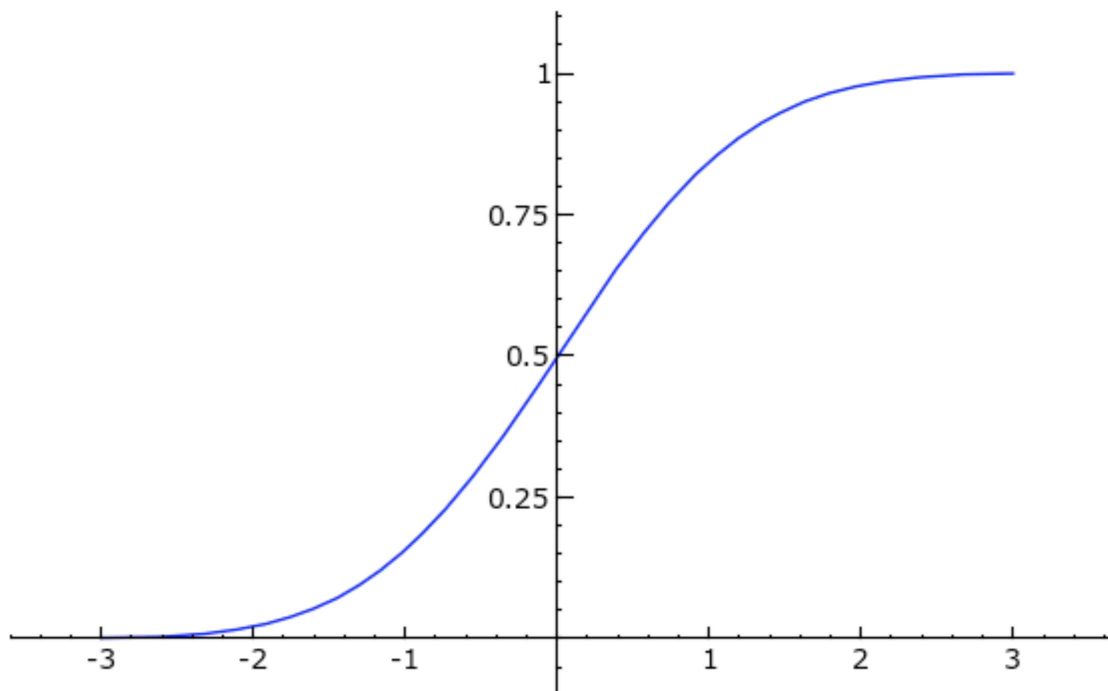
```
n
```

```
<scipy.stats.distributions.rv_frozen object at 0x8848310>
```

```
plot(n.pdf, -3, 3).show(ymin=0, ymax=0.5)
```




```
plot(n.cdf, -3, 3).show(ymin=0)
```




```
# The entropy of a distribution measures its "disorder".
# It is by definition the integral of  - pdf(x) * log(pdf(x)):
n.entropy()
```

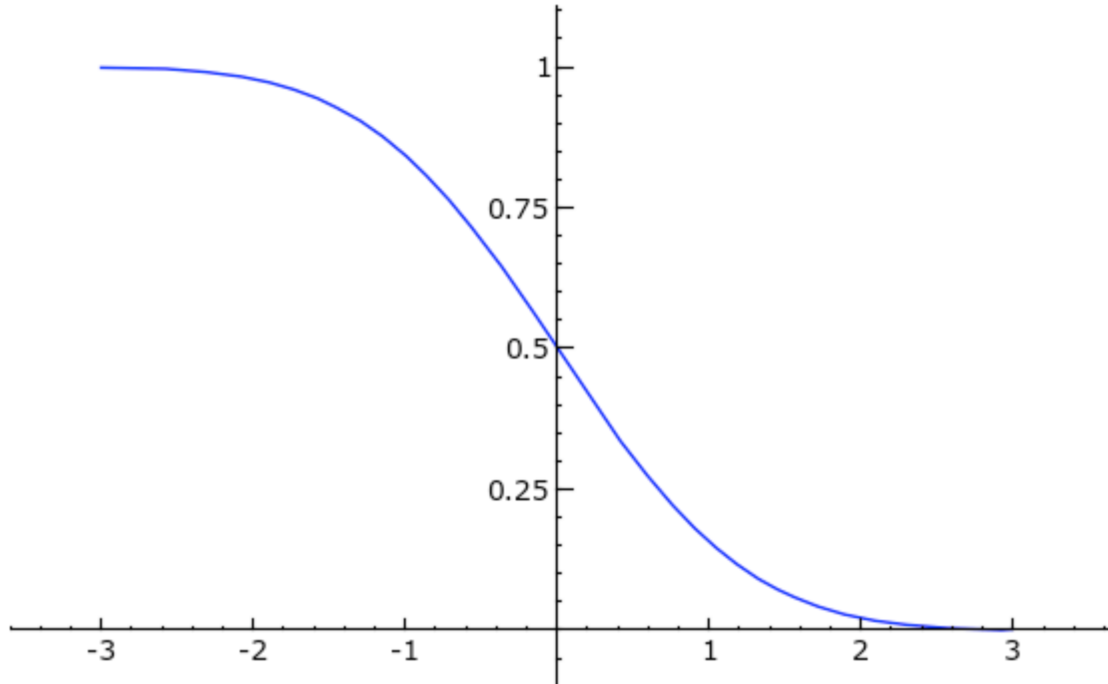
```
array(1.4189385332046727)
```

```
integral_numerical(lambda x: - n.pdf(x) * math.log(n.pdf(x)), -10,
10)
```

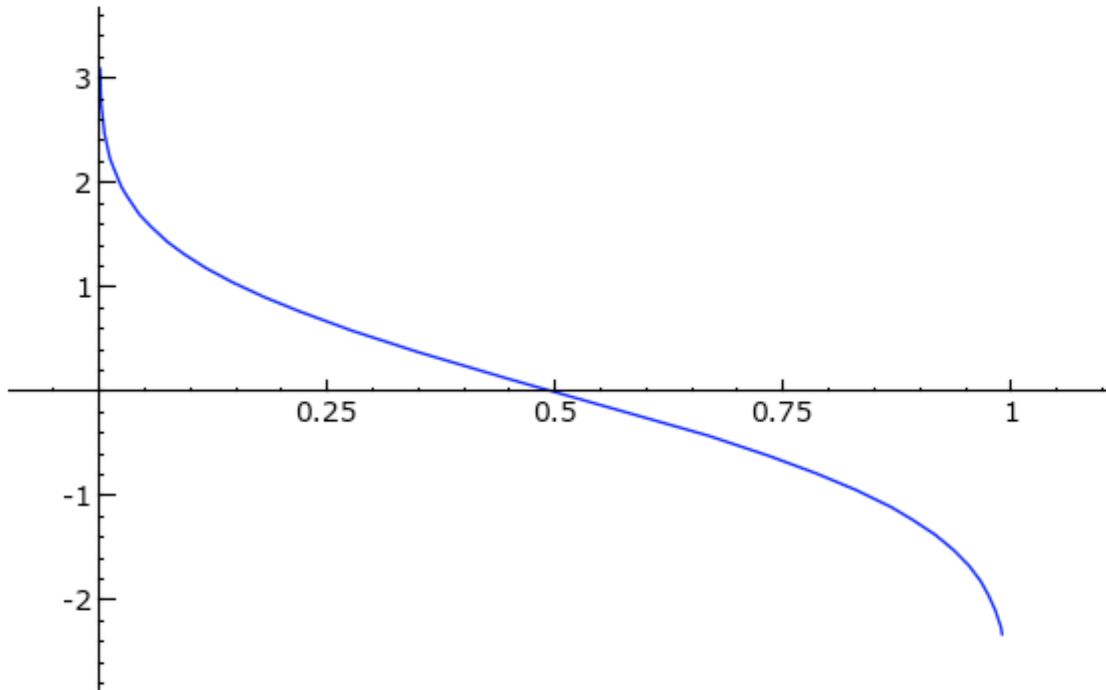
```
(1.4189385332046724, 1.5753382180282074e-14)
```

```
# The survival function of the a distribution is
# the integral from t to infinity of the density function.
# If cdf is the cumulative distribution, then sf(t) = 1 - F(t).
# This measure the probability that T >= t.

plot(n.sf, -3, 3).show(ymin=0)
```




```
# The inverse survival function is the inverse of the survival
function:
plot(n.isf, 0.001,0.99).show(xmin=0)
```



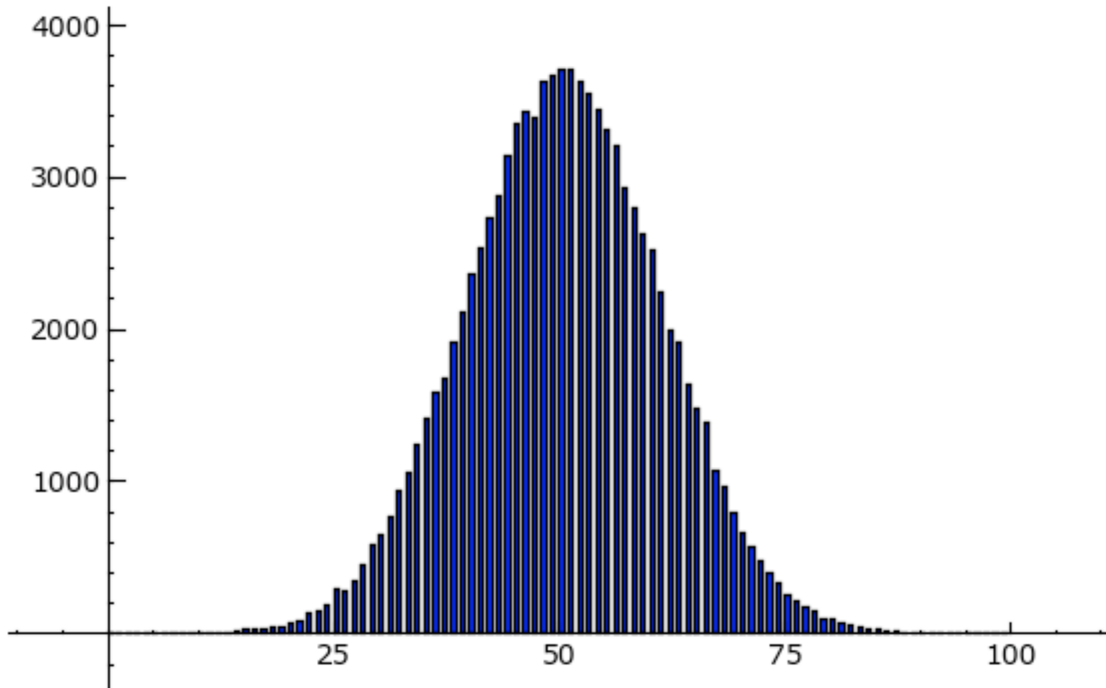

```
# rvs -- samples random values distributed as given by the
distribution
n.rvs(10)
```

```
array([-0.60137756, -0.48778485, -0.70903973,  0.81877853,
0.86778174,
       -0.14627077, -1.16265023,  0.30674602,  0.48502231,
1.22423788])
```

```
# very fast in some cases (but very slow sometimes too).
time w = n.rvs(10^5)
```

```
Time: CPU 0.01 s, Wall: 0.02 s
```

```
bar_chart(scipy.stats.histogram(w,100)[0])
```




```
# The n-th moment of a probability distribution f(x) is the
integral of
#     x^n * f(x)
# from -oo to oo.

# This is unfortunately broken in scipy.stats for the normal
distribution.
n.moment(0)
```

```
Traceback (click to the left for traceback)
...
TypeError: moment() got an unexpected keyword argument 'size'
```

```
n.moment(2)
```

```
Traceback (click to the left for traceback)
...
TypeError: moment() got an unexpected keyword argument 'size'
```

```
# Try to work around the problem:
n.dist.moment(0, *n.args)
```

```
1.0
```

```
numerical_integral(lambda x: n.pdf(x), -99, 99)
```

```
(0.99999999999999989, 7.3469192985879716e-12)
```

```
n.dist.moment(1, *n.args)
```

```
0.0
```

```
numerical_integral(lambda x: x*n.pdf(x), -99, 99)
```

```
(0.0, 5.282835432102192e-19)
```

```
# seems to work... but actually sometimes gives wrong answers...
n.dist.moment(2, *n.args)
```

```
0.0
```

```
numerical_integral(lambda x: x^2*n.pdf(x), -99, 99)
```

```
(0.99999999999999956, 9.4760945619804947e-10)
```

```
n.dist.moment(4, *n.args)
```

```
3.0
```

```
numerical_integral(lambda x: x^4*n.pdf(x), -99, 99)
```

```
(2.9999999999999996, 1.3111362030804319e-09)
```

```
# seems to work... but actually soon gives wrong answers
n.dist.moment(5, *n.args)
```

```
Traceback (click to the left for traceback)
...
AttributeError: 'module' object has no attribute 'integrate'
```

```
numerical_integral(lambda x: x^5*n.pdf(x), -99, 99)
```

```
(0.0, 3.5619409945211177e-16)
```

```
# probability mass function  -- probability that the random variable
# is a specific value.  I think this should give a "not defined error"
# since it should only be for a discrete random variable (or return 0).
n.pmf(1)
```

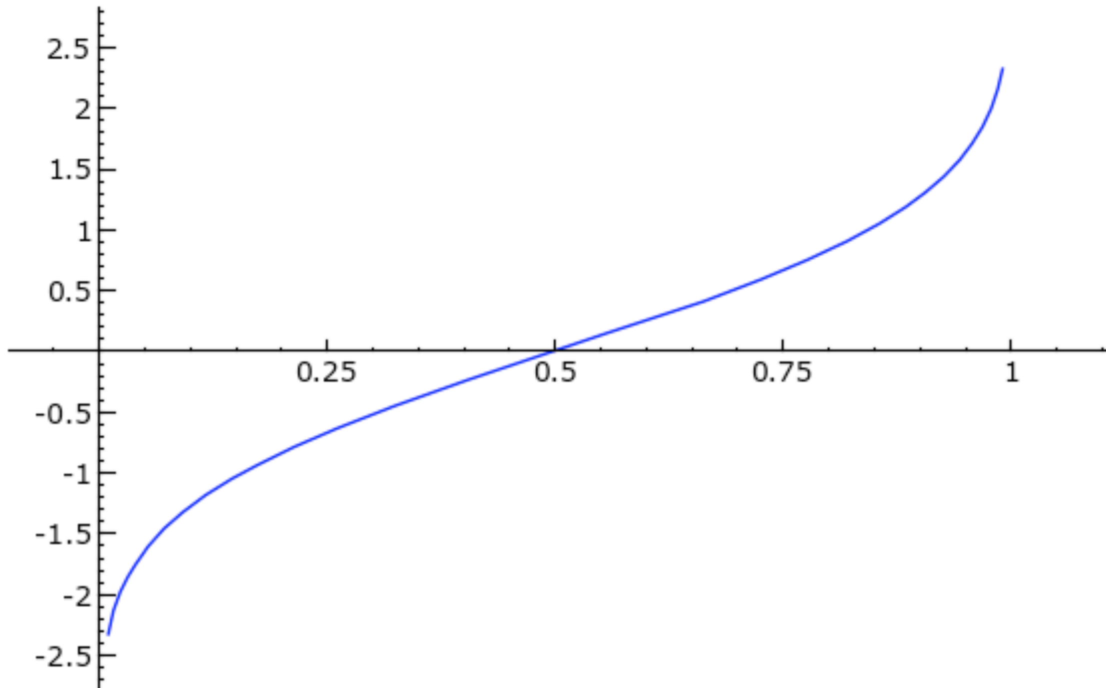
```
Traceback (click to the left for traceback)
...
AttributeError: 'norm_gen' object has no attribute 'pmf'
```

```
# ppf -- the "percent point function", which is the inverse of the
# cumulative distribution function; "percentiles".
# I.e., "when does the cummulative distribution function get to a certain value"?

# The plot below says: "75% of normally distributed values are less than 0.67"

print n.ppf(float(0.75))
plot(n.ppf, 0.01,0.99).show(xmin=0)
```

```
0.674489750196
```




```
# stats returns mean, variance, and sometimes (?) skew and kurtosis

n.stats()
```

```
(array(0.0), array(1.0))
```

```
>>> # from the source
"Disclaimers:  The function list is obviously incomplete and,
worse, the
functions are not optimized.  All functions have been tested (some
more
so than others), but they are far from bulletproof.  Thus, as with
any
free software, no warranty or guarantee is expressed or implied.
:-)  A
few extra functions that don't appear in the list below can be
found by
interested treasure-hunters.  These functions don't necessarily
have
```

```
both list and array versions but were deemed useful"
```

```
v = n.rvs(100)
```

```
# The mean -- the classical "average".

scipy.stats.mean(v)
```

```
0.10832274563246694
```

```
# The harmonic mean
scipy.stats.hmean(v)
```

```
-2.9979532429053095
```

```
scipy.stats.hmean([1,2,0])
```

```
Traceback (click to the left for traceback)
...
ZeroDivisionError: float division
```

```
# The geometric mean, which is the n-th root of (x_0 * ... * x_n)
scipy.stats.gmean([1..10])
```

```
4.5287286881167654
```

```
v
```

```
array([ 0.6352313 , -2.74305546,  0.23279514,  0.55641052,
-1.83930802,
       -0.02831757, -1.24177275, -0.32360579,  0.05510738,
-1.6586782 ,
        0.57082531,  1.68184578, -0.77633882,  0.87807009,
0.50773898,
        2.13950834, -0.7858098 , -1.37511357, -0.68977451,
-0.20984626,
        0.21328201,  0.05485112,  2.07075191, -0.3981853 ,
-0.9919303 ,
       -0.98475559, -0.943483  ,  0.3762825 ,  0.58204237,
-0.09923454,
       -0.30966291,  0.02823091,  1.01030658,  0.65921989,
-1.15215078,
       -1.55545166, -0.6306082 ,  1.51076331,  0.1725328 ,
-0.22600167,
        1.01721412, -0.69592177, -0.0977495 ,  1.0197558 ,
0.59372615,
       -0.81435141,  1.20194648, -0.34834079,  0.51773903,
1.80602891,
        0.75048913,  0.91635459, -0.8196556 ,  1.38377389,
1.33266656,
       -0.59453844,  0.74769506, -0.06721443,  1.21583508,
0.57842726,
       -0.81090007, -0.019981  , -1.02610834,  1.68057037,
-0.42166734,
        1.43781697,  0.32168116,  0.16062242,  1.07433282,
1.29269717,
```

```
        0.57986803,  0.28775614,  0.31114162, -0.50652343,
1.00951456,
        0.06693564, -0.51460016,  0.93954855, -1.4257728 ,
-0.15077741,
        0.80028853,  2.32555492,  0.58648602, -0.43079115,
1.07772596,
        2.08538682,  0.16377104, -1.88367861, -1.20063545,
-0.30771167,
        0.766689  , -0.35327534, -0.38100907, -0.49434124,
0.84337743,
       -1.79894001, -0.46966755, -0.10931582, -0.64199078,
1.35160498])
```

```
# The median, which is the value in the middle:
scipy.stats.median(v)
```

```
0.061021506216104894
```

```
# The mode is the (least) value that occurs the most frequently.
scipy.stats.mode(v)
```

```
(array([-2.74305546]), array([ 1.]))
```

```
min(v)
```

```
-2.743055459281901
```

```
scipy.stats.mode([1,2,5,-3,2,2,8,5,5])
```

```
(array([2], dtype=object), array([ 3.]))
```

```
scipy.stats.moment(v,0)
```

```
1.0
```

Statistical Functions

Scipy.stats also has many standard statistical functions by Gary Strangman and Travis Oliphant.
Take a look at the source code: stats.py.txt.

```
scipy.stats.moment(v,2)
```

```
0.99814107225933613
```

```
# variation --
scipy.stats.variation(v)
```

```
9.2230869698037754
```

```
# standard deviation
scipy.stats.std(v)
```

```
1.0041032343900007
```

```
# standard deviation
scipy.stats.stderr(v)
```

```
0.10041032343900007
```

```
scipy.stats.skew(v)
```

```
array(-0.10748369673169043)
```

```
scipy.stats.skew(v)
```

```
array(-0.10748369673169043)
```

```
scipy.stats.kurtosis(v)
```

```
-0.22644040941499721
```

```
scipy.stats.normaltest(v)
```

```
(0.27047656340717907, 0.87350774597416958)
```

```
scipy.stats.histogram(v,5)
```

```
(array([  1.,  14.,  45.,  35.,   5.]), -3.3766317561637109,
1.2671525937636203, 0)
```

```
scipy.stats.ttest_1samp(v, 0)
```

```
(1.078800883439776, 0.28329792853188568)
```

```
scipy.stats.ttest_1samp(v, 2)
```

```
(-18.839469783370824, 1.6537036035471282e-34)
```

```
scipy.stats.chisquare(v)
```

```
(921.45104560585378, 1.5728348207769911e-133)
```

```
# THERE ARE MANY MORE STATISTICAL FUNCTIONS
```

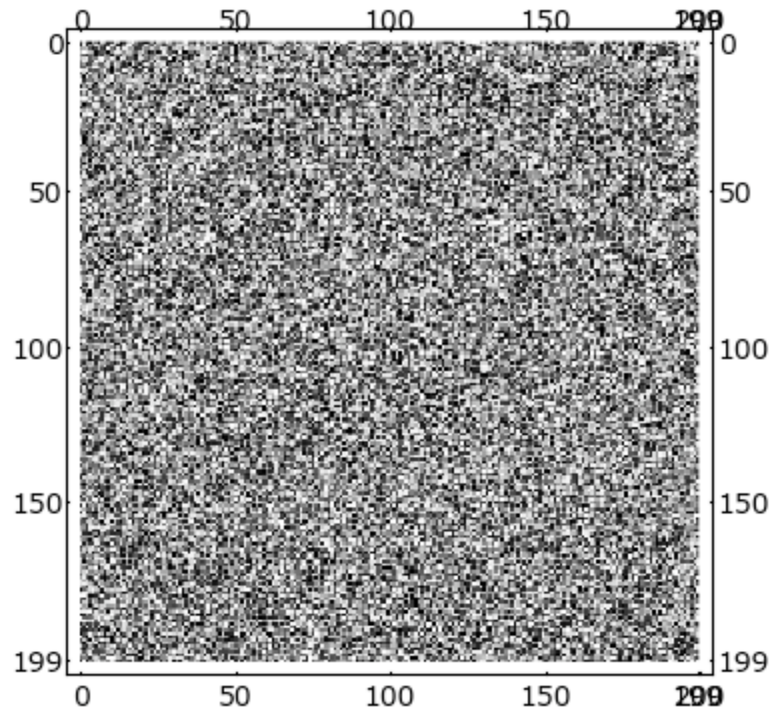
Statistical functions on nd arrays

numpy has excellent statistical functionality for nd arrays.

```
import numpy
```

```
# matrix with uniform random entries on [-1,1]
v = numpy.random.rand(200,200)
```

```
matrix_plot(matrix(v))
```




```
numpy.mean(v)
```

```
0.50021682381507104
```

```
numpy.mean(v)
```

```
0.50021682381507104
```

```
v = numpy.random.rand(10)
```

```
numpy.corrcoef(v,v)
```

```
array([[ 1.,  1.],
       [ 1.,  1.]])
```

```
numpy.corrcoef(v[1:],v[:-1])
```

```
array([[ 1.        , -0.02031074],
       [-0.02031074,  1.        ]])
```

```
v = [1..10]
```

```
numpy.corrcoef(v[1:],v[:-1])
```

```
array([[ 1.,  1.],
       [ 1.,  1.]])
```

```
# matrix with uniform random entries on [-1,1]
v = scipy.stats.norm().rvs(200^2)
```

```
len(v)
```

```
40000
```

```
v.shape = (200,200)
```

```
matrix_plot(matrix(v))
```