STAT221Week10

STAT 221 Week 10: Limits, Convergence, and Estimation

STAT 221 2010 S1: Monte Carlo Methods

Inference and Estimation: The Big Picture

The Markov Chains we discussed last week fit into our Big Picture, which is about inference and estimation and especially inference and estimation problems where computational techniques are helpful.

	Point estimation	Set estimation
Parametric	MLE of finitely many parameters *done*	Confidence intervals, via the central limit theorem
Non-parametric (infinitely many parameters)	*coming up ...*	*coming up ...*
One/Many-dimensional Integrals (finite-dimensional)	*coming up ...*	*coming up ...*

But before we move on we have to discuss what makes it all work: the idea of limits - where do you get to if you just keep going?

Limits

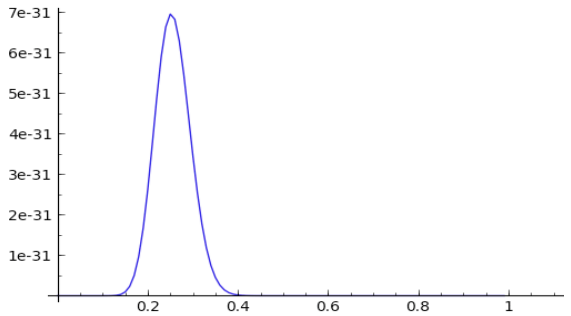
Last week we described a Markov Chain, informally, as a system which "jumps" among several states, with the next state depending (probabilistically) only on the current state. Since the system changes randomly, it is generally impossible to predict the exact state of the system in the future. However, the statistical and probailistic properties of the system's future can be predicted. In many applications it is these statistical properties that are important. We saw how we could find a steady state vector:

$$\mathbf{s} = \lim_{n \to \infty} \mathbf{p}^{(n)}$$

(And we noted that $\mathbf{p}^{(n)}$ only **converges** to a strictly positive vector if $\mathbf{P}$ is a **regular** transition matrix.)

The week before, we talked about the likelihood function and maximum likelihood estimators for making point estimates of model parameters. For example for the $Bernoulli(\theta^*)$ RV (a $Bernoulli$ RV with true but possibly unknown parameter θ^*, we found that the likelihood function was $L_n(\theta) = \theta^{t_n}(1-\theta)^{(n-t_n)}$ where $t_n = \sum_{i=1}^{n} x_i$. We also found the maxmimum likelihood estimator (MLE) for the $Bernoulli$ model, $\widehat{\theta}_n = \frac{1}{n}\sum_{i=1}^{n} x_i$.

We demonstrated these ideas using samples simulated from a $Bernoulli$ process with a secret θ^*. We had an interactive plot of the likelihood function where we could increase n, the number of simulated samples or the amount of data we had to base our estimate on, and see the effect on the shape of the likelihood function. The animation belows shows the changing likelihood function for the Bernoulli process with unknown θ^* as n (the amount of data) increases.



Likelihood function for Bernoulli process, as n goes from 1 to 1000 in a continuous loop

For large n, you can probably make your own guess about the true value of θ^* even without knowing t_n. As the animation progresses, we can see the likelihood function 'homing in' on $\theta = 0.3$.

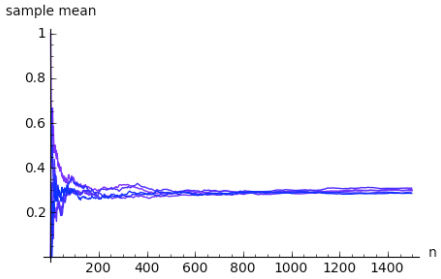
We can see this in another way, by just looking at the sample mean as n increases. An easy way to do this is with running means: generate a very large sample and then calculate the mean first over just the first observation in the sample, then the first two, first three, etc etc (running means were discussed in an earlier worksheet if you want to go back and review them in detail in your own time). Here we just define a function so that we can easily generate sequences of running means for our $Bernoulli$ process with the unknown θ^*.

```
%auto
def bernoulliSecretThetaRunningMeans(n, mySeed = None):
    '''Function to give a list of n running means from Bernoulli with unknown theta.

    Param n is the number of running means to generate.
    Param mySeed is a value for the seed of the random number generator, defaulting
to None
    Note: the unknown theta parameter for the Bernoulli process is defined in
bernoulliSampleSecretTheta
    Return a list of n running means.'''

    sample = bernoulliSampleSecretTheta(n, simSeed = mySeed)
    from pylab import cumsum # we can import in the middle of code
    csSample = list(cumsum(sample))
    samplesizes = range(1, n+1,1)
    return [RR(csSample[i])/samplesizes[i] for i in range(n)]
```

Now we can use this function to look at say 5 different sequences of running means (they will be different, because for each iteration, we will simulate a different sample of $Bernoulli$ observations).

```
nToGenerate = 1500
iterations = 5
xvalues = range(1, nToGenerate+1,1)
for i in range(iterations):
    redshade = 0.5*(iterations - 1 - i)/iterations # to get different colours for
the lines
    bRunningMeans = bernoulliSecretThetaRunningMeans(nToGenerate)
    pts = zip(xvalues,bRunningMeans)
    if (i == 0):
        p = line(pts, rgbcolor = (redshade,0,1))
    else:
```

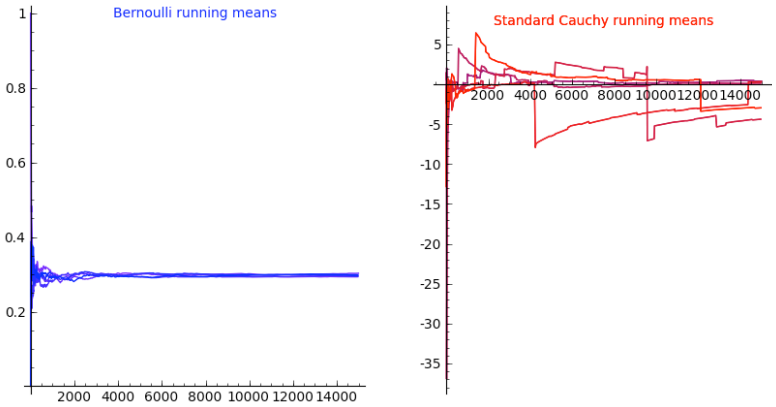
```
        p += line(pts, rgbcolor = (redshade,0,1))
show(p, figsize=[5,3], axes_labels=['n','sample mean'])
```



What we notice is how the different lines **converge** on a sample mean of close to 0.3.

Is life always this easy? Unfortunately no. In the plot below we show the well-behaved running means for the *Bernoulli* and beside them the running means for simulated standard *Cauchy* random variables. They are all over the place, and each time you re-evaluate the cell you'll get different all-over-the-place behaviour.

```
nToGenerate = 15000
iterations = 5
g = twoRunningMeansPlot(nToGenerate, iterations) # use hidden function to make plot
show(g,figsize=[10,5])
```



We talked about the Cauchy in more detail in an earlier worksheet. If you cannot recall the detail and are interested, go back to that in your own time. The message here is that although with the Bernoulli process, the sample means converge as the number of observations increases, with the Cauchy they do not.

We talked about $\mathbf{p}^{(n)}$, for $\mathbf{p}$ our Markov Chain transition matrix, **converging**. We talked about sample means **converging** (or not). What do we actually mean by 'converge'? These ideas of convergence and limits are fundamental to using Monte Carlo techniques: we need to be able to justify that the way we are attacking a problem will give us the 'right' answer. (At its very simplest, how do we justify that, by generating lots of simulations, we can get to some good approximation for a probability or an integral or a sum?) The advantages of an MLE as a point estimate in parametric estimation all come back to limits and convergence (remember how the likelihood function 'homed in'). And, as we will see when we do non-parametric estimation, limits and convergence are also fundamental there.

Limits of a Sequence of Real Numbers

A sequence of real numbers $x_1, x_2, x_3, \ldots$ (which we can also write as $\{x_i\}_{i=1}^{\infty}$) is said to converge to a limit $a \in \mathbb{R}$,

$$\lim_{i\to\infty} x_i = a$$

if for every natural number $m \in \mathbb{N}$, a natural number $N_m \in \mathbb{N}$ exists such that for every $j \geq N_m$, $|x_j - a| \leq \frac{1}{m}$

What is this saying? $|x_j - a|$ is measuring the closeness of the jth value in the sequence to a. If we pick bigger and bigger m, $\frac{1}{m}$ will get smaller and smaller. The definition of the limit is saying that if a is the limit of the sequence then we can get the sequence to become as close as we want ('arbitrarily close') to a, and to stay that close, by going far enough into the sequence ('for every $j \geq N_m$, $|x_j - a| \leq \frac{1}{m}$')

($\mathbb{N}$, the natural numbers, are just the 'counting numbers' $\{1, 2, 3, \ldots\}$.)

Take a trivial example, the sequence $\{x_i\}_{i=1}^{\infty} = 17, 17, 17, \ldots$

Clearly, $\lim_{i\to\infty} x_i = 17$, but let's do this formally:

For every $m \in \mathbb{N}$, take $N_m = 1$, then

$\forall\, j \geq N_m = 1, |x_j - 17| = |17 - 17| = 0 \leq \frac{1}{m}$, as required.

($\forall$ is mathspeak for 'for all' or 'for every')

What about $\{x_i\}_{i=1}^{\infty} = \frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \ldots$, i.e., $x_i = \frac{1}{i}$?

$\lim_{i\to\infty} x_i = \lim_{i\to\infty} \frac{1}{i} = 0$

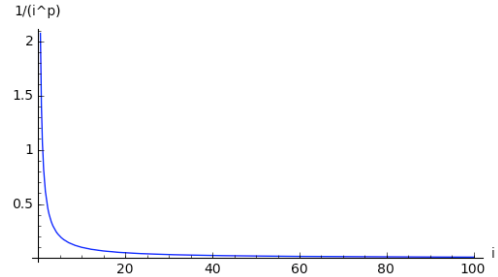
For every $m \in \mathbb{N}$, take $N_m = m$, then $\forall\, j \geq m$, $|x_j - 0| \leq |\frac{1}{m} - 0| = \frac{1}{m}$

You try

Think about $\{x_i\}_{i=1}^{\infty} = \frac{1}{1^p}, \frac{1}{2^p}, \frac{1}{3^p}, \ldots$ with $p > 0$. The limit $\lim_{i\to\infty} \frac{1}{i^p} = 0$, provided $p > 0$.

You can draw the plot of this very easily using the Sage symbolic expressions we have already met ('f.subs(...)' allows us to substitute a particular value for one of the symbolic variables in the symbolic function f, in this case a value to use for p).

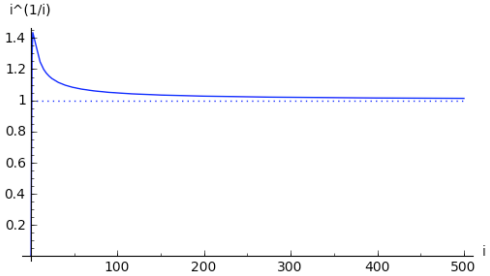
```
var('i, p')
f = 1/(i^p)
plot(f.subs(p=1), (x, 0, 100), axes_labels=('i',f)).show(figsize=[6,3]) # make and
show plot, note we can use f in the label
```



What about $\{x_i\}_{i=1}^{\infty} = 1^{\frac{1}{2}}, 2^{\frac{1}{2}}, 3^{\frac{1}{2}}, \ldots$. The limit $\lim_{i\to\infty} i^{\frac{1}{i}} = 1$.

This one is not as easy to see intuitively, but again we can plot it with Sage.

```
var('i')
f = i^(1/i)
n=500
p=plot(f.subs(p=1), (x, 0, n), axes_labels=('i',f)) # main plot
p+=line([(0,1),(n,1)],linestyle=':') # add a dotted line at height 1
p.show(figsize=[6,3]) # show the plot
```



Finally, $\{x_i\}_{i=1}^{\infty} = p^{\frac{1}{1}}, p^{\frac{1}{2}}, p^{\frac{1}{3}}, \ldots$, with $p > 0$. The limit $\lim_{i\to\infty} p^{\frac{1}{i}} = 1$ provided $p > 0$.

You can cut and paste (with suitable adaptations) to try to plot this one as well ...

```

```

(end of You Try)

back to the real stuff ...

Limits of Functions

We say that a function $f(x) : \mathbb{R} \to \mathbb{R}$ has a limit $L \in \mathbb{R}$ as x approaches a:

$$\lim_{x\to a} f(x) = L$$

provided $f(x)$ is arbitrarily close to L for all $(\forall)$ values of x that are sufficiently close to but not equal to a.

For example

Consider the function $f(x) = (1+x)^{\frac{1}{x}}$

$\lim_{x\to 0} f(x) = \lim_{x\to 0}(1+x)^{\frac{1}{x}} = e \approx 2.71828\cdots$

even though $f(0) = (1+0)^{\frac{1}{0}}$ is undefined!

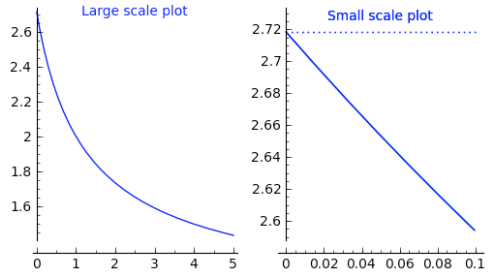
```
# x is defined as a symbolic variable by default by Sage so we do not need var('x')
f = (1+x)^(1/x)
f.subs(x=0) # this will give you an error message
```

```
Traceback (click to the left of this block for traceback)
...
RuntimeError: power::eval(): division by zero
```

You can get some idea of what is going on with two plots on different scales

```
f = (1+x)^(1/x)
n1=5
p1=plot(f.subs(p=1), (x, 0.001, n1), axes_labels=('x',f)) # main plot
t1 = text("Large scale plot", (n1/2,e), rgbcolor='blue',fontsize=10)
```

```
n2=0.1
p2=plot(f.subs(p=1), (x, 0.0000001, n2), axes_labels=('x',f)) # main plot
p2+=line([(0,e),(n2,e)],linestyle=':') # add a dotted line at height e
t2 = text("Small scale plot", (n2/2,e+.01), rgbcolor='blue',fontsize=10)
show(graphics_array((p1+t1,p2+t2)),figsize=[6,3]) # show the plot
```



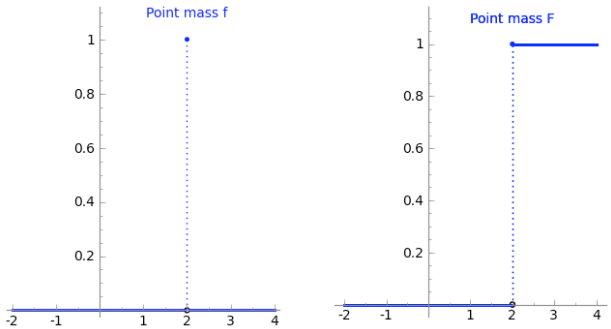
all this has been laying the groundwork for the topic of real interest to us ...

Limit of a Sequence of Random Variables

We want to be able to say things like $\lim_{i\to\infty} X_i = X$ in some sensible way. X_i are some random variables, X is some 'limiting random variable', but what do we mean by 'limiting random variable'?

To help us, lets introduce a very very simple random variable, one that puts all its mass in one place.

```
theta = 2.0
show(graphics_array((pmfPointMassPlot(theta),cdfPointMassPlot(theta))),figsize=
[8,4]) # show the plots
```



This is known as the $Point\,Mass(\theta)$ random variable, $\theta \in (R)$: the density $f(x)$ is 1 if $x = \theta$ and 0 everywhere else

$$f(x;\theta) = \begin{cases} 0 & \text{if } x \neq \theta \\ 1 & \text{if } x = \theta \end{cases}$$

$$F(x;\theta) = \begin{cases} 0 & \text{if } x < \theta \\ 1 & \text{if } x \geq \theta \end{cases}$$

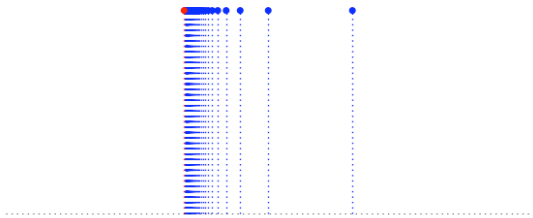
So, if we had some sequence $\{\theta_i\}_{i=1}^{\infty}$ and $\lim_{i\to\infty} \theta_i = \theta$

and we had a sequence of random variables $X_i \sim Point\,Mass(\theta_i)$, $i = 1, 2, 3, \ldots$

then we could talk about a limiting random variable as $X \sim Point\,Mass(\theta)$:

i.e., we could talk about $\lim_{i\to\infty} X_i = X$

```
# mock up a picture of a sequence of point mass rvs converging on theta = 0
ptsize = 20
i = 1
theta_i = 1/i
p = points((theta_i,1), rgbcolor="blue", pointsize=ptsize)
p += line([(theta_i,0),(theta_i,1)], rgbcolor="blue", linestyle=':')
while theta_i > 0.01:
    i+=1
    theta_i = 1/i
    p += points((theta_i,1), rgbcolor="blue", pointsize=ptsize)
    p += line([(theta_i,0),(theta_i,1)], rgbcolor="blue", linestyle=':')
p += points((0,1), rgbcolor="red", pointsize=ptsize)
p += line([(0,0),(0,1)], rgbcolor="red", linestyle=':')
p.show(xmin=-1, xmax = 2, ymin=0, ymax = 1.1, axes=false, gridlines=[None,[0]],
figsize=[7,3])
```



Now, we want to generalise this notion of a limit to other random variables (that are not necessarily $Point\ Mass(\theta_i)$ RVs)

What about one many of you will be familiar with - the 'bell-shaped curve' - the $Gaussian(\mu, \sigma^2)$ or $Normal(\mu, \sigma^2)$ RV?

The probability density function (PDF) $f(x)$ is given by

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

The two parameters, μ and σ, are sometimes referred to as the location and scale parameters.

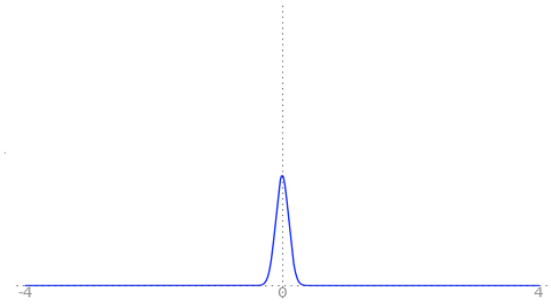
To see why this is, use the interactive plot below to have a look at what happens to the shape of the density function $f(x)$ when you change μ or increase or decrease σ:

```
@interact
def _(my_mu=input_box(0, label='mu') ,my_sigma=input_box(1,label='sigma')):
    ''□<script type="text/javascript" src="http://sage.math.canterbury.ac.nz
/javascript/tiny_mce/themes/advanced/langs/en.js"></script>9;Interactive function to
plot the normal pdf and ecdf.'''

    if my_sigma > 0:
        html('<h4>Normal('+str(my_mu)+','+str(my_sigma)+'<sup>2</sup>)</h4>')
        var('mu sigma')
        f = (1/(sigma*sqrt(2.0*pi)))*exp(-1.0/(2*sigma^2)*(x - mu)^2)
        p1=plot(f.subs(mu=my_mu,sigma=my_sigma), (x, my_mu - 3*my_sigma - 2, my_mu +
3*my_sigma + 2), axes_labels=('x','f(x)'))
        show(p1,figsize=[8,3])
    else:
        print "sigma must be greater than 0"
```

Consider the sequence of random variables $X_1, X_2, X_3, \ldots$, where

$X_1 \sim Normal(0, 1)$
$X_2 \sim Normal(0, \frac{1}{2})$
$X_3 \sim Normal(0, \frac{1}{3})$
$X_4 \sim Normal(0, \frac{1}{4})$
$\vdots$
$X_i \sim Normal(0, \frac{1}{i})$
$\vdots$

We can use the animation below to see how the PDF $f_i(x)$ looks as we move through the sequence of X_i (the animation only goes to $i = 25$, $\sigma = 0.04$ but you get the picture ...)



Normal curve animation, looping through $\sigma = \frac{1}{i}$ for $i = 1, \ldots, 25$

We can see that the probability mass of $X_i \sim Normal(0, \frac{1}{i})$ increasingly concentrates about 0 as $i \to \infty$ and $\frac{1}{i} \to 0$

Does this mean that $\lim_{i\to\infty} X_i = Point\ Mass(0)$?

No, because for any i, however large, $P(X_i = 0) = 0$ because X_i is a **continuous** RV (for any continous RV X, for any $x \in \mathbb{R}$, $P(X = x) = 0$).

So, we need to refine our notions of convergence when we are dealing with random variables

Convergence in Distribution

Let $X_1, X_2, \ldots$ be a sequence of random variables and let X be another random variable. Let F_i denote the distribution function (DF) of X_i and let F denote the distribution function of X.

Now, **if** for any real number t at which F is continuous,

$$\lim_{i\to\infty} F_i(t) = F(t)$$

(in the sense of the convergence or limits of functions we talked about earlier)

Then we can say that the sequence or RVs X_i, $i = 1, 2, \ldots$ **converges to** X **in distribution** and write $X_i \xrightarrow{d} X$.

An equivalent way of defining convergence in distribution is to go right back to the meaning of the probabilty space 'under the hood' of a random variable, a random variable X as a mapping from the sample space Ω to the real line ($X : \Omega \to \mathbb{R}$), and the sample points or outcomes in the sample space, the $\omega \in \Omega$. For $\omega \in \Omega$, $X(\omega)$ is the mapping of ω to the real line $\mathbb{R}$. We could look at the set of ω such that $X(\omega) \leq t$, i.e. the set of ω that map to some value on the real line less than or equal to t, $\{\omega : X(\omega) \leq t\}$.

Saying that for any $t \in \mathbb{R}$, $\lim_{i\to\infty} F_i(t) = F(t)$ is the equivalent of saying that for any $t \in \mathbb{R}$,

$$\lim_{i\to\infty} P\left(\{\omega : X_i(\omega) \leq t\}\right) = P\left(\{\omega : X(\omega) \leq t\right)$$

Armed with this, we can go back to our sequence of $Normal$ random variables $X_1, X_2, X_3, \ldots$, where

$X_1 \sim Normal(0, 1)$

$X_2 \sim Normal(0, \frac{1}{2})$
$X_3 \sim Normal(0, \frac{1}{3})$
$X_4 \sim Normal(0, \frac{1}{4})$
$\vdots$
$X_i \sim Normal(0, \frac{1}{i})$
$\vdots$

and let $X \sim Point\,Mass(0)$,

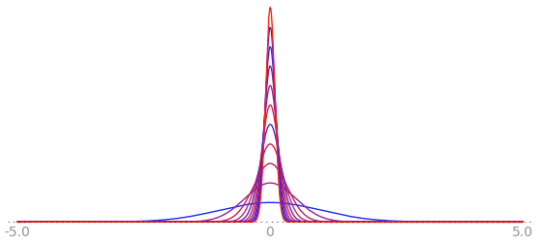
and say that the X_i **converge in distribution** to the $x \sim Point\,Mass$ RV X,

$$X_i \xrightarrow{d} X$$

What we are saying with convergence in distribution, informally, is that as i increases, we increasingly expect to see the next outcome in a sequence of random experiments becoming better and better modeled by the limiting random variable. In this case, as i increases, the $Point\,Mass(0)$ is becoming a better and better model for the next outcome of a random experiment with outcomes $\sim Normal(0, \frac{1}{i})$.

```
# mock up a picture of a sequence of converging normal distributions
my_mu = 0
upper = my_mu + 5; lower = -upper;     # limits for plot
var('mu sigma')
stop_i = 12
html('<h4>N(0,1) to N(0, 1/'+str(stop_i)+')</h4>')
f = (1/(sigma*sqrt(2.0*pi)))*exp(-1.0/(2*sigma^2)*(x - mu)^2)
p=plot(f.subs(mu=my_mu,sigma=1.0), (x, lower, upper), rgbcolor = (0,0,1))
for i in range(2, stop_i, 1): # just do a few of them
    shade = 1-11/i # make them different colours
    p+=plot(f.subs(mu=my_mu,sigma=1/i), (x, lower, upper), rgbcolor = (1-shade, 0,
shade))
textOffset = -0.2 # offset for placement of text -  may need adjusting
p+=text("0",(0,textOffset),fontsize = 10, rgbcolor='grey')
p+=text(str(upper.n(digits=2)),(upper,textOffset),fontsize = 10, rgbcolor='grey')
p+=text(str(lower.n(digits=2)),(lower,textOffset),fontsize = 10, rgbcolor='grey')
p.show(axes=false, gridlines=[None,[0]], figsize=[7,3])
```

```
N(0,1) to N(0, 1/12)
```



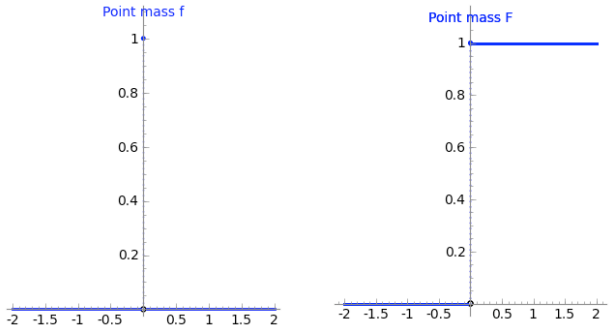
There is an interesting point to note about this convergence:

We have said that the $X_i \sim Normal(0, \frac{1}{i})$ with distribution functions F_i converge in distribution to $X \sim Point\,Mass(0)$ with distribution function F, which means that we must be able to show that for any real number t at which F is continuous,

$$\lim_{i \to \infty} F_i(t) = F(t)$$

Note that for any of the $X_i \sim Normal(0, \frac{1}{i})$, $F_i(0) = \frac{1}{2}$, and also note that for $X \sim Point, Mass(0)$, $F(0) = 1$, so clearly $F_i(0) \neq F(0)$. What has gone wrong? Nothing: we said that we had to be able to show that $\lim_{i \to \infty} F_i(t) = F(t)$ for any $t \in \mathbb{R}$ at which F is **continuous**, but the $Point\,Mass(0)$ distribution function F is not continous at 0!

```
theta = 0.0
show(graphics_array((pmfPointMassPlot(theta),cdfPointMassPlot(theta))),figsize=
[8,4]) # show the plots
```



Convergence in Probability

Let $X_1, X_2, \ldots$ be a sequence of random variables and let X be another random variable. Let F_i denote the distribution function (DF) ofX_i and let F denote the distribution function of X.

Now, **if** for any real number $\varepsilon > 0$,

$$\lim_{i \to \infty} P\left(|X_i - X| > \varepsilon\right) = 0$$

Then we can say that the sequence X_i, $i = 1, 2, \ldots$ **converges to** X **in probability** and write $X_i \xrightarrow{P} X$.

Or, going back again to the probability space 'under the hood' of a random variable, we could look the way the X_i maps each outcome $\omega \in \Omega$, $X_i(\omega)$, which is some point on the real line, and compare this to mapping $X(\omega)$.

Saying that for any $\varepsilon \in \mathbb{R}$, $\lim_{i \to \infty} P\left(|X_i - X| > \varepsilon\right) = 0$ is the equivalent of saying that for any $\varepsilon \in \mathbb{R}$,

$$\lim_{i \to \infty} P\left(\{\omega : |X_i(\omega) - X(\omega)| > \varepsilon\}\right) = 0$$

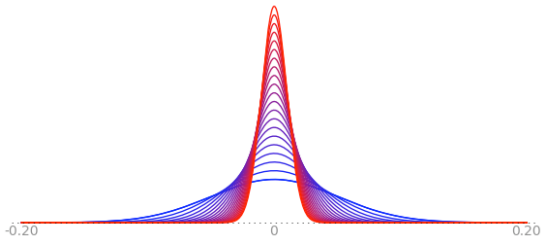
Informally, we are saying X is a limit in probabilty if, by going far enough into the sequence X_i, we can ensure that the mappings $X_i(\omega)$ and $X(\omega)$ will be arbitrarily close to each other on the real line for all $\omega \in \Omega$.

Note that convergence in distribution is implied by convergence in probability: convergence in distribution is the weakest form of convergence; any sequence of RV's that converges in probability to some RV X also converges in distribution to X (but not necessarily vice versa).

```
# mock up a picture of a sequence of converging normal distributions
my_mu = 0
var('mu sigma')
upper = 0.2; lower = -upper
i = 20 # start part way into the sequence
lim = 100 # how far to go
stop_i = 12
html('<h4>N(0,1/'+str(i)+') to N(0, 1/'+str(lim)+')</h4>')
f = (1/(sigma*sqrt(2.0*pi)))*exp(-1.0/(2*sigma^2)*(x - mu)^2)
p=plot(f.subs(mu=my_mu,sigma=1.0/i), (x, lower, upper), rgbcolor = (0,0,1))
for j in range(i, lim+1, 4): # just do a few of them
    shade = 1-(j-i)/(lim-i) # make them different colours
```

```
    p+=plot(f.subs(mu=my_mu,sigma=1/j), (x, lower,upper), rgbcolor = (1-shade, 0,
shade))
textOffset = -1.5 # offset for placement of text -  may need adjusting
p+=text("0",(0,textOffset),fontsize = 10, rgbcolor='grey')
p+=text(str(upper.n(digits=2)),(upper,textOffset),fontsize = 10, rgbcolor='grey')
p+=text(str(lower.n(digits=2)),(lower,textOffset),fontsize = 10, rgbcolor='grey')
p.show(axes=false, gridlines=[None,[0]], figsize=[7,3])
```

```
N(0,1/20) to N(0, 1/100)
```



For our sequence of *Normal* random variables $X_1, X_2, X_3, \ldots$, where

$X_1 \sim Normal(0, 1)$
$X_2 \sim Normal(0, \frac{1}{2})$
$X_3 \sim Normal(0, \frac{1}{3})$
$X_4 \sim Normal(0, \frac{1}{4})$
$\vdots$
$X_i \sim Normal(0, \frac{1}{i})$
$\vdots$

and $X \sim Point\,Mass(0)$,

It can be shown that the X_i **converge in probability** to $X \sim Point\,Mass(0)$ RV X,

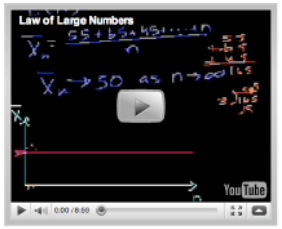
$$X_i \xrightarrow{P} X$$

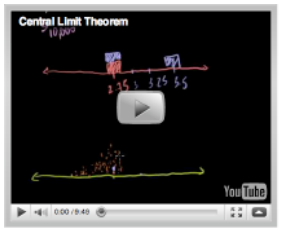
(the formal proof of this involves Markov's Inequality, which is beyond the scope of this course).

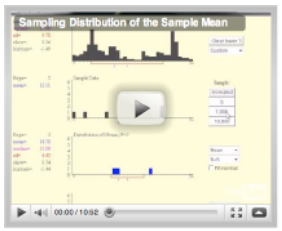
Some Basic Limit Laws in Statistics

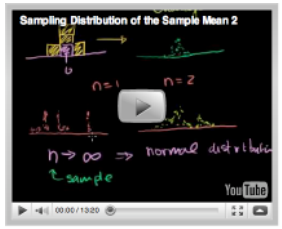
Intuition behind Law of Large Numbers and Central Limit Theorem

Take a look at the Khan academy videos on the Law of Large Numbers and the Central Limit Theorem. This will give you a working idea of these theorems. In the sequel, we will strive for a deeper understanding of these theorems on the basis of the two notions of convergence of sequences of random variables we just saw.









Weak Law of Large Numbers

Remember that a statistic is a random variable, so a sample mean is a random variable. If we are given a sequence of independent and identically distributed RVs, $X_1, X_2, \ldots \overset{IID}{\sim} X_1$, then we can also think of a sequence of random variables $\overline{X}_1, \overline{X}_2, \ldots, \overline{X}_n, \ldots$ (n being the sample size).

Since $X_1, X_2, \ldots$ are IID, they all have the same expection, say $E(X_1)$ by convention.

If $E(X_1)$ exists, then the sample mean $\overline{X}_n$ converges in probability to $E(X_1)$ (i.e., to the expectatation of any one of the individual RVs):

$$\text{If} \quad X_1, X_2, \ldots \overset{IID}{\sim} X_1 \text{ and if } E(X_1) \text{ exists, then } \overline{X}_n \xrightarrow{P} E(X_1)\,.$$

Going back to our definition of convergence in probability, we see that this means that for any real number $\varepsilon > 0$,
$\lim_{n\to\infty} P\left(|\overline{X}_n - E(X_1)| > \varepsilon\right) = 0$

Informally, this means that means that, by taking larger and larger samples we can make the probability that the average of the observations is more than ε away from the expected value get smaller and smaller.

Proof of this is beyond the scope of this course, but we have already seen it in action when we looked at the *Bernoulli* running means. Have another look, this time with only one sequence of running means. You can increase n, the sample size, and change θ. Note that the seed for the random number generator is also under your control. This means that you can get **replicable samples**: in particular, in this interact, when you increase the sample size it looks as though you are just adding more to an existing sample rather than starting from scratch with a new one.

```
@interact
def
_(nToGen=slider(1,1500,1,100,label='n'),my_theta=input_box(0.3,label='theta'),rSeed=input
seed')):
    '''Interactive function to plot running mean for a Bernoulli with specified n,
theta and random number seed.'''

    if my_theta >= 0 and my_theta <= 1:
        html('<h4>Bernoulli('+str(my_theta.n(digits=2))+')</h4>')
        xvalues = range(1, nToGen+1,1)
        bRunningMeans = bernoulliRunningMeans(nToGen, myTheta=my_theta, mySeed=rSeed)
        pts = zip(xvalues, bRunningMeans)
        p = line(pts, rgbcolor = (0,0,1))
        p+=line([(0,my_theta),(nToGen,my_theta)],linestyle=':',rgbcolor='grey')
        show(p, figsize=[5,3], axes_labels=['n','sample mean'],ymax=1)
    else:
        print 'Theta must be between 0 and 1'
```

Central Limit Theorem

You have probably all heard of the Central Limit Theorem before, but now we can relate it to our definition of **convergence in distribution**.

Let $X_1, X_2, \ldots \overset{IID}{\sim} X_1$ and suppose $E(X_1)$ and $V(X_1)$ both exist,

then

$$\overline{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i \xrightarrow{d} X \sim Normal\left(E(X_1), \frac{V(X_1)}{n}\right)$$

And remember $Z \sim Normal(0,1)$?

Consider $Z_n := \dfrac{\overline{X}_n - E(\overline{X}_n)}{\sqrt{V(\overline{X}_n)}} = \dfrac{\sqrt{n}\left(\overline{X}_n - E(X_1)\right)}{\sqrt{V(X_1)}}$

If $\overline{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i \xrightarrow{d} X \sim Normal\left(E(X_1), \frac{V(X_1)}{n}\right)$, then $\overline{X}_n - E(X_1) \xrightarrow{d} X - E(X_1) \sim Normal\left(0, \frac{V(X_1)}{n}\right)$

and $\sqrt{n}\left(\overline{X}_n - E(X_1)\right) \xrightarrow{d} \sqrt{n}\left(X - E(X_1)\right) \sim Normal\left(0, V(X_1)\right)$

so $Z_n := \dfrac{\overline{X}_n - E(\overline{X}_n)}{\sqrt{V(\overline{X}_n)}} = \dfrac{\sqrt{n}\left(\overline{X}_n - E(X_1)\right)}{\sqrt{V(X_1)}} \xrightarrow{d} Z \sim Normal\left(0,1\right)$

Thus, for sufficiently large n (say $n > 30$), probability statements about $\overline{X}_n$ can be approximated using the *Normal* distribution.

The beauty of the CLT, as you have probably seen from other courses, is that $\overline{X}_n \xrightarrow{d} Normal\left(E(X_1), \frac{V(X_1)}{n}\right)$ does not require the X_i to be normally distributed.

We can try this with our *Bernoulli* RV generator. First, a small number of samples:

```
theta, n, samples = 0.6, 10, 5 # concise way to set some variable values
sampleMeans=[] # empty list
for i in range(0, samples, 1):  # loop
    thisMean = QQ(sum(bernoulliSample(n, theta)))/n # get a sample and find the mean
    sampleMeans.append(thisMean) # add mean to the list of means
sampleMeans    # disclose the sample means
```

```
[3/5, 7/10, 2/5, 7/10, 4/5]
```

You can use the interactive plot to increase the number of samples and make a histogram of the sample means. According to the CLT, for lots of reasonably-sized samples we should get a nice symmetric bell-curve-ish histogram centred on θ. You can adjust the number of bins in the histogram as well as the number of samples, sample size, and θ.

```
import pylab
@interact
def _(samples=slider(1,3000,1,100,label='number of samples'),
nToGen=slider(1,1500,1,100,label='sample size
n'),my_theta=input_box(0.3,label='theta'),Bins=5):
    '''Interactive function to plot distribution of sample means for a Bernoulli
process.'''

    if my_theta >= 0 and my_theta <= 1 and samples > 0:
        sampleMeans=[] # empty list
        for i in range(0, samples, 1):

            thisMean = RR(sum(bernoulliSample(nToGen, my_theta)))/nToGen
            sampleMeans.append(thisMean)
        pylab.clf() # clear current figure
        n, bins, patches = pylab.hist(sampleMeans, Bins, normed=true)
        pylab.ylabel('normalised count')
        pylab.title('Normalised histogram for Bernoulli samples')
        pylab.savefig('myHist') # to actually display the figure
        pylab.show()
```

```
        #show(p, figsize=[5,3], axes_labels=['n','sample mean'],ymax=1)
    else:
        print 'Theta must be between 0 and 1, and samples > 0'
```

But although the X_i do not have to be $\sim Normal$ for $\overline{X}_n \xrightarrow{d} X \sim Normal\left(E(X_1), \frac{V(X_1)}{n}\right)$, remember that we said "Let $X_1, X_2, \ldots \overset{IID}{\sim} X_1$ and suppose $E(X_1)$ and $V(X_1)$ both exist", then,

$$\overline{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i \xrightarrow{d} X \sim Normal\left(E(X_1), \frac{V(X_1)}{n}\right)$$

This is where is all goes horribly wrong for the standard *Cauchy* distribution (any *Cauchy* distribution in fact): neither the expectation nor the variance exist for this distribution. The Central Limit Theorem cannot be applied here. In fact, if $X_1, X_2, \ldots \overset{IID}{\sim}$ standard *Cauchy*, then $\overline{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i \sim$ standard *Cauchy*.

You try

Try looking at samples from two other RVs where the expectation and variance do exist, the *Uniform* and the *Exponential*:

```
import pylab
@interact
def _(samples=input_box(100,label='number of samples'),
nToGen=slider(1,1500,1,100,label='sample size
n'),my_theta1=input_box(2,label='theta1'),my_theta2=input_box(4,label='theta1'),Bins=5):
    '''Interactive function to plot distribution of sample means for a
Uniform(theta1, theta2) process.'''

    if (my_theta1 < my_theta2) and samples > 0:
        sampleMeans=[] # empty list
        for i in range(0, samples, 1):

            thisMean = RR(sum(uniformSample(nToGen, my_theta1, my_theta2)))/nToGen
            sampleMeans.append(thisMean)
        pylab.clf() # clear current figure
        n, bins, patches = pylab.hist(sampleMeans, Bins, normed=true)
        pylab.ylabel('normalised count')
        pylab.title('Normalised histogram for Uniform samples')
        pylab.savefig('myHist') # to actually display the figure
        pylab.show()
        #show(p, figsize=[5,3], axes_labels=['n','sample mean'],ymax=1)
    else:
        print 'theta1 must be less than theta2, and samples > 0'
```

```
import pylab
@interact
def _(samples=input_box(100,label='number of samples'),
nToGen=slider(1,1500,1,100,label='sample size
n'),my_lambda=input_box(2,label='lambda'),Bins=5):
    '''Interactive function to plot distribution of sample means for an
Exponential(lambda) process.'''

    if my_lambda > 0 and samples > 0:
        sampleMeans=[] # empty list
        for i in range(0, samples, 1):

            thisMean = RR(sum(exponentialSample(nToGen, my_lambda)))/nToGen
            sampleMeans.append(thisMean)
        pylab.clf() # clear current figure
        n, bins, patches = pylab.hist(sampleMeans, Bins, normed=true)
        pylab.ylabel('normalised count')
        pylab.title('Normalised histogram for Exponential samples')
        pylab.savefig('myHist') # to actually display the figure
```

```
        pylab.show()
        #show(p, figsize=[5,3], axes_labels=['n','sample mean'],ymax=1)
    else:
        print 'lambda must be greater than 0, and samples > 0'
```

More **You try**

`random` for sampling and sequence manipulation

The python `random` module, available in Sage, provides a useful way of taking samples if you have already generated a 'population' to sample from, or otherwise playing around with the elements in a sequence. See http://docs.python.org/library/random.html for more details. Here we will try a few of them.

The aptly-named `sample` function allows us to take a sample of a specified size from a sequence. We will use a list as our sequence:

```
pop = range(1, 101, 1) # make a population
sample(pop, 10) # sample 10 elements from it at random
```

Each call to `sample` will select unique elements in the list (not that 'unique' here means that it will not select the element at any particular position in the list more than once, but if there are duplicate elements in the list, such as with a list [1,2,4,2,5,3,1,3], then you may well get any of the repeated elements in your sample more than once). `sample` samples with replacement, which means that repeated calls to `sample` may give you samples with the same elements in.

```
popWithDuplicates = range(1, 11, 1)*4 # make a population with repeated elements
popWithDuplicates
```

```
for i in range (5):
    print sample(popWithDuplicates, 10)
```

Try experimenting with `choice`, which allows you to select one element at random from a sequence, and `shuffle`, which shuffles the sequence in place (i.e, the ordering of the sequence itself is changed rather than you being given a re-ordered copy of the list). It is probably easiest to use lists for your sequences. See how `shuffle` is creating permutations of the list. You could use `sample` and `shuffle` to emulate permuations of k objects out of n ...

You may need to check the documentation to see how use these functions.

```
help(sample)
```
docs-0.html

```
help(shuffle)
```
docs-0.html

```
help(choice)
```
docs-0.html

Strictly optional

Animations in Sage

These cells show how the animations were made. If you want to use them, you should uncomment the final two lines in each animation-making cell. The cells with the `gif` function command will also need uncommenting if you want to do that part of it. Note that there will a delay of the order of 30-60 seconds while the plots are made and a similar delay for the gif. The final two lines are commented out in the published worksheet so that you have to read this warning about delays before you evaluate the cells!

Making animations is definitely not part of this course, but if you are interested you can find out more on the Sage reference manual page on animations and in the Sage wiki animate section.

```
v = [] # an empty list to store our animation in
animDelay = 40 # delay between frames for animation
n = 2 # starting value for n
inc = 1 # starting value to increment n by
while n < 1000: # a while loop - you are not expected to know about these
    v.append(plotBernoulliLikelihoodSecretTheta(n)) # add a plot (made by a hidden
function) to the list of plots
    n = n + inc                 # increment n
    if inc < 30: inc = inc+2    # increment the increment if it is less than 30
#animLik = animate(v, xmin=0, xmax=1.1, ymin=0) # animate
#animLik.show(delay = animDelay, iterations = 1)
```

```
#animLik.gif(delay=40) #Animation to gif
```

```
v = [] # an empty list to store our animation in
animDelay = 50 # delay between frames for animation
i = 1 # starting value for i
my_mu = 0
my_sigma = 1/i
stop_sigma = 0.04
upper = my_mu + 3*my_sigma + 1; lower = -upper
var('mu sigma')
f = (1/(sigma*sqrt(2.0*pi)))*exp(-1.0/(2*sigma^2)*(x - mu)^2)
y_max = f.subs(x=0,mu=0,sigma=stop_sigma)
while my_sigma > stop_sigma: # a while loop
    p1=plot(f.subs(mu=my_mu,sigma=my_sigma), (x, lower, upper))
    textOffset = -0.25 # offset for placement of text -  may need adjusting
    p1+=text("0",(0,textOffset),fontsize = 10, rgbcolor='grey')
    p1+=text(str(upper),(upper,textOffset),fontsize = 10, rgbcolor='grey')
    p1+=text(str(lower),(lower,textOffset),fontsize = 10, rgbcolor='grey')
    v.append(p1) # add a plot to the list of plots
    i = i +1 # increment i
    my_sigma = 1/i #recalculate sigma
#animNormal = animate(v, xmin=lower, xmax=upper, ymax=y_max,fontsize=0,axes=false,
gridlines=[[0],[0]], axes_labels=('x','f(x)')) # animate
#animNormal.show(delay = animDelay, iterations = 1)
```

```
#animNormal.gif(delay=50) #animation to gif
```

```

```

```

```

```

```

```
%hide
%auto
def likelihoodBernoulli(theta, n, tStatistic):
    '''Bernoulli likelihood function.
    theta in [0,1] is the theta to evaluate the likelihood at.
    n is the number of observations.
    tStatistic is the sum of the n Bernoulli observations.
    return a value for the likelihood of theta given the n observations and
tStatistic.'''
    retValue = 0 # default return value
    if (theta >= 0 and theta <= 1): # check on theta
        mpfrTheta = RR(theta) # make sure we use a Sage mpfr
        retValue = (mpfrTheta^tStatistic)*(1-mpfrTheta)^(n-tStatistic)
    return retValue
```

```
%hide
%auto

def bernoulliFInverse(u, theta):
    '''A function to evaluate the inverse CDF of a bernoulli.

    Param u is the value to evaluate the inverse CDF at.
    Param theta is the distribution parameters.
    Returns inverse CDF under theta evaluated at u'''

    return floor(u + theta)

def bernoulliSample(n, theta, simSeed=None):
    '''A function to simulate samples from a bernoulli distribution.

    Param n is the number of samples to simulate.
    Param theta is the bernoulli distribution parameter.
    Param simSeed is a seed for the random number generator, defaulting to 30.
    Returns a simulated Bernoulli sample as a list.'''

    set_random_seed(simSeed)
    us = [random() for i in range(n)]
    set_random_seed(None)
    return [bernoulliFInverse(u, theta) for u in us] # use bernoulliFInverse in a
list comprehension

def bernoulliSampleSecretTheta(n, theta=0.30, simSeed=30):
    '''A function to simulate samples from a bernoulli distribution.

    Param n is the number of samples to simulate.
    Param theta is the bernoulli distribution parameter.
    Param simSeed is a seed for the random number generator, defaulting to 30.
    Returns a simulated Bernoulli sample as a list.'''

    set_random_seed(simSeed)
    us = [random() for i in range(n)]
    set_random_seed(None)
    return [bernoulliFInverse(u, theta) for u in us] # use bernoulliFInverse in a
list comprehension
```

```
%hide
%auto
```

```
def bernoulliRunningMeans(n, myTheta, mySeed = None):
    '''Function to give a list of n running means from bernoulli with specified
theta.

    Param n is the number of running means to generate.
    Param myTheta is the theta for the Bernoulli distribution
    Param mySeed is a value for the seed of the random number generator, defaulting
to None.'''

    sample = bernoulliSample(n, theta=myTheta, simSeed = mySeed)
    from pylab import cumsum # we can import in the middle of code
    csSample = list(cumsum(sample))
    samplesizes = range(1, n+1,1)
    return [RR(csSample[i])/samplesizes[i] for i in range(n)]
```

```
%hide
%auto
#return a plot object for BernoulliLikelihood using the secret theta bernoulli
generator
def plotBernoulliLikelihoodSecretTheta(n):
    '''Return a plot object for BernoulliLikelihood using the secret theta bernoulli
generator.

    Param n is the number of simulated samples to generate and do likelihood plot
for.'''

    thisBSample = bernoulliSampleSecretTheta(n) # make sample
    tn = sum(thisBSample) # summary statistic
    from pylab import arange
    ths = arange(0,1,0.01) # get some values to plot against
    liks = [likelihoodBernoulli(t,n,tn) for t in ths] # use the likelihood function
to generate likelihoods
    redshade = 1*n/1000 # fancy colours
    blueshade = 1 - redshade
    return line(zip(ths, liks), rgbcolor = (redshade, 0, blueshade))
```

```
%hide
%auto
def cauchyFInverse(u):
    '''A function to evaluate the inverse CDF of a standard Cauchy distribution.

    Param u is the value to evaluate the inverse CDF at.'''

    return RR(tan(pi*(u-0.5)))

def cauchySample(n):
    '''A function to simulate samples from a standard Cauchy distribution.

    Param n is the number of samples to simulate.'''

    us = [random() for i in range(n)]
    return [cauchyFInverse(u) for u in us]
```

```
%hide
%auto
def cauchyRunningMeans(n):
    '''Function to give a list of n running means from standardCauchy.

    Param n is the number of running means to generate.'''

    sample = cauchySample(n)
    from pylab import cumsum
    csSample = list(cumsum(sample))
    samplesizes = range(1, n+1,1)
    return [RR(csSample[i])/samplesizes[i] for i in range(n)]
```

```
%hide
%auto
def twoRunningMeansPlot(nToPlot, iters):
    '''Function to return a graphics array containing plots of running means for
Bernoulli and Standard Cauchy.

    Param nToPlot is the number of running means to simulate for each iteration.
    Param iters is the number of iterations or sequences of running means or lines
on each plot to draw.
    Returns a graphics array object containing both plots with titles.'''
    xvalues = range(1, nToPlot+1,1)
    for i in range(iters):
        shade = 0.5*(iters - 1 - i)/iters # to get different colours for the lines
        bRunningMeans = bernoulliSecretThetaRunningMeans(nToPlot)
        cRunningMeans = cauchyRunningMeans(nToPlot)
        bPts = zip(xvalues, bRunningMeans)
        cPts = zip(xvalues, cRunningMeans)
        if (i < 1):
            p1 = line(bPts, rgbcolor = (shade, 0, 1))
            p2 = line(cPts, rgbcolor = (1-shade, 0, shade))
            cauchyTitleMax = max(cRunningMeans) # for placement of cauchy title
        else:
            p1 += line(bPts, rgbcolor = (shade, 0, 1))
            p2 += line(cPts, rgbcolor = (1-shade, 0, shade))
            if max(cRunningMeans) > cauchyTitleMax: cauchyTitleMax =
max(cRunningMeans)
    titleText1 = "Bernoulli running means" # make title text
    t1 = text(titleText1, (nToGenerate/2,1), rgbcolor='blue',fontsize=10)
    titleText2 = "Standard Cauchy running means" # make title text
    t2 = text(titleText2, (nToGenerate/2,ceil(cauchyTitleMax)+1),
rgbcolor='red',fontsize=10)
    return graphics_array((p1+t1,p2+t2))
```

```
%hide
%auto
def pmfPointMassPlot(theta):
    '''Returns a pmf plot for a point mass function with parameter theta.'''

    ptsize = 10
    linethick = 2
    fudgefactor = 0.07 # to fudge the bottom line drawing
    pmf = points((theta,1), rgbcolor="blue", pointsize=ptsize)
    pmf += line([(theta,0),(theta,1)], rgbcolor="blue", linestyle=':')
    pmf += points((theta,0), rgbcolor = "white", faceted = true, pointsize=ptsize)
    pmf += line([(min(theta-2,-2),0),(theta-0.05,0)],
rgbcolor="blue",thickness=linethick)
    pmf += line([(theta+.05,0),(theta+2,0)], rgbcolor="blue",thickness=linethick)
    pmf+= text("Point mass f", (theta,1.1), rgbcolor='blue',fontsize=10)
    pmf.axes_color('grey')
    return pmf

def cdfPointMassPlot(theta):
    '''Returns a cdf plot for a point mass function with parameter theta.'''

    ptsize = 10
    linethick = 2
    fudgefactor = 0.07 # to fudge the bottom line drawing
    cdf = line([(min(theta-2,-2),0),(theta-0.05,0)],
rgbcolor="blue",thickness=linethick) # padding
    cdf += points((theta,1), rgbcolor="blue", pointsize=ptsize)
    cdf += line([(theta,0),(theta,1)], rgbcolor="blue", linestyle=':')
    cdf += line([(theta,1),(theta+2,1)], rgbcolor="blue", thickness=linethick) #
padding
```

```
    cdf += points((theta,0), rgbcolor = "white", faceted = true, pointsize=ptsize)
    cdf+= text("Point mass F", (theta,1.1), rgbcolor='blue',fontsize=10)
    cdf.axes_color('grey')
    return cdf
```

```
%hide
%auto
def uniformFInverse(u, theta1, theta2):
    '''A function to evaluate the inverse CDF of a uniform(theta1, theta2)
distribution.

    u, u should be 0 <= u <= 1, is the value to evaluate the inverse CDF at.
    theta1, theta2, theta2 > theta1, are the uniform distribution parameters.'''

    return theta1 + (theta2 - theta1)*u

def uniformSample(n, theta1, theta2):
    '''A function to simulate samples from a uniform distribution.

    n > 0 is the number of samples to simulate.
    theta1, theta2 (theta2 > theta1) are the uniform distribution parameters.'''

    us = [random() for i in range(n)]

    return [uniformFInverse(u, theta1, theta2) for u in us]
```

```
%hide
%auto
def exponentialFInverse(u, lam):
    '''A function to evaluate the inverse CDF of a exponential distribution.

    u is the value to evaluate the inverse CDF at.
    lam is the exponential distribution parameter.'''

    # log without a base is the natural logarithm
    return (-1.0/lam)*log(1 - u)

def exponentialSample(n, lam):
    '''A function to simulate samples from an exponential distribution.

    n is the number of samples to simulate.
    lam is the exponential distribution parameter.'''

    us = [random() for i in range(n)]

    return [exponentialFInverse(u, lam) for u in us]
```

```

```