Math & Random Libraries

ECE 476 Advanced Embedded Systems

Jake Glower - Lecture #14

Please visit Bison Academy for corresponding
lecture notes, homework sets, and solutions

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

Note:

- MicroPython uses a subset of the libraries used in Python
- The Raspberry Pi Pico doesn't have as much memory as a Raspberry Pi

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...
     Natural constant
     exp(x) is the only function equal to its derivative
     Also shows up in interest calculations, calculus, etc.

nan  not-a-number.
     nan is not equal to anything other than non

inf  infinity
```

Note on NaN

nan can 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:

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

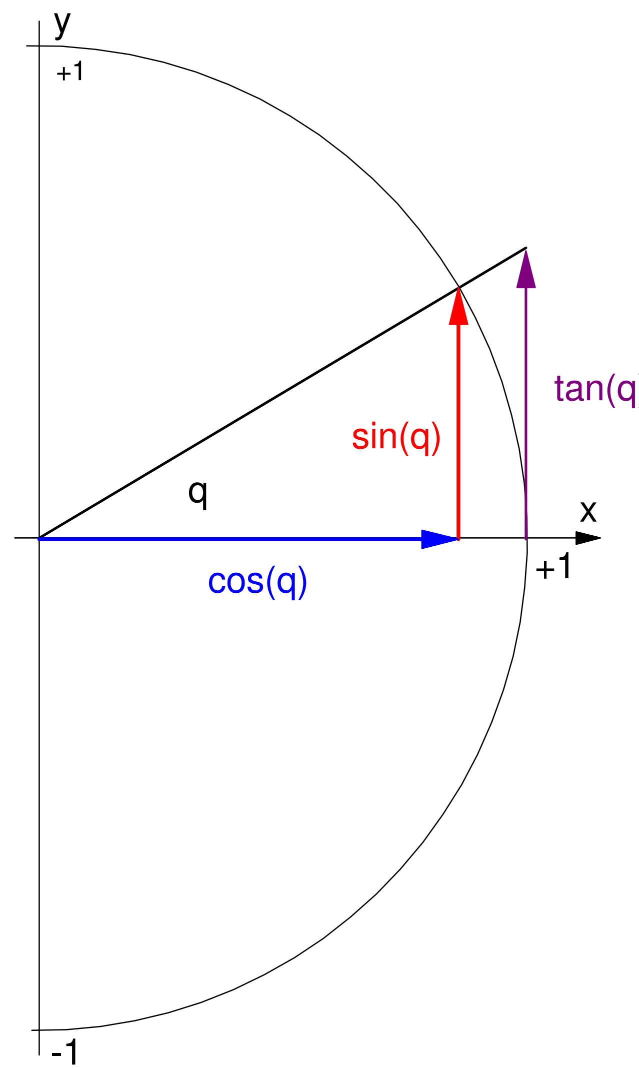
From this point onwards, you can just work with a single matrix, G, to describe a dynamic system.

Trig Functions:

```
sin, cos, tan,
asin, acos, atan, atan2(x,y)
y = degrees(x)     y = x*180/pi
y = radians(x)     y = x*pi/180
```

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 Execution Time

- Can be found using ticks_us()

Write a test program

- 14,339us to loop 1000 times
- 59,410us adding a cos() funciton

Take the difference

- 45,071us for 1000 cos() functions
- 45.071us per cos() function

execution time = 45.072us

- cos()

```
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.

$$\cos(jx) = \cosh(x)$$

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.

Statistics Functions

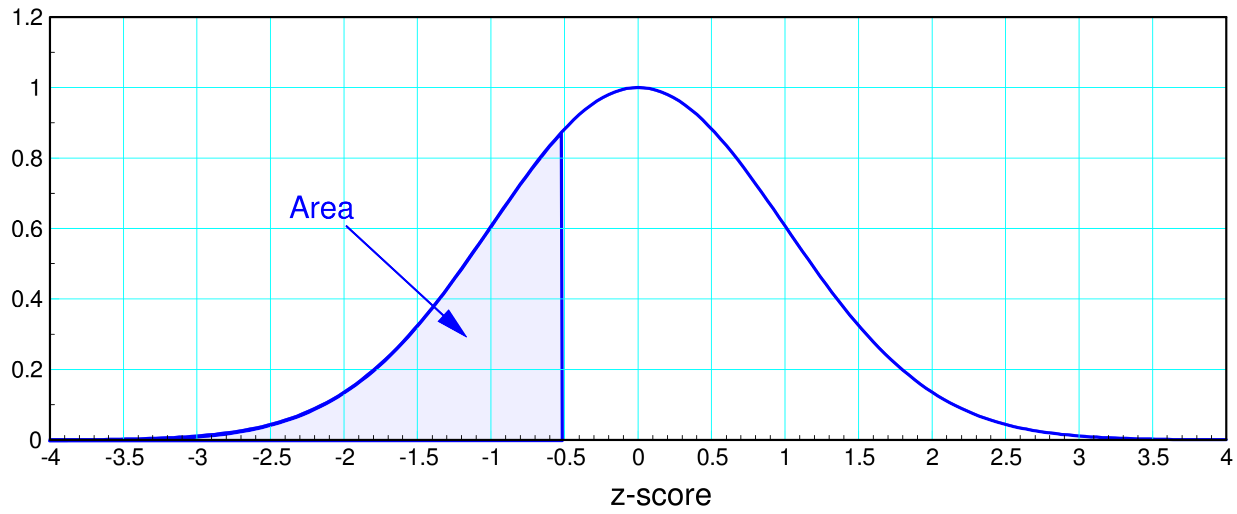
```
factorial(x)          factorial(x) = 1 * 2 * 3 * ... * x
gamma(x)              factorial for non-integers

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)

```
Area = ( 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 a random number with step size

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

seed(a)                specify the starting seed for random numbers
                       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

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

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

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

$$Y = ABC$$

Convolution Example:

```
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:

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

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

pdf's & cdf's

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

The random library has several probability functions

These are described by pdf's and cdf's

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

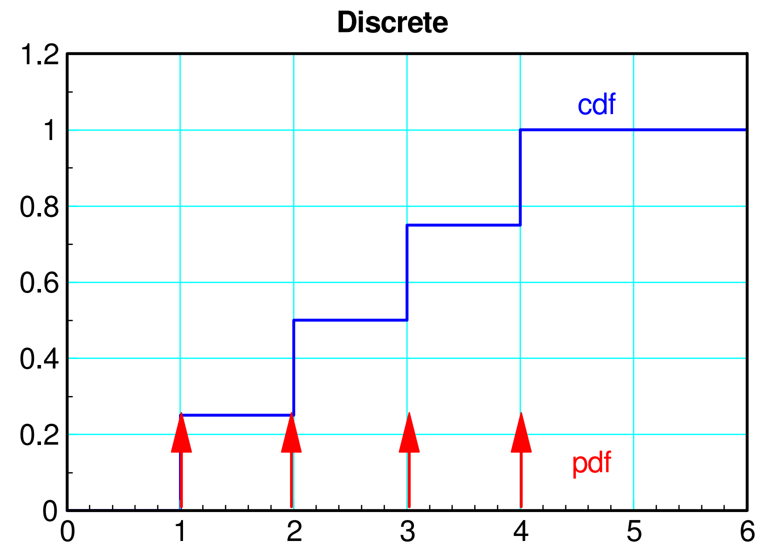
$$y(a) = p(x = a)$$

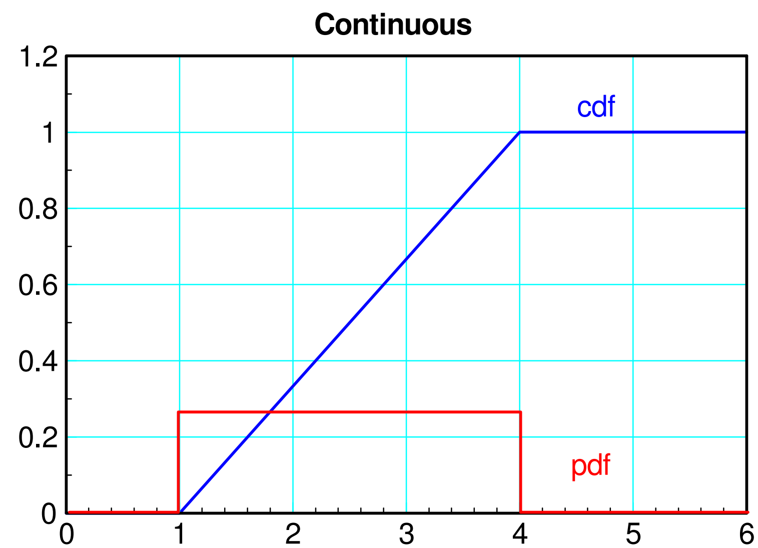
- **cdf:** A cdf is the probability that a random variable is less than x

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

Both can be discrete or continuous:

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





Mean, Standard Deviation, and Variance

The mean of a pdf is

- The average or
- The center of mass

$$\mu = \sum p_i \cdot x_i$$

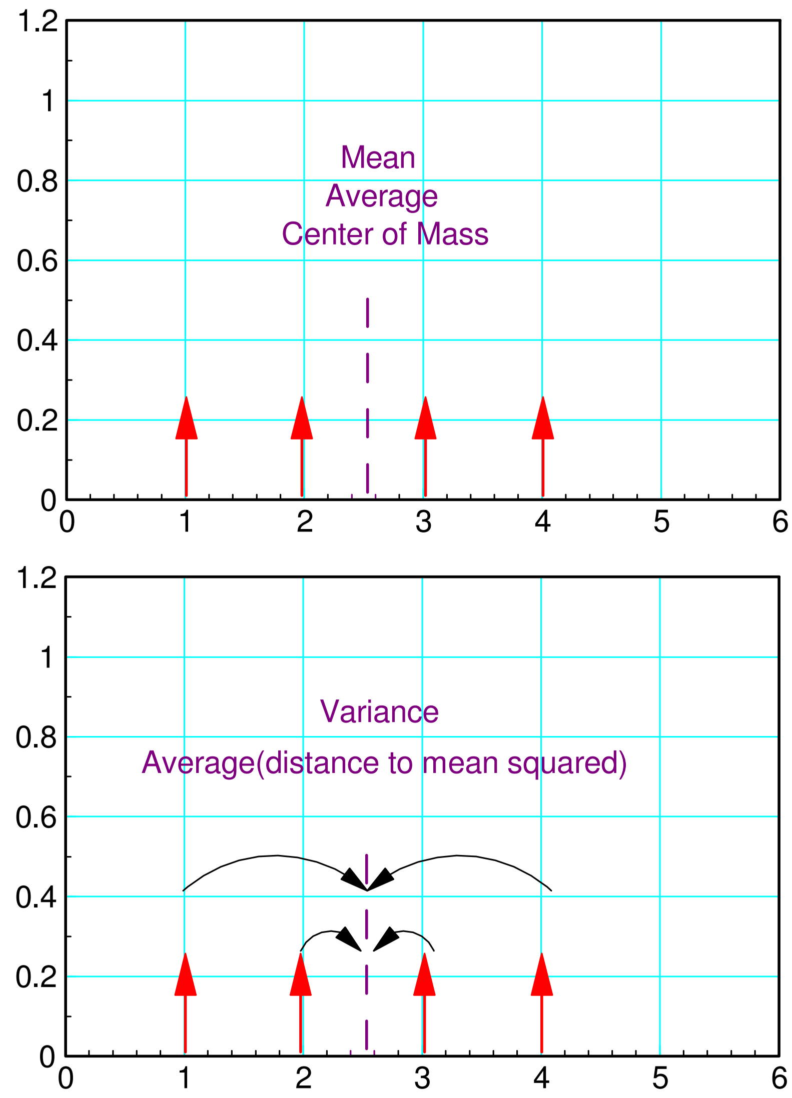
The variance of a pdf is

- A measure of the spread
- The avg distance to the mean squared

$$\sigma^2 = \sum p_i \cdot (x_i - \mu)^2$$

The standard deviation is

$$\sigma = \sqrt{\sigma^2}$$



Discrete Random Distributions

Bernoulli Trial: Toss a coin toss

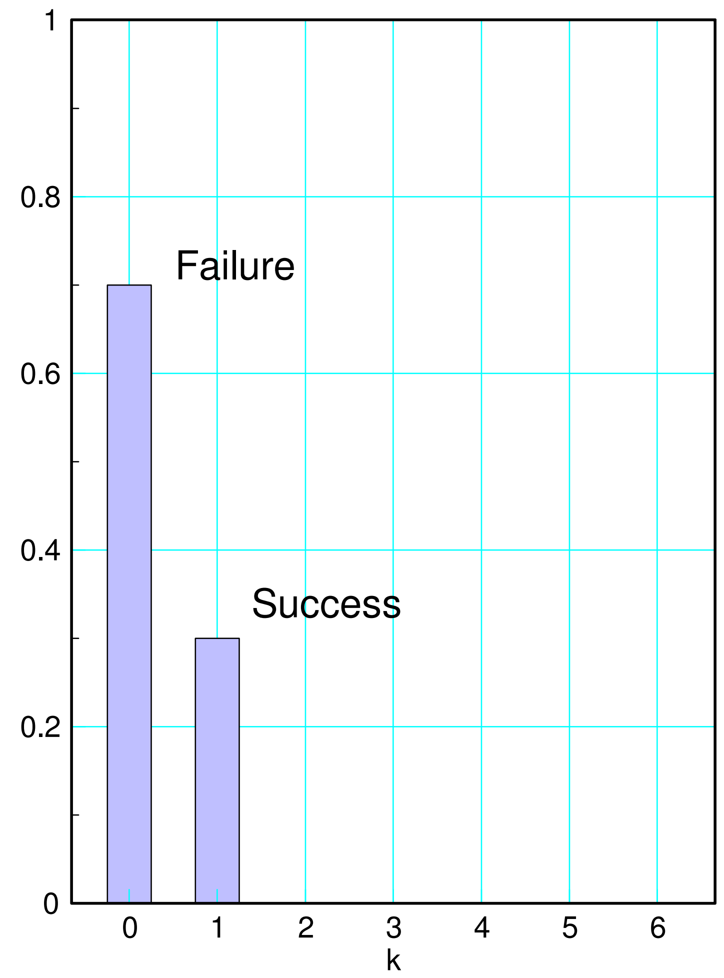
- The outcome is binary 1 or 0.

Examples

- A coin toss: p = 1/2
- Roll 6-sided die to get a one: p = 1/6
- Bet on red on Roulette: p = 15/31
- Bet on 10-black in Roulette: p = 1/31

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

- 1: success
- 0: failure



Bernoulli Trials in Python

Use the random function in the random library

Example:

- Flip a coin
- p(success) = 0.7

```
from random import random

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

shell

```
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.

Binomial Example

Roll ten 6-sided dice

- N = 10

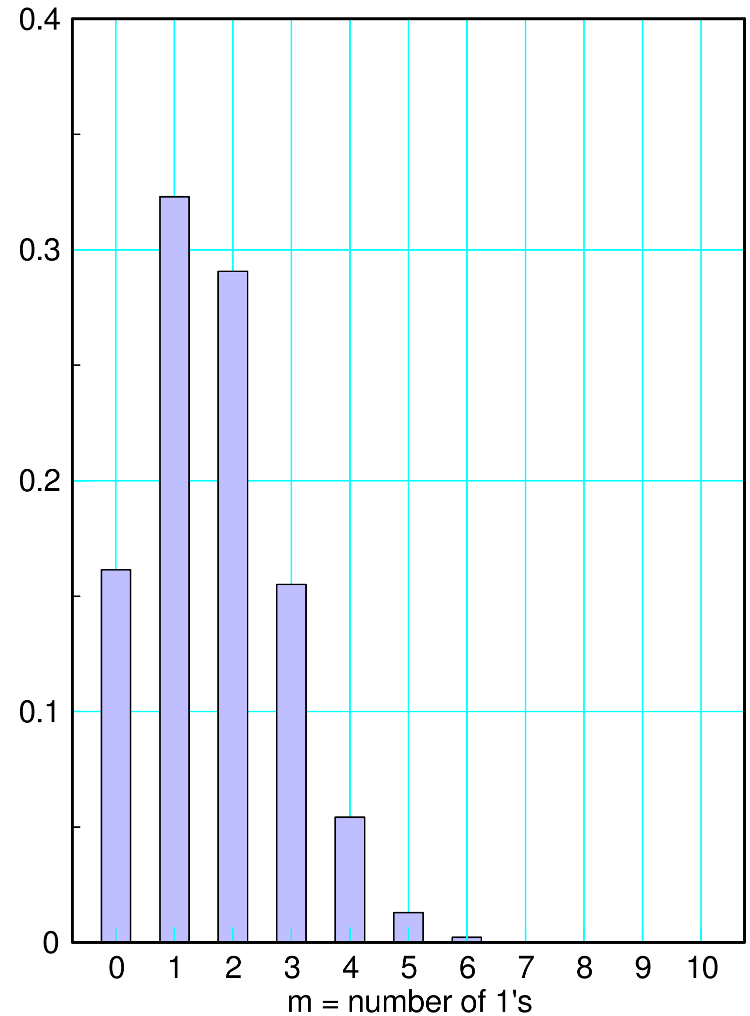
Count the number of ones

- p = 1/6

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

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

The pdf for m successes in ten trials is:



Binomial pdf in Python

Create a function which

- flips a coin n times

with p = probability of success

Example:

- p = 0.6
- Flip ten coins
- Count the number of successes

In the shell window:

- 1st column = trial number
- 2nd column = number of ones

```
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)
```

shell

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

Uniform Distribution:

- All numbers have equal probability.

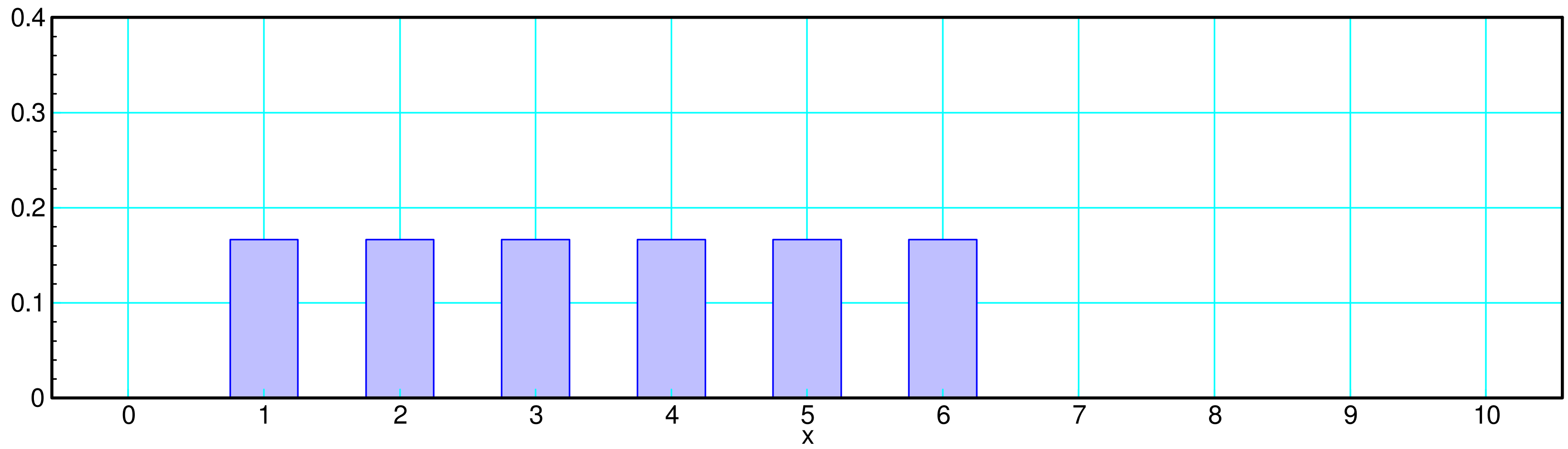
There are several Python commands to do this.

```
y = randrange(6)+1              pick a number from 1..6

y = randrange(1,7,1)            pick a random number from 1..6

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

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



Uniform Distribution Example

Add up multiple dice to cast D&D spells:

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

Dice(n, m)

- roll n dice
- with m sides
- take the sum

```
from random import randrange

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)
```

Insect Swarm:

- four 10-sided dice (4d10)

Ice Storm:

- 2d8 + 4d6

```
i   InsectSwarm  IceStorm
0       31          22
1       17          17
2       32          25
3       26          23
4       33          18
```

Exponential Distribution:

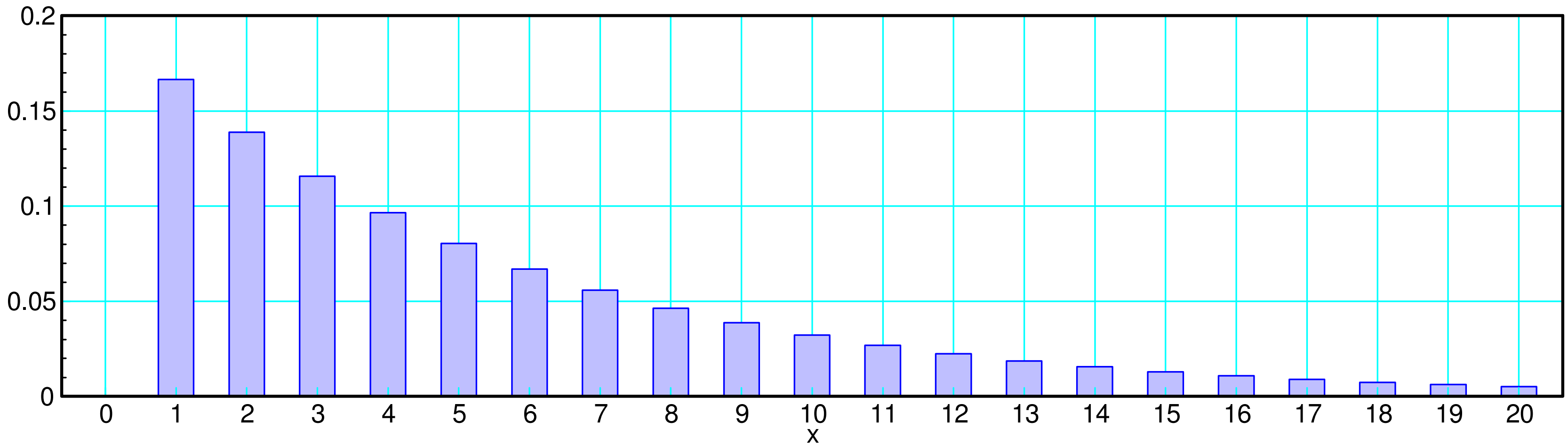
Running an Bernoulli trial until you get one success.

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 until your car doesn't start

The pdf for an exponential distribution is:

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



Exponential pdf in Python (take 1)

Use a while-loop

- Run an experiment until you get a success
- Matches the definition of an exponential distribution
- Can take a long time when p is small

Shell Window:

- col 1: trial number
- col 2: number of rolls until you get a 1 on a 6-sided die (p = 1/6)

```
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
```

Exponential pdf in Python (take 2)

Use the CDF

- Pick a random number from 0 to 1 (y)
- Use the CDF to compute x
  - cdf = integral of pdf

Gives the same result

Takes fewer clocks

- Not as obvious what's going on

```
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:

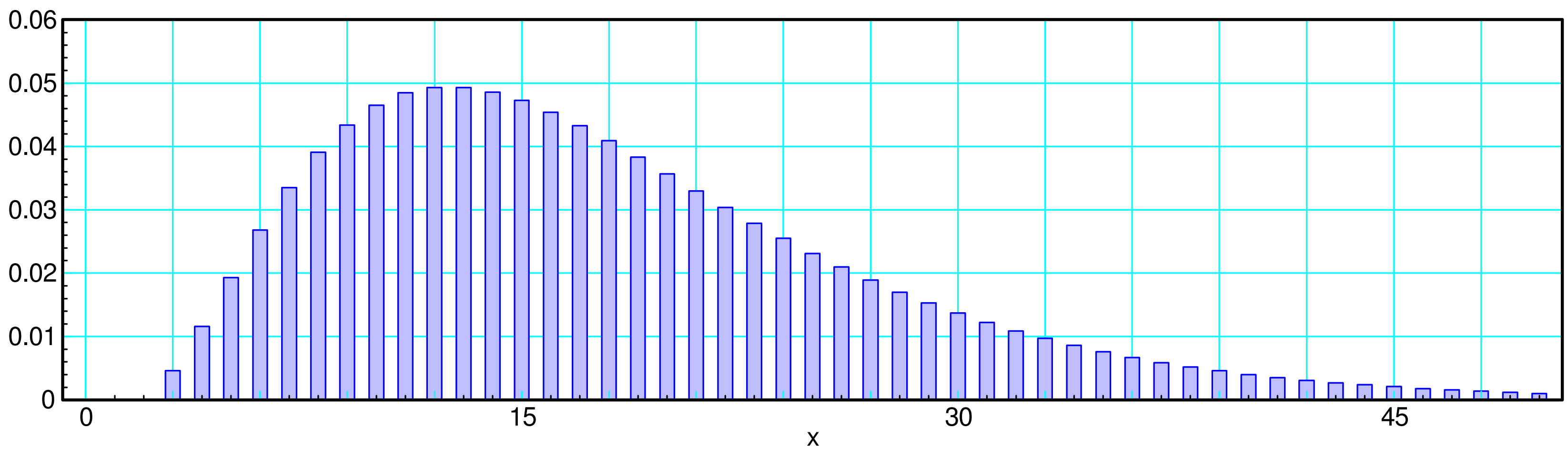
- Time until r successes for an exponential distribution

Examples:

- 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

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



In Python, repeat the exponential pdf r times

Example: Roll a 6-sided die until you get three 1's

- p = 1/6
- r = 3

In the shell window

- Column 1 = trial number
- Column 2 = number of rolls

```
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)
```

shell

```
0    12
1    20
2    9
3    17
4    23
```

Continuous Random Distributions

You can also do continuous distributions with Python.

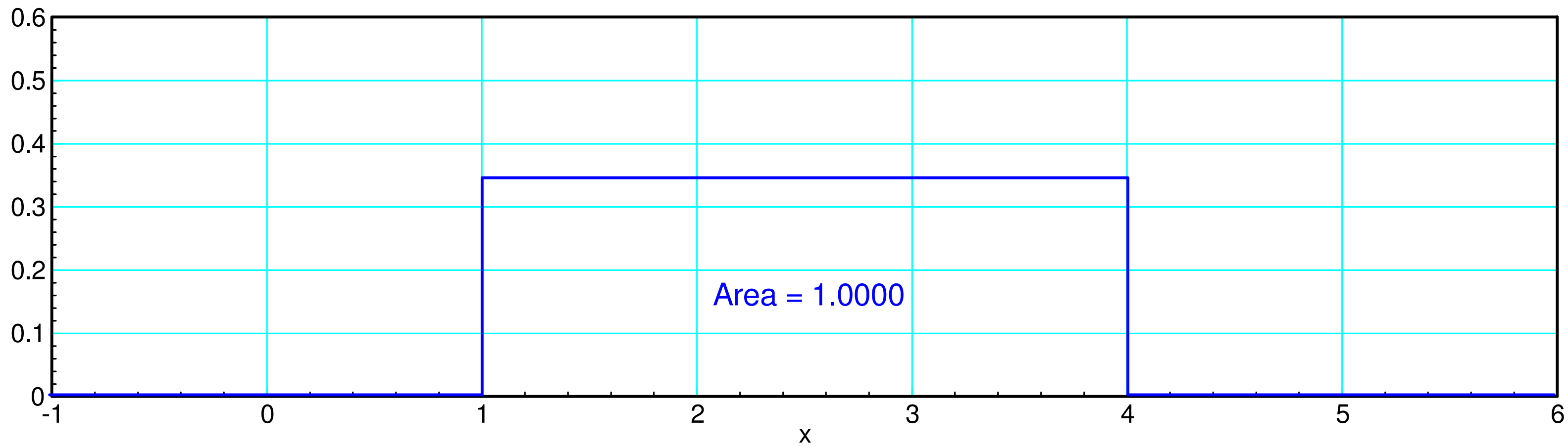
Uniform Distribution: Equal probability over a range of (a,b).

Examples:

- Modeling a resistor with 5% tolerance:
  - R can take any value from 95% to 105% of rated value
  - Equal probability over this range
- The time that you press a button, measured to 1us, mod ten

Example: uniform distribution over the range of (1,4)

- note: the area must be one to be a valid pdf



Uniform Distribution in Python

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
- Assumes the event has a fixed probability over any small time interval

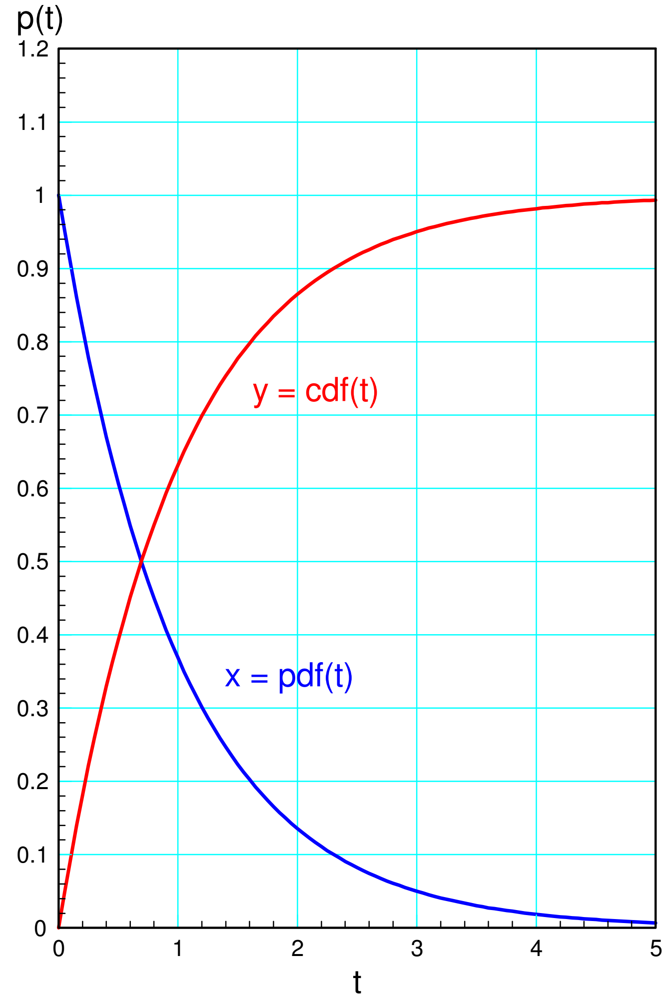
Examples:

- 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 and cdf are:

$$pdf(t) = \begin{cases} ae^{-at} & 0 < t < \infty \\ 0 & otherwise \end{cases}$$

$$cdf(t) = \begin{cases} 1 - e^{-at} & 0 < t < \infty \\ 0 & otherwise \end{cases}$$



Exponential in Python

The cdf lets you compute t with an exponential distribution

- pick y in the range of (0,1)
- compute t using the cdf

Shell Window

- col 1: trial number
- col 2: t with an exponential pdf
- mean = 6 seconds
  - p = 1/6

```
from random import random
from math import log

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

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
- Assumes the event has a fixed probability over any small time interval

Examples include

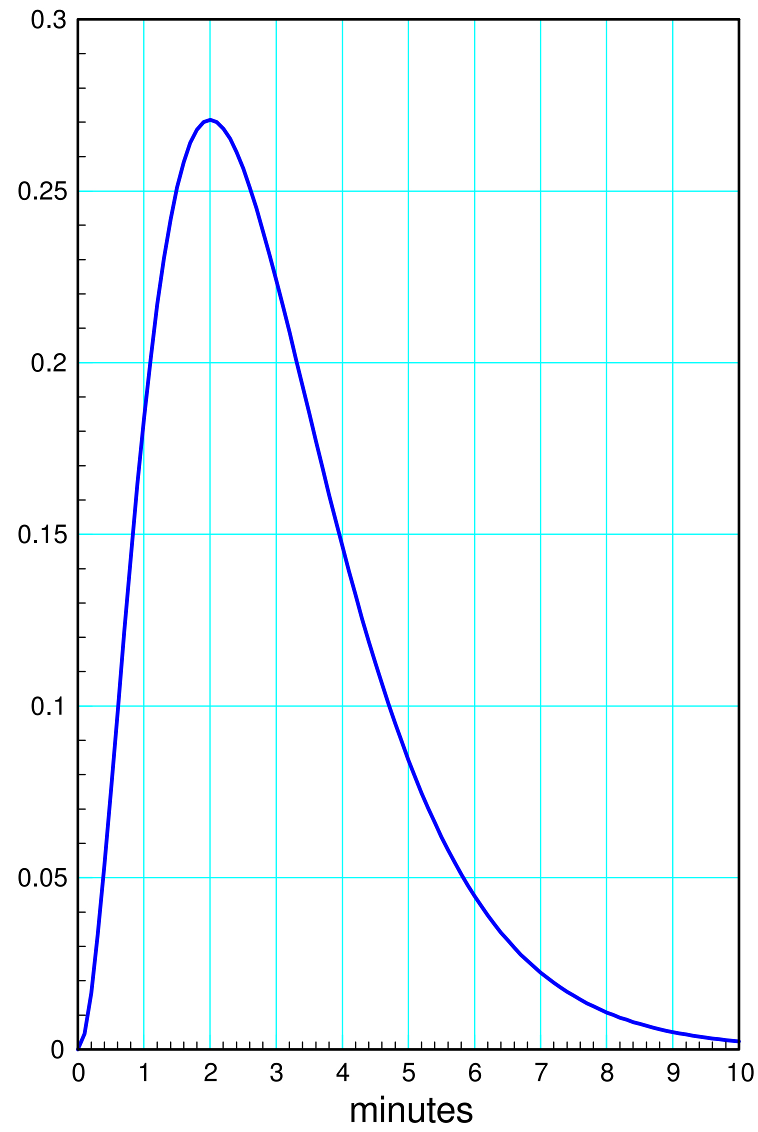
- The duration of k phone calls
- The time until k atoms decays
- The time until k customers arrive

The pdf for a gamma distribution is

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

Example:

- Time of three phone calls (k=3)
- Each phone call has a mean of one minute



Gamma distribution in Python

Find the time of an exponential distribution k times

Example:

- mean = 6 seconds
  - p = 1/6
- time for three events
  - k = 3

Shell Window

- col 1: trial number
- con 2: time until three events

```
from random import random
from math import log

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

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:

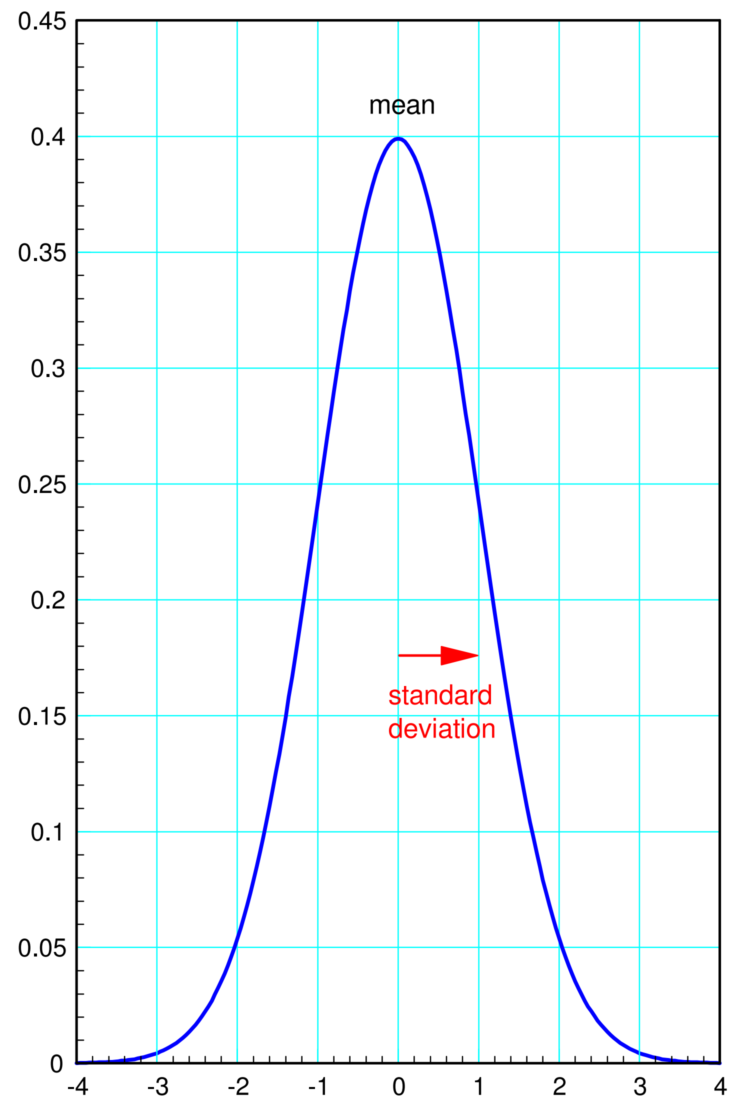
- a.k.a.Gaussian distribution
- Bell-shaped curve you're probably familiar with.

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

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



Normal Distribution (cont'd)

- Probably the most important distribution in all of statistics

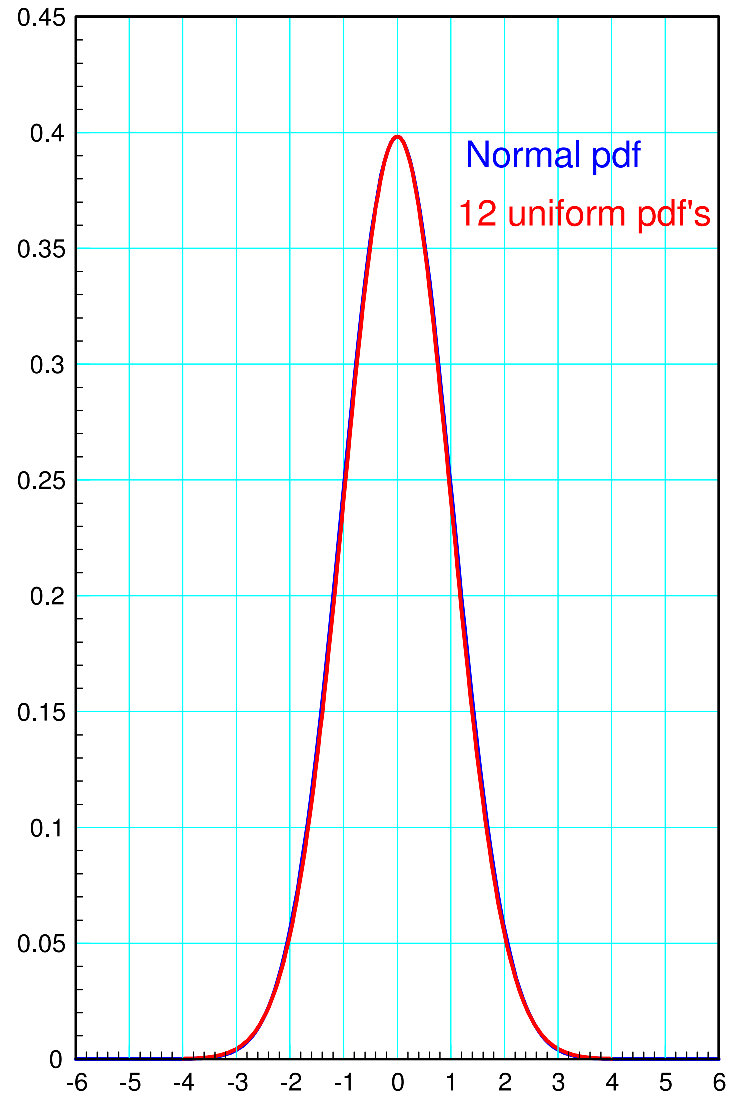
Central Limit Theorem:

- Under fairly general assumptions
- All distributions converge to a normal distribution
- Normal + Normal = Normal

Example:

- Add twelve uniform distributions
- Subtract six

The pdf looks very much like a normal distribution



Normal Distribution: Area of tail

- z-Score

The area of a tail tells you the probability that y < x

- Find the z-score
  - distance to the mean in terms of standard deviations
- Find the area of the tail
  - uses the error function
  - standard function in Python

```
from math include erf

x = 10
s = 5

z = (10 - 3) / s

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

print('z score = ',z)
print('area of tail = ',p)
```

Example:

- mean = 10
- standard deviation = 5
- p(y < 3) = ?

shell

```
z score = 1.400
area of tail = 0.0808
```

Normal Distribution in Python

Python does not have a randn function like Matlab

randn can be approximated

- add twelve uniform distributions (variance = 1)
- subtract six (mean = 0)

```
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
```

Flickering Candle

Write a program which causes an LED to flicker

- looks like a flickering candle

Use

- PWM to set the brightness
- A normal pdf to vary the PWM
- Change the PWM every 50ms

```
# Candle Flicker
# Create a flickering LED
# connected to pin 16

from machine import Pin, PWM
from time import sleep_ms
from random import random

Candle = Pin(16, Pin.OUT)
Candle = PWM(Pin(16))
Candle.freq(1000)

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

while(1):
    x = 32000 + randn()*10000
    Candle.duty_u16(int(x))
    sleep_ms(50)
```

Summary

The math library allows you to use

- Trig functions
- Exponentials
- Constants (pi, tau, e)

The random library allows you to

- Generate random numbers
- Generate variables with a uniform distribution

With come coding, you can generate other distributions

- Exponential
- Gamma,
- Poisson,
- Normal
- Other