Math 246 Unit 9: Random numbers, and a simple example of simulation

Brenton LeMesurier, Revised December 7, 2015

Random numbers are often useful both for simulation of physical processes and for generating a coleciot of test cases. Here we will do a mathematical simulation: approximating π on the basis that the unit circle occupies a fraction $\pi/4$ of the 2×2 square enclosing it.

Actualy, the best we can do is *pseudo-random* numbers, generated by algorithms that actually produce a very long but eventually repeating sequence of numbers.

Module random within package numpy

The pseudo-random number generator we use are provided by package numpy in its module random – full name `numpy.random`. We introduce the abbreviation "npr" for this, along with the standard abbreviation "np" for numpy:

In [1]:

```
import numpy as np
import numpy.random as npr
```

This module contains numerous random number generators; here we look at just a few.

Uniformly distributed real numbers: `numpy.random.rand`

First, the function `rand` (full name `numpy.random.rand`) generates uniformly diatributed real numbers in the interval $[0, 1)$. To generate a single value, use it with no argument:

In [2]:

```
n_samples = 4
for sample in range(n_samples):
    print(npr.rand())
```

```
0.7181735107188684
0.030607117127731787
0.30117305899677893
0.5227383377487944
```

To generate an array of values all at once, one can specify how many as the first and only input argument:

```
In [3]:
```

```
numbers = npr.rand(n_samples)
print(numbers)
```

```
[ 0.99607832  0.72084518  0.43720712  0.84177438]
```

We can also generate multi-dimensional arrays, by giving the lengths of each dimension as arguments:

```
In [4]:
```

```
numbers2d = npr.rand(2,3)
print('A two-dimensional array of random numbers:\n', numbers2d)
numbers3d = npr.rand(2,3,4)
print('A three-dimensional array of random numbers:\n', numbers3d)
```

```
A two-dimensional array of random numbers:
 [[ 0.93302322  0.02388334  0.93848828]
 [ 0.84817969  0.12852778  0.59739349]]
A three-dimensional array of random numbers:
 [[[ 0.19527196  0.10562377  0.19599746  0.959718  ]
  [ 0.76737001  0.16849823  0.49688107  0.54024254]
  [ 0.61811175  0.22379335  0.14904507  0.80413901]]

 [[ 0.79253451  0.79796145  0.0512036   0.53773175]
  [ 0.57747377  0.28497849  0.05243709  0.46889627]
  [ 0.05070672  0.29107842  0.97633367  0.12880998]]]
```

Normally distributed real numbers: `numpy.random.randn`

The function `randn` has the same interface, but generates numbers with the standard normal distribution of mean zero, standard deviation one:

```
In [ ]:
```

```
print('Ten normally distributed values:\n', npr.randn(20))
```

```
Ten normally distributed values:
 [ 0.93313917  0.63348417  0.45688394 -0.70140356 -1.79858511  0.485
453
 -1.07562837 -0.54633824 -0.80697793  0.62853872  0.70684386  1.3616
281
 -0.32520941 -1.00624555 -0.54698334 -0.18441301 -0.01914501  0.6084
6587
 -0.09658742 -0.22564081]
```

In [ ]:

```
n_samples = 10**8
normf_samples = npr.randn(n_samples)
mean = sum(normf_samples)/n_samples
print('The mean of these', n_samples, 'samples is', mean)
standard_deviation = np.sqrt((sum(normf_samples**2) - mean**2)/n_samples)
print('and their standard deviation is', standard_deviation)
```

Random integers: `numpy.random.random_integers`

One can generate integers, uniformly distributed betwen specified lower and upper values:

In [ ]:

```
n_dice = 30
dice_rolls = npr.random_integers(1, 6, n_dice)
print(n_dice, 'random dice rolls:\n', dice_rolls)
# Count each outcome: this needs a list instead of an array:
dice_rolls_list = list(dice_rolls)
for value in (1, 2, 3, 4, 5, 6):
    count = dice_rolls_list.count(value)
    print(value, 'occured', count, 'times')
```

Things average out with more rolls:

In [ ]:

```
n_dice = 10**8
dice_rolls = npr.random_integers(1, 6, n_dice)
# Count each outcome: this needs a list instead of an array:
dice_rolls_list = list(dice_rolls)
for value in (1, 2, 3, 4, 5, 6):
    count = dice_rolls_list.count(value)
    print(value, 'occured a fraction', count/n_dice, 'of the time')
```

Exercise

Approximate π as follows:

- generate a **long** list of random points in the square $[-1, 1] \times [-1, 1]$ that circumscribes the unit circle, by generating successive random values for both the x and y coordinates.
- compute the fraction of these that are inside the unit circle, which should be approximately $\pi/4$.
- Multiply by four and there you are!

It takes a lot of samples to get decent accuracy, so experiment with successively more of them.

Also, repeat several times with each choice of number of samples, to see the variation, as an indication of accuracy.