IX-Randomness

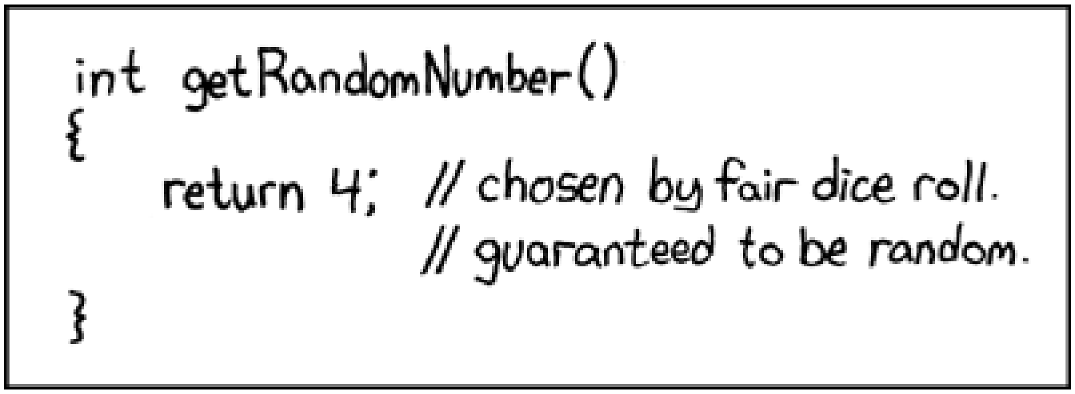
December 7, 2014

1 IX-Randomness

```
In [1]: from IPython.display import Image
```

```
In [2]: Image('http://imgs.xkcd.com/comics/random_number.png')
```

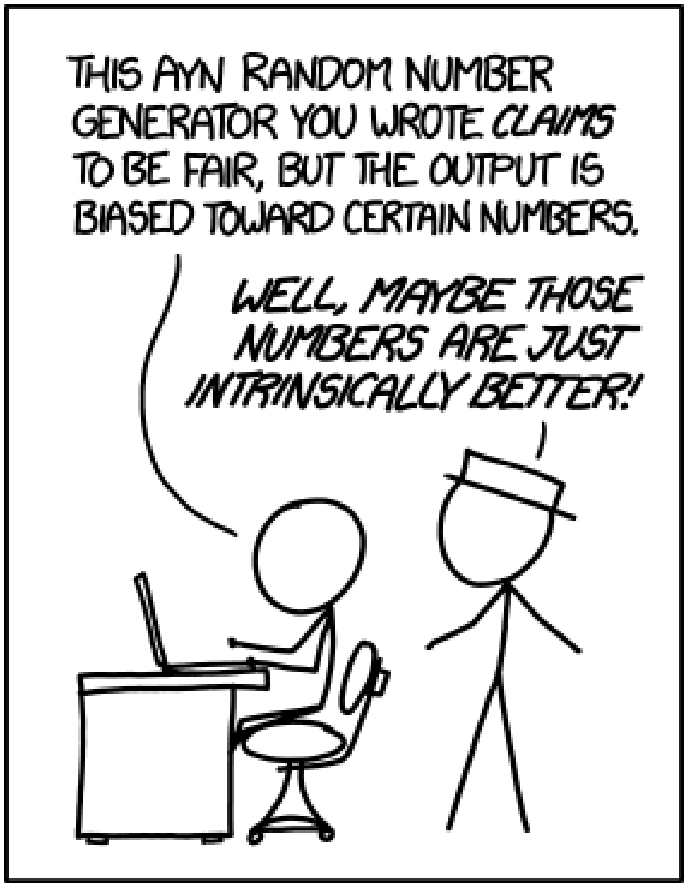
```
Out[2]:
```



```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

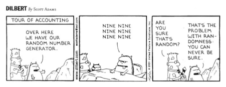
```
In [3]: Image('http://imgs.xkcd.com/comics/ayn_random.png')
```

```
Out[3]:
```


THIS AYN RANDOM NUMBER GENERATOR YOU WROTE *CLAIMS* TO BE FAIR, BUT THE OUTPUT IS BIASED TOWARD CERTAIN NUMBERS.
WELL, MAYBE THOSE NUMBERS ARE JUST INTRINSICALLY BETTER!

```
In [4]: Image('http://www.random.org/analysis/dilbert.jpg')
```

```
Out[4]:
```


DILBERT By Scott Adams
TOUR OF ACCOUNTING
OVER HERE WE HAVE OUR RANDOM NUMBER GENERATOR.
NINE NINE NINE NINE NINE NINE
ARE YOU SURE THAT'S RANDOM?
THAT'S THE PROBLEM WITH RANDOMNESS: YOU CAN NEVER BE SURE.
www.dilbert.com

```
In [15]: import numpy as np
```

The *random* subpackage is the package responsible for all random functions in the numpy environment

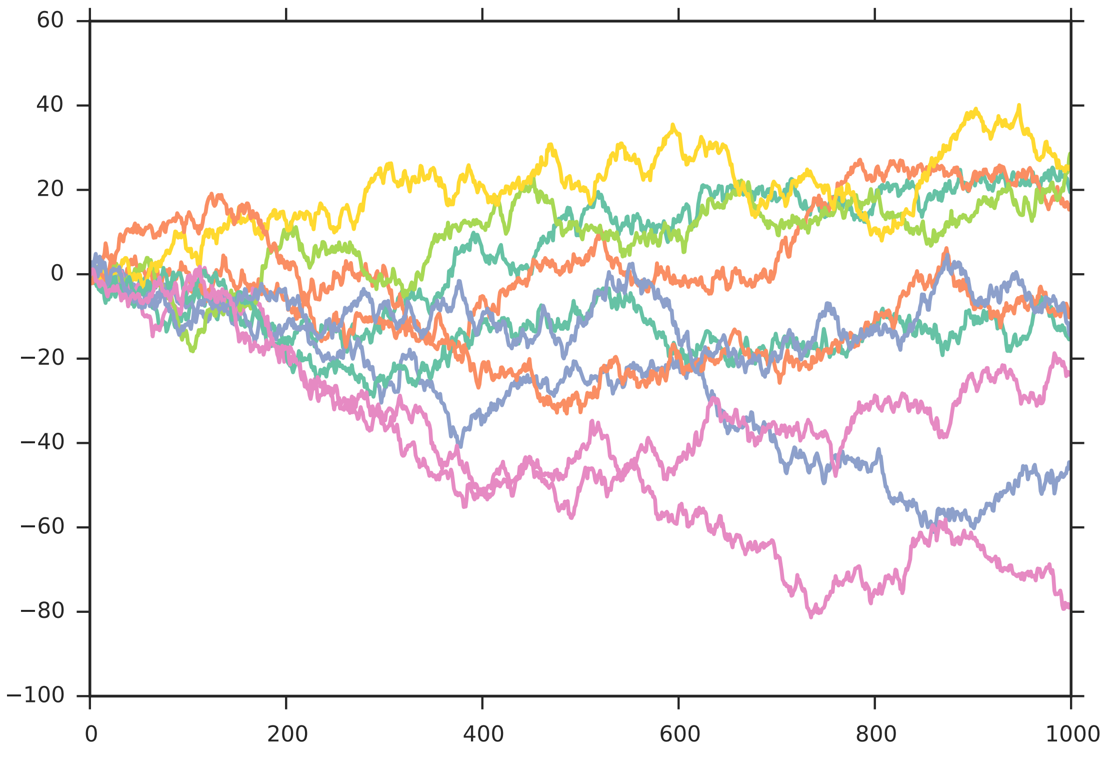
```
In [16]: #help(np.random)
```

Let's generate a couple of random paths

```
In [26]: Nt = 1000
         Npaths = 10
         rv = np.random.randn(10,1000)
```

```
In [27]: rv.shape
```

```
Out[27]: (10, 1000)

In [28]: paths = rv.cumsum(1)

In [29]: paths.shape

Out[29]: (10, 1000)

In [21]: %matplotlib inline
         from matplotlib.pyplot import *
         import seaborn as sns
         sns.set(style='ticks', palette='Set2')

In [22]: plot(paths.T)

Out[22]: [<matplotlib.lines.Line2D at 0x7f09d8b351d0>,
          <matplotlib.lines.Line2D at 0x7f09d8b23650>,
          <matplotlib.lines.Line2D at 0x7f09d8b238d0>,
          <matplotlib.lines.Line2D at 0x7f09d8b23110>,
          <matplotlib.lines.Line2D at 0x7f09d8a9f110>,
          <matplotlib.lines.Line2D at 0x7f09d8a9f2d0>,
          <matplotlib.lines.Line2D at 0x7f09d8b3e6d0>,
          <matplotlib.lines.Line2D at 0x7f09d8a9f650>,
          <matplotlib.lines.Line2D at 0x7f09d8a9f810>,
          <matplotlib.lines.Line2D at 0x7f09d8a9f9d0>]

/home/jpsilva/anaconda/lib/python2.7/site-packages/matplotlib/font_manager.py:1279: UserWarning: findfor
  (prop.get_family(), self.defaultFamily[fontext]))
```



```
In [38]: from scipy import stats
```

We can easily get a list of available distributions (continuous and discrete) by using *dir* or by checking on the online documentation

```
In [45]: list_distributions = dir(stats)
         len(list_distributions)

Out[45]: 243
```

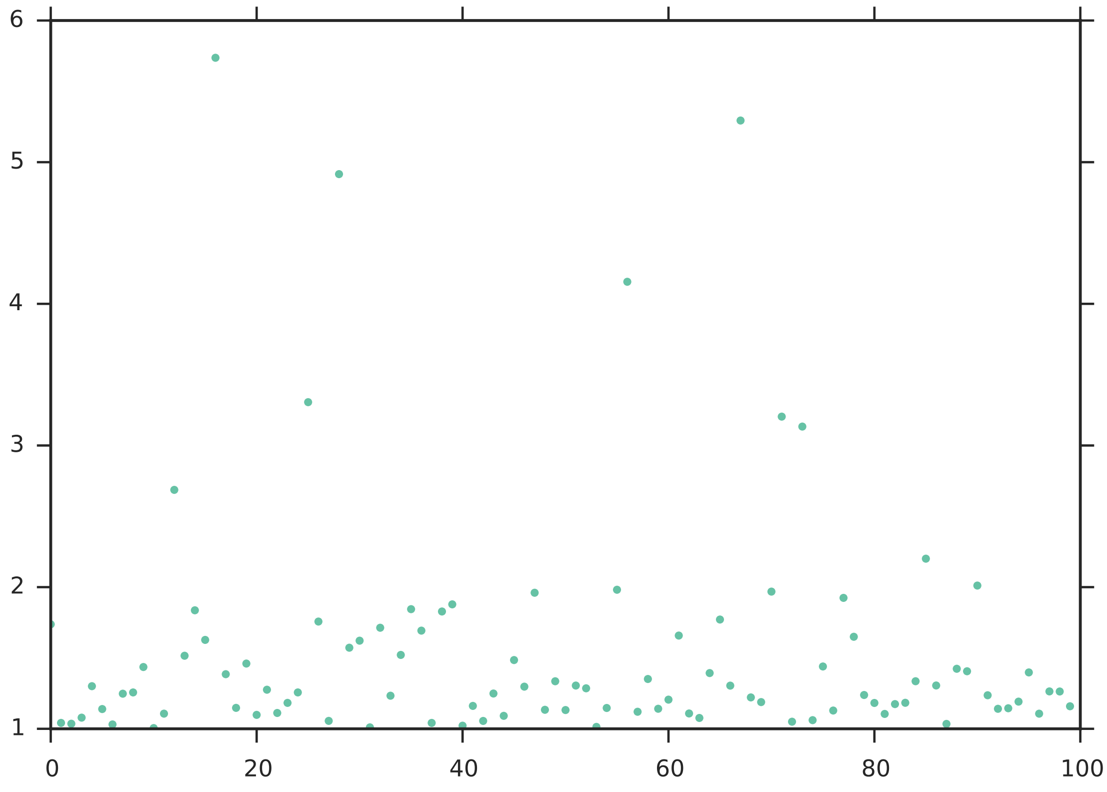
Let's choose a Pareto distribution with parameter $\alpha = 3$

$$pdf_{pareto}(x) = \frac{\alpha}{x^{\alpha+1}}$$

```
In [51]: alpha = 3
         X = stats.pareto(alpha)
```

We can sample...

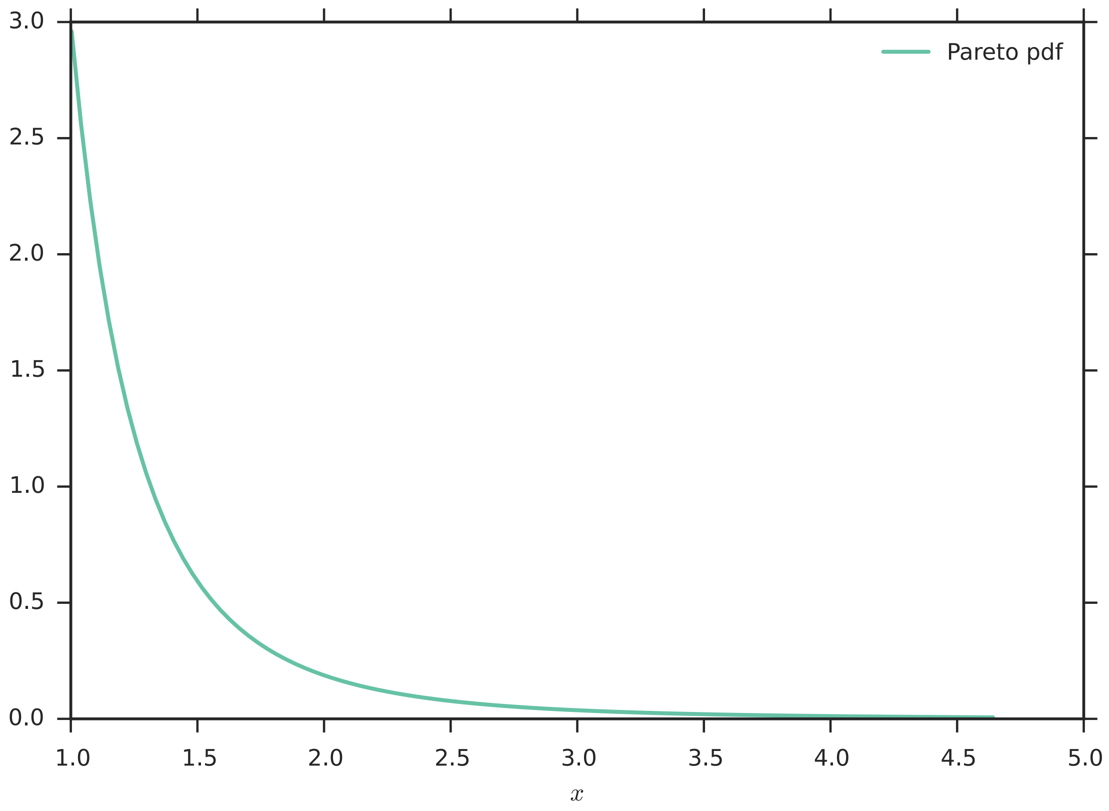
```
In [50]: n = 100
         plot(X.rvs(n),'.')

Out[50]: [<matplotlib.lines.Line2D at 0x7f09d6ce03d0>]
```



...have a look at the probability density function...

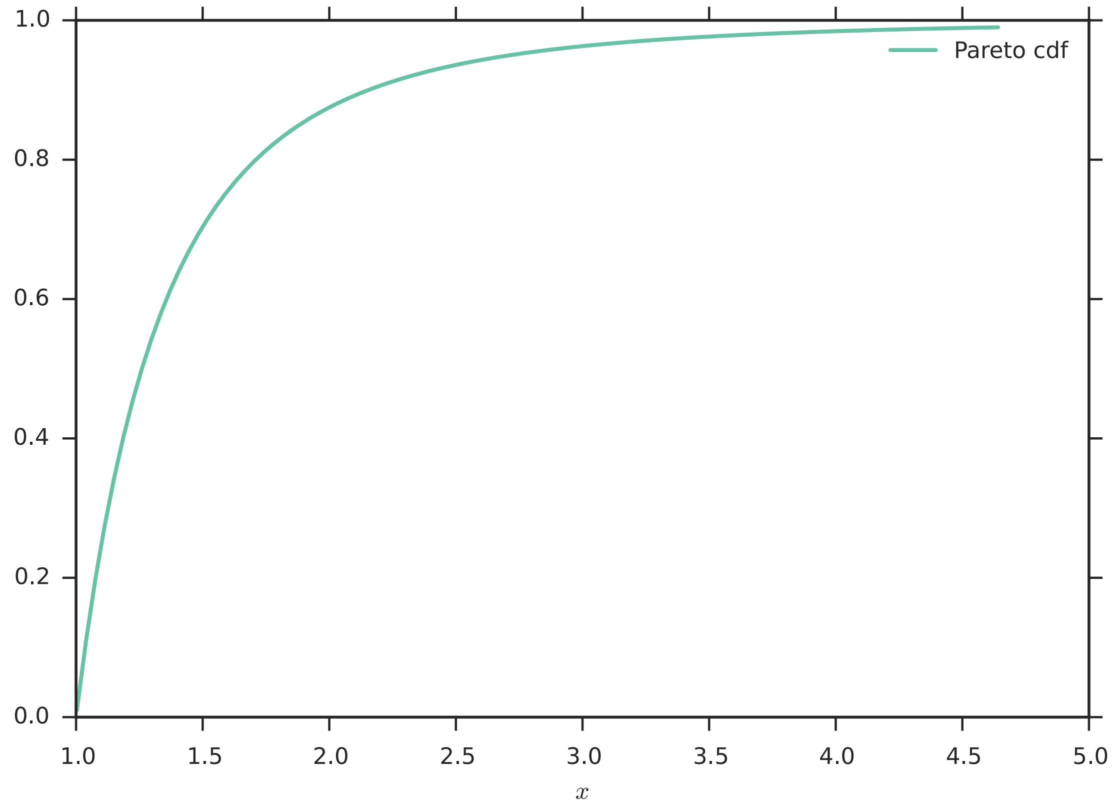
```
In [58]: x = np.linspace(X.ppf(0.01), X.ppf(0.99), 100)
         plot(x, X.pdf(x),label='Pareto pdf')
         legend()
         xlabel('$x$')
```

```
Out[58]: <matplotlib.text.Text at 0x7f09d691bf90>
```



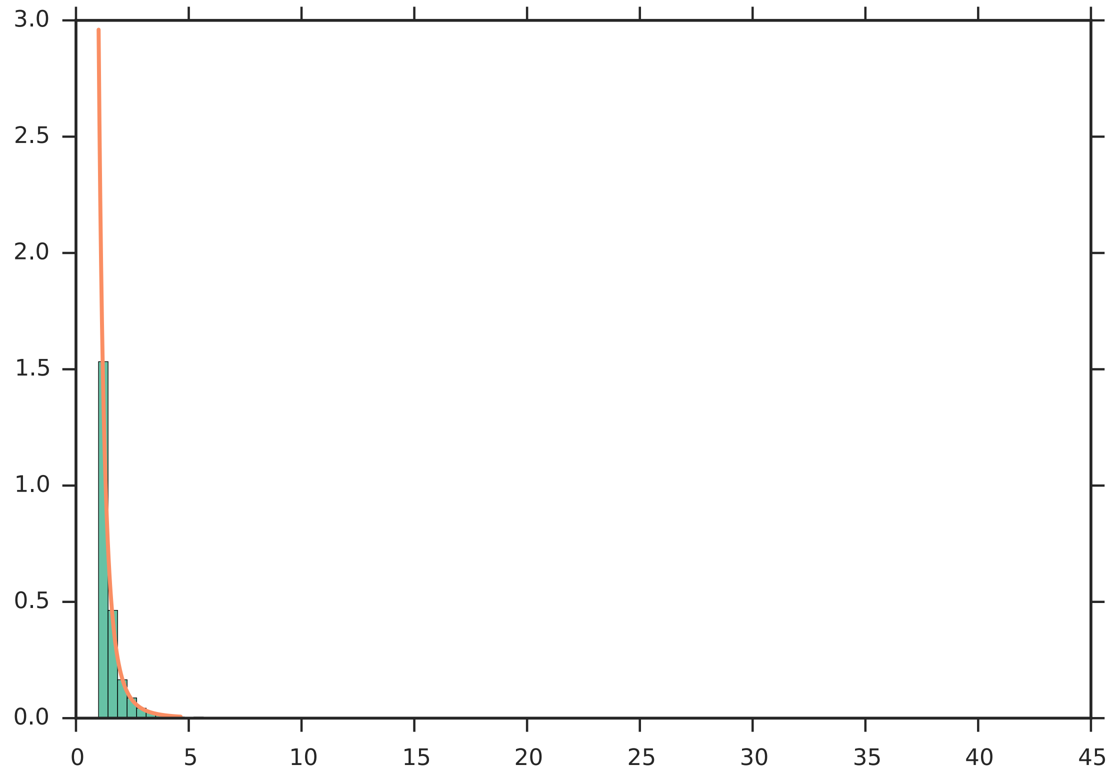
. . . have a look at the cumulative density function. . .

```
In [60]: plot(x, X.cdf(x),label='Pareto cdf')
         legend()
         xlabel('$x$')

Out[60]: <matplotlib.text.Text at 0x7f09d67ef390>
```



...compare the histogram of the previously generated sample with the pdf...

```
In [68]: fig, ax = subplots()
         ax.hist(X.rvs(10000),normed=True,bins=100)
         ax.plot(x,X.pdf(x))

Out[68]: [<matplotlib.lines.Line2D at 0x7f09d6016590>]
```



(let's just see the influence of the number of bins in the non-parametric estimation)

```
In [86]: %matplotlib
         from matplotlib.widgets import Slider

         samples = X.rvs(10000)

         ax = axes([0.1,0.25,0.8,0.6])
         ax.hist(samples,normed=True)
         ax.plot(x,X.pdf(x))
         sl = Slider(axes([0.1,0.1,0.8,0.1]),'Bins',1,200,valinit=10,valfmt='%d')

         def update(data):
             data = int(data)
             ax.cla()
             ax.hist(samples,normed=True,bins=data)
             ax.plot(x,X.pdf(x))
             ax.draw()

         sl.on_changed(update)
```

```
Using matplotlib backend: Qt4Agg
```

```
Out[86]: 0
```

```
In [87]: X.stats(moments='mv')
```

```
Out[87]: (array(1.5), array(0.75))
```

```
In [23]: from scipy.stats import kstest

In [30]: kstest(rv,'norm')


        ---------------------------------------------------------------------------
    ValueError                                Traceback (most recent call last)

        <ipython-input-30-f62508e716a2> in <module>()
    ----> 1 kstest(rv,'norm')


        /home/jpsilva/anaconda/lib/python2.7/site-packages/scipy/stats/stats.pyc in kstest(rvs, cdf, ar
       3433
       3434     if alternative in ['two-sided', 'greater']:
    -> 3435         Dplus = (np.arange(1.0, N+1)/N - cdfvals).max()
       3436         if alternative == 'greater':
       3437             return Dplus, distributions.ksone.sf(Dplus,N)


        ValueError: operands could not be broadcast together with shapes (10,) (10,1000)


In [31]: kstest(rv,'norm')


        ---------------------------------------------------------------------------
    ValueError                                Traceback (most recent call last)

        <ipython-input-31-f62508e716a2> in <module>()
    ----> 1 kstest(rv,'norm')


        /home/jpsilva/anaconda/lib/python2.7/site-packages/scipy/stats/stats.pyc in kstest(rvs, cdf, ar
       3433
       3434     if alternative in ['two-sided', 'greater']:
    -> 3435         Dplus = (np.arange(1.0, N+1)/N - cdfvals).max()
       3436         if alternative == 'greater':
       3437             return Dplus, distributions.ksone.sf(Dplus,N)


        ValueError: operands could not be broadcast together with shapes (10,) (10,1000)


In [33]: rv.apply(kstest,axis=1)


        ---------------------------------------------------------------------------
    AttributeError                            Traceback (most recent call last)

        <ipython-input-33-0e6a60028561> in <module>()
    ----> 1 rv.apply(kstest,axis=1)


        AttributeError: 'numpy.ndarray' object has no attribute 'apply'
```

```
In [34]: np.apply_along_axis(lambda x: kstest(x,'norm'),1,rv)

Out[34]: array([[ 0.02462349,  0.57921451],
               [ 0.02313629,  0.65813139],
               [ 0.02690737,  0.46175665],
               [ 0.01927064,  0.85160859],
               [ 0.02455753,  0.58267687],
               [ 0.0551436 ,  0.00439168],
               [ 0.02582343,  0.51737188],
               [ 0.02793807,  0.41210544],
               [ 0.02163583,  0.73737606],
               [ 0.02397197,  0.61361516]])

In [35]: map(lambda x: kstest(x,'norm'),rv)

Out[35]: [(0.024623490897107081, 0.57921450514349404),
          (0.02313628625770614, 0.65813139144955546),
          (0.026907370554564158, 0.46175664577782749),
          (0.019270644257675884, 0.85160859135592293),
          (0.02455753392874805, 0.58267687044366556),
          (0.055143604679474989, 0.0043916809490187614),
          (0.025823431496813432, 0.51737188246826193),
          (0.027938071951015719, 0.41210543959431489),
          (0.021635830354071017, 0.73737606358434737),
          (0.023971974347128389, 0.61361516398657845)]

In [36]: %%timeit
         np.apply_along_axis(lambda x: kstest(x,'norm'),1,rv)

100 loops, best of 3: 6.61 ms per loop

In [37]: %%timeit
         map(lambda x: kstest(x,'norm'),rv)

100 loops, best of 3: 6.4 ms per loop

In [23]: np.apply_along_axis??

In [93]: t_statistic, p_value = stats.ttest_ind(X.rvs(size=1000), X.rvs(size=1000))
         print p_value

0.433472011446

In [90]:

In []:

In []:

In []:

In []:

In []:

In [25]: from os import urandom

In [26]: ur = urandom(16)
In [27]: ur.encode('hex')

Out[27]: 'a745c7bc678df931b1ac1945baf2c7d1'

In [28]: import random

In [29]: random.random()

Out[29]: 0.9962189820806353
```

What about choosing from some predetermined set?

```
In [1]: from os import urandom

In [9]: urandom(2).encode('hex')

Out[9]: '27b9'

In [10]: a = urandom(64)
         a.encode('base-64')

Out[10]: '1Ln50JCxw1DmD87MiiHHZDv0xjNpcG/h0uiGS2Y4kG6h1g0qAoySBKYiGX31SxD1weDkaF0YkTwa\nu1Akx92EHg==\n'

In [11]: from base64 import b64encode
         from os import urandom

         random_bytes = urandom(64)
         token = b64encode(random_bytes).decode('utf-8')
         token

Out[11]: u'H+y/fHdAumBMVJr3CKG6PJX/dg1su8WEc48z9I9MYECc0KDqiXz9+WoPS0/rlt/cJdyyZAvw68mPa+uFyEbpZw=='

In [3]: %load_ext version_information
        %version_information numpy

The version_information extension is already loaded. To reload it, use:
  %reload_ext version_information

Out[3]:
```

Software	Version
Python	2.7.8 —Anaconda 2.1.0 (64-bit)— (default, Aug 21 2014, 18:22:21) [GCC 4.4.7 20120313 (Red Hat 4.4.7-1)
IPython	2.3.0
OS	posix [linux2]
numpy	1.9.1
Thu Dec 04 14:48:14 2014 CET	

```
In [1]: from IPython.core.display import HTML
        def css_styling():
            styles = open("./styles/custom.css", "r").read()
            return HTML(styles)
        css_styling()

Out[1]: <IPython.core.display.HTML at 0x7fc5082118d0>

In []:
```