04_05_randomness

Unknown Author

April 1, 2014

Part I

Randomness : Monte Carlo techniques

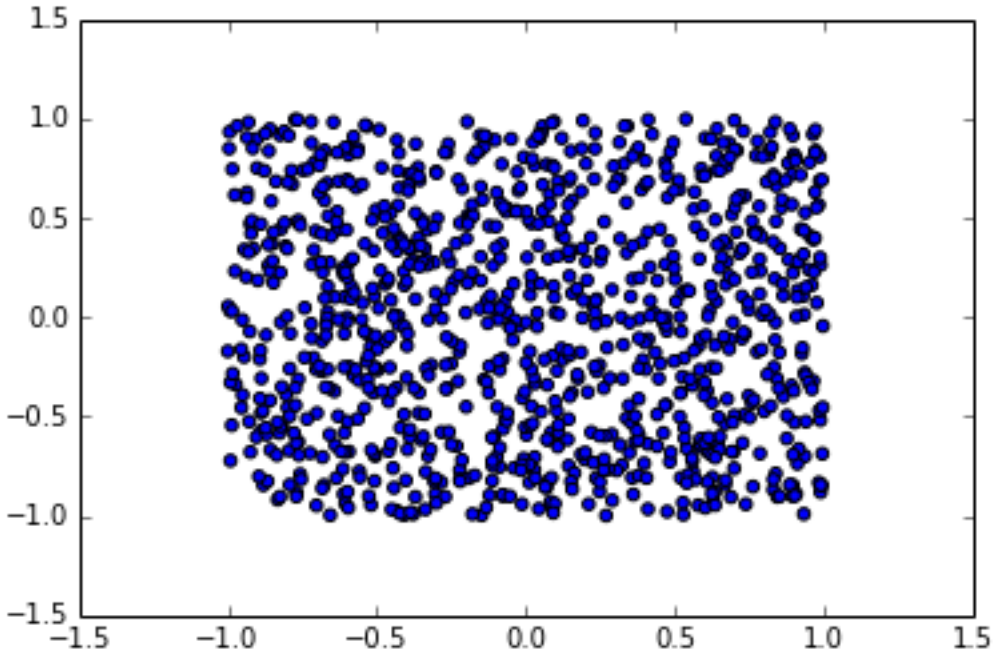
Finding the area

Let us find the area of the disc. We know that a disc is given by the equation $x^2 + y^2 \leq 1$. So if we generate (x, y) uniform randomly on the $[-1, 1] \times [-1, 1]$ square, the proportion of the points lying in the disc will give the area.

In [1]:
```
# I'll put all the imports here so that it is easy to reference what we
# are importing

import random

# This is to plot points to show what is happening
import matplotlib.pylab

# This we need to make vectorized versions of Monte Carlo simulations.
import numpy
```

In [2]:
```
%matplotlib inline
```

In [3]:
```
x = []
y = []
for i in range(1000) :
    x.append(random.uniform(-1, 1))
    y.append(random.uniform(-1, 1))
plt = matplotlib.pylab.scatter(x, y)
matplotlib.pylab.show(plt)
```



In [4]:
```
def area_circle(N) :
    i = 0
    incircle = 0
    areaofsquare = 2*2 # 2 = length of [-1, 1]
    while i < N :
        x = random.uniform(-1, 1)
        y = random.uniform(-1, 1)
        if x*x + y*y <= 1 :
            incircle += 1
        i += 1
    area = (float(incircle)/N) * areaofsquare
    return area
```

In [5]:
```
print area_circle(1000000)
```

```
3.141052
```

1 Monte Carlo Simulations

What we did above was some kind of Monte Carlo simulation. We estimated the probability of a random point on the aforementioned square to lie on the circle. That probability turned out to be π.

However computations only seem to make sense when the number of iterations are large. Because of that, I explain how to do a numpy vectorized version of the code which will work faster. Though for the present examples it doesn't matter much, for huge simulations the vectorized version, if done properly, can save you a lot of time.

In [6]:
```
def vect_area_circle(N) :
    # Create a 2xN aarray of random numbers
    r = numpy.random.random((2, N)) * 2 - 1
    x = r[0, :]
    y = r[1, :]
    incircle = (x*x + y*y < 1)
    no_incircle = numpy.sum(incircle)
    return 4.0*float(no_incircle)/N
```

In [7]:
```
print vect_area_circle(1000000)
```
```
3.142084
```

In [8]:
```
%timeit      area_circle(1000000)
%timeit vect_area_circle(1000000)
```
```
1 loops, best of 3: 5.98 s per loop
1 loops, best of 3: 358 ms per loop
```

Simulations of probabilistic experiments

Consider the very simple game where you throw a dice. You gain n points if the dice shows n eyes. The first person to achieve more than or equal to 50 points will win the game.

To code this game, it is useful to introduce another random function, random.randint. `random.randint(a, b)`. This returns a uniformly distributed random number uniformly distributed between `a` and `b` with the end points included. As an example, I print 100 random integers between `1` and `6`.

In [9]:
```
no = [0, 0, 0, 0, 0, 0]
for _ in range(100) :
    r = random.randint(1, 6)
    no[r - 1] += 1
    print r, ",",
print
print "-"*10, "Distribution", "-"*10
print "1 : %3d \t 2 : %3d \t 3 : % 3d \n4 : %3d \t 5 \
: %3d \t 6 : %3d\nSum = %d" % tuple(no + [sum(no)])
```
```
1 , 6 , 2 , 4 , 2 , 4 , 6 , 5 , 4 , 2 , 3 , 6 , 6 , 6 , 6 , 2 , 2 , 5
, 2 , 2 , 2 , 5 , 1 , 2 , 3 , 6 , 3 , 4 , 5 , 6 , 6 , 4 , 5 , 2 , 2 ,
6 , 3 , 4 , 5 , 1 , 5 , 1 , 1 , 6 , 1 , 6 , 4 , 4 , 1 , 2 , 1 , 3 , 3
, 5 , 1 , 2 , 3 , 2 , 1 , 2 , 3 , 6 , 3 , 5 , 1 , 2 , 6 , 6 , 3 , 5 ,
3 , 6 , 6 , 2 , 6 , 5 , 6 , 2 , 1 , 6 , 4 , 1 , 1 , 2 , 5 , 2 , 5 , 3
, 6 , 6 , 6 , 4 , 1 , 2 , 6 , 1 , 1 , 5 , 6 , 4 ,
---------- Distribution ----------
1 :  17          2 :  21          3 :  12
4 :  11          5 :  14          6 :  25
Sum = 100
```

In [10]:
```
def stupid_game(withplayer=True) :
    """withplayer : If the user is going to input his value.
    Setting this to False will make the computer play with
    itself.
    """
    t = 50 # total to reach
    c = 0 # Computer's score
    p = 0 # player's score
    toss = random.randint(1, 2)
    if toss == 1 :
        print "Computer wins the toss."
    else :
        print "Player wins the toss."

    # Play if toss = 1
    if toss == 1 :
        throw = random.randint(1, 6)
        c += throw

    while c < t and p < t :
        if withplayer :
            throw = input("Throw : ")
            # Penalize cheaters : You can write
            # all your AI Stuff here.
            if throw > 6 :
```

```
                throw -= 100
        else :
            throw = random.randint(1, 6)
        p += throw

        throw = random.randint(1, 6)
        c += throw
        if c > p :
            print "Scores : COMP : %3d;   Player : %3d." % (c, p)
        elif c < p :
            print "Scores : Comp : %3d;   PLAYER : %3d." % (c, p)
        else :
            print "Scores : Comp : %3d;   Player : %3d." % (c, p)


    if c >= t and p >= t :
        print "There is a tie! :)"
    elif c >= t :
        print "The computer wins."
    else :
        print "The player wins."
    return None
```

In [11]:
```
stupid_game(False)
```

```
Player wins the toss.
Scores : COMP :   3;   Player :   2.
Scores : Comp :   4;   PLAYER :   6.
Scores : Comp :  10;   Player :  10.
Scores : COMP :  15;   Player :  14.
Scores : Comp :  16;   PLAYER :  17.
Scores : Comp :  17;   PLAYER :  22.
Scores : Comp :  23;   PLAYER :  25.
Scores : Comp :  25;   PLAYER :  26.
Scores : COMP :  30;   Player :  28.
Scores : COMP :  35;   Player :  33.
Scores : Comp :  36;   Player :  36.
Scores : Comp :  38;   PLAYER :  41.
Scores : Comp :  44;   PLAYER :  45.
Scores : Comp :  46;   PLAYER :  49.
Scores : Comp :  49;   PLAYER :  50.
The player wins.
```

Another example can be a simple random walk. Instead of doing graphics, let us do it using character drawings. We'll have an array of say 41 characters with a $\star$ at the 21st place and blanks everywhere else. Then we generate a random number which takes values -1 and 1 with equal probability. If it is -1 we go one character left, otherwise we go one character forward. If we hit an end, we always go away from the end.

In [12]:
```
def genm1p1() :
    """Generate -1 and 1 with equal probability."""
    r = random.randint(0,1)
    return 2*r - 1
```

In [13]:
```
def random_step(path, pm) :
    """Given a path and position, it gives a new
    path with updated position"""
    n = len(path)
    # Find the current position, cp
    cp = -1
    hitl = 0
    hitr = 0
    for i in range(n) :
```

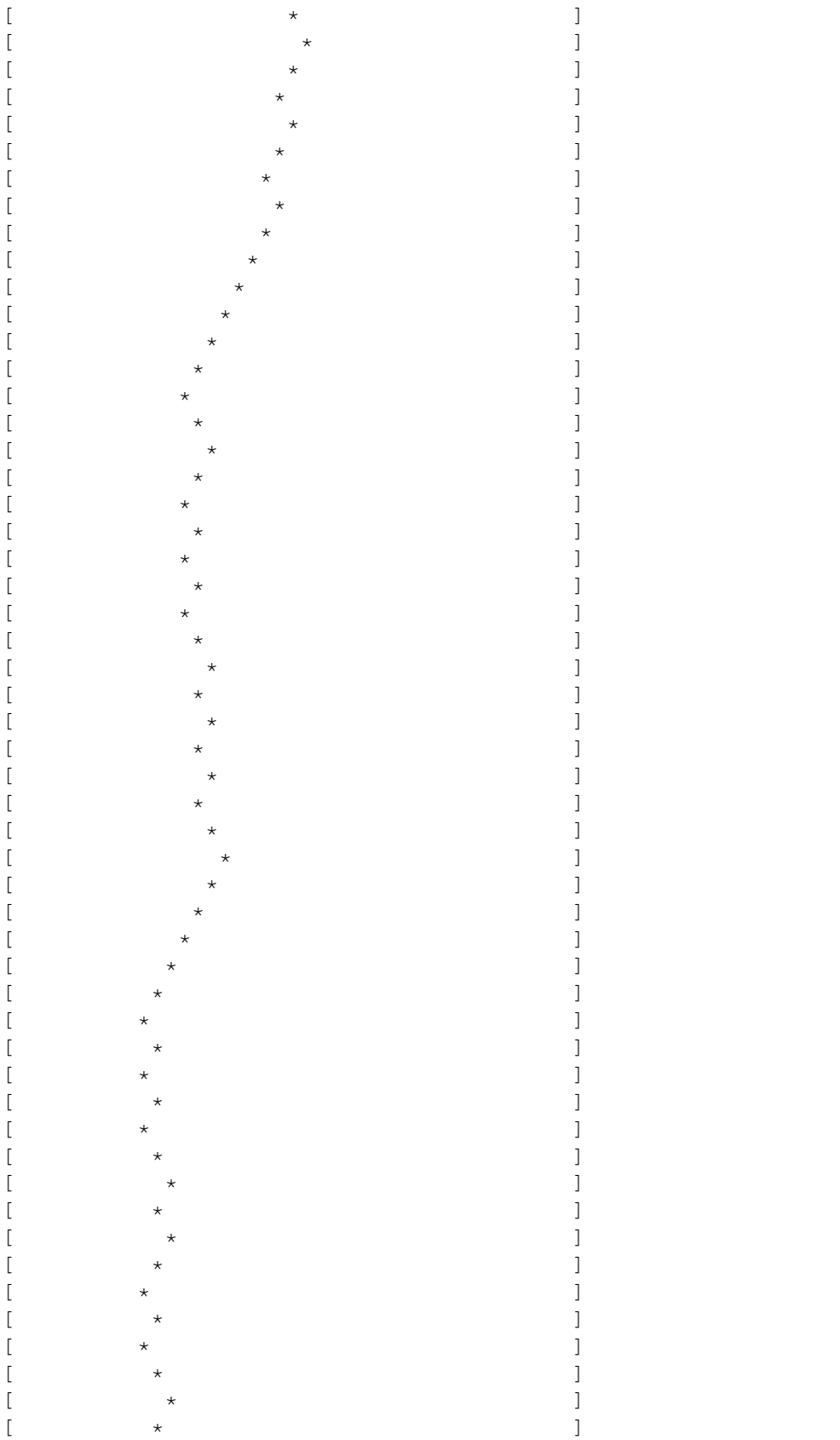
```
        if path[i] == '*' :
            cp = i
    if cp == -1 :
        print "Cannot find current position."
        retval = None
    else :
        if cp == n-1 :
            np = cp - 1 # np : new position
            hitr += 1
        elif cp == 0 :
            np = cp + 1
            hitl += 1
        else :
            np = cp + pm
        path[cp] = ' '
        path[np] = '*'
        retval = (path, hitl, hitr)
    return retval
```

In [14]:
```
def displaypath(path) :
    pathstr = ""
    for i in range(len(path)) :
        pathstr += path[i]
    print '[' + pathstr + ']'
    return None
```

In [15]:
```
def random_walk(len_path=41, no_steps=100, init_pos = -1) :
    hitl = 0
    hitr = 0
    if init_pos < 0 :
        init_pos = int(len_path/2 + 1)

    path = [' ']*len_path
    path[init_pos] = '*'
    displaypath(path)
    for _ in range(no_steps) :
        pm = genm1p1()
        (path, hl, hr) = random_step(path, pm)
        hitl += hl
        hitr += hr
        if path == None :
            break
        else :
            displaypath(path)
    print "Left wall hits : %d; Right wall hits : %d." % (hitl, hitr)
    return None
```

In [16]:
```
random_walk()
```

```
[                         *                         ]
[                        *                          ]
[                       *                           ]
[                        *                          ]
[                         *                         ]
[                        *                          ]
[                       *                           ]
[                        *                          ]
[                         *                         ]
[                        *                          ]
[                       *                           ]
[                        *                          ]
[                         *                         ]
[                        *                          ]
[                       *                           ]
```



```
[           *                                    ]
[          *                                     ]
[         *                                      ]
[        *                                       ]
[       *                                        ]
[        *                                       ]
[         *                                      ]
[          *                                     ]
[         *                                      ]
[        *                                       ]
[       *                                        ]
[        *                                       ]
[         *                                      ]
[        *                                       ]
[         *                                      ]
[       *                                        ]
[        *                                       ]
[       *                                        ]
[        *                                       ]
[       *                                        ]
[      *                                         ]
[     *                                          ]
[      *                                         ]
[       *                                        ]
[      *                                         ]
[     *                                          ]
[       *                                        ]
[      *                                         ]
Left wall hits : 0; Right wall hits : 0.
```

2 Monte Carlo Integration

The method relies of the fact that if $x_1, x_2, \ldots, x_n$ are n uniformly distributed random numbers between a and b, then

$$\int_a^b f(x)dx \simeq \frac{(b-a)}{n} \sum_{i=1}^{n} f(x_i) \tag{1}$$

Let us try this.

In [17]:
```
def mc_int(f, a, b, n) :
    s = 0
    for i in range(n) :
        x = random.uniform(a, b)
        s += f(x)
    integral = (float(b-a)/n) * s
    return integral
```

In [18]:
```
print mc_int(lambda x : 2*x, 0, 10, 10000)
```
```
99.719487342
```

In [19]:
```
# Vectorized version

def mc_int_vec(f, a, b, n) :
    x = numpy.random.uniform(a, b, n)
    s = numpy.sum(f(x))
    i = (float(b-a)/n) * s
    return i
```

In [20]:
```
totval = 0
for i in range(100) :
    intval = mc_int_vec(lambda x : 1 + 2*x, 0, 10, 10000)
    totval += intval
    #print intval, ":",
#print
print "Avg val = ", (totval/100)
```
```
Avg val =  110.061918581
```