Probability and Simulation

Statistics 426: SAS Programming

Module 12

2021

Random variables

Quantities that vary at random are called random variables; they are akin to a mathematical function, and many process generate random variables. Think of capture-and-release methods to study birds and measure their beak length, every different bird measured will vary to some degree. When a process is repeated, it is typical for a new proportion to occur. The proportion of successes is a quantity that will vary during repeated sampling.

In science random variable are important because they are typically counts or measurement produced by some process by which scientific data are generated. Science is full of quantities that vary: amount of daily rainfall at a weather station, household income among households in a city, growth yields of wheat under certain conditions, distances of galaxies from Earth, the number of craters in 100-km^2 areas on Venus, and the list can go on. Think about a process and most of the time they could be considered random variables.

Probability

What can possibly be learned from processes that have varying levels of outcomes? Look for variability in the patterns of the outcomes.

The study of ways to make reliable conclusions from outcomes of random variables is the branch of **science** called *statistics*. The main driving force of statistics is in building and testing mathematical models of how the outcomes arise, and for use as hypotheses about our understanding of the processes involved. The study of patterns and randomness is a branch of mathematics called *probability*.

Probability

Law of Large Numbers (LLN): if a random process could be repeated many, many, many, *many* times, the *long-run* proportion of successes will stabilize and get closer and closer to the theoretical probability. For a small number of repetitions, considerable variation can be expected to happen.

and the next day. . . and the next day. . . and the next day. . .

LLN

RAND('FXN',parm1,parm2,. . .) - function will return with the same random sequence each time for the particular seed function will return with a vector of 0s and 1s (0=failure, 1=success) The idea is to use the function to simulate samples of size 5, then size 6, size 7, and so on until maybe 500. For each block, the mean is calculated and look at the variability in the means decreasing.

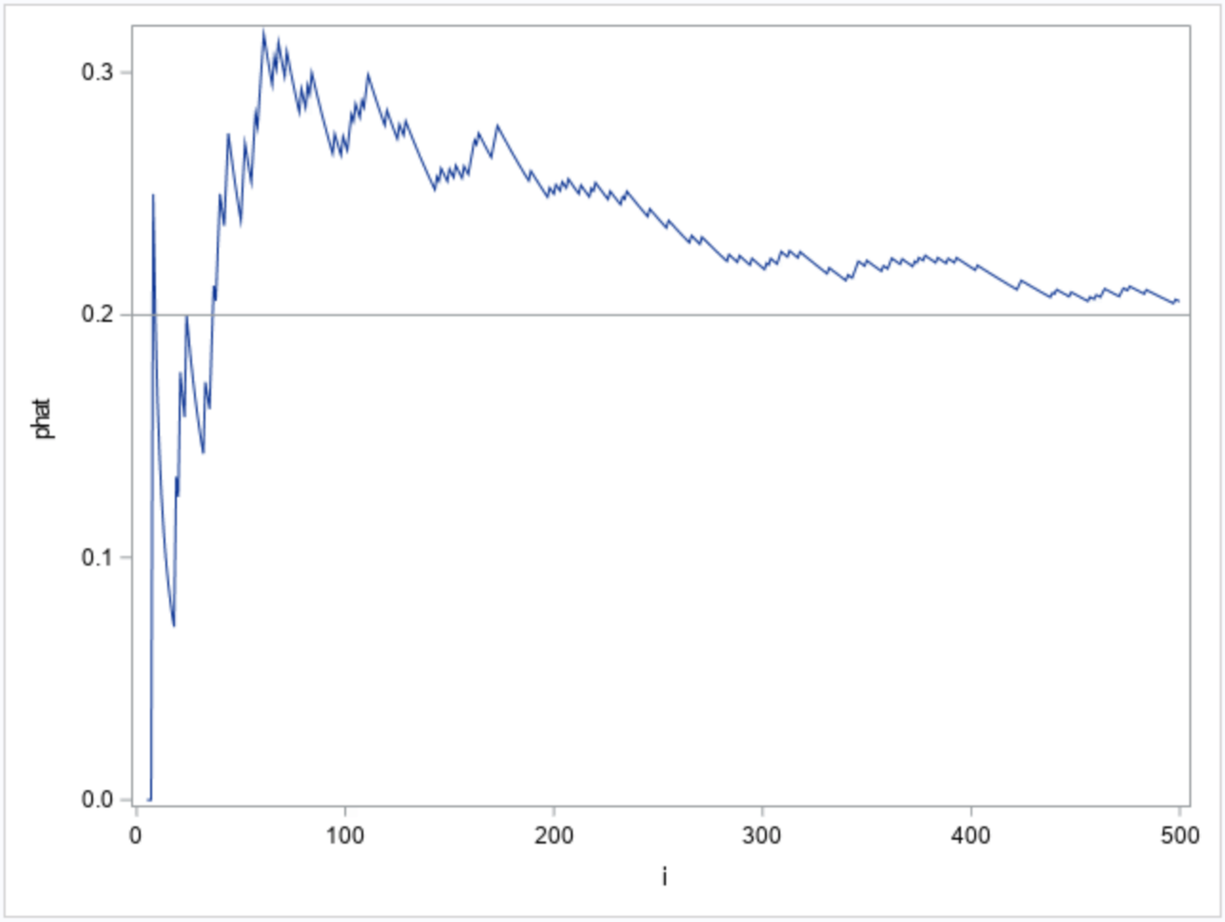
Simulation of LLN

```
data lln;
call streaminit(7777);
```

```
do i=5 to 500;
 x=rand('bern',0.2);
 output;
end;
run;
data lln;
 set lln;
 sum+x;
 phat=sum/_n_;
run;

proc sgplot data=lln;
series x=i y=phat;
refline 0.2/axis=y;
run;
```

LLN plot



LLN

You can see in the graph, as we take more and more samples, the variation is reducing and the probability will converge to a single number (the theoretical probability).

In practice, an event probability might never actually be known but rather estimated from many trials.

Probability distribution of counts

Rather than an average now, suppose we keep track of the number of successes. In simulated samples of size 30 for 10,000 iterations, the collection of the possible outcomes $(0, 1, \ldots, 30)$ of the random variable (successes out of n trials) along with the probabilities of the outcomes is called the probability distribution of a sample.

Binomial distribution

Basically what was just simulated (the sum of the outcomes) is a type of probability distribution called the *binomial distribution.* A random process that produces a binomial distribution has the following properties (1) There are a fixed number of trials n, with only two possible outcomes (success (S) or failure (F)) (2) For each trial, the probability of success p is the same (3) Trials are independent of each other (the outcome on any one trial has no effect on the outcome of any other trial) (4) The random variable is the count of y successes from n trials.

Binomial distribution function

The probability of y successes from n trials is

$$P(Y = y) = \binom{n}{y} p^y (1-p)^{n-y}$$

$$\binom{n}{y} = \frac{n!}{y!(n-y)!}$$

The mean ($E(Y)$), variance ($V(Y)$), and standard deviation ($SD(Y)$):

$$E(Y) = np \qquad V(Y) = np(1-p) \qquad SD(Y) = \sqrt{V(Y)}$$

SAS functions for binomial distribution

`PDF ('BINOMIAL',m,p,n)`$= P(Y = y)$ *or* $P(X = x)$
`PROBBNML(p,n,m)`$= P(Y \leq y)$ (less than or equal to argument; a range)

`p`: probability of success $(0 \leq p \leq 1)$
`n`: sample size
`m`: is a numeric constant, variable, or expression that specifies an integer number of successes (argument)

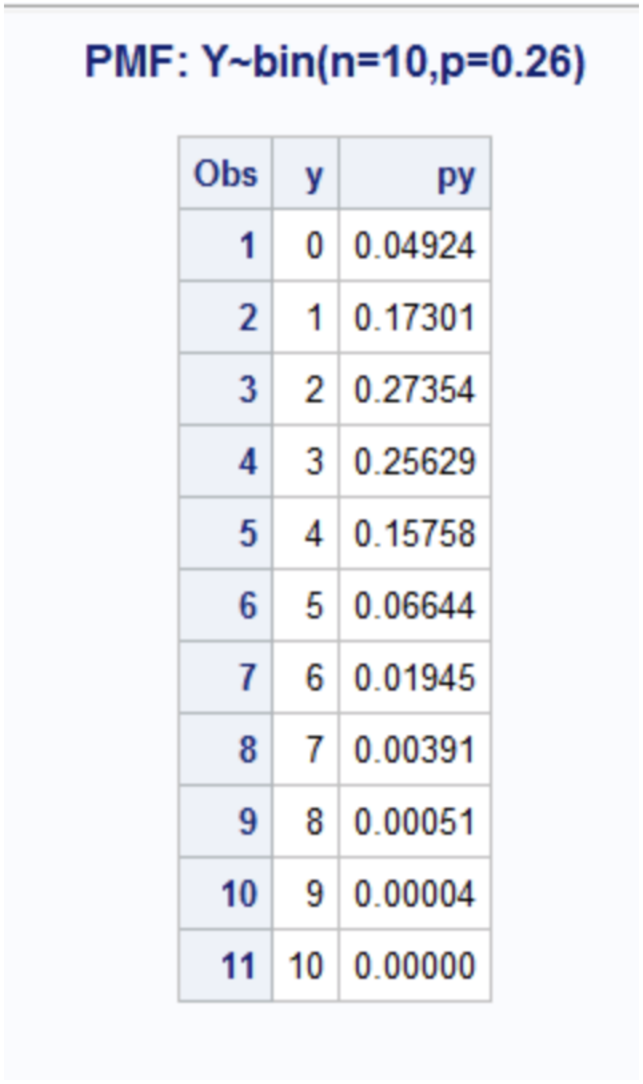
$P(Y > y) = 1 - binom(y)$ and $P(Y \geq y) = 1 - P(Y < y)$

PMF example

```
title 'PMF: Y~bin(n=10,p=0.26)';
data args;
p=.26;
n=10;
do y=0 to n;
output; end;
run;
data bin1;
set args;
py=PDF('BINOMIAL',y,p,n);
run;
proc print data=bin1;
var y py;
```

```
run;
```

PMF bin



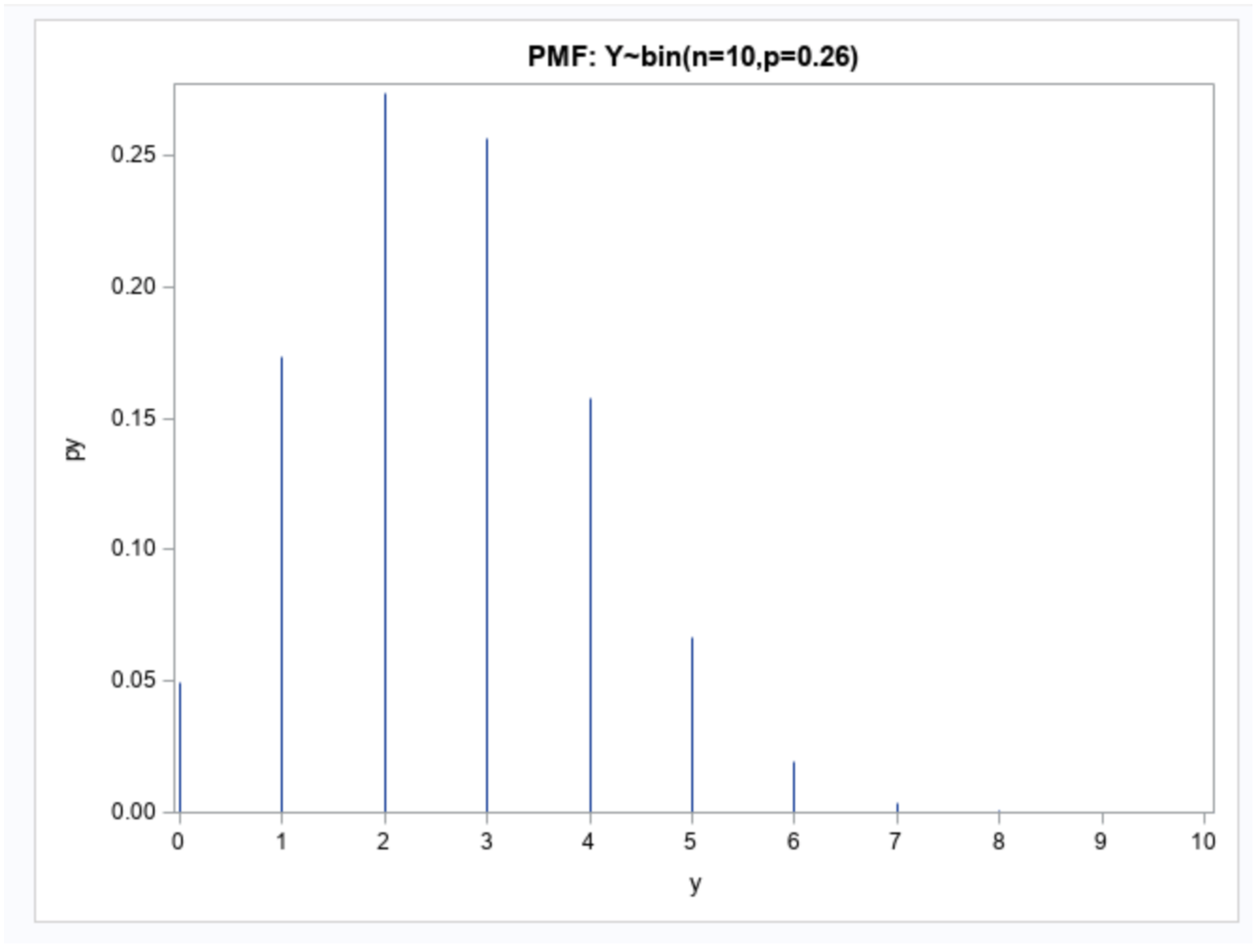
PMF: Y~bin(n=10,p=0.26)

Obs	y	py
1	0	0.04924
2	1	0.17301
3	2	0.27354
4	3	0.25629
5	4	0.15758
6	5	0.06644
7	6	0.01945
8	7	0.00391
9	8	0.00051
10	9	0.00004
11	10	0.00000

PMF plot

```
proc sgplot data=bin1;
needle x=y y=py;
xaxis values=(0 to 10);
run; quit;
title;
```

PMF bin plot

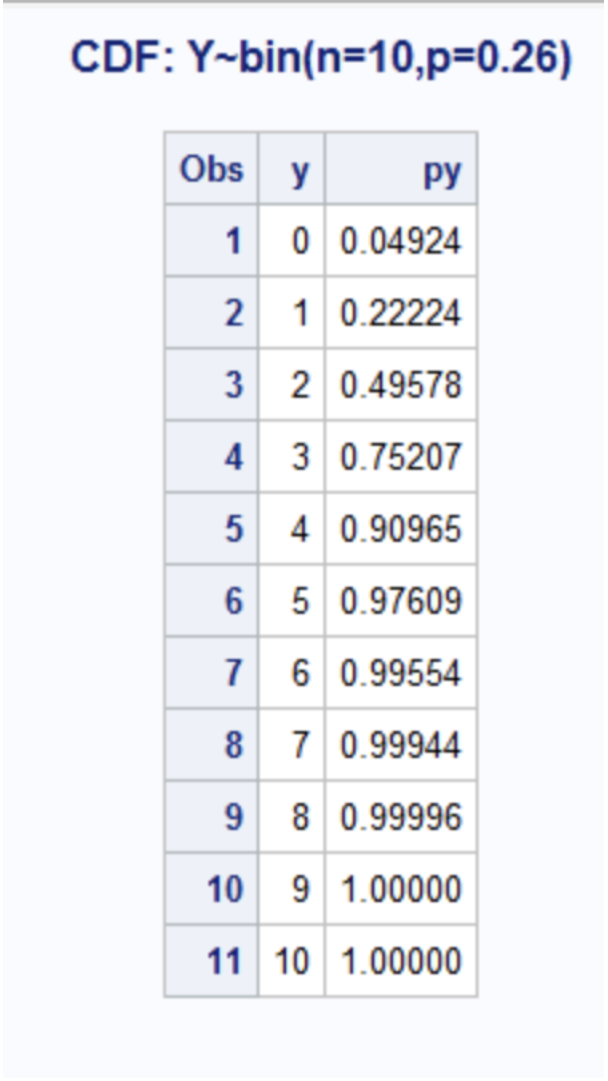


plot.png

CDF example

```
title 'CDF: Y~bin(n=10,p=0.26)';
data bin2;
set args;
cdfy=probbnml(p,n,y);
run;
proc print data=bin2 label;
var y cdfy;
label cdfy='py';
run;
```

CDF bin



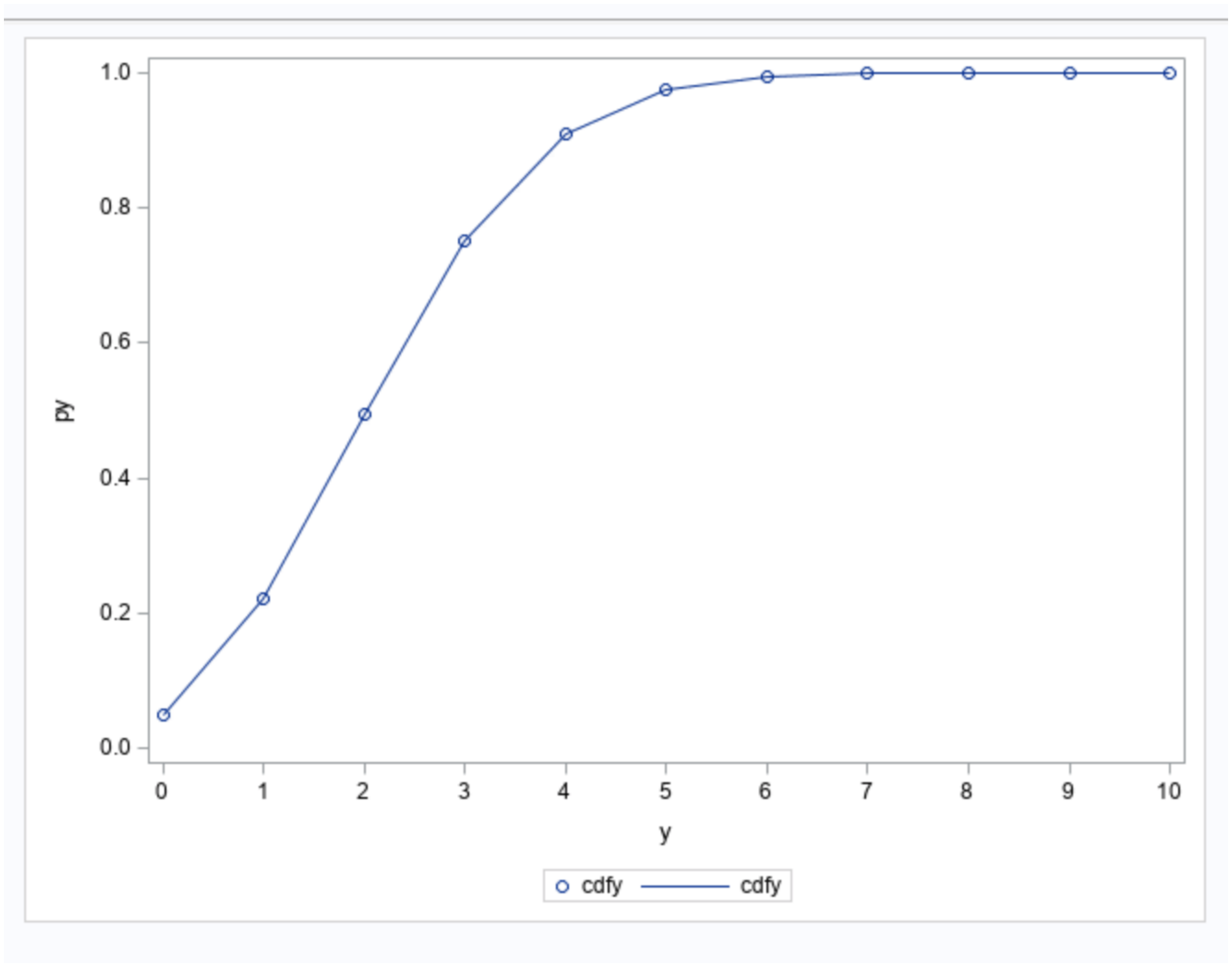
CDF: Y~bin(n=10,p=0.26)

Obs	y	py
1	0	0.04924
2	1	0.22224
3	2	0.49578
4	3	0.75207
5	4	0.90965
6	5	0.97609
7	6	0.99554
8	7	0.99944
9	8	0.99996
10	9	1.00000
11	10	1.00000

CDF plot

```
proc sgplot data=bin2;
scatter x=y y=cdfy;
series x=y y=cdfy;
xaxis values=(0 to 10);
yaxis label='py';
run; quit;
title;
```

CDF bin plot



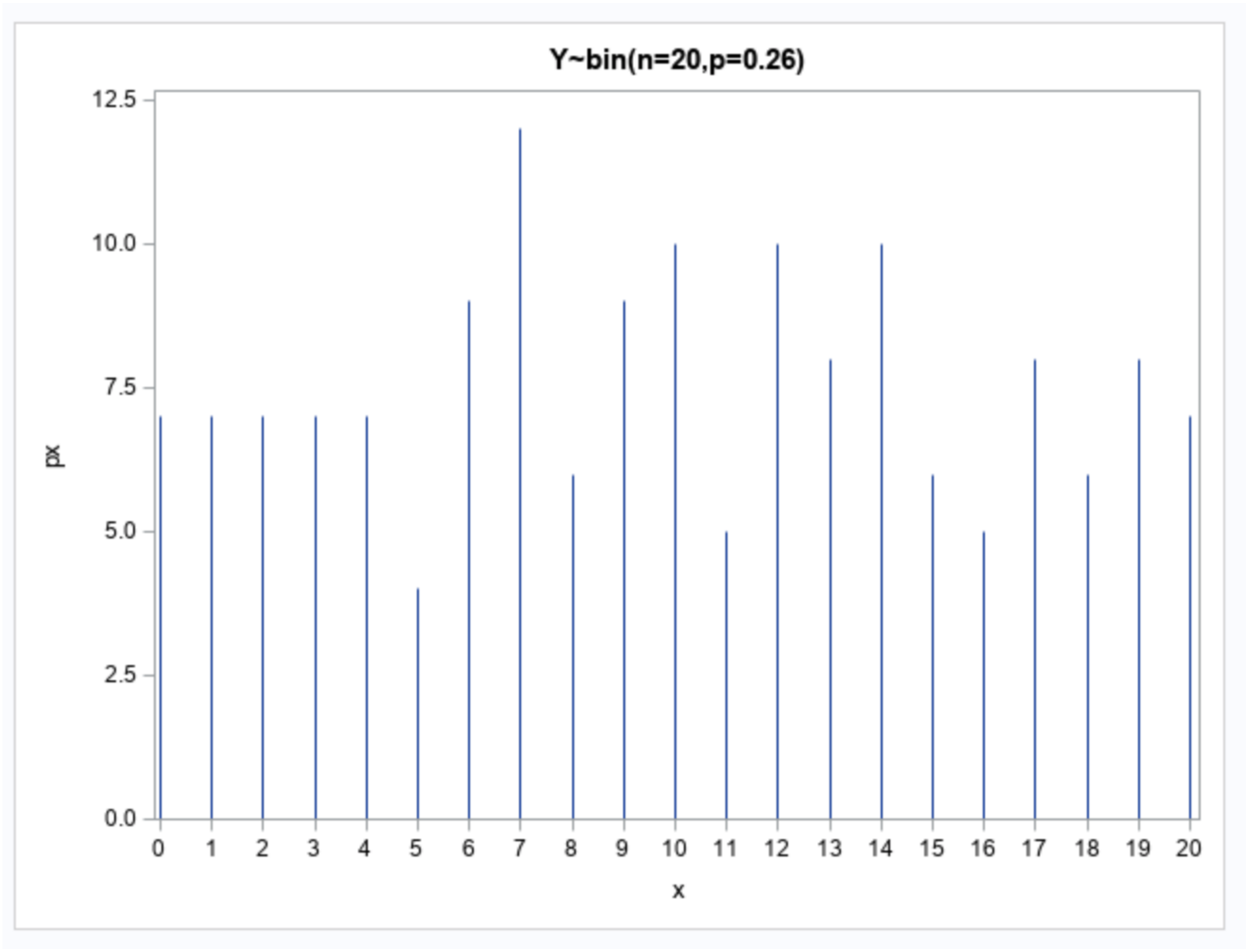
plot.png

random sample

Generates 20 random values from binom with n=30,p=.26

```
title 'Y~bin(n=20,p=0.26)';
data rbin;
do x=0 to 20;
 px=ranbin(7777,30,0.26);
 output;
end;
run;
proc sgplot data=rbin;
needle x=x y=px;
xaxis values=(0 to 20);
run; quit;
title;
```

Random bin sample plot



plot1.png

Probability distributions of measurements

Life is discrete; every measurement device we have is limited to a certain degree of accuracy, like to the second decimal place. The set of measurements that can be output by the device is a discrete set, meaning we can count the members of the set. Even the set of numbers that can be stored in the floating point system on a computer is a discrete set (granted, it is a *large* set).

Discrete vs. continuous

Members of the set of real numbers in a interval, say between 0 and 1, cannot be "counted" like the members of a discrete set. Real numbers are a mathematical abstraction beyond our everyday experience; there are numerous circumstances in science in which modeling the numerical output of some phenomenon as a set of real numbers offers great calculation convenience at little expense of realism.

In probability, if the range of output of a random process is "fine-grained" with many possible values, we can often achieve great simplification by using a *continuous probability distribution* to model the process. A random variable with a continuous probability distribution takes a real values within a range (interval) of possible values.

Uniform distribution

When a random variable has a continuous probability distribution, we assign probabilities to intervals instead of individual numbers. In earlier modules, we have used the *uniform distribution*. The random variable U takes a real value between 0 and 1 (in computers U takes on a value from any of the possible 16-decimal place numbers between 0 and 1). The probability that U takes on a value between a and b is $b - a$ where $0 \leq a \leq b \leq 1$, that is

$$P(a \leq U \leq b) = b - a$$

Normal distribution

For probabilities for area to the left, use `PROBNORM()`; for area to the right use `1-PROBNORM()`; for area between two values (a<b) use `PROBNORM(b)-PROBNORM(a)`.

$P(Y = y) = dne$ because Y is continuous

`PROBNORM(x)`
`x`: (y) argument; z-score or formula; mean and standard deviation are assumed $Z \sim N(\mu = 0, \sigma = 1)$

$$Y \sim N(\mu, \sigma)$$

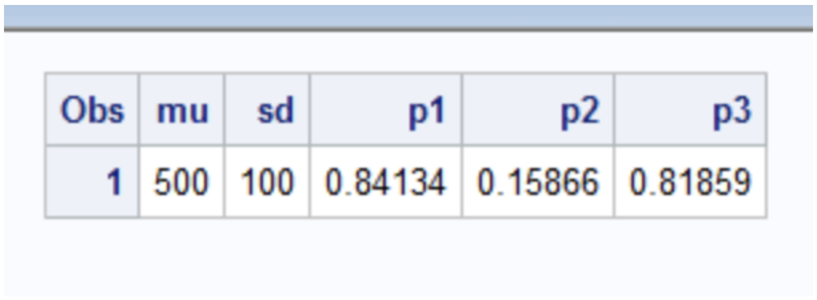
With $z = \frac{y-\mu}{\sigma}$

Normal example

$Y \sim N(500, 100)$

```
data norms;
mu=500;
sd=100;
p1=probnorm((600-mu)/sd); *P(Y<600);
p2=1-probnorm((600-mu)/sd); *P(Y>600);
p3=probnorm((700-mu)/sd)-probnorm((400-mu)/sd); *P(400<Y<700);
run;
proc print data=norms;
run;
```

Normal probabilities



Obs	mu	sd	p1	p2	p3
1	500	100	0.84134	0.15866	0.81859

Central Limit Theorem (CLT)

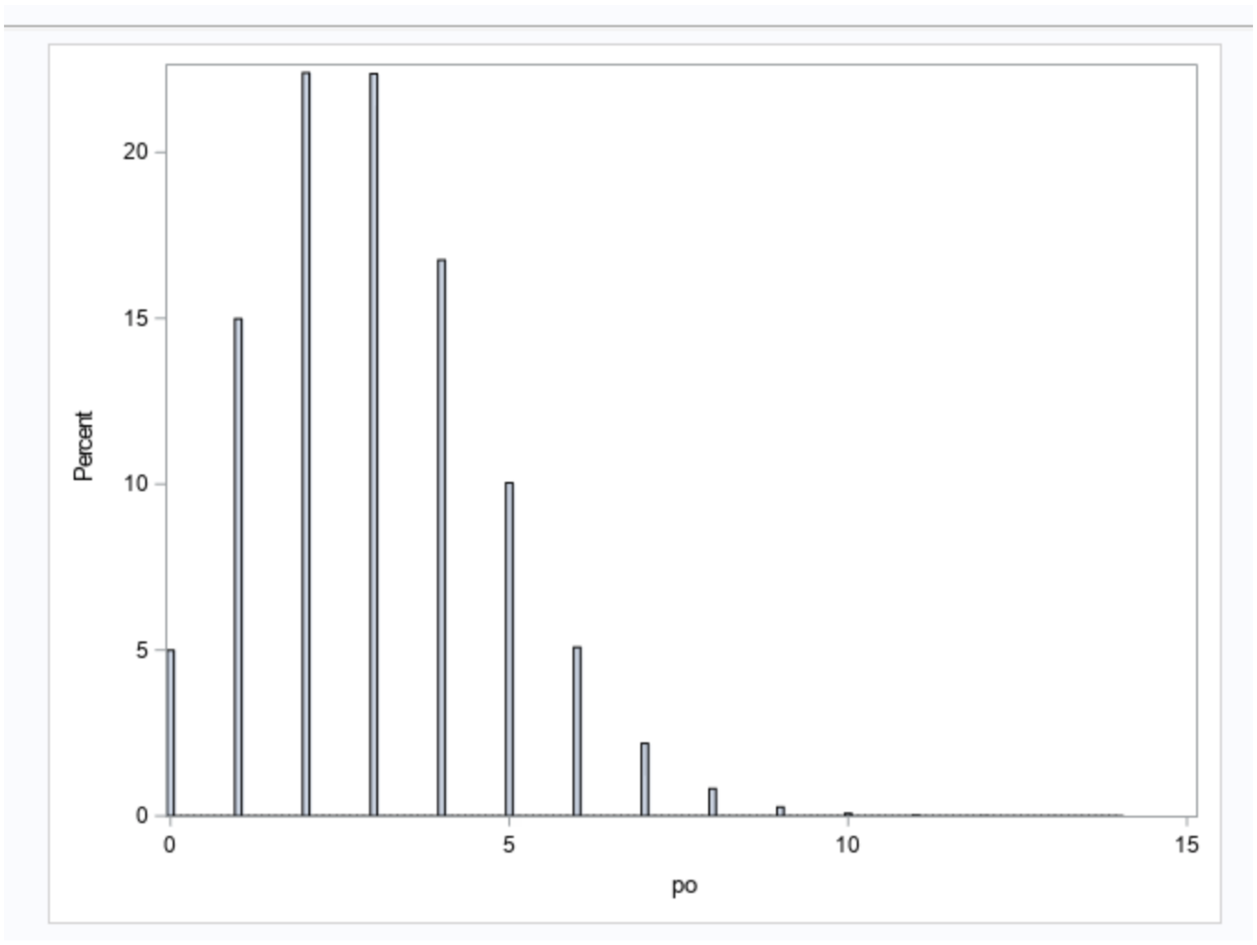
defn: The sampling distribution of the sample mean (or other sample statistic) will be approximately normal with mean μ and standard deviation of the sampling distribution of the sample mean $\sigma_{\bar{y}} = \sigma/\sqrt{n}$ (aka standard error), provided n is sufficiently large.

Examples of CLT with the poisson and binomial distributions.

CLT Poisson

```
%let NumSample=100;
%let SampleSize=10000;
%let lambda=3;
data pois;
 call streaminit(7777);
  do sample=1 to &NumSample;
   do n=1 to &SampleSize;
    po=rand("Poisson",&lambda);
    output;
   end;
  end;
run;
proc sgplot data=pois;
histogram po;
run; quit;
```

Random poisson

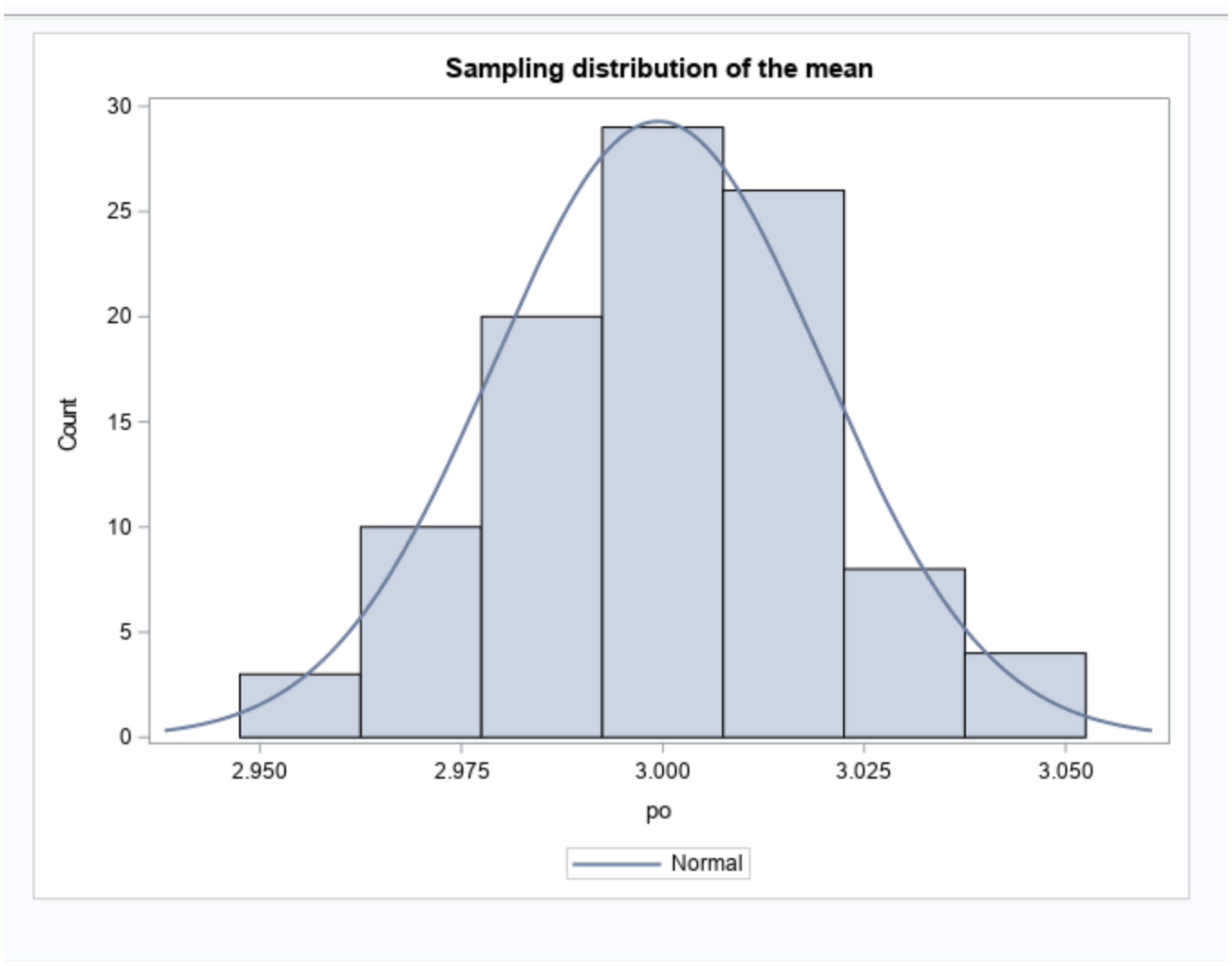


plot.png

Poisson means

```
proc means data=pois noprint mean;
   class sample;
   output out=pois_means(where=( _TYPE_=1 & _STAT_='MEAN'));
run;
proc print data=pois_means;
run;
title 'Sampling distribution of the mean';
proc sgplot data=pois_means;
   histogram po / scale=count;
   density po / type=normal;
run;
title;
```

Poisson means



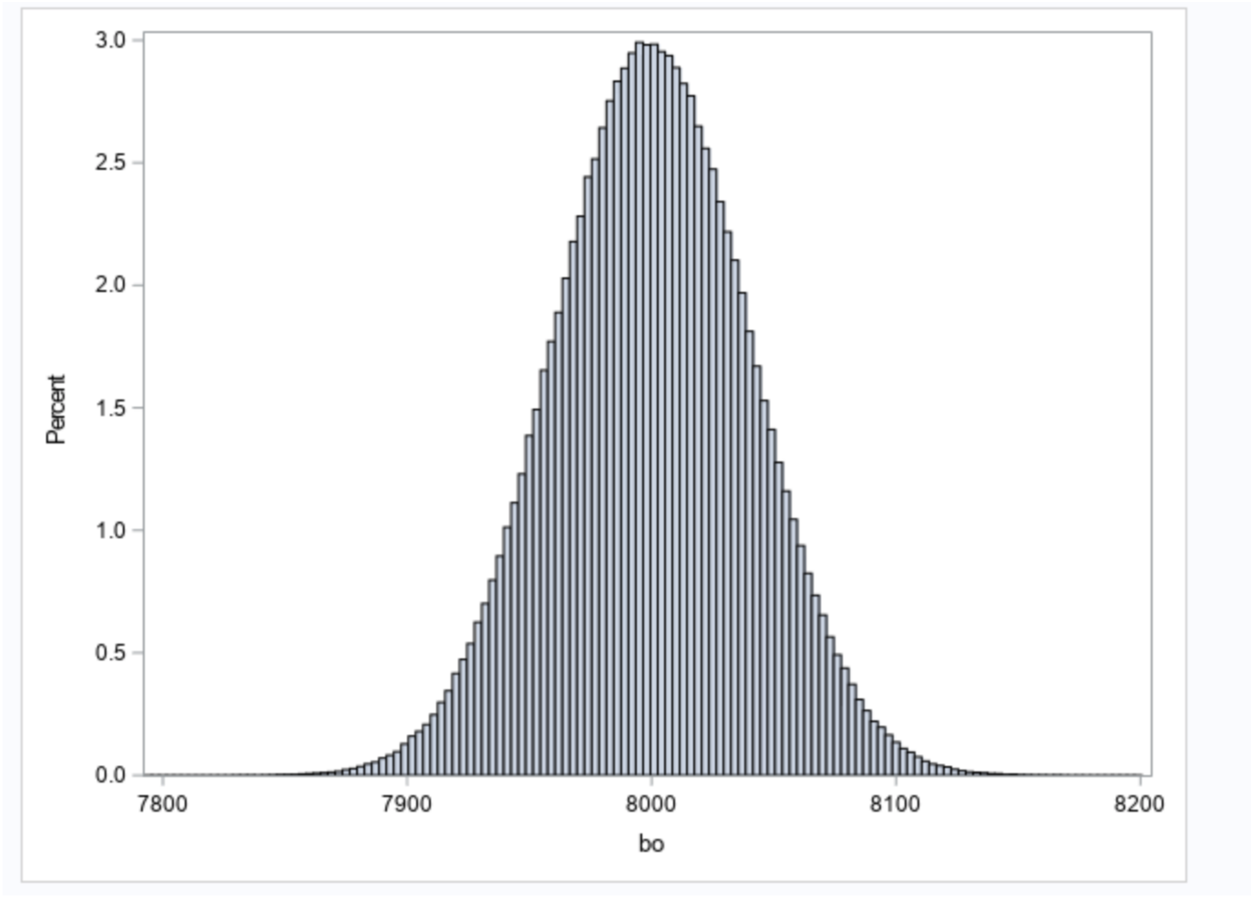
plot.png

CLT binomial

```
%let NumSample=100;
%let SampleSize=10000;
%let p=.8;
data bin;
```

```
 call streaminit(7777);
  do sample=1 to &NumSample;
   do n=1 to &SampleSize;
    bo=rand("Binomial",&p,&SampleSize);
    output;
   end;
  end;
run;
proc sgplot data=bin;
histogram bo;
run; quit;
```

Random binomial

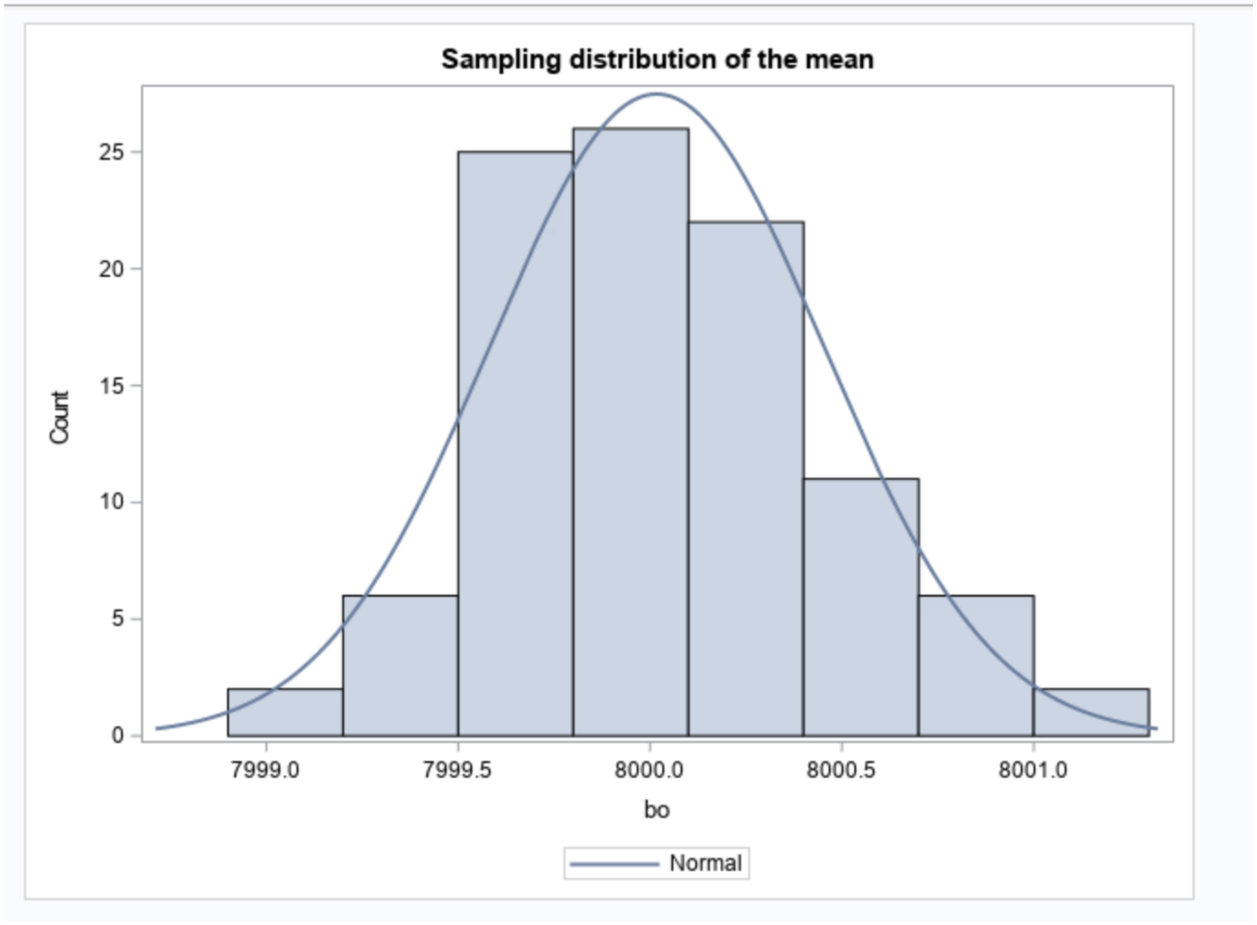


plot2.png

Binomial means

```
proc means data=bin noprint mean;
   class sample;
   output out=bin_means(where=( _TYPE_=1 & _STAT_='MEAN'));
run;
proc print data=bin_means;
run;
title 'Sampling distribution of the mean';
proc sgplot data=bin_means;
   histogram bo / scale=count;
   density bo / type=normal;
run;
title;
```

Binomial means



plot.png