Numerical Python

CS*101* `lec13`

`Random Numbers`

Announcements

quiz: `quiz13` due on Thurs 31/10

lab: `lab06` on Fri 01/11

hw: `hw07` due 06/11

exam: `exam02` from `lec06-13` on `13 Nov` @ `8.00pm`

MCQ, short question / programming

Roadmap

Week 1 (9/9): L0 Intro; L1 PY - Intro	Week 2 (16/9): L2 PY – Data& Bool; L3 PY – if & Loops	Week 3 (23/9): L4 PY - Lists; L5 PY - Funcs	30/9: L6 PY - Files
9/10: L7 PY - Muta; 12/10: Exam 01	Week 6 (14/10): L8 PY - Dict; L9 PY - IO	Week 7 (21/10): L10 PY - Numpy; L11 PY - Plot	Week 8 (28/10): L12 PY - State; L13 PY - Rand
Week 9 (4/11): L14 PY – De-bug; L15 PY - Eqn	Week 10 (11/11): L16 PYA I; Exam 02	Week 11 (18/11): L17 PY - Opt II; L18 PY - Mod	Week 12 (25/11): L19 PY – Sym I; L20 PY – Sym II
Week 13 (2/12): L21 PY – Sym III; L22 ML - Intro	Week 14 (9/12): L23 ML - IO; L24 ML - Poly	Week 15 (16/12): L25 ML - Stats; L26 ML - Fit	Week 16 (23/12): Review
Week 17 (30/12): Finals			

PIt Recap

Plt.plot

Which is correct?

```
plt.plot(x,y,'ro')
plt.plot(x,y,color='red',linestyle='o')
plt.plot(x,y,color='red', linestyle='-')
```

Plt.plot

Which is correct?

```
plt.plot(x,y,'ro')
```
★★★

```
plt.plot(x,y,color='red',linestyle='o')
```

```
plt.plot(x,y,color='red', linestyle='-')
```
★★★

State Recap

Modeling

1. Model a experiment/process in computer without performinng the REAL physical experiment and see what happens during this process.
2. We used the example of releasing a ball from the table. Through modeling, we can ask
    a. When will the ball hit the ground?
    b. When and where it will reach the highest speed?
    c. How many times can it bounce after hitting the ground?
    d. others....
3. We can solve these questions using:
    1. Analytical Solution

Modeling

A Use analytical equation (if available).

$$y(t) = y_0 + v_0 t + \frac{a}{2} t^2$$

$$y_0 = 1$$

$$v_0 = 0$$

$$a = -9.8$$

subject to

$$y(t) \geq 0$$

Modeling

Input

```
# Parameters of simulation
n = 100      # number of data points to plot
start = 0.0 # start time, s
end = 1.0   # ending time, s
a = -9.8    # acceleration, m*s**-2

# State variable initialization
t = np.linspace(start,end,n+1) # time, s
```

Model

```
y = 1.0 + a/2 * t**2

for i in range(1,n+1):
    if y[i] <= 0: # ball has hit the ground
        y[i] = 0
... more needed...
```

Modeling

1. Model a experiment/process in computer without performinng the REAL physical experiment and see what happens during this process.
2. We used the example of releasing a ball from the table. Through modeling, we can ask
    a. When will the ball hit the ground?
    b. When and where it will reach the highest speed?
    c. How many times can it bounce after hitting the ground?
    d. others....
3. We can solve these questions using:
    1. Analytical Solution
    2. Finite difference

Modeling

We can use finite difference is that in this problem as:

$$v_n(t) = \frac{dy_n}{dt} \approx \frac{y_{n+1} - y_n}{t_{n+1} - t_n} \rightarrow y_{n+1} = y_n + v_n\left(t_{n+1} - t_n\right) \;\; (eqn\ 1)$$

$$a = \frac{dv_n}{dt} \approx \frac{v_{n+1} - v_n}{t_{n+1} - t_n} \rightarrow v_{n+1} = v_n + a\left(t_{n+1} - t_n\right) \;\; (eqn\ 2)$$

$$v_{n=0} = 0 \qquad\qquad y_{n=0} = 1 \qquad\qquad a = -9.8$$

subject to

$$y(t) \geq 0$$

We can put

$$v_{n+1} \;\; (eqn\ 2) \;\; into\ v_n \;\; (eqn\ 1)$$

to get the "next"

$$y_{n+1}\ at\ n = n + 1$$

Modeling

Input

```
# Parameters of simulation
n = 100      # number of data points to plot
start = 0.0 # start time, s
end = 1.0    # ending time, s
a = -9.8     # acceleration, m*s**-2

# State variable initialization
t = np.linspace( start,end,n+1 )    # time, s
y = np.zeros( n+1 )                 # height, m
v = np.zeros( n+1 )                 # velocity, m*s**-1
y[ 0 ] = 1.0                        # initial condition, m
```

Model

```
for i in range( 1,n+1 ):
    v[ i ] = v[ i-1 ] + a*( t[ i ]-t[ i-1 ] )
    y[ i ] = y[ i-1 ] + v[ i ] * ( t[ i ]-t[ i-1 ] )

    if y[ i ] <= 0: # ball has hit the ground
        v[ i ] = 0
        y[ i ] = 0
...more needed...
```

Numerical Methods

A. Numerical Differentiation:

Forward difference, Backward difference, and others....

I) Forward difference,

$$\frac{dy_n}{dt} \approx \frac{y_{n+1} - y_n}{t_{n+1} - t_n}$$

II) Backward difference,

$$\frac{dy_n}{dt} \approx \frac{y_n - y_{n-1}}{t_n - t_{n-1}}$$

Objectives

Using **Numpy and Plot (lec10 and 11)**:

A. Explain how random numbers are generated from deterministic algorithms.
B. Distinguish the three basic random distributions (uniform, normal, integer) by sight.
C. Understand when to apply each of the three basic random distributions in constructing models and simulations (uniform, discrete, normal).

Randomness

Randomness

A philosophical question: what is randomness?

What are some sources of true randomness?

Randomness

A philosophical question: what is randomness?

What are some sources of true randomness?

Consider the following two sequences:

$$7\,8\,5\,3\,9\,8\,1\,6\,3\,3\,9\,7\,4\,4\,8\,3\,0\,9\,6\,1\,5\,6\,6\,0\,8\,4 \ldots$$

$$+1, -\frac{1}{3}, +\frac{1}{5}, -\frac{1}{7}, +\frac{1}{9}, -\frac{1}{11}, +\frac{1}{13}, -\frac{1}{15}, \ldots$$

Randomness

A philosophical question: what is randomness?

What are some sources of true randomness?

Consider the following two sequences:

$$7\,8\,5\,3\,9\,8\,1\,6\,3\,3\,9\,7\,4\,4\,8\,3\,0\,9\,6\,1\,5\,6\,6\,0\,8\,4\ldots$$

$$+1, -\frac{1}{3}, +\frac{1}{5}, -\frac{1}{7}, +\frac{1}{9}, -\frac{1}{11}, +\frac{1}{13}, -\frac{1}{15}, \ldots$$

These are derived from the same rule (π/4)—but one seems “random” to us.

Randomness

Pseudo-random numbers come from computer formulae.

The formula uses a *seed* (often the system clock time) to start the sequence.

It then returns a new number unpredictable to you (but predictable to the formula!) each time you query the function.

This means that anybody who has the seed value will be able to generate the same sequence of random numbers.

Randomness

Pseudo-random numbers come from computer formulae.

The formula uses a *seed* (often the system clock time) to start the sequence.

It then returns a new number unpredictable to you (but predictable to the formula!) each time you query the function.

This means that anybody who has the seed value will be able to generate the same sequence of random numbers.

Not-so-random random number!

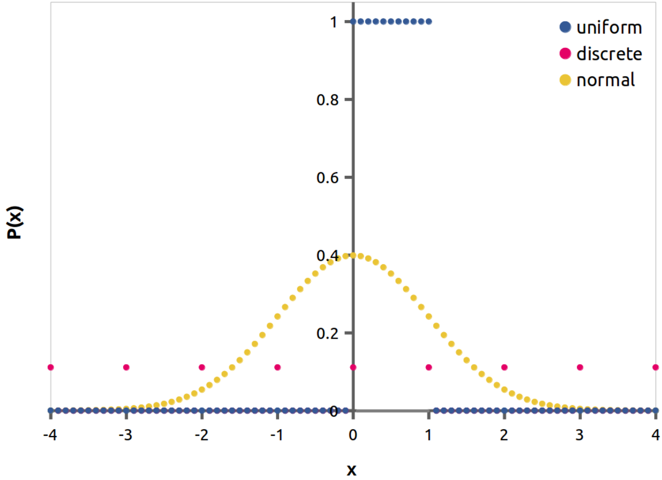
Randomness from numpy

NumPy uses the *Mersenne twister*, based on prime number distributions (but you don't need to know this).

Dozens of distributions are available—let's see a few.

Randomness

There are different ways a number can be random, or distributed.

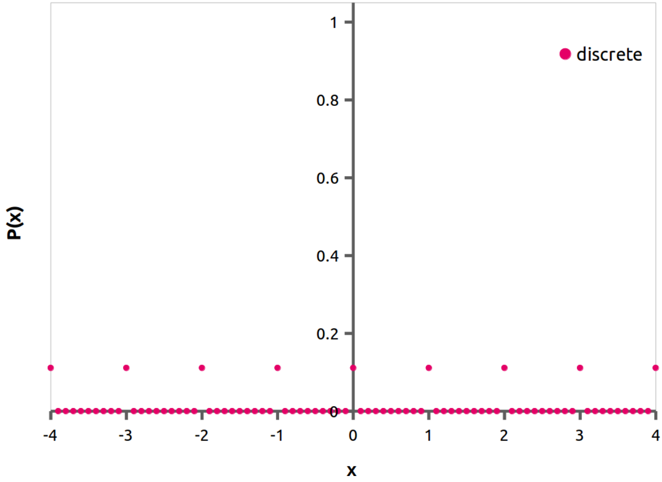


Discrete distribution

`randint` returns a random (pseudo-random) `integer` in a range (which works the same as `range`).

```
np.random.randint( 10 )  #random int between [0,10)
np.random.randint( 1,7 ) #random int between [1,7)
np.random.randint( 0,10, size=(5,5) )
np.random.randint( 0,10, size=(5,5) ) + 1
```

Discrete distribution



Uniform distribution

`uniform` returns a random `float` in the range $[0,1)$.

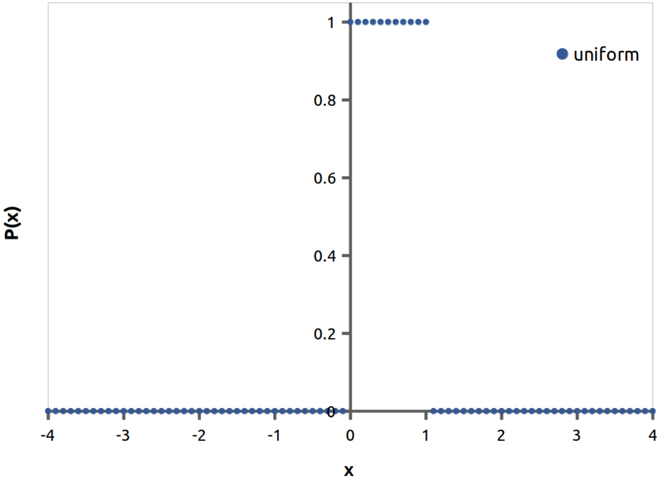
This is called a *uniform* random distribution.

```
np.random.uniform()      # random number, [0,1)
np.random.uniform(a,b)   # random number, [a,b)

np.random.uniform( size=(4,3) ) # in array

x = np.random.uniform( size=(10000,) )
or
x = np.random.uniform( size=10000)
```

Uniform distribution

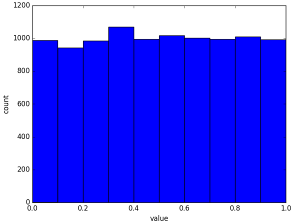


Uniform distribution

`uniform` returns a random `float` in the range $[0, 1)$.

This is called a *uniform* random distribution.

```
x = np.random.uniform( size=(10000,) )
plt.hist( x,bins=10 )
plt.show()
```



Normal distribution

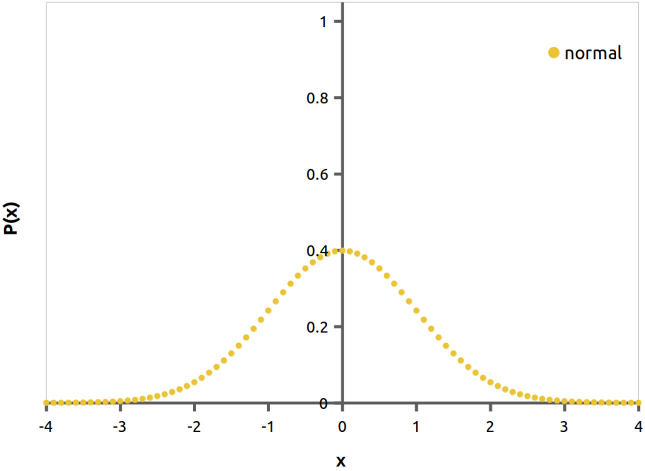
`normal` returns a random `float` selected from the *normal* distribution with mean 0 and standard deviation 1.

```
np.random.normal()          # random normal number

np.random.normal() + 1.0    # mean 1.0
np.random.normal( loc=1.0 )

(np.random.normal()) * 4    # variance 4.0
np.random.normal( scale=4.0 )
```

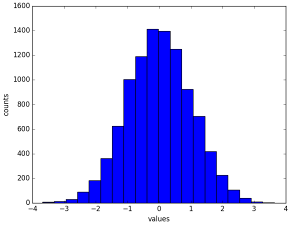
Normal distribution



Normal distribution

`normal` returns a random number selected from the *normal* distribution with mean 0 and variance 1.

```
x = np.random.normal( size=(10000,) )
plt.hist( x,bins=20 )
plt.show()
```



hist

`hist` (matplotlib) creates a *histogram*.

Histograms plot the number of times a value occurs in a data set.

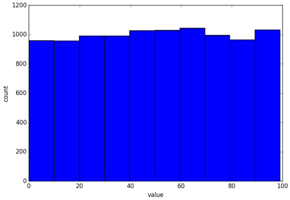
> It COUNTS the number of times a value occurs
> Then PLOTS it

hist

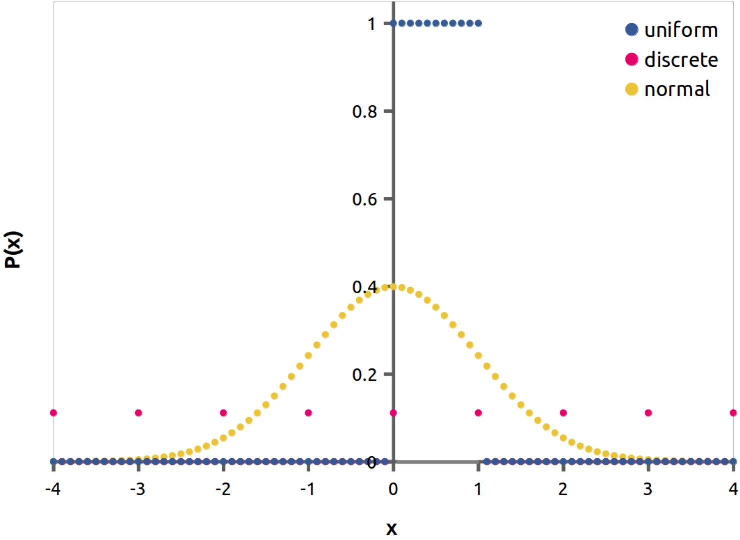
`hist` (matplotlib) creates a *histogram*.

Histograms plot the number of times a value occurs in a data set.

```
x = np.random.randint(0,100,size=(10000,1))
plt.hist(x)
plt.show()
```



Randomness



Question 1

Which command will generate

```
array([[21, 17],
       [19,  3],
       [ 7, 14]])
```

A. np.random.randint(1, 23)
B. np.random.normal(size=(3, 2)) * 10 + 3
C. np.random.randint(1, 23, size=(3, 2))
D. np.random.normal() * 10 + 3
E. np.random.uniform(1, 23, size=(3, 2))

Question 1

Which command will generate

```
array([[21, 17],
       [19,  3],
       [ 7, 14]])
```

A. np.random.randint(1, 23)
B. np.random.normal(size=(3, 2)) * 10 + 3
C. np.random.randint(1, 23, size=(3, 2)) ⋆⋆
D. np.random.normal() * 10 + 3
E. np.random.uniform(1, 23, size=(3, 2))

Question 2

```
x = np.random.randint( 0, 10, size=10000 )
count = [0]*10
for i in x:
    count[i] += 1

print(count)
plt.hist( count, bins=10)
plt.show()
```

Question 2

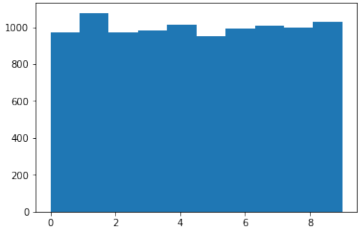
```
x = np.random.randint( 0, 10, size=10000 )
count = [0]*10
for i in x:
    count[i] += 1

print(count)
plt.hist( count, bins=10)
plt.show()
```

If count = [973, 1077, 973, 981, 1015, 950, 994, 1010, 997, 1030], What will be plotted?

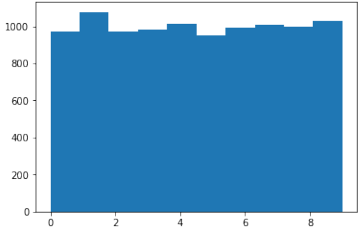
Question 2

If count = [973, 1077, 973, 981, 1015, 950, 994, 1010, 997, 1030], What will be plotted?

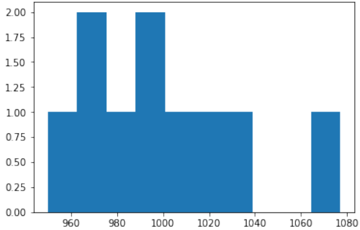


Question 2

If count = [973, 1077, 973, 981, 1015, 950, 994, 1010, 997, 1030], What will be plotted?



nope!



This! plt.hist() counts then plots!

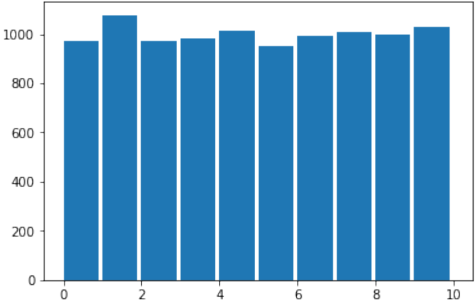
Question 2

But it is difficult to see the different bars!
Use

```
plt.hist( x, bins=(range(0,11)), rwidth=0.8)
```

> `bins` = the number of bins or the actual different bin intervals, here bins = [0,1), [1,2)...[9,10), [10,11)

> `rwidth` =displayed bar width as a fraction of the bin width



Example - Write your first game!

Number guessing:

```
import numpy as np
number = np.random.randint( 10 )+1
guess = input('Guess the number between 1 and 10:')
while int( guess ) != number:
  guess = input( 'Nope.  Try again, my grandma
                 can do better than you' )
print('You did it. Ok, you are better than
                      my grandma. Happy?')
```

np.random.choice()

`choice` randomly samples a one-dimensional array or list

```
x = [ 'red','yellow','blue','jale',
      'ulfire','octarine' ]
y = np.random.choice(x)        # random color
```

np.random.choice()

`choice` randomly samples a one-dimensional array or list

```
x = [ 'red','yellow','blue','jale',
      'ulfire','octarine' ]
y = np.random.choice(x)       # random color
```

ans: Any item inside the list `x`

np.random.choice()

`choice` by *default* samples *with replacement* - it can select the same item again the next time!

`choice` randomly samples a one-dimensional array but can do so *without replacement*.

Replacement means pulling a card from a deck and putting it back before drawing again.

```
x = np.array( range( 1,53 ) )
c = np.random.choice( x, size=5, replace=False )
```

np.random.choice()

- `choice` by *default* samples *with replacement* - it can select the same item again the next time!
- `choice` randomly samples a one-dimensional array but can do so *without replacement*.
- Replacement means pulling a card from a deck and putting it back before drawing again.

```
x = np.array( range( 1,53 ) )
c = np.random.choice( x, size=5, replace=False )
```

Ans: This code chooses five items one at a time from x array but no replacement after each choice.

np.random.shuffle()

`shuffle` randomly reorders an array in place.

```
x = np.array( range( 1,53 ) )
y = np.random.shuffle(x)
```

np.random.shuffle()

`shuffle` randomly reorders an array in place.

```
x = np.array( range( 1,53 ) )
y = np.random.shuffle(x)
```

What is `y`?

np.random.shuffle()

`shuffle` randomly reorders an array in place.

```
x = np.array( range( 1,53 ) )
y = np.random.shuffle(x)
```

What is `y`?

None!

The code above shuffles a deck of cards but does not select anything from it.

But `x` is changed!

Question 3

Which of the following will *not* reproduce the behavior of a six-sided dice in `c`?

A
```
c = np.random.normal( 6 ) + 1
```

B
```
x = np.array( range( 1,7 ) )
c = np.random.choice( x )
```

C
```
c = np.random.randint( 6 )+1
```

D
```
d = np.random.uniform() * 6
c = int(d) + 1
```

Question 3

Which of the following will *not* reproduce the behavior of a six-sided dice in `c`?

A `c = np.random.normal( 6 ) + 1` ⋆

B
```
x = np.array( range( 1,7 ) )
c = np.random.choice( x )
```

C `c = np.random.randint( 6 )+1`

D
```
d = np.random.uniform() * 6
c = int(d) + 1
```

Question 4

```
x = np.array([ ['red',1],['yellow',2],['blue',3],
    ['orange',4],['green',5],['pink',6] ])
y = np.random.shuffle(x[:,1])
print(y)
```

What are the values for x and y?

Question 4

```
x = np.array([ ['red',1],['yellow',2],['blue',3],
   ['orange',4],['green',5],['pink',6] ])
y = np.random.shuffle(x[:,1])
print(y)
```

What are the values for x and y?

y = None

x = array with all first element unchanged but second elements shuffled.

Maybe (because the order is random)

```
x = array([ ['red',5],
            ['yellow',1],
            ['blue',3],
            ['orange',2],
            ['green',6],
            ['pink',4] ])
```

What else?

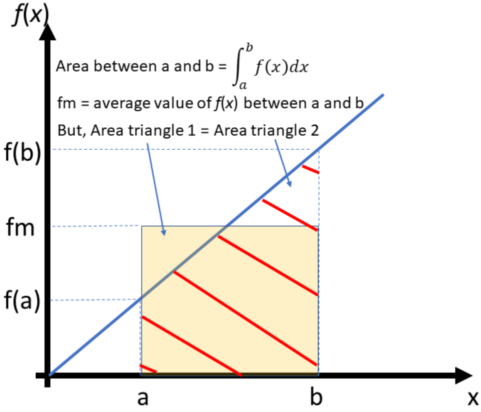
Our first game example using random numbers was pretty lame. What else can we do?

> Monte Carlo integration

Monte Carlo integration

The mean theorem in calculus for integration: If f_m is the average value of $f(x)$ between [a,b], Then

$$\int_a^b f(x)dx = f_m(b-a)$$



Monte Carlo integration

How to find f_m `which is the average value of` $f(x)$ `between [a,b]`?

1. All we need is to get many, many, many, many points between a and b and sum up the corresponding f(x) values, then take the average.

2. This is similar to if you want to know the average weight for all the Year 1 students, you will need to get the weight of many Year 1 students, add them up and average it.

   > The more students, the more accurate. No particular student is preferred.

Monte Carlo integration

To do this averaging in a computer, we can use random numbers to get N ordinates x_0,x_1,...x_{N-1}, get $f(x_0)$,$f(x_1)$,...$f(x_{N-1})$ and sum them up to find f_m:

$$f_m = x_0 + x_1 + x_2 + ...x_{N-1}$$

$$f_m = \frac{1}{N}\sum_{j=0}^{N-1} f(x_j)$$

Monte Carlo integration

```
def mcInt(f,a,b,N):
    import numpy as np
    x = 0.
    fsum = 0.
    for i in range(N):
        x = np.random.uniform(a,b)
        fsum += f(x)
    return fsum/N*np.abs(b-a)
```

What else?

Our first toy example was pretty lame. What else can we do?

> Monte Carlo integration

> Random walk

Random walk

How to generate a program that randomly plot a point up or down or right or left from original point?

```
import numpy as np
import matplotlib.pyplot as plt

x = np.zeros( ( 100,1 ) )
y = np.zeros( ( 100,1 ) )
```

Random walk

```
for i in range( 1,len(x) ):
    dir = np.random.randint(4)
    if dir == 0:
        x[i] = x[i-1]
        y[i] = y[i-1]+1
    if dir == 1:
        x[i] = x[i-1]+1
        y[i] = y[i-1]
    if dir == 2:
        x[i] = x[i-1]
        y[i] = y[i-1]-1
    if dir == 3:
        x[i] = x[i-1]-1
        y[i] = y[i-1]
plt.plot(x,y)
plt.show()
```

What else?

Our first toy example using random numbers was pretty lame. What else can we do?

> Monte Carlo integration

> Random walk

> Others: scientific applications (quantum mechanics).

Summary

A. Different distributions and ways to get random sampling using numpy

```
np.random.uniform( size=( i,j ) )
np.random.normal( size=( x,y ) )
np.random.randint( k,size=( z,a ) )
```

B. `plt.hist` to count and plot the number of times a number appeared

C. `shuffle` and `choice` commands

```
np.random.shuffle( «container» )
np.random.choice( «container» )
```

D. Applications: Monte Carlo Integration, random walk...