14. Math & Random Libraries

Introduction:

The math and random libraries include a bunch of useful routines. In this lecture, we'll go over some of these functions as well as writing our own routines to expand these libraries.

Good descriptions of these two libraries are available here:

- https://docs.python.org/3/library/math.html
- https://docs.python.org/3/library/random.html

Math Library

The content of the math library can be found using the script shell. This is a subset of what's available in Python 3.

```
>>> import math
>>> dir(math)
['__class__', '__name__', 'pow', '__dict__', 'acos', 'acosh', 'asin',
'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial',
'floor', 'fmod', 'frexp', 'gamma', 'inf', 'isclose', 'isfinite',
'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log2', 'modf',
'nan', 'pi', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau',
'trunc']
```

A brief description of these functions is as follows...

Constants: Several constants are defined in the math library:

```
pi      3.14159...
        the ratio of a circle's circumference to its radius
tau     6.283185...
        the ratio of a circle's circumference to its diameter
e       2.718281...
        solution to
nan     not-a-number.
        nan is not equal to anything other than non
inf     infinity
```

nan cab be used as a place holder. For example, in controls systems, the dynamics of a system can be written in state-space form as

$$sX = AX + BU$$

$$Y = CX + DU$$

This is a little cumbersome since you need to keep track of four matrices for any given system: {A, B, C, D}. You can store these as a single matrix using *nan* as space holders:

$$G = \begin{bmatrix} A & nan & B \\ nan & nan & nan \\ C & nan & D \end{bmatrix}$$

From this point onwards, you can just work with a single matrix, G, to describe a dynamic system.

Trig Functions:

```
sin, cos, tan, asin, acos, atan, atan2(x,y)
degrees(x) x = radians, result is degrees
radians(x) x = degrees, result is radians
```

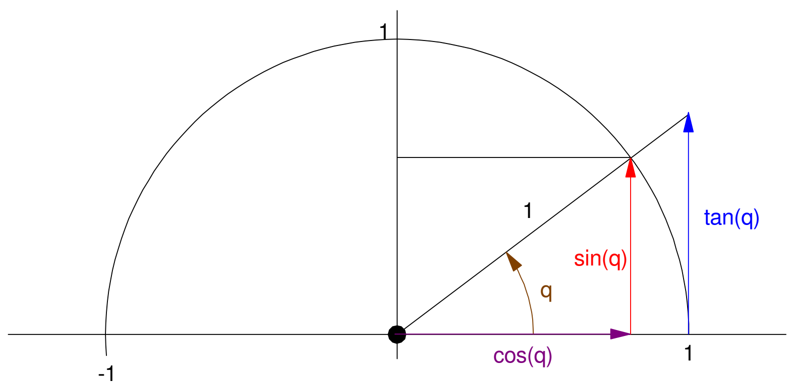
Trig functions are all about the unit circle

- cos(q) the x-coordinate of the vector $1\angle q$
- sin(q) the y-coordinate of the vector $1\angle q$
- tan(q) the y-coordinate of the tangent line with an angle of q to the origin

They can also be defined using the complex exponential:

$$\cos(x) = \left(\frac{e^{jx}+e^{-jx}}{2}\right)$$

$$\sin(x) = \left(\frac{e^{jx}-e^{-jx}}{2j}\right)$$



Trig functions

The execution time of trig functions can be found using the ticks_us() function. Measuring the time to do 1000 cosine functions results in

- 14339us to execute 1000 loops without the cos() function (comment out this line)
- 59410us to execute 1000 loops with the cos() function

The difference is 45,071us for executing 1000 cos() functions, or 45.071us per cos() function

cos() execution time = 45.072us

```
import math
import time

x0 = time.ticks_us()
for i in range(0,1000):
    x = i*0.01
    y = math.cos(x)
x1 = time.ticks_us()

print(x1-x0, 'us')
```

Shell

```
# removing the cos() function
14399 us

#including the cos() function
59410 us
```

Hyperbolic Functions:

```
sinh, cosh, tanh, asinh, acosh, atanh
```

Hyperbolic functions result from the sine or cosine of a complex number. They also result in nature quite often. For example,

- The shape of a soap film is a cosh() function
- The shape of a hanging chain is a cosh() function

This can be derived when you take a course on calculus of variations.

The shape of a hanging chain (or wire or transmission line) is a hyperbolic cosine function

Statistics Functions

```
factorial(x)       factorial(x) = 1 * 2 * 3 * ... * x
gamma(x)           factorial for non-integers

erf(x)             Error function for x
```

$$= \frac{2}{\sqrt{\pi}} \int_0^x e^{-x^2} dx$$

Note: The error function is used to compute the probability associated with a z-score in normal distributions (more on this later)

```
p = ( erf(z/sqrt(2) + 1) / 2
```

Exponential

```
exp(x)      e^x
expm1(x)    e^x - 1
2 ** x      2^x   (standard python syntax for raising to a power)

log(x)      log base e (natural log)
log2        log base 2
log10       log base 10
```

Rounding

```
ceil(2.3)          round up
    3
floor(2.3)         round down
    2
```

Other

```
sqrt(x)            square root

x ** 0.7           standard python syntax for raising to a power
```

Random Library

Functions in the random library can be found using the dir command. This is a subset of what's available on Python 3

```
>>> import random
>>> dir(random)
['__class__', '__init__', '__name__', '__dict__', 'choice',
'getrandbits', 'randint', 'random', 'randrange', 'seed', 'uniform']
```

These functions are:

```
randint(a,b)       returns in integer in the range of [a,b]

random()           returns a float in range of (0,1)

randrange(start, stop, step)  returns number from (start, stop) step size

randrange(stop)    returns a number from 0..stop

seed(a)            specify the starting seed for the random number generator
                   if a is not passed, uses system time

uniform(a,b)       returns a float in the range of (a,b)
```

General Stuff

Additional functions can be added by writing your own subroutines.

Convolution:

Convolution appears several places:

- The output of a signal going through a filter is the convolution of the input and the filter's impulse response
- Multiplication of polynomials is convolution
- Combining probability-density functions (pdf's) is convolution

Example: Multiply out the following polynomials

$$A = 3 + 2x + x^2$$

$$B = 5 + 2x^2$$

$$C = 6 - 3x + 4x^2 + 7x^3$$

$$Y = ABC$$

```
def conv(A, B):
    nA = len(A)
    nB = len(B)
    nC = nA + nB - 1
    C = []
    for n in range(0,nC):
        C.append(0)
        for k in range(0,nA):
            if( ( (n-k) >= 0) & ( (n-k) < nB ) & (k < nA) ):
                C[n] += A[k]*B[n-k]
    return(C)

A = [3,2,1]
B = [5,0,2]
C = [6,-3,4,7]
AB = conv(A,B)
Y = conv(AB,C)
print(Y)
```

Shell

```
[90 15 96 136 114 87 36 14]
```

The result is

$$Y = 90 + 15x + 96x^2 + 136x^3 + 114x^4 + 87x^5 + 36x^6 + 14x^7$$

Combinations & Permutations:

These count the number ways you can arrange m items selected from a population of n

- Where order does not matter (n choose m) and
- Where order does matter (n pick m)

$$nCm = \frac{n!}{m!\cdot(n-m)!}$$ n choose m. order does not matter

$$nPm = \frac{n!}{(n-m)!}$$ n pick m. order does matter

```
from math import factorial

# Comninatorics:  n choose m
def nCm(n,m):
    y = int( ( factorial(n) / factorial(m) ) / factorial(n-m) )
    return(y)

# permitations: N pick M
def nPm(n,m):
    y = int( factorial(n) / factorial(n-m) )
    return(y)
```

Example: How many distinct volleyball teams can you make with 20 people?

In this case, order doesn't matter.

N = 20 choose 6

$$N = \frac{20!}{6! \cdot 24!} = 38,760$$

How many volleyball teams can you make where each person is assigned a specific position? (1st player is setter, 2nd is outside hitter, etc.)

In this case, order does matter.

N = 20 pick 6

$$N = \frac{20!}{6!} = 27,907,200$$

Probability Density Functions (pdf) & Cumulative Distribution Functions (cdf)

The random library also has several discrete and continuous probability density functions (pdf's). Before we talk about these, let's first define what pdf's and a cdf's are.

- **pdf:** A pdf is the probability that a random variable is equal to x

$$y(a) = p(x = a)$$

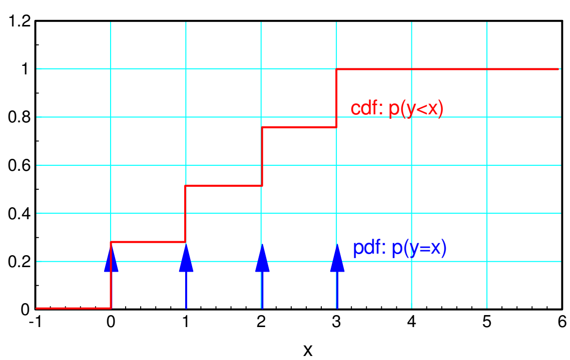
- **cdf:** A cdf is the probability that a random variable is less than x

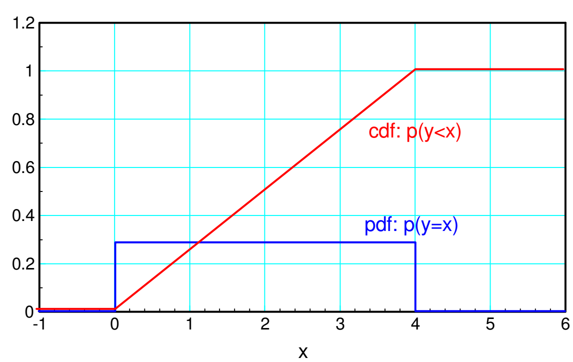
$$Y(a) = p(x < a)$$

Both of these can be discrete or continuous:

- **discrete:** x can only take on certain values, such as integers
- **continuous:** x can take on any value

For example, the pdf and cdf typically look something like this:





pdf and cdf for a discrete and a continuous probability function

The mean of a pdf is the average

$$\mu = \Sigma p_i \cdot x_i \qquad \textit{mean}$$

The variance is a measure of the spread (distance to the mean)

$$\sigma^2 = \Sigma p_i \cdot (x_i - \mu) \qquad \textit{variance}$$

$$\sigma = \sqrt{\sigma^2} \qquad \textit{standard deviation}$$

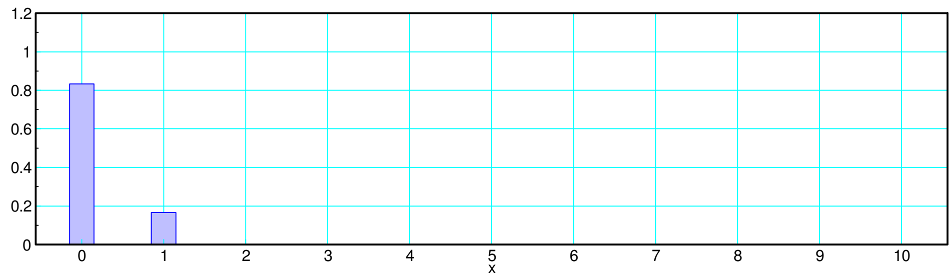
Discrete Random Distributions

Bernoulli Trial: A Bernoulli trial is a coin toss: the outcome is binary 1 or 0. Examples include

- A coin toss: p = 1/2
- Roll a die and see if you get a six: p = 1/6
- Roulette wheel betting on red: p = 15/31
- Roulette wheel betting on 10-black: p = 1/31
- A single game of tennis: probability of winning that game is p

The pdf for a Bernoulli trial only has two possible outcomes:

- 1: success
- 0: failure



pdf for a Bernoulli trial with p = 1/6

In Python:

```
from random import random

p = 0.7
for i in range(0,5):
    if (random() < p):
        Win = 1
    else
        Win = 0
    print(i,Win)
```

```
1     1
2     1
3     0
4     1
5     0
```

Binomial Distribution: Conduct n Bernoulli trials and count the number of successes.

- Flip a coin 10 times and count the number of heads
- Roll a six-sided die 10 times and count the number of ones

The pdf for a binomial distribution is

$$p(x = m) = \binom{n}{m} (p)^m (1-p)^{n-m}$$

where

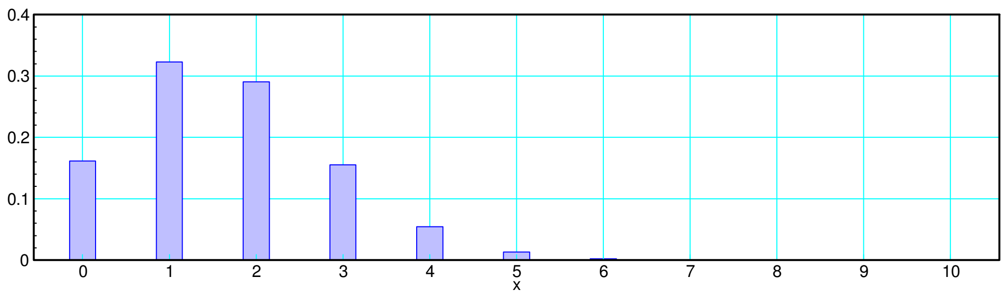
- n = number of Bernoulli trials
- m = number of successes, and
- p = the probability of a success.

For example, the probability of rolling ten six-sided dice and getting three ones is

$$p(m = 3) = \binom{10}{3} \left(\frac{1}{6}\right)^3 \left(\frac{5}{6}\right)^7$$

$$p(m = 3) = 0.1550$$

The pdf for m successes in ten trials is:



Probability of rolling x ones when rolling ten six-sided dice (binomial distribution)

In Python, you can create a function, binomial, which

- flips a coin n times
- with a probability of a success for any given flip being p:

For example, the following code flips a coin ten times with p=0.6. This experiment is repeated five times:

```
from random import random

def binomial(p, n):
   x = 0
   for i in range(0,n):
       if(random.random() < p):
           x += 1
   return(x)

p = 0.6
for i in range(0,5):
    y = binomial(p, 10)
    print(i, y)
```

```
0   6
1   5
2   4
3   6
4   5
```

Note that the results (second column) is different each time you run the experiment: this is a random process.

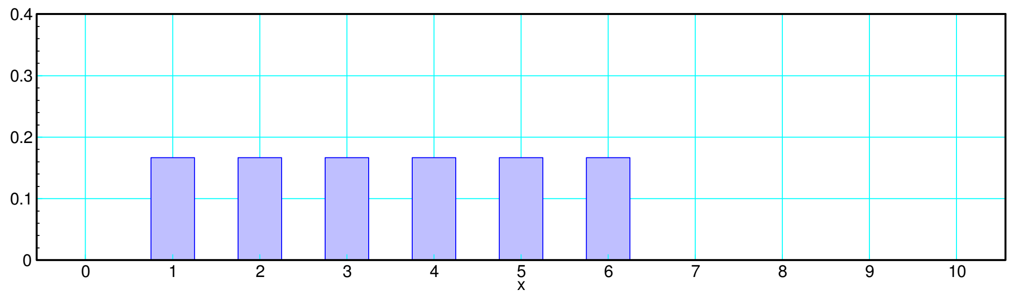
Uniform Distribution: All numbers have equal probability. There are several Python commands to do this.

```
y = randrange(6)  pick a number from 0..6

y = randrange(1,6,1)            pick a random number from 1..6

Die = [1,2,3,4,5,6]
y = choice(Die)                 pick a random value from Die
```

For example, the pdf for rolling a fair six-sided die should be:



pdf for a fair 6-sided die. All numbers have a probability of 1/6

This allows you to generate random numbers for various die rolls. For example the D&D spells do:

- Insect Swarm: four 10-sided dice (4d10)
- Ice Storm: two 8-sided dice plus four 6-sided dice (2d8 + 4d6)

```
from random import random

def Dice(n, sides):
   x = 0
   for i in range(0,n):
       x += randrange(1, sides, 1)
   return(x)

for i in range(0,5):
    InsectSwarm = Dice(4, 10)
    IceStorm = Dice(2, 8) + Dice(4, 6)
    print(i, InsectSwarm, IceStorm)
```

```
0   6
1   5
2   4
3   6
4   5
```

Exponential Distribution: An exponential distribution is where you keep running an Bernoulli trial until you get one success. Examples would be:

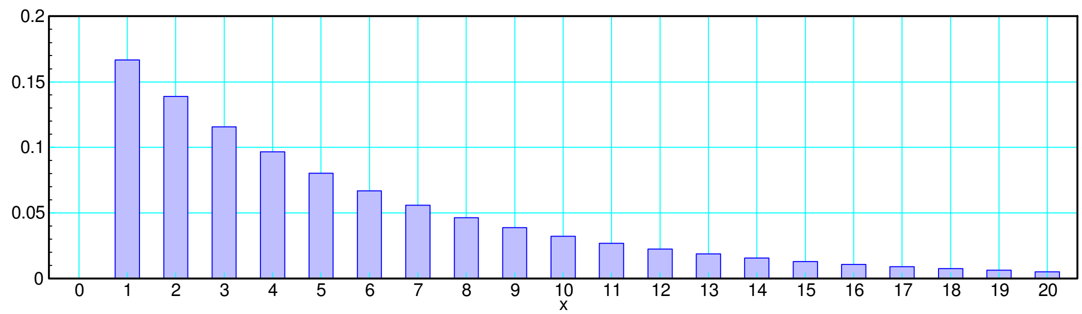
- The number of coin tosses until you get a heads (p = 1/2)
- The number of times you roll a 6-sided die until you roll a one (p = 1/6)
- The number of days you do the dishes until someone notices
- The number of days your car doesn't start

The pdf for an exponential distribution is:

$$p(n) = p(1-p)^{n-1}$$

(you have to fail n-1 times followed by one success).

For example, if p = 1/6, the pdf for an exponential distribution is as follows. Note that the pdf for an exponential distribution goes to infinity.



pdf for an exponential distribution with p = 1/6

In Python, there are two ways to create a trial with an exponential distribution.

Option #1: Run the experiment with a while-loop.

```
from random import random

def exponential(p):
    x = 0
    y = 1
    while(y > p):
        x += 1
        y = random()
    return(x)

p = 1/6

for i in range(0,5):
    y = exponential(p)
    print(i, y)
```

shell

```
0 18
1 14
2 4
3 2
4 5
```

This works, but when p is small, it may take a long time to execute. A more efficient method is to use the cdf for an exponential distribution:

$$Y(x) = ceil\left(\frac{-\ln(1-x)}{p}\right)$$

For example, if p = 1/6 and x = 0.8

```
y = ceil(9.656) = 10
```

By generating a random number for x in the range of (0,1), you can compute y with an exponential distribution without having to loop.

In Python:

```
from random import random
from math import ceil, log

def exponential(p):
    y = random()
    x = ceil( -log(1-y) / p )
    return(x)


p = 1/6

for i in range(0,5):
    y = exponential(p)
    print(i, y)
```

shell

```
0 1
1 11
2 12
3 1
4 3
```

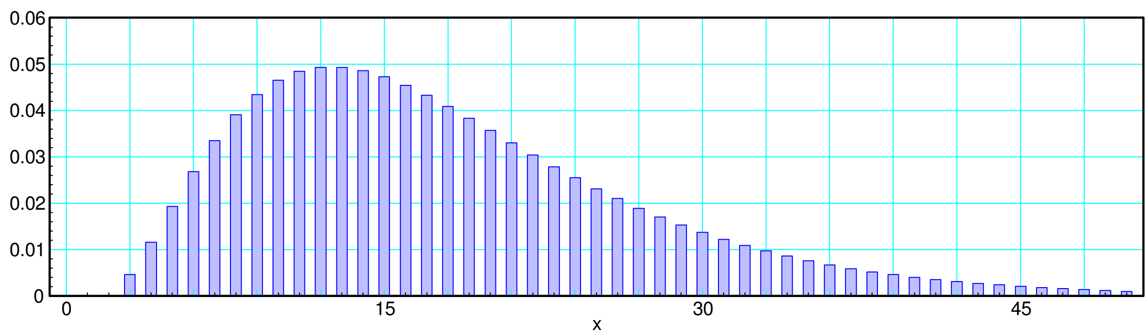
Pascal Distribution: An exponential distribution is where you keep running an Bernoulli trial until you get r successes. Examples would be:

- The number of coin tosses until you get three heads (p = 1/2)
- The number of times you roll a 6-sided die until you roll three ones (p = 1/6)
- The number of days you do the dishes until three people notice
- The number of days until your car doesn't start three times (and you trade it in)

The pdf for a Pascal distribution is

$$p(x) = \binom{x-1}{r-1} p^r (1-p)^{x-r}$$

For example, the pdf for it taking x die tosses to get three ones looks like:



pdf for a Pascal distribution: the number of rolls until you get three ones

In Python, repeat the exponential pdf r times

```
from random import random
from math import ceil, log

def exponential(p):
    y = random()
    x = ceil( -log(1-y) / p )
    return(x)


p = 1/6
r = 3

for i in range(0,5):
    y = 0
    for j in range(0,r):
        y += exponential(p)
    print(i, y)
```

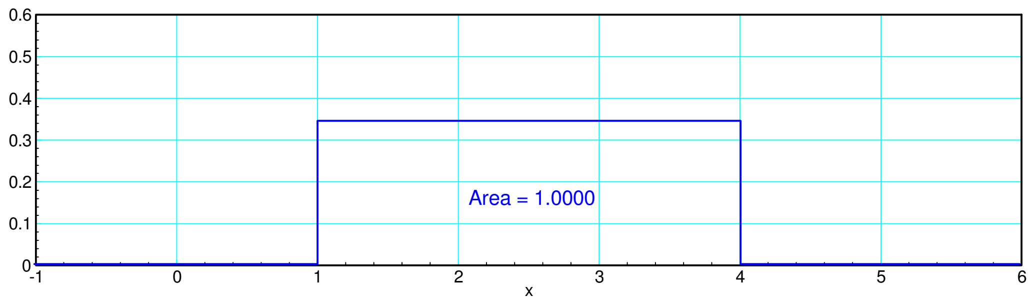
Continuous Random Distributions

You can also do continuous distributions with Python.

Uniform Distribution: A uniform distribution has equal probability over a range of (a,b). Examples would be:

- Modeling a resistor with 5% tolerance: its value is a uniform distribution over the range of 95% to 105% of rated value
- The time that you press a button, measured to 1us, mod ten

For example, a uniform distribution over the range of (1,4) looks like this: (note: the area must be one to be a valid pdf).



Uniform distribution over the range of (1,4)

This is a built-in function in Python

- A uniform distribution over the range of (0,1) is the function *random()*
- A uniform distribution over the range of (a,b) is the function *uniform()*

```
>>> random.random()
0.7870027

>>> random.uniform(5,6)
5.801835
```

Exponential Distribution: The time until an event happens, assuming the event has a fixed probability over any small time interval. Examples include

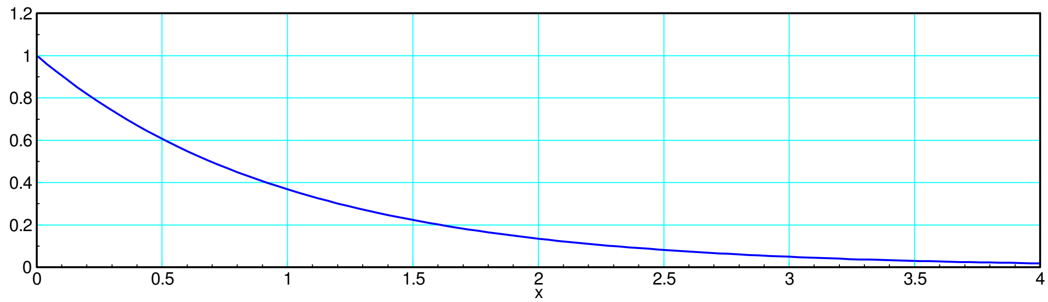
- The duration of a phone call
- The time until an atom decays
- The time until a customer arrives at a store
- The time it takes to serve a customer

The pdf for an exponential distribution is

$$f_X(x) = \begin{cases} ae^{-ax} & 0 < x < \infty \\ 0 & otherwise \end{cases}$$

The mean of an exponential distribution is 1/a.

For example, the pdf for an exponential distribution with a mean of 1 is:



pdf for an exponential distribution with mean = 1 (1/a = 1)

In Python, x can be computed by using a random number over the range of (0,1) and using the cdf to find x:

```
def exponential(p):
    y = random.random()
    x = - math.log(1-y) / p
    return(x)

p = 1/6

for i in range(0,5):
    y = exponential(p)
    print(i, y)
```

shell

```
0   10.1154
1   14.1735
2    0.8148
3   14.6771
4    6.0039
```

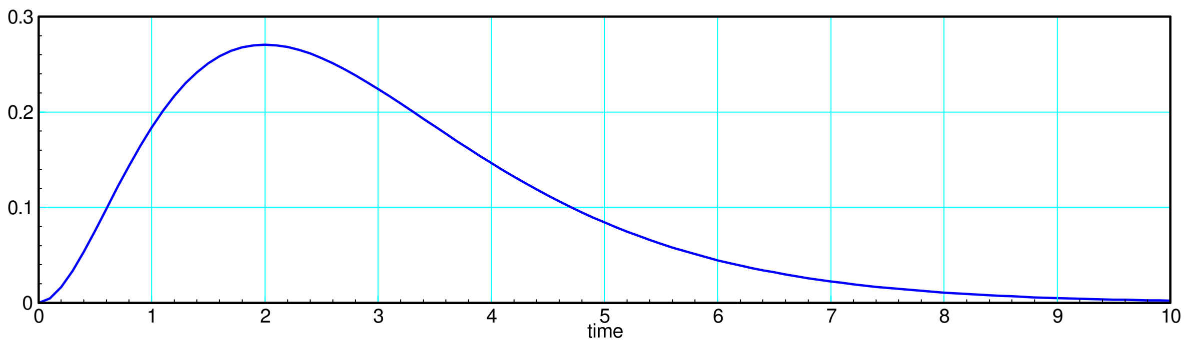
Gamma Distribution: The time until k events happen, assuming the event has a fixed probability over any small time interval. Examples include

- The duration of k phone calls
- The time until k atoms decays
- The time until k customers arrives at a store

The pdf for a gamma distribution is

$$f_X = \left(\frac{a^k}{(k-1)!}\right) x^{k-1} e^{-ax}$$

For example, if the mean time between events is one second (1/a = 1), the pdf for the time until three events happen is:



pdf for a Gamma distribution with k = 3 events and 1/a = 1 second

In Python, repeat an exponential distribution k times

```
def exponential(p):
    y = random.random()
    x = - math.log(1-y) / p
    return(x)

p = 1/6
k = 3

for i in range(0,5):
    y = 0
    for j in range(0,k):
        y += exponential(p)
    print(i, y)
```

shell

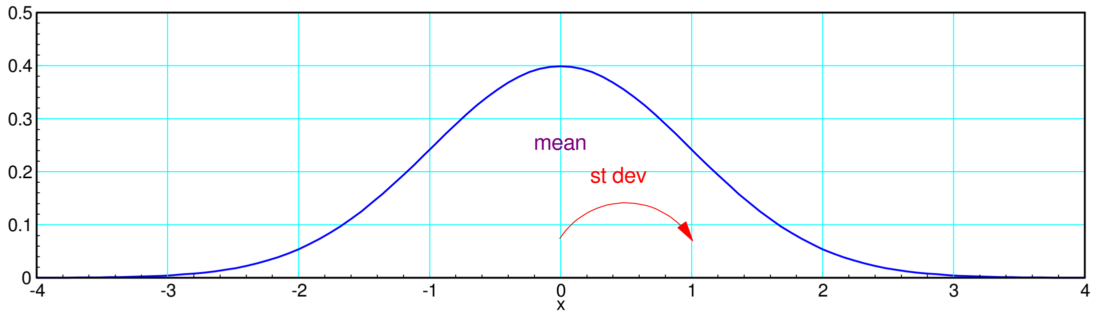
```
 0   19.6494
 1   14.8661
 2    2.4232
 3   18.1395
 4   20.4999
```

Normal Distribution: The normal distribution (also known as the Gaussian distribution) is the bell-shaped curve you're probably familiar with. The normal distribution is defined by two terms:

- μ: The mean or average
- σ: The standard deviation (a measure of the spread)
- σ^2: The variance

The pdf for a normal distribution is

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)$$



pdf for a standard normal distribution (mean = 0, standard deviation = 1)

The normal distribution is probably the most important distribution in all of statistics. The Central Limit Theorem states that, under fairly general assumptions, all distributions converge to a normal distribution. When you add a normal distribution to a normal distribution, you get a normal distribution.

The area of the tails for a normal distribution can be found in Python using the error function

```
tail(x) = ( math.erf(-x / sqrt(2)) + 1 ) / 2
```

This is useful when you want to the find the probability of an event which is x standard deviations away from the mean.

Python does not have a rand function which outputs a normal distribution. This can be approximated by adding twelve uniform distributions.

- A uniform distribution over the range of (0,1) has a mean of 0.5 and a variance of 1/12
- The sum of twelve uniform distributions has a mean of 12*0.5 (6) and a variance of 12 * 1/12 (1)
- Subtract six and the distribution looks like a standard normal pdf with mean of 0 and variance of 1

```
def randn():
    x = -6
    for i in range(0,12):
       x += random.random()
    return(x)

for i in range(0,5):
    y = randn()
    print(i, y)
```

shell

```
0   -0.3209
1   -1.0091
2    0.0923
3   -0.0394
4    0.9420
```

Summary

In Python, you can create a wide variety of random distributions. Most of these use the random() function from the random library.