

## CDF/PDF for density

- $f(a)$  is a PDF:

$$Pr(a \leq X \leq b) = \int_a^b f(x)dx$$

- $F(a)$  is a CDF:

$$F(a) = Pr(X \leq a) = \int_{-\infty}^a f(x)dx$$

- $$f(a) = \left. \frac{d}{dx} F(x) \right|_{x=a}$$

## CDF for empirical distribution

We have a sample  $x_1, \dots, x_n$

$$\hat{F}(a) = \frac{1}{n} |\{i \text{ such that } x_i \leq a\}|$$

- What about the PDF for empirical distribution? We cannot take the derivative... The closest we have to an empirical pdf is the histogram.

## Calculating the probability of a segment

- The true probability

$$P(a \leq X \leq b) = \int_a^b f(x)dx = \int_{-\infty}^b f(x)dx - \int_{-\infty}^a f(x)dx =$$

- The empirical probability. We have a sample  $x_1, \dots, x_n$

$$\hat{P}(a \leq X \leq b) = \frac{1}{n} |\{i \text{ such that } a \leq x_i \leq b\}| =$$

## Empirical CDFs vs. histograms

- The histogram converges to the density function
- The empirical CDF converges to the true CDF
- The convergence of the CDF is much faster.

```
In [1]: 1 %pylab inline
        2 import random
```

Populating the interactive namespace from numpy and matplotlib

```

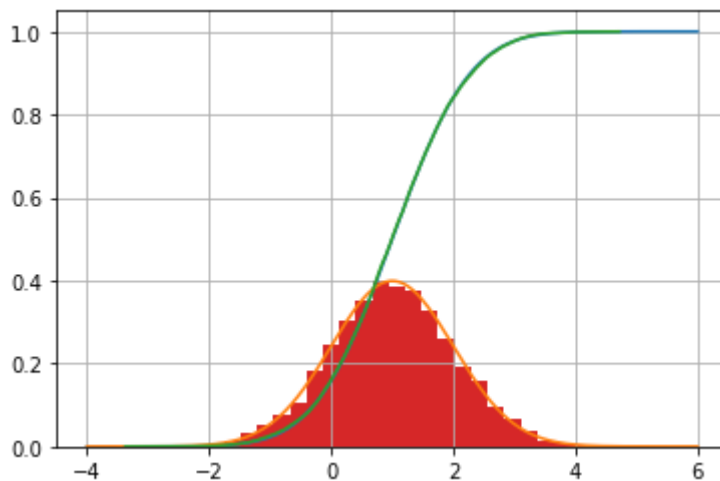
In [2]: 1 from scipy.stats import norm
2 def plot_normal(mu=1,sigma=1, n=20, m=100, plot_pdf_model=True,plot_cdf
3         plot_pdf_empir=True,plot_cdf_empir=True):
4     s = np.random.normal(mu, sigma, n)
5
6     xmin=-4; xmax=6; delta=1/n
7     x=arange(xmin,xmax,delta)
8     if plot_cdf_model:
9         _cdf=norm.cdf(x,loc=mu,scale=sigma)
10        plot(x,_cdf)
11    if plot_pdf_model:
12        _pdf=norm.pdf(x,loc=mu,scale=sigma)
13        plot(x,_pdf)
14    grid()
15    if plot_cdf_empir:
16        q=sorted(s)
17        P=arange(0,1,1/s.shape[0])
18        plot(q,P)
19    if plot_pdf_empir:
20        plt.hist(s, 30, density=True);
21
22    return

```

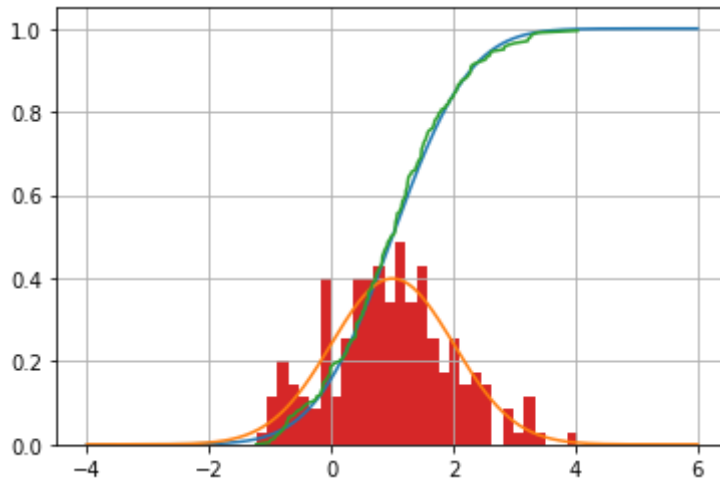
```

In [3]: 1 plot_normal(n=10000)

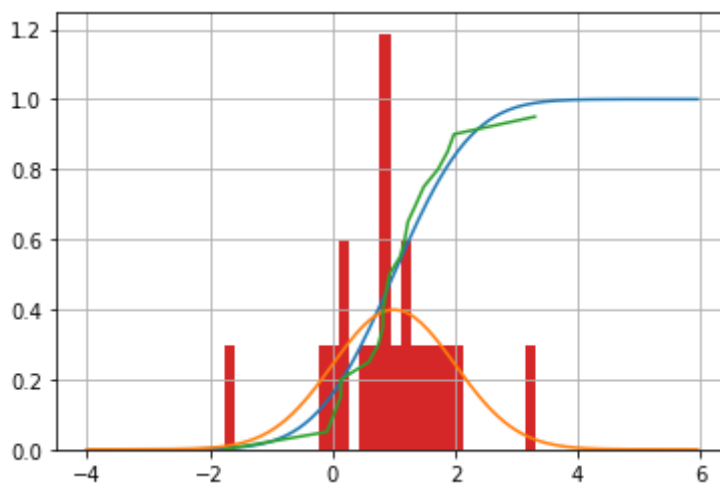
```



```
In [4]: 1 plot_normal(n=200)
```



```
In [5]: 1 plot_normal(n=20)
```



## Sorting data that comes from a distribution

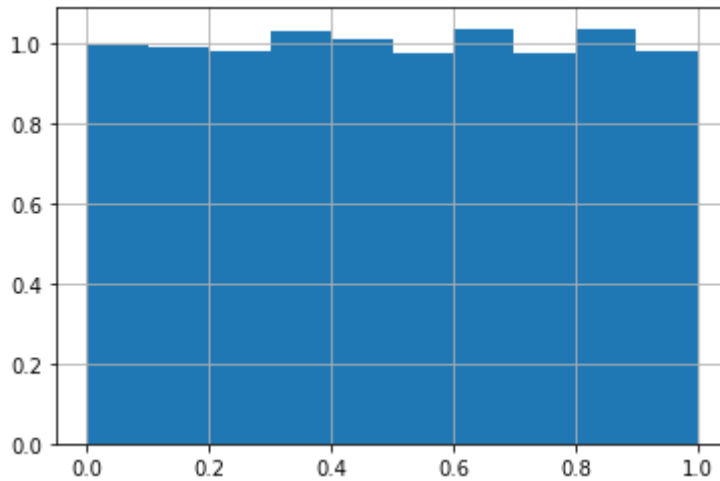
The best sorting time for arbitrary data:  $O(n \log n)$  (quicksort)

For data that is sampled from a fixed distribution (Independent Identically Distributed or IID) we can sort in  $O(n)$  time!

### Easiest case: Uniform distribution.

```
In [6]: 1 n=10000
        2 R=array([random.uniform(0,1) for i in range(n)])
```

```
In [7]: 1 hist(R,density=True);
        2 grid()
```



### Create a list of lists

- Size of (outside) list is  $n$ .
- Each inside list starts empty.

```
In [8]: 1 m=n #int(n/10)
        2 L=[] for i in range(m)
        3
        4 for r in R:
        5     i=int(r*m) # scale random number from [0,1] to [0,n]
        6     L[i].append(r) #append number to list at location i
```

```
In [9]: 1 L[:2]
```

```
Out[9]: [[], [0.0001494277885513151]]
```

### Size of fullest bin

```
In [10]: 1 lengths=[len(l) for l in L]
          2 min(lengths),max(lengths)
```

```
Out[10]: (0, 8)
```

## sort each short list and concatenate

```
In [11]: 1 _sorted=[]
          2 for l in L:
          3     _sorted +=sorted(l)
```

```
In [12]: 1 _sorted[:10]
```

```
Out[12]: [0.0001494277885513151,
          0.0007499316913494036,
          0.0014788538473297086,
          0.0016874161325813875,
          0.0018594589400154904,
          0.0019749541714617624,
          0.0020417289951969453,
          0.002168231059543224,
          0.0021831743262641368,
          0.0022116091240896862]
```

```
In [13]: 1 resort=sorted(_sorted)  # checking that the order is good.
          2 resort==_sorted
```

```
Out[13]: True
```

## What about distributions other than uniform

We can use the CDF of a distribution to transform it into a uniform distribution.

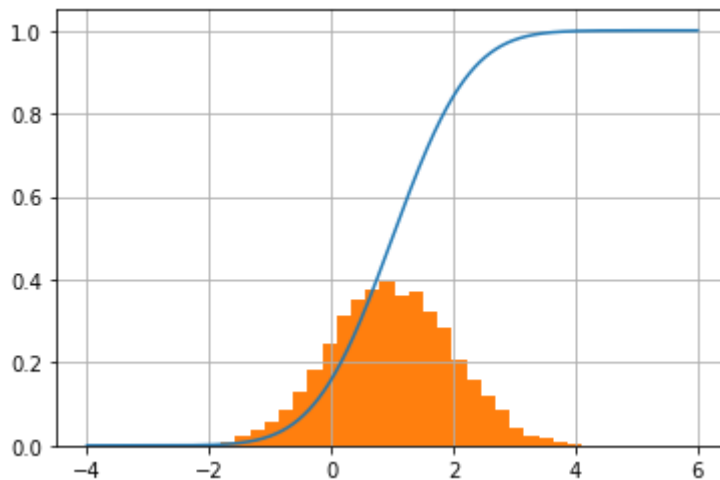
```
In [14]: 1 from scipy.stats import norm
```

## Transforming the distribution to a uniform distribution

We use  $F$  to denote the CDF

- $F(x) = P(X \leq x)$
- 
- $X$  is a random variable, therefor  $F(X)$  is a random variable.
- 
- What is the distribution of the RV  $F(X)$  ?
- $0 \leq F(X) \leq 1$ , Therefor  $P(0 \leq F(X) \leq 1) = ?$
- What is the probability that  $0 \leq A \leq F(X) \leq B \leq 1$

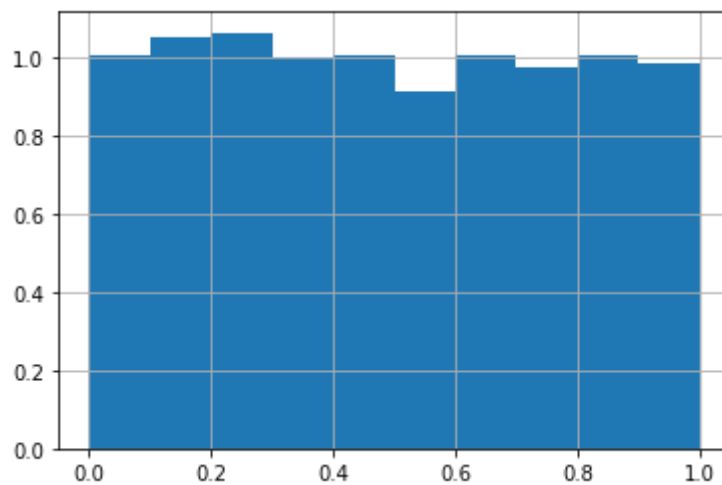
```
In [15]: 1 #figure(figsize=[15,10])
2 mu=1; sigma=1; n=10000; m=100
3 s = np.random.normal(mu, sigma, n)
4
5 xmin=-4; xmax=6; delta=1/n
6 x=arange(xmin,xmax,delta)
7 cdf=norm.cdf(x,loc=mu,scale=sigma)
8 plot(x,cdf)
9 grid()
10 plt.hist(s, 30, density=True);
```



- $F(x)$  is a non-decreasing function from  $(-\infty, +\infty)$  to  $[0, 1]$ . Therefore for any  $0 \leq A \leq B \leq 1$  there exists  $a \leq b$  such that  $F(a) = A, F(b) = B$ .
- $P(a < X \leq b) = P(A \leq F(X) \leq B)$
- On the other hand  $P(a \leq X \leq b) = F(b) - F(a) =$
- Therefore, for any  $0 \leq A \leq B \leq 1$ :  

$$P(A \leq F(X) \leq B) = B - A$$
- Which implies that the distribution of  $F(X)$  is ?

```
In [16]: 1
          2 index=np.array(x.shape[0]*(s-xmin)/(xmax-xmin),dtype=np.int)
          3 scaled=cdf[index]
          4 hist(scaled,density=True);
          5 grid()
```



**We can now use the sorting method for the uniform distribution**

```
In [17]: 1 m=100
          2 L=[] for i in range(m)
          3
          4 for j in range(s.shape[0]):
          5     r=s[j]
          6     _scale=scaled[j]
          7     i=int(_scale*m)
          8     L[i].append(r)
```

```
In [18]: 1 _sorted=[]
          2 for l in L:
          3     _sorted +=sorted(l)
```

```
In [19]: 1 _sorted[:10]
```

```
Out[19]: [-2.519609128168893,  
          -2.3544165734214726,  
          -2.3470217906548867,  
          -2.1142300462692667,  
          -2.1085060675418363,  
          -2.0771503919752417,  
          -2.053087616509429,  
          -2.053028113499684,  
          -2.0053252271060855,  
          -1.996303937273491]
```

```
In [20]: 1 resort=sorted(_sorted)    # checking that the order is good.  
        2 resort==_sorted
```

```
Out[20]: True
```

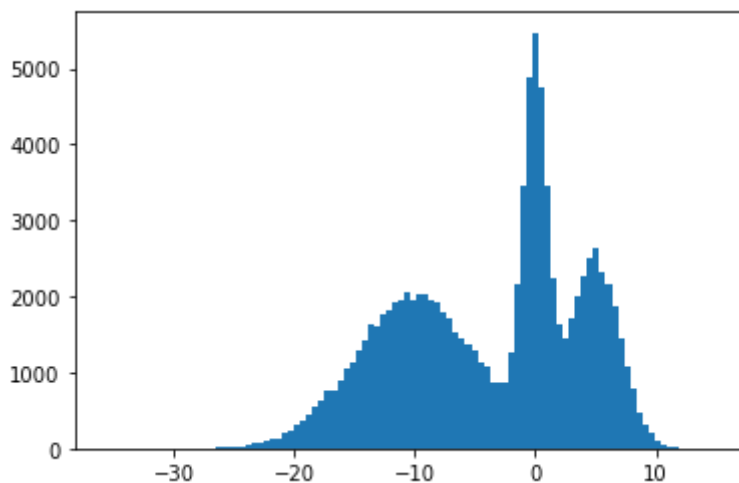
## Sorting when the distribution is not known

When the distribution is not know, we can use the empirical CDF

```
In [28]: 1 ## Lets generate a complicated distribution  
        2 n=25000  
        3 s1 = np.random.normal(0, 1, n)  
        4 s2 = np.random.normal(5, 2, n)  
        5 s3 = np.random.normal(-10, 5, 2*n)  
        6 s=concatenate([s1,s2,s3])  
        7 s.shape
```

```
Out[28]: (100000,)
```

```
In [29]: 1 hist(s,bins=100);
```



## Calculating the CDF requires sorting

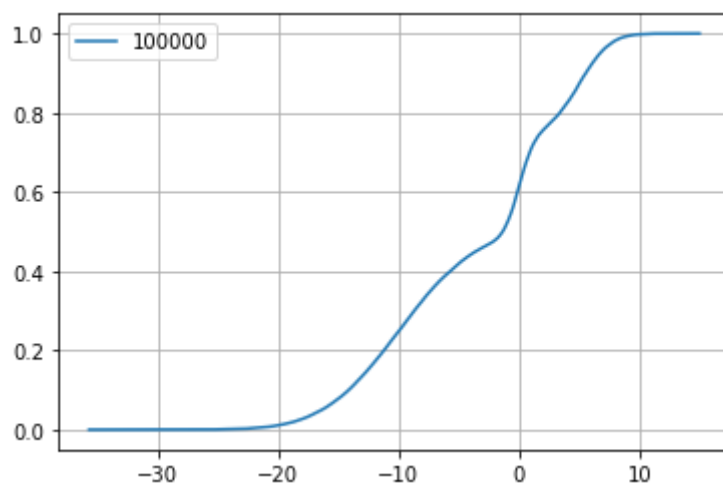


So calculating the CDF requires sorting and  $O(n)$  sorting requires knowing the CDF ...

Are we stuck in an infinite loop?

```
In [30]: 1 def plot_CDF(s):  
2         x=sorted(s)  
3         p=arange(0,1,1/s.shape[0])  
4         plot(x,p,label=str(s.shape[0]))
```

```
In [31]: 1 plot_CDF(s)  
2 grid()  
3 legend();
```



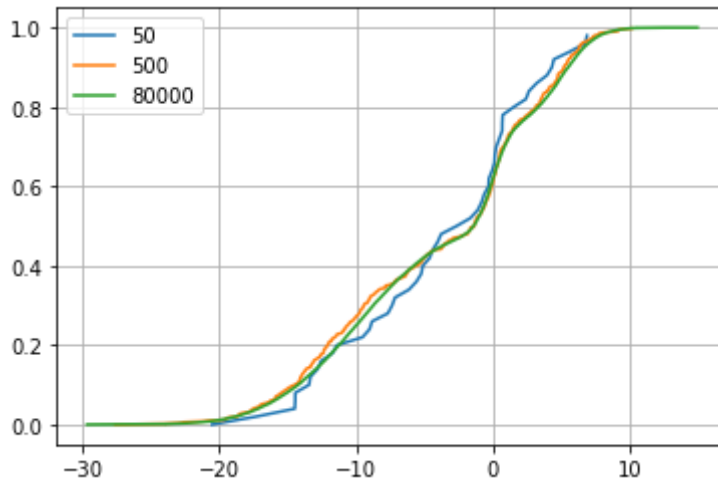
## Are we stuck?

- **No!** we can approximate the CDF using a sample
- Estimating the CDF does not require many examples (proof: more advanced probability)

```
In [32]: 1 from numpy.random import choice
```

```
In [33]: 1 #figure(figsize=[15,10])
2 for m in [50,500,80000]:
3     sample=choice(s,m)
4     plot_CDF(sample)
5 grid()
6 legend()
```

Out[33]: <matplotlib.legend.Legend at 0x15c20e4c0>



## Problem in HW5

Write a program that sorts  $s$  in linear time using bucket sort.

The steps for the algorithm are:

1. Sample 1000 points from  $s$
2. Compute 99 pivot points at the 1%,2%, ,99% of the distribution
3. Use the pivot points to define 100 buckets.
4. collect into each bucket the elements of  $s$  that are between  $i\%$  and  $(i+1)\%$
5. Compute the sizes of the least filled and the most filled bucket. How do these sizes impact the running time?
6. use python sort on each bucket.
7. concatanate the buckets to form the sorted list.

## Submitting the homework

You can use this notebook to test your code. However, to submit the code write it out into the homework and submit through grade-scope.

```
In [ ]: 1
```

