Estimation and Likelihood, Maximum Likelihood Estimators, Symbolic Expressions in Sage

Monte Carlo Methods

Estimation and Likelihood

Likelihood is one of the most fundamental concepts in **statistical inference**. You may have already met some of its applications in other statistics courses, if not there is no sweat! We will see the ideas from scratch. The following three so-called decision problems are our main attractions in this likelihood tour.

- **Point estimation**. A 'single best guess' at some quantity of interest.
- **Set estimation**. Guess a set that traps some quantity of interest with a high probability, for example, a confidence interval.
- **Hypothesis testing**. Attempt to reject a falsifiable null hypothesis, i.e., make scientific progress through Popper's falsifiability, an important concept in science and the philosophy of science.

Likelihood also comes into regression, classification, risk minimisation ...

There are two types of estimation. In this worksheet we are looking at **parametric estimation**. The other kind of estimation is non-parametric estimation.

Parameters, Models and Real Life

jsMath

What do we mean by parametric estimation? In parametric estimation, we assume that the data comes from a particular type of probability distribution and we try to estimate the *parameters* of that distribution. What are *parameters*, in a statistical sense? Remember the $Bernoulli(\theta)$ random variable? A $Bernoulli$ distribution has one parameter, usually denoted as θ. We talked about modelling events such as the outcome of a toss of a coin as using the $Bernoulli(\theta)$ random variable. If the coin is fair then, in our model, $\theta = \frac{1}{2}$. If the random variable X takes value 1 when the fair coin lands heads, then we model $P(X = 1) = \frac{1}{2}$.

When we speak about the probability of observing events such as the outcome of a toss of a coin, we are assuming some kind of *model*. In the case of a coin, the model is a $Bernoulli$ RV. This model would have one *parameter*, the probability of the coin landing heads.

When we introduced the $Exponential$ distribution, we talked about the $Exponential(\lambda)$ RV, or the $Exponential$ parameterised by λ. Distributions can be parameterised by more than one quantity. We have already met the $Uniform(\theta_1, \theta_2)$ - this has two parameters, θ_1 and θ_2. Another distribution you may be familiar with, although we have not discussed it in this course, is the Normal distribution which is parameterised by μ and σ. The symbols like θ, λ, μ, σ are conventionally used for the parameters of these distributions. It is useful to become familiar with these conventions (see for example Wikipedia on the Exponential or Normal)

There are many applications of computational statistics which involve models of real life events, and it is not enough to say "this can be modeled with a $Bernoulli$ RV", or "Orbiter bus inter-arrival times can be modelled with an $Exponential$ RV". We also have to choose parameters for our models.

We also remind ourselves that the probabilty density function (probability mass function for a discrete random variable) and distribution function depend on the parameters when we write them. In the case of the $Bernoulli$, for example, the probability mass function is denoted by $f(x; \theta)$.

In real life, we are usually not trying to do "textbook" examples like calculating the probability of an event given a distribution and parameter value(s). We may be able to see the outcomes of a process, but we can never know exactly what that process is, all we can do is try to find a useful model for it. We are trying to use the information available to us in the form of observations or data to make our models, including guessing/estimating values for the model parameters. We have to turn our thinking around and focus on what the data can tell us about the model. In particular, in this course, we focus on what the data can tell us about model parameters - parametric estimation. Now is a good time to reflect on these words of the renowned Statisticians:

All models are wrong, but some are useful --- **George Edward Pelham Box**

The only math I did not use is the one I did not know -- **Lucien Le Cam**

The Likelihood Function

Likelihood, as we said above, is a fundamental concept in statistical inference ("inference" - making inferences from observations, making guesses based on information in data).

In informal terms, likelihood is "the likelihood of the parameters given the data". We can talk about a **likelihood function** where the domain of the likelihood function is all the possible values for the parameters (remember, a function is a mapping from a *domain* to a *range*): the likelihood function is a mapping from possible values for the parameters to the likelihood of those parameters given the data.

The **likelihood function** of θ based on n observations $x_1, x_2, \ldots, x_n$ is denoted $L_n(\theta)$. We have said that it is a mapping from "all possible values for the parameters", i.e. all possible values for θ, to the likelihood of those parameters given the data $x_1, x_2, \ldots, x_n$. In our formal notation, if we know that θ must be somewhere in some parameter space $\mathbf{\Theta}$, then $L_n(\theta)$ is a mapping from $\mathbf{\Theta}$ to the real numbers $\mathbb{R}$:

$$L_n(\theta) : \mathbf{\Theta} \rightarrow \mathbb{R}$$

For example, in the case of a $Bernoulli(\theta)$ RV, we know that the parameter θ must be between 0 and 1, or $\theta \in [0, 1]$. In the case of an $Exponential(\lambda)$ random variable parameterised by λ, we know $\lambda > 0$, i.e., jsMath

$\lambda \in (0, \infty)$

We will focus on the likelihood function for independent and identically distributed (IID) random variables.

Suppose we have $X_1, X_2, \ldots, X_n$ as n independent random variables and they are all identically distributed with $f(x;\theta)$. We would write this as $X_1, X_2, \ldots, X_n \overset{IID}{\sim} f(x;\theta)$. Then,

$$X_1, X_2, \ldots, X_n \overset{IID}{\sim} f(x;\theta)$$

implies that

$$L_n(\theta) := L_n(x_1, x_2, \ldots, x_n; \theta) = f(x_1, x_2, \ldots, x_n; \theta) = f(x_1;\theta) f(x_2;\theta) \ldots f(x_n;\theta) := \prod_{i=1}^{n} f(x_i;\theta)$$

$f(x_1, x_2, \ldots, x_n; \theta)$ is termed the joint density of $X_1, X_2, \ldots, X_n$ given θ

When $X_1, X_2, \ldots, X_n \overset{IID}{\sim} f(x;\theta)$ the joint density $f(x_1, x_2, \ldots, x_n; \theta)$ is the product of the individual densities $\prod_{i=1}^{n} f(x_i;\theta)$.

So when $X_1, X_2, \ldots, X_n \overset{IID}{\sim} f(x;\theta)$,

$$L_n(\theta) := \prod_{i=1}^{n} f(x_i;\theta)$$

The likelihood Function for the $Bernoulli(\theta)$ RV

We can make all this theory a little more real by considering the $Bernoulli$ RV. In the last worksheet, we wrote function to be able to simulate samples from a $Bernoulli(\theta)$ RV given some value for the parameter θ and the number of samples required. Suppose we used this function, with a small adaptation, to simulate some samples now using θ^* - but there is a catch: you don't know what the value of θ^* is.

```
n = 20
bSample = bernoulliSampleSecretTheta(n)
bSample
```

What we have is $X_1, X_2, \ldots, X_n \overset{IID}{\sim} Bernoulli(\theta^*)$ where $\theta^* \in [0,1]$ is the true, unknown value of the parameter θ responsible for producing all those observations.

Recall that the $Bernoulli(\theta)$ RV X has probability mass function (PMF), for $x \in \{0,1\}$, $f(x;\theta)$:

jsMath

$$f(x;\theta) = \theta^x(1-\theta)^{1-x} = \begin{cases} \theta & \text{if } x = 1, \\ 1-\theta & \text{if } x = 0, \\ 0 & \text{otherwise} \end{cases}$$

So, for $x_1, x_2, \ldots, x_n \in \{0, 1\}$, the joint density of n IID $Bernoulli(\theta)$ RVs is:

$$\begin{aligned} f(x_1, x_2, \ldots, x_n; \theta) := \prod_{i=1}^{n} f(x_i; \theta) &= \prod_{i=1}^{n} \theta^{x_i}(1-\theta)^{1-x_i} \\ &= \theta^{\sum_{i=1}^{n} x_i}(1-\theta)^{(n-\sum_{i=1}^{n} x_i)} \end{aligned}$$

$\sum_{i=1}^{n} x_i$ is a bit of a mouthful, so lets summarise this as $t_n = \sum_{i=1}^{n} x_i$

We can use t_n to make our likelihood function a little more user-friendly:

$$L_n(\theta) = \theta^{t_n}(1-\theta)^{(n-t_n)}$$

What we have actually done is to define a statistic of the data. Remember that a statistic is a function of the data. We will call our statistic (note the big T) T_n. The n subscript reminds us that it is a function of n observations.

T_n is a function of the data, a mapping from the data space $\mathbf{X}$ to the space $\mathbf{T}_n$:

$$T_n(X_1, \ldots, X_n) = \sum_{i=1}^{n} X_i : \mathbf{X} \rightarrow \mathbf{T}_n$$

(If you are wondering what the space $\mathbf{T}_n$ is for the $Bernoulli$, think about the range of possible values of $\sum_{i=1}^{n} X_i$ when each X_i can only be 0 or 1.)

We have some actual observations $x_1, \ldots, x_n$ so we have a **realisation** of our statistic $T_n(x_1, \ldots, x_n) = t_n = \sum_{i=1}^{n} x_i$

We can easily use Sage to calculate t_n for us, using the `sum` function. For example, for the small sample of 20 simulated Bernoulli observations above:

```
tn = sum(bSample)
tn
```

We can also write ourselves a Sage function to calculate the likelihood of a specified value of θ given n and t_n.

jsMath

```
%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
```

(We use `RR(theta)` above to make sure that we use a souped-up Sage multi-precision floating-point real (mpfr) number type in our calculation, which will improve the precision of the calculation of the likelihoods.)

You try in class

You should be able to understand what the `likelihoodBernoulli` function is doing and be able to write this kind of Sage function for yourselves. Why do we need to check that the value for `theta` passed to the function is between 0 and 1? How does the function deal with a situation where it is asked to evaluate a likelihood for `theta` < 0 or `theta` > 1?

(end of You Try)

Let's look at a very simple situation where we have one observation ($n = 1$) and it is a 0. What is the realisation of T_1, t_1?

```
bernoulliSample0 = [0]
tn = sum(bernoulliSample0)
tn
```

You try in class

Try going back to the $Bernoulli$ likelihood function $L_n(\theta) = \theta^{t_n}(1-\theta)^{(n-t_n)}$ to calculate the likelihood of $\theta = 0$ **without using** the Sage function. Think about what the likelihood function is doing.

When you have done that, check that you get the same answer using our `likelihoodBernoulli` Sage function:

jsMath

```
tryTheta0 = 0 # a value of theta to find the likelihood for
n = len(bernoulliSample0) # find n as the length of the sample list
tn = sum(bernoulliSample0) # find tn as the sum of the samples
# calculate the likelihood of theta=tryTheta0=0
likelihoodBernoulli(tryTheta0, n, tn)
```

What about $\theta = 1$? What is the likelihood of $\theta = 1$ when we have observed 0? Think back to what the θ parameter means in a $Bernoulli$ distribution:

The $Bernoulli(\theta)$ RV X has probability mass function (PMF), for $x \in \{0, 1\}$, $f(x; \theta)$:

$$f(x;\theta) = \theta^x(1-\theta)^{1-x} = \begin{cases} \theta & \text{if } x = 1, \\ 1-\theta & \text{if } x = 0, \\ 0 & \text{otherwise} \end{cases}$$

Remember that the idea behind the likelihood function is "what is the likelihood of a parameter value given our data?"

When you have worked out the answer using $L_n(\theta)$, check using our `likelihoodBernoulli` Sage function:

```
tryTheta1 = 1 # a value of theta to find the likelihood for
n = len(bernoulliSample0) # find n as the length of the sample list
tn = sum(bernoulliSample0) # find tn as the sum of the samples
# calculate the likelihood of theta=tryTheta0=0
likelihoodBernoulli(tryTheta1, n, tn)
```

What about $\theta = \frac{1}{2}$?

```
tryTheta1 = 0.5 # a value of theta to find the likelihood for
n = len(bernoulliSample0) # find n as the length of the sample list
tn = sum(bernoulliSample0) # find tn as the sum of the samples
# calculate the likelihood of theta=tryTheta0=0
likelihoodBernoulli(tryTheta1, n, tn)
```

Try to sketch the likelihood function over $\theta \in [0, 1]$ for our one observation of 0.

Now, what if instead of observing a 0 in our one-observation case, we had observed a 1?

What is our realisation of T_n now? What is our intuition about the likelihood of $\theta = 0$? $\theta = 1$?

Again try to sketch the likelihood function for our single observation of 1.

We could use `likelihoodBernoulli` and a for loop to calculate the likelihood for some different values of θ without repeating code:

jsMath

```
bernoulliSample1 = [1]
n = len(bernoulliSample1) # find n as the length of the sample list
tn = sum(bernoulliSample1) # find tn as the sum of the samples
from pylab import arange # import arange from pylab
for t in arange(0, 1.1, 0.2):
    # calculate the likelihood of theta=tryTheta0=0
    print "If we observe", bernoulliSample1, "The likelihood of theta=", t,
" is ", likelihoodBernoulli(t, n, tn)
```

Or, we could use a list comprehension to get a list of likelihoods corresponding to the list of possible values of θ:

```
ths = arange(0, 1.1, 0.2)
[likelihoodBernoulli(t,len(bernoulliSample1),sum(bernoulliSample1)) for t in
ths]
```

If you have forgotten about the arange function, it is a useful way of getting a sequence of values in steps. We have used the statement "from pylab import arange" because we don't want the whole of the pylab libary, just arange. Check the documentation if you need a reminder about `arange`.

```
help(arange)
```

docs-0.html

You will see that the arange function can take values for `start, stop`, and `step`. You will also see that (like `range`), the list you get will go to just *below* the value you specifiy for `stop`. To get a list of values from 0 to 1 in steps of 0.2 our `stop` value was 1.1. 1.2 would also have worked, but a `stop` value of 1.3 or 1.4 would have given us a list from 0 to 1.2 in steps of 0.2.

Try some different values in `arange` if you want to check how it works again.

```
from pylab import arange
arange(0, 1.1, 0.2)
```

(end of You Try)

Now, we look at a possible sample of $n = 2$ observations from a Bernoulli process with unknown θ:

```
smallBSample = [0,1]
```

What is the realisation of the T_n statistic?

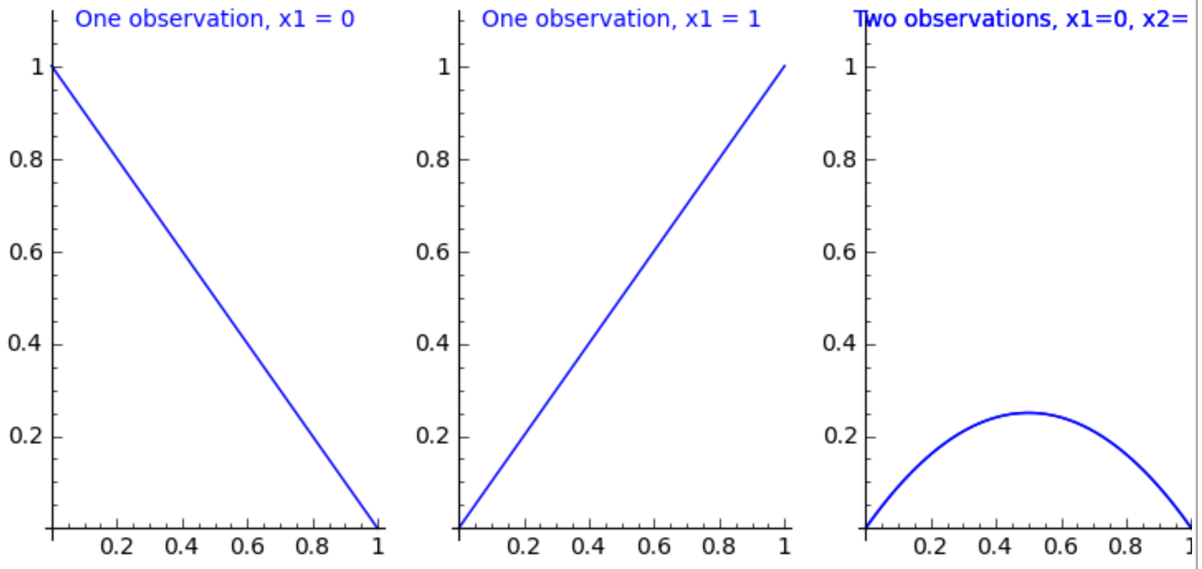
```
tn = sum(smallBSample) # what is tn
tn
```

jsMath

Use $L_n(\theta) = \theta^{t_n}(1-\theta)^{(n-t_n)}$ to think about the likelihood of some possible values for θ given this data. Think what the shape of the likelihood function might be.

In the visualisation below we have used our `likelihoodBernoulli` function to plot the likelihood function for the cases where we have a single observation 0, a single observation 1, and a small sample 0, 1.

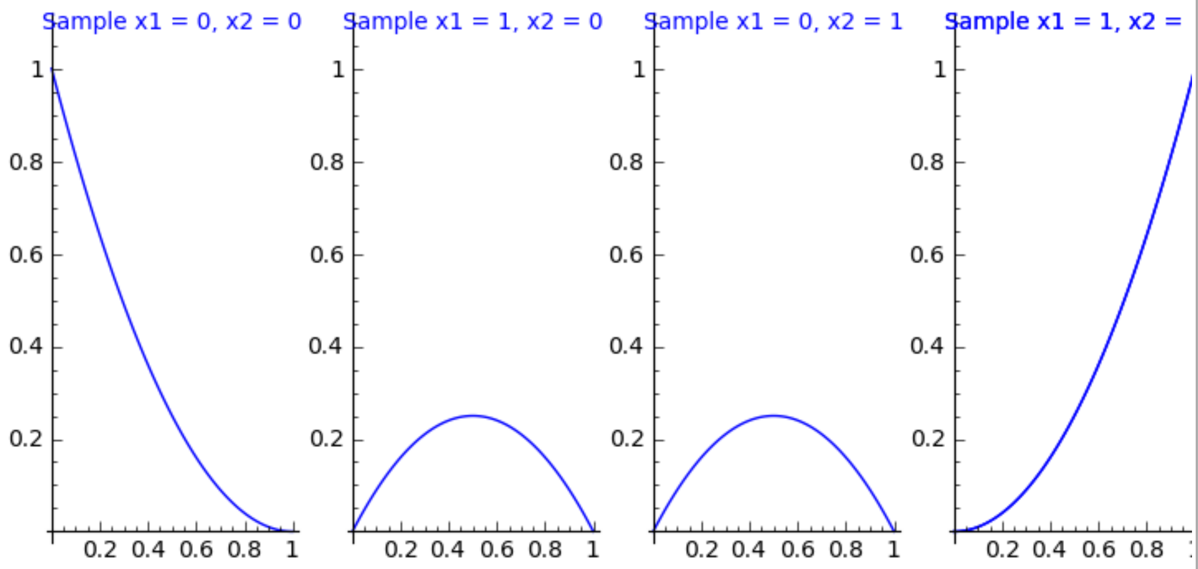
```
bernoulliSample0 = [0] # make sure we know about our samples
bernoulliSample1 = [1]
smallBSample = [0,1]
from pylab import arange
ths = arange(0,1.01,0.01) # get some values to plot against
p1 = line(zip(ths, [likelihoodBernoulli(t,
len(bernoulliSample0),sum(bernoulliSample0)) for t in ths]))
t1 = text("One observation, x1 = 0", (0.5,1.1), fontsize=10)
p1 = p1+t1
p2 = line(zip(ths, [likelihoodBernoulli(t,
len(bernoulliSample1),sum(bernoulliSample1)) for t in ths]))
t2 = text("One observation, x1 = 1", (0.5,1.1), fontsize=10)
p2 = p2+t2
p3 = line(zip(ths, [likelihoodBernoulli(t,
len(smallBSample),sum(smallBSample)) for t in ths]))
t3 = text("Two observations, x1=0, x2=1", (0.5,1.1), fontsize=10)
```



There are other samples we could get if we draw a sample of size $n = 2$ from a $Bernoulli$ RV. In the visualisation below we plot the likelihood functions for the four unique possible samples:

jsMath

```
smallBSample1 = [0,0]
smallBSample2 = [1,0]
smallBSample3 = [0,1]
smallBSample4 = [1,1]
listOfSamples = [smallBSample1, smallBSample2, smallBSample3, smallBSample4]
# a list of lists
from pylab import arange
ths = arange(0,1.01,0.01) # get some values to plot against
l_plots = [] # an empty list of plots
for sample in listOfSamples: # go through the list of samples one by one
    ptitle = text("Sample x1 = " + str(sample[0]) + ", x2 = " +
str(sample[1]), (0.5,1.1), fontsize=10)
    l_plots.append(line(zip(ths, [likelihoodBernoulli(t,
```



We can see that the shape of the likekihood function depends on the sample: we are looking at the likelihood for the parameter **given the actual data**.

You do not have to know how to do these plots or the interactive plot below, but you should understand what a list comprehension statement like

```
[likelihoodBernoulli(t,len(smallBSample),sum(smallBSample)) for t in ths]
```

is doing, when `ths = arange(0, 1.01, 0.01)` are some possible values for $\theta \in [0, 1]$

What happens as we increase the sample size n? In the interactive plot below, we use our `bernoulliSampleSecretTheta` to simulate samples of size n with an unknown θ. You can have a look at the effect of increasing n:

jsMath

```
@interact
def _(n=(2,(2..1000))):
    '''Interactive function to plot the bernoulli likelihood for different
n.'''
    if n > 0:
        thisBSample = bernoulliSampleSecretTheta(n) # make sample
        n = len(thisBSample) # what is n
        tn = sum(thisBSample)
        print "Likelihood function for n = ", n , " and tn = ", tn
        from pylab import arange
        ths = arange(0,1,0.01) # get some values to plot against
        p = line(zip(ths, [likelihoodBernoulli(t, n, tn) for t in ths]))
        show(p)
    else:
```

evaluate

Remember that for each n you try, the shape of the likelihood function will depend on the t_n for the sample simulated by `bernoulliSampleSecretTheta`.

The Log-likelihood Function

Working with products, as in $L_n(\theta) = \prod_{i=1}^{n} f(x_i;\theta)$ for $X_1, X_2, \ldots, X_n \overset{IID}{\sim} f(x_i;\theta)$ (i.e., n independent and identically distributed random variables), can be inconvenient. Taking logs can be useful here. The log-likelihood function for some parameter θ is $l_n(\theta)$ and it is literally the log of the likelihood function $L_n(\theta)$:

$$l_n(\theta) := log(L_n(\theta))$$

You will probably be aware from other courses that $\log(a \times b) = \log(a) + \log(b)$.

In Sage, using the `log` function without specifying a base gives the natural logarithm (logarithm to base *e*) of a value.

```
a = 5.0
b = 6.0
log(a*b)
```

```
log(a) + log(b)
```

```
help(log)
```

    docs-0.html

The Sage `log` function provides a default parameter value of `None` for the base, and if the base is `None` the natural log is calculated. If you specify a value for the base when you use the `log` function, you will get the logarithm using this base (default parameter values and `None` were discussed in the last worksheet: go back that in your own time if you need to).

jsMath

```
log(a, 10) # log to base 10
```

We can generalise this into a useful trick: "the log of products is the sum of logs"

$$\log\left(\prod_{j=1}^{n} y_j\right) = \sum_{j=1}^{n}\left(\log(y_j)\right)$$

So, if we have n IID (independent and identically distributed) random variables, the log-likelihood function is

$$X_1, X_2, \ldots, X_n \overset{IID}{\sim} f(x;\theta) \Longrightarrow l_n(\theta) = \log(L_n(\theta)) := \log\left(\prod_{i=1}^{n} f(x_i;\theta)\right) = \sum_{i=1}^{n} \log(f(x_i;\theta))$$

In the case of $X_1, X_2, \ldots, X_n \overset{IID}{\sim} Bernoulli(\theta)$, n independent and identically distributed $Bernoulli$ RVs,

$$l_n(\theta) = \log(L_n(\theta)) = \log\left(\theta^{t_n}(1-\theta)^{(n-t_n)}\right) = t_n \log(\theta) + (n-t_n)\log(1-\theta)$$

Here we are using the fact that $\log(a^c) = c\log(a)$.

If this is not familiar to you, consider $\log(a^2) = \log(a \times a) = \log(a) + \log(a) = 2\log(a)$

and then think what happens if you have $\log(a^3) = \log(a \times a^2) = \log(a) + \ log(a^2) = \log(a) + 2log(a) = 3\log(a)$, etc etc.

You try in class

Write down for yourself the steps to prove that the log-likelihood function $l_n(\theta) = t_n\log(\theta) + (n-t_n)\log(1-\theta)$ for n IID $Bernoulli$ RVs.

(end of You Try)

Log is a **monotone** function (also known as a monotonic function). What does it mean when we say that something is a monotone function? In words, it means that the function preserves the given order. For once, putting something as a formula may make it easier to understand than the words. If f is some monotone function and we have two values a and b in the domain of f (values that f can be applied to) such that $a \leq b$, then $f(a) \leq f(b)$.

So, if $a \leq b$, $\log(a) \leq \log(b)$: log preserves order. If we calculatete the likelihood L_n for two different possible values of θ, say θ_a and θ_b and find that $L_n(\theta_a) \leq L_n(\theta_b)$, then we know that $l_n(\theta_a) = \log(L_n(\theta_a)) \leq l_n(\theta_b) = \log(L_n(\theta_b))$.

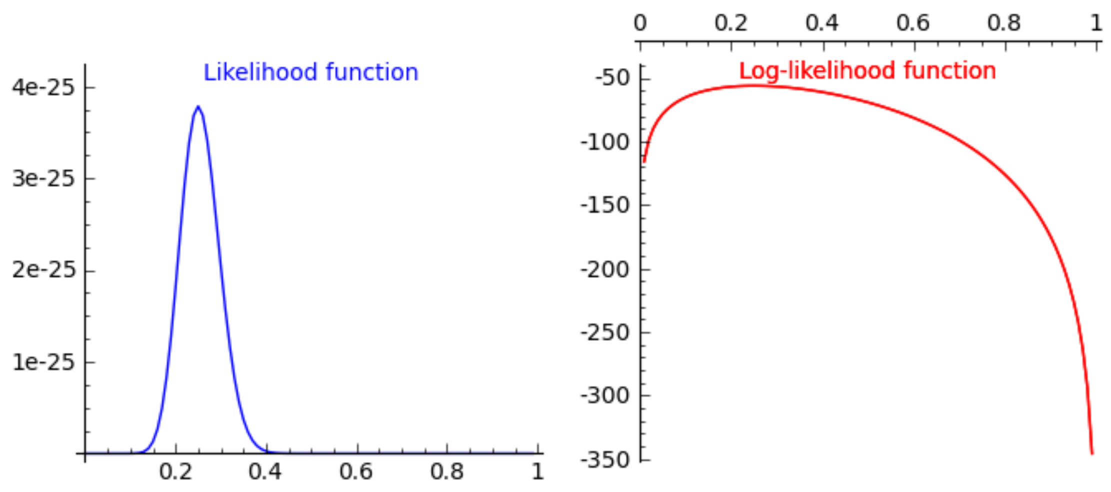
We can see this if we adapt our interactive plot for the $Bernoulli$ likelihood function:

jsMath

```
@interact
def _(n=(2,(2..1000))):
    '''Interactive function to plot the bernoulli likelihood for different
n.'''
    if n > 0:
        thisBSample = bernoulliSampleSecretTheta(n) # make sample
        n = len(thisBSample) # what is n
        tn = sum(thisBSample)
        print "n = ", n , " and tn = ", tn
        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]
        p1 = line(zip(ths, liks))
        p1 += text("Likelihood function", (0.5, max(liks)*1.1))
        thsForLog = arange(0.01,1,0.01) # fudge to avoid log(0) get some
values to plot log against
        logliks = [log(likelihoodBernoulli(t,n,tn)) for t in thsForLog]
        p2 = line(zip(thsForLog, logliks), rgbcolor="red")
        p2 += text("Log-likelihood function", (0.5, max(logliks)*0.8),
rgbcolor="red")
        show(graphics_array([p1, p2]),figsize=[8,3])
    else:
```

n 2

```
n =  100  and tn =  25
```



You try in class

What has this got to do with computational statistics? Computational statistics is often used in inference. Likelihood, because it is a fundamental concept in inference, is therefore very important for computational statistics. If you evaluate the cell below you'll get the titles of 30 articles in the journal *Computational Statistics and Data Analysis*, all involving likelihood - these 30 articles are only a random selection of over 1000 such articles in that journal alone over the past 10 years!

(If you want to get rid of the print out, just comment out the line "`print eachline`" and re-evaluate the cell).

```
filename = 'LikelihoodArticles.txt'
articlesFile = open(DATA+filename) # open the file - likelihoodArticles is
now a file object
for eachline in articlesFile:    # for loop
    # read and successive lines of the file
    print eachline
```

(end of You Try)

Maximum Likelihood Estimator

So now, we know about the likelihood function, the function that tells us about the likelihood of parameter values given the data, and we know about the log-likelihood function.   How do either of these help us to make an estimate of a parameter value?

How about estimating a parameter with the value that maximises the likelihood function?   This is called the **Maximum Likelihood Estimator** (MLE).

And, because log is a monotone function, we know that if some particular value for the unknown parameter maximises the likelihood function, then it will also maximise the log-likelihood function.

Formally,

Let $X_1,\ldots,X_n \sim f(x_1,\ldots,x_n;\theta^*)$ $X_1,\ldots,X_n$ have joint density $f(x_1,\ldots,x_n;\theta^*)$ where θ^* is the true but possibly unknown parameter value.

The maximum likelihood estimator or MLE $\widehat{\Theta}_n$ of the fixed and possibly unknown parameter true parameter $\theta^* \in \Theta$ is a function that returns the value of θ that maximises the likelihood function.

As we saw when we looked at the different possible unique samples of size $n = 2$ from a Bernoulli RV, the shape of the likelihood function depends on the data. The maximum likelihood estimator, the value of θ which maximises the likelihood function (or log-likelihood function) is clearly a function of data itself.

$$\widehat{\Theta}_n := \widehat{\Theta}_n(X_1,X_2,\ldots,X_n) := \underset{\theta\in\Theta}{argmax}\, L_n(\theta)$$

Equivalently, the maximum likelihood estimator is the value of θ that maximises the log-likelihood function:

$$\widehat{\Theta}_n := \underset{\theta\in\Theta}{argmax}\, l_n(\theta)$$

$\underset{\theta in \Theta}{argmax}\, L_n(\theta)$ is the value of $\theta \in \Theta$ that maximises $L_n(\theta)$. $argmax$ is doing what we try to do by eye when we look at the shape of a likelihood function and try to see which value of θ corresponds to the function's highest point.

How do we find the value which maximises the likelihood function, or log-likelihood function? What do we usually do when we want to find the value of a parameter which maximises a function?   We find the turning point(s) of the function by taking the derivative of the function with respect to the parameter (for maximums, we are looking for turning poir jsMath where the slope of the function changes from positive to negative, which we could check with a second derivative, but let's just concentrate on finding the derivative for the moment).

Consider finding the maximum likelihood estimator for $X_1, X_2, \ldots, X_n \overset{IID}{\sim} Bernoulli(\theta^*)$ (n independent Bernoulli random variables, identically distributed with the same true parameter value θ^*):

We found that the likelihood function $L_n(\theta) = \theta^{t_n}(1-\theta)^{(n-t_n)}$ and the log-likelihood function $l_n(\theta) = t_n \log(\theta) + (n - t_n)\log(1-\theta)$.

It is much easier to work with the log-likelihood function when we are taking the derivative with respect to θ:

$$\begin{aligned} \frac{\partial}{\partial\theta} l_n(\theta) &= \frac{\partial}{\partial\theta}\ t_n \log(\theta) + \frac{\partial}{\partial\theta}\ (n-t_n)\log(1-\theta) \\ &= \frac{t_n}{\theta} - \frac{n-t_n}{1-\theta} \end{aligned}$$

Here, we are using the useful fact that $\frac{\partial \log(\theta)}{\partial\theta} = \frac{1}{\theta}$ (and $\frac{\partial \log(1-\theta)}{\partial\theta} = \frac{-1}{1-\theta}$)

Now, set $\frac{\partial}{\partial\theta} l_n(\theta) = 0$ and solve for θ to obtain the maximum likelihood estimate $\widehat{\theta}_n$:

$$\frac{\partial}{\partial\theta} l_n(\theta)) = 0 \iff \frac{t_n}{\theta} = \frac{n-t_n}{1-\theta} \iff \frac{1-\theta}{\theta} = \frac{n-t_n}{t_n} \iff \frac{1}{\theta} - 1 = \frac{n}{t_n} - 1 \iff \widehat{\theta}_n = \frac{t_n}{n}$$

What was t_n? $t_n = \sum_{i=1}^{n} x_i$, so we can see that $\widehat{\theta}_n = \frac{1}{n}\sum_{i=1}^{n} x_i$

In general, the maximum likelihood estimator as a function of the RVs $X_1, X_2, \ldots, X_n$ is:

$$\widehat{\Theta}_n(X_1, X_2, \ldots, X_n) = \frac{1}{n} T_n(X_1, X_2, \ldots, X_n) = \frac{1}{n}\sum_{i=1}^{n} X_i = \overline{X}_n$$

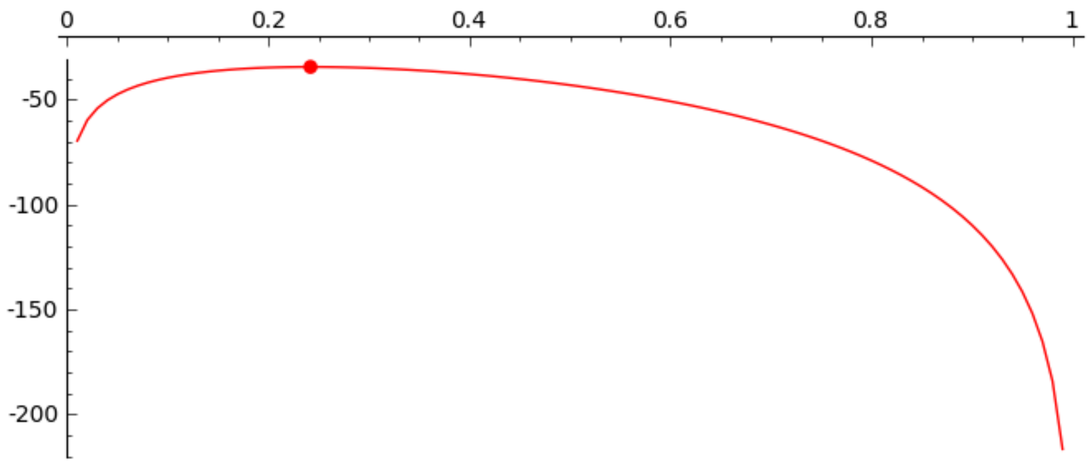
Now, let's look an another version of the interactive plot of the log-likelihood function for a sample of size n from a $Bernoulli$ process with unknown θ^*, but this time we will show the maximum point on the function and the maximum likelihood estimator (MLE):

jsMath

```
@interact
def _(n=(2,(2..2000))):
    '''Interactive function to plot the bernoulli likelihood for different
n.'''
    if n > 0:
        thisBSample = bernoulliSampleSecretTheta(n) # make sample
        n = len(thisBSample) # what is n
        tn = sum(thisBSample)
        from pylab import arange
        thsForLog = arange(0.01,1,0.01) # fudge to avoid log(0) get some
values to plot log against
        logliks = [log(likelihoodBernoulli(t,n,tn)) for t in thsForLog]
        p = line(zip(thsForLog, logliks), rgbcolor="red")
        MLE = RR(tn/n)
        MLEpt = (MLE, log(likelihoodBernoulli(MLE,n,tn)))
        p += points(MLEpt,rgbcolor="red", pointsize=30)
        print "Log-likelihood function for n = ", n , " and tn = ", tn, ":
MLE = ",MLE.n(digits = 3)
        show(p,figsize=[8,3])
    else:
```

n 2

```
Log-likelihood function for n =  62  and tn =  15 : MLE =  0.242
```



What happens as you gradually increase n? What do you think the true value θ^* is?

Example : New Zealand Lotto Data

jsMath

Now, we are going to apply what we have learned about maximum likelihood estimates to the the NZ lotto data we looked at earlier in the course. Specifically, we are interested in whether Ball One is odd or even. This can be considered as a Bernoulli random variable where the outcome is 1 if the number drawn for Ball One is odd, and 0 if the number drawn is even. The observed outcomes of the draws for Ball One are modelled as independently and identically distributed (IID) realisations of the $Bernoulli(\theta^*)$ random variable. Thus our probability model is:

$$X_1, X_2, \ldots, X_n \overset{IID}{\sim} Bernoulli(\theta^*), \text{ where, } \theta^* \in [0,1]$$

We have provided the functions needed for you to access the data, so all you have to do is to evaluate the cell below to get a list of the Ball one data between 1987 and 2008:

```
listBallOneData = getLottoBallOneData()
```

Remember that we can find how many observations we have using the `len` function:

```
len(listBallOneData)
```

Now we can get to what we are really interested in - whether the number drawn is odd or even. You'll recall that we can get a 1 to represent an odd number and a 0 to represent an even number with the modulus operator %.

```
bernoulliBallOneOdd = [x % 2 for x in listBallOneData]
bernoulliBallOneOdd
```

We want to start our investigation of the relative number of odd and even numbers that occur in the draws for Ball one by visualising the outcome data in terms of the proportion of odd numbers that are observed in the Ball One draws. One way to find the number of occurrences of a particular value in a list is to use the list object`count(...)` method:

```
bernoulliBallOneOdd.count(1) # find how many 1s there are in the list
bernoulliBallOneOdd
```

```
help(list.count)
```

docs-0.html

Note also that since our Bernoulli random variable outcomes are 1 or 0, we can also count how many odd numbers are drawn by simply adding up the outcomes: every odd number contributes 1 to the sum and the total is therefore the number of odd numbers drawn. Doing this over all 1114 observations should give us the same value as counting the number of 1s

```
sum(bernoulliBallOneOdd)
```

This is the equivalent of $t_n = \sum_{i=1}^{n}$, the realisation of the statistic $T_n(X_1, X_2, \ldots, X_n) = \sum_{i=1}^{n} X_i$.

jsMath

We saw that the Bernoulli likelihood function is $L_n(\theta) = \theta^{t_n}(1-\theta)^{(n-t_n)}$ and the log-likelihood function is $l_n(\theta) = log(L_n(\theta)) = t_n\, log(\theta) + (n-t_n)\, log(1-\theta)$

With our Bernoulli model, our maximum likelihood estimate $\widehat{\theta}_n$ for the parameter θ, the probability that a ball is odd, is $\frac{t_n}{n}$ which we can see is the same as the proportion of odd-numbered balls in the sample.

Using the sum(...) function makes it very easy for us to explore how, as we look at more and more draws (samples), the proportion of odd-numbered balls settles down.

Remember the pylab function `cumsum` which you can use to calculate the *cumulative sum* of an array or 'array-like object' (i.e. an object that the function can convert into an array, such as a list or tuple)? We can use this to give us the cumulative sum of the number of odd-numbered balls in the sample:

```
csBernoulliBallOneOdd = list(pylab.cumsum(bernoulliBallOneOdd))
csBernoulliBallOneOdd
```

Note that our lotto data and counts are `sage.rings.integer.Integer` types, but the values returned by the `pylab.cumsum` function are `numpy.int64` ('plain integer') types. pylab does not use the fancy Sage value types.

```
type(bernoulliBallOneOdd[0])
```

```
type(csBernoulliBallOneOdd[0])
```

What we want to do is to visualise the changing proportion of odd numbers as we get more and more samples. The proportion of odd numbers is the number of odds (calculated as the sum of the outcomes in the samples we have so far) over the total number of samples so far. To help plotting this we make ourselves a sequence of sample sizes, going up from 1 to 1114:

```
sampleSize = range(1,1115,1)
sampleSize
```

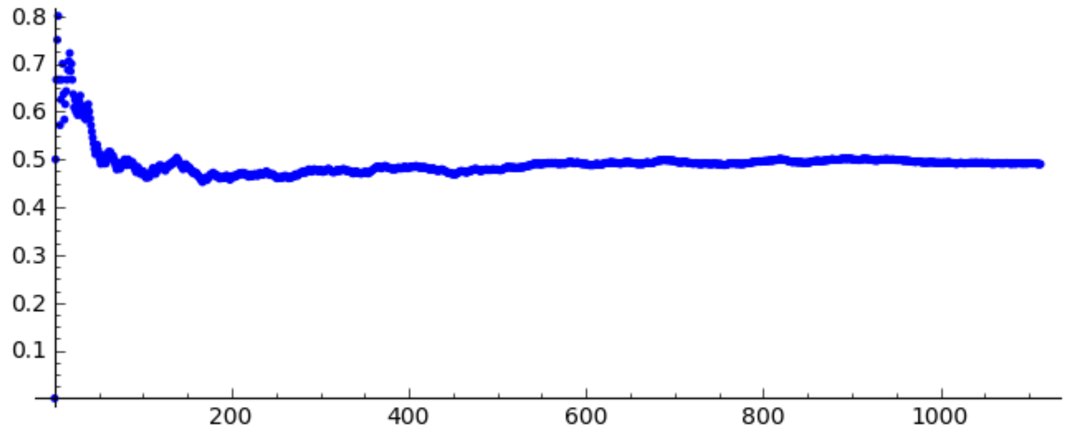
Imagine you are at the end of week one of the draws (8 January 1987). You'll have seen one draw and Ball One was 4, i.e., even. The proportion of odd numbers is from this one draw is 0 (= 0/1). Then week two comes along, there is another draw, the number is 3 (odd) and so the proportion of odds in the 2 samples so far is 1/2. Then week 3 (11, odd), so the proportion is 2/3, etc etc etc. After each draw, we are dividing the cumulative sum of the outcomes by the number of draws to date. If we kept doing this week after week we'd get something like this, ending up with 546 odds out of 1114 observations which simplifies to 273/557 (we are making the results display as rational numbers by converting the plain int types returned by the `pylab.cumsum` function to Sage integers using the `ZZ(...)` function to make the results easier to interpret).

```
[ZZ(csBernoulliBallOneOdd[i])/sampleSize[i] for i in range(1114)]
```

With a slight variation on our list comprehension, we can make a list of points out of this, plotting the proportion of odds on the y-axis against the number of observations on the x-axis:

jsMath

```
show(points([(i, ZZ(csBernoulliBallOneOdd[i])/sampleSize[i]) for i in
range(1114)]), figsize=[8,3])
```



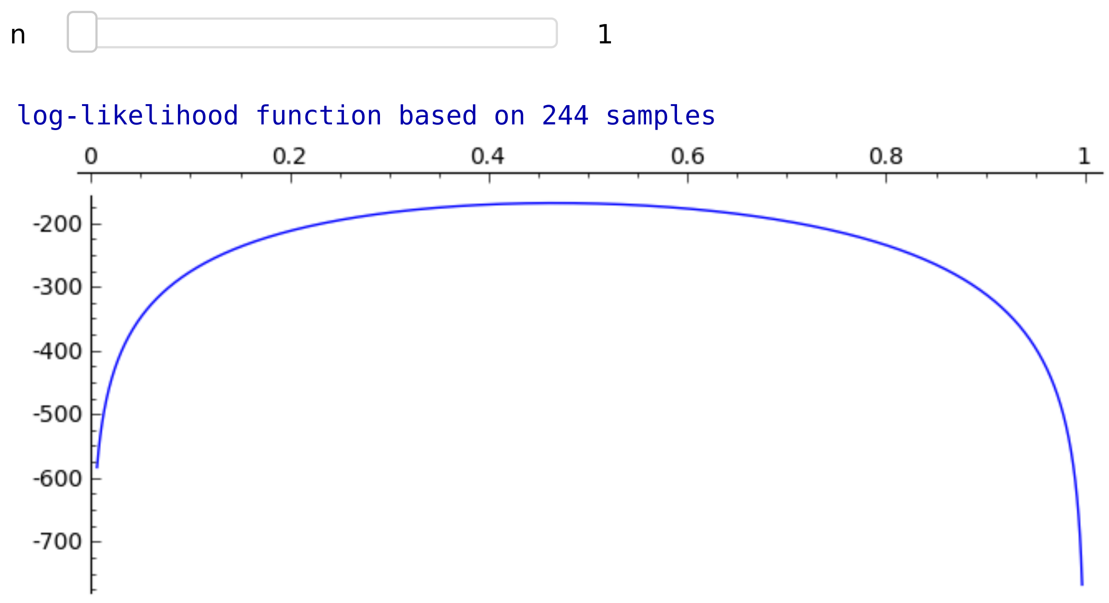
We have effectively plotted the maximum likelihood estimate or MLE $\widehat{\theta}_n$ for θ in our $Bernoulli$ model over increasing values of n.

We can also look at the shape of the whole log-likelihood function, not just the value that maximises it.

This interactive plot draws the log-likelihood function for samples based on for different values of n. Again, for $n = 1$, you have only observed the first draw, for $n = 2$ you have observed two draws, etc etc.

```
@interact
def _(n=(1..1114)):
    '''Interactive function to plot sample-size specific log likelihood
function.'''
    if n == 1:
        print "Log-likelihood function based on first sample"
    else:
        print "log-likelihood function based on", n,  "samples"
    tn = csBernoulliBallOneOdd[n-1]
    theta = var('theta')
    show(plot((tn * log(theta) + (n - tn) * log(1-theta)),
theta,0,1),figsize=[8,3])
```

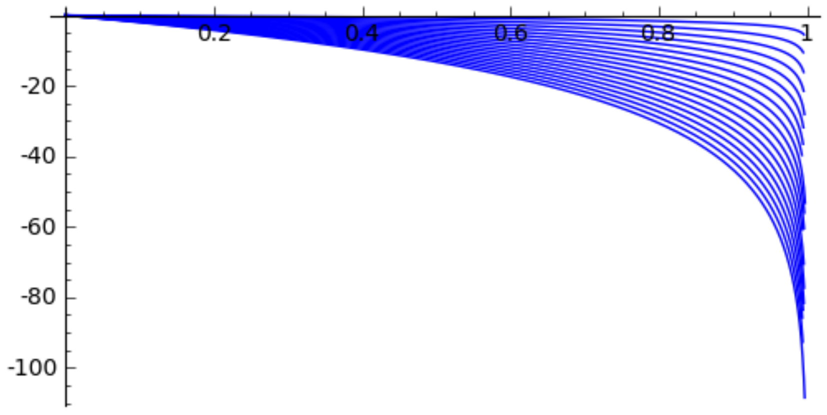
jsMath



Try changing n to see how the shape of the log-likelihood function changes as we get more and more observations.

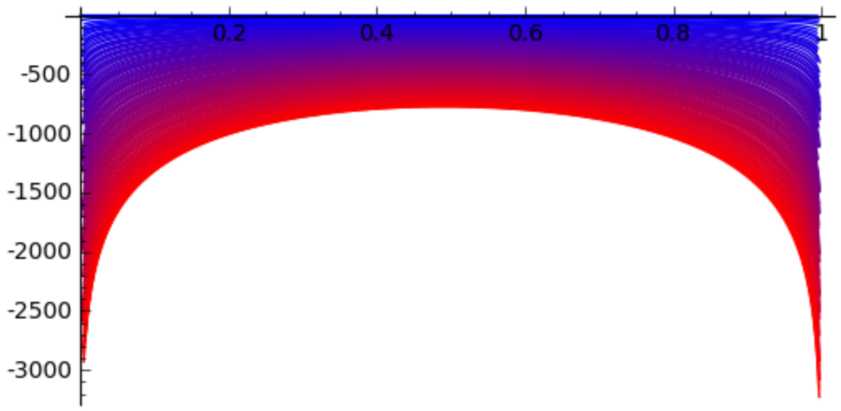
We can also show the log likelihood functions for a number of different value of n all on the same plot. The first cell below shows log-likelihood functions for $n = 1$ to $n = 20$. This is where the log-likelihood moves around most as n changes.

```
theta = var('theta')
n = 1
tn = csBernoulliBallOneOdd[n-1]
p = plot((tn * log(theta) + (n-tn)*log(1-theta)), theta,0,1)
for n in range(2,20,1):
    Tn = csBernoulliBallOneOdd[n-1]
    p += plot((tn * log(theta) + (n-tn)*log(1-theta)), theta,0,1)
show(p, figsize=[6,3])
```



Looking at larger values of n, we can see the shape of the log-likelihood function settling down as n increases. Colour has been used to show the changing values of n which result in each curved line on the plot: bluer shades for smaller n, moving to red as n increases.

```
theta = var('theta')
n=1
Tn = csBernoulliBallOneOdd[n-1]
p = plot((Tn * log(theta) + (n-Tn) * log(1-theta)), theta,0,1, rgbcolor=
(0,0,1))
for n in range(10,1111,10):
    Tn = csBernoulliBallOneOdd[n-1]
    redshade = 1*n/1114.0
    blueshade = 1 - redshade
    p += plot((Tn * log(theta) + (n-Tn) * log(1-theta)), theta,0,1, rgbcolor
= (redshade, 0, blueshade))
n = 1114
p += plot((Tn * log(theta) + (n-Tn) * log(1-theta)), theta,0,1,
```



Using Sage for Basic Algebra and Calculus

When we wanted to differentiate the log-likelihood $l_n(\theta)$ above, we did it for ourselves, but Sage could have helped us to do even that.

Sage can be used to find solutions to equations and for basic calculus. The secret is to create a *symbolic expression* using the `var(...)` function. The details of the symbolic rings used for symbolic expressions in Sage are beyond the scope of this course, but they do offer us some useful features. It is probably easiest to think of var as a way to tell Sage that something is a variable name without having to assign that variable to an actual value. Then, using the function `solve` we can solve equations, i.e. use Sage to find the value(s) of a variable which would solve the equation of interest, or expressions for one variable in terms of other variables involved in the equation(s).

The examples used here are taken from the book Sage Tutorial (version 3.4), The Sage Group.

```
x=1
```

```
type(x)
```

```
x = var('x') # symbolic expression
type(x)
```

jsMath

```
help(var)
```

docs-0.html

```
help(solve)
```

docs-0.html

Let's try a simple example, solving $x^2 + 3x + 2 = 0$ for x.

```
solve(x^2 + 3*x + 2, x)
```

We can also use `solve` if we have variables instead of known values for the coefficients.

```
x, a, b, c = var('x a b c')
solve([a*(x^2) + b*x + c == 0], x)
```

And we can solve a system of equations for several variables.

```
x, y = var('x y')
solve([x+y==6, x-y==4], x, y)
```

Sometimes Sage cannot find an exact solution to the equation, as in the following cell:

```
theta = var('theta')
solve([cos(theta) == sin(theta)], theta)
```

Then we can try using the `find_root` function to find a numerical solution. Note that as well as the equation, you have to pass `find_root(...)` values for the end points of the interval within which you want it to search for the solution. In this case we are searching within the interval $[0, \frac{\pi}{2}]$.

```
find_root(cos (theta) == sin(theta), 0, pi/2)
```

```
help(find_root)
```

docs-0.html

We can also use calculus with our symbolic expressions. We differentiate with the `diff` function or method.

```
u = var('u')
diff(sin(u), u)
```

jsMath

We can also tackle higher derivatives, such as the fourth derivative of $\sin(x^2)$ in the following cell.

```
x = var('x')
diff(sin(x^2), x, 4)
```

```
help(diff)
```

docs-0.html

Partial derivatives can also be found:

```
x, y = var('x y')
f = x^2 + 17*y^2
f.diff(x) # differentiate f with respect to x
```

Let's try a simple example.

```
x, y = var('x y')
f = x^2 + 17*y^2
f.diff(y) # differentiate f with respect to y
```

The `integral(...)` function does integration.

```
x = var('x')
integral(x*sin(x^2), x)
```

Symbolic Expressions for the Maximum Likelihood Estimator using Sage

We can use these Sage capabilities to help us to find maximum likelihood estimators. We will first have to find an expression for the likelihood of the parameter(s) in terms of of some statistic or statistics of the observations. We then take logs to get a log-likelihood function (since logs are usually easier to work with). Then, with the Sage `diff` function and the `solve` function, we have some powerful tools to then help us to differentiate and find the value at which the differential is 0.

Let's start with the Bernoulli log-likelihood function $l_n(\theta) = log(L_n(\theta))$ = $t_n\ log(\theta)$ + $(n - t_n)\ log(1 - \theta)$ and first of all find an expression for the differential of this with respect to θ

```
theta, n, tn = var('theta n tn')
logL = tn*log(theta) + (n-tn)*log(1-theta) # Bernoulli log likelihood
dlogL = logL.diff(theta)
dlogL
```

And then solve for θ when the differential is zero:

jsMath

```
solve([dlogL == 0], theta)
```

Magic! We get the expression for $\widehat{\theta}_n$ that we derived before!

You try in class

Try `diff`, `integral`, and `solve` on some other functions

(end of You Try)

The Maximum Likelihood Principle

Do we always use the MLE? No, not always, but we can follow the same principle of using the "most likely" possible value.

Example: The Most-Likely of Three Coins

Suppose there are three coins in a bag, but they are not all fair coins. Using our $Bernoulli$ model for the probability of getting a head on a single toss:

- Coin 1 has $\theta = \frac{1}{4}$
- Coin 2 has $\theta = \frac{3}{4}$
- Coin 3 has $\theta = \frac{1}{2}$

The coins are otherwise identical - you can't tell by look or feel which is which.

You have to choose a single coin from the bag and guess which one it is. To help your guess, you can toss it three times and observe the results. Then you make your 'single best guess': is it Coin 1, Coin 2, or Coin 3?

The experiment

$X_1, X_2, X_3 \overset{IID}{\sim} Bernoulli(\theta^*)$, with $\theta^* \in \{\frac{1}{4}, \frac{3}{4}, \frac{1}{2}\}$

(i.e., our model is three $Bernoulli$ random variables, independent and identically distributed with true parameter value θ^* which is known to be one of either $\frac{1}{4}$, $\frac{3}{4}$, or $\frac{1}{2}$.)

Suppose the results of your three tosses are $x_1 = 1$, $x_2 = 0$, $x_3 = 0$ (a head, a tail, a tail)

jsMath

According to what we have learned so far, the MLE $\widehat{\theta}_3 = \frac{t_n}{n} = \frac{1+0+0}{3} = \frac{1}{3}$

But now we have a problem: we know that $\theta^* \in \{\frac{1}{4}, \frac{3}{4}, \frac{1}{2}\}$ and $\widehat{\theta}_3 \notin \{\frac{1}{4}, \frac{3}{4}, \frac{1}{2}\}$ ($\notin$ means "is not in").

So, the MLE $\widehat{\theta}_n = \frac{t_n}{n} = \frac{1}{n}\sum_{i=1}^{n} x_i$ is not a good idea if we have a finite set of possible values for θ. Remember that we derived the MLE for the $Bernoulli$ on the continuous parameter space $[0,1]$ ("$\theta^* \in [0,1]$ the true but unknown value ..."), which is rather different to a situation where $\theta^* \in \{\frac{1}{4}, \frac{3}{4}, \frac{1}{2}\}$, a finite set.

In this situation, we take each of the possible values in the set in turn and say "if θ was this value, what is the likelihood":

(Coin 1) $L_3(\frac{1}{4} = f(x_1, x_2, x_3; \theta = \frac{1}{4}) = f(1,0,0; \theta = \frac{1}{4}) = \frac{1}{4} \times (1 - \frac{1}{4}) \times (1 - \frac{1}{4}) = \frac{1}{4} \times \frac{3}{4} \times \frac{3}{4} = \frac{9}{64}$

(Coin 2) $L_3(\frac{3}{4} = f(x_1, x_2, x_3; \theta = \frac{3}{4}) = f(1,0,0; \theta = \frac{3}{4}) = \frac{3}{4} \times (1 - \frac{3}{4}) \times (1 - \frac{3}{4}) = \frac{3}{4} \times \frac{1}{4} \times \frac{1}{4} = \frac{3}{64}$

(Coin 3) $L_3(\frac{1}{2} = f(x_1, x_2, x_3; \theta = \frac{1}{2}) = f(1,0,0; \theta = \frac{1}{2}) = \frac{1}{2} \times (1 - \frac{1}{2}) \times (1 - \frac{1}{2}) = \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} = \frac{1}{8} = \frac{8}{64}$

So, the "most likely" estimate following the three tosses on the basis of the **maximum likelihood principle** is $\widehat{\theta}_3 = \frac{1}{4}$ (Coin 1).

Maximum Likelihood and the $Exponential(\lambda)$ RV

The $Exponential$ is parameterised by λ. We have seen that, for a given $\lambda \in (0, \infty)$, an $Exponential(\lambda)$ random variable has the following PDF f and DF F:

$$f(x;\lambda) = \lambda e^{-\lambda x}$$

$$F(x;\lambda) = 1 - e^{-\lambda x}$$

You try in class

On paper, try to work out $f(x_1, x_2, \ldots, x_n; \lambda)$, the joint density of $X_1, X_2, \ldots, X_n \overset{IID}{\sim} f(x;\lambda)$ remembering that for for $X_1, \ldots, X_n$ IID (independent and identically distributed), the joint density is the product of the individual densities, i.e., $\prod_{i=1}^{n} f(x_i; \lambda)$.

Here is the start:

The joint density of n IID $Exponential(\lambda)$ RVs is:

jsMath

$$f(x_1, x_2, \ldots, x_n; \lambda) := \prod_{i=1}^{n} f(x_i; \lambda) \quad = \quad \prod_{i=1}^{n} \lambda e^{-\lambda x_i}$$
$$= \quad ?$$

Hint: $\prod_{i=1}^{n} \lambda e^{-\lambda x_i} = \left(\prod_{i=1}^{n} \lambda \right) \left(\prod_{i=1}^{n} e^{-\lambda x_i} \right)$

Another hint: $e^a e^b = e^{a+b}$

When you have got the joint density, use it to get the likelihood function $L_n(\lambda) = f(x_1, f_2, \ldots, x_n; \lambda)$ and then show that the log-likelihood function for the $Exponential$ is

$$l_n(\lambda) = \log(L_n(\lambda)) = n \log(\lambda) - \lambda t_n$$

where $t_n = \sum_{i=1}^{n} x_i$

Try differentiating $l_n(\lambda)$ with respect to λ.

Compare what you have got with the answer from Sage using `diff`:

```
lam, n, tn = var('lam n tn')
logL = log(lam^n) - lam*tn # Exponential log likelihood
dlogL = logL.diff(lam)
dlogL
```

(Note that we can't use a variable name `lambda` in Sage: in Sage "lambda" is reserved for a special use.)

Can you show that the maximum likelihood estimator $\widehat{\lambda}_n = \frac{n}{t_n} = \frac{n}{\sum_{i=1}^{n} x_i}$ by solving $\frac{\partial l_n(\lambda)}{\partial \lambda} = 0$?

Compare this to the answer from Sage using `solve`:

```
solve([dlogL == 0], lam)
```

(end of You Try)

jsMath

Example: Inter-earthquake Times for New Zealand Earthquakes

The $Exponential$ is often suitable for modelling the time between occurences of some event, like the time between buses at a bus-stop. We could try to model the inter-earthquake times of New Zealand earthquakes as IID $Exponential(\lambda^*)$ random variables and use our earthquakes data to calculate an MLE for λ^*.

We start by getting the some earthquake data from the file again:

```
myFilename = 'earthquakes_1July2009_19Mar2010.csv'
myData = getData(myFilename,headerlines=1,sep=',')
print "Got earthquakes data"
```

Checking the shape of the array will tell us how many rows it has:

```
myData.shape
```

One of the columns in the array of data is the earthquake time. We have written a function to extract the inter-earthquake times in seconds (i.e., seconds between each earthquake) and give it to you as a list. We can specify the minimum and maximum latitude and longitudes for the earthquakes. You don't need to be able to write such a function yourselves so we have hidden it so that you can concentrate on the data.

```
interQuakesSecs = interQuakeTimes(myData, -50, -30, 150, 200)
```

```
help(interQuakeTimes)
```

docs-0.html

Note that some of the rows in the whole data set were outside our latitude and longitude ranges, so the actual number of inter-earthquake times we have is smaller than the number of rows in the complete data array. We can find how many inter-earthquake times we have from the length of the list:

```
len(interQuakesSecs)
```

```
pylab.mean(interQuakesSecs) # mean inter-earthquake times in seconds
```

```
pylab.mean(interQuakesSecs)/60 # mean inter-earthquake times in minutes
```

Our (very simplistic) model is $X_1, X_2, \ldots, X_n \overset{IID}{\sim} Exponential(\lambda^*)$ with observations $x_1, x_2, \ldots, x_n$ in the list `interQuakesSecs` and n the number of elements in the list.

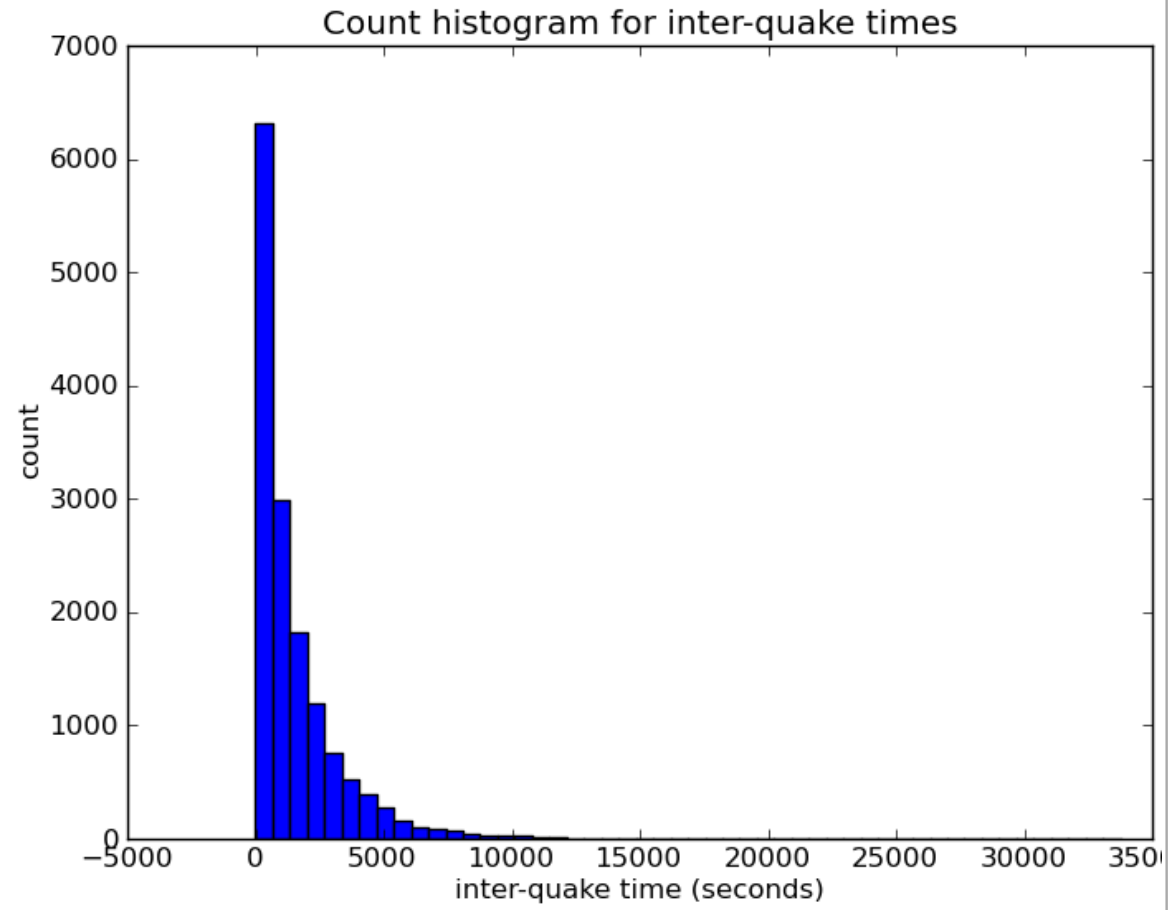
The `sum` function can tell us $t_n = \sum_{i=1}^{n} x_i$

jsMath

```
sum(interQuakesSecs)
```

We can get a quick look at the shape of the data using a histogram:

```
pylab.clf() # clear current figure
n, bins, patches = pylab.hist(interQuakesSecs, 50) # make the histogram
(don't have to have n, bins, patches = ...)
pylab.xlabel('inter-quake time (seconds)') # use pyplot methods to set
labels, titles etc similar to as in matlab
pylab.ylabel('count')
pylab.title('Count histogram for inter-quake times')
pylab.savefig('myHist') # seem to need to have this to be able to actually
display the figure
pylab.show() # and finally show it
```



Our 'best guess' point estimate $\widehat{\lambda}_n$ of $\lambda^* \in (0, \infty)$ is $\frac{n}{t_n}$

jsMath

```
n = len(interQuakesSecs)
tn = sum(interQuakesSecs)
bestGuessSecs = n/tn
bestGuessSecs
```

Can you see how $\widehat{\lambda}_n$ relates to the sample mean? Think about the Expectation of an $Exponential$ RV.

```
pylab.mean(interQuakesSecs) # mean inter-earthquake times in seconds again
```

```
1/bestGuessSecs
```

What about converting our data to give the inter-quake times in minutes rather than seconds:

```
interQuakesMins = [t/60 for t in interQuakesSecs]
```

And getting the best guess for λ^* with the units as minutes:

```
n = len(interQuakesMins)
tn = sum(interQuakesMins)
bestGuessMins = n/tn
bestGuessMins
```

Can you see how $\widehat{\lambda}_n$ relates to the sample mean? Think about the Expectation of an $Exponential$ RV.

```
pylab.mean(interQuakesMins) # mean inter-earthquake times in minutes
```

```
1/bestGuessMins
```

What about the relationship between the best guess values for λ^* when we are measuring time in different units. Does this fit with the idea of λ as a "rate"?

```
bestGuessSecs*60
```

You try

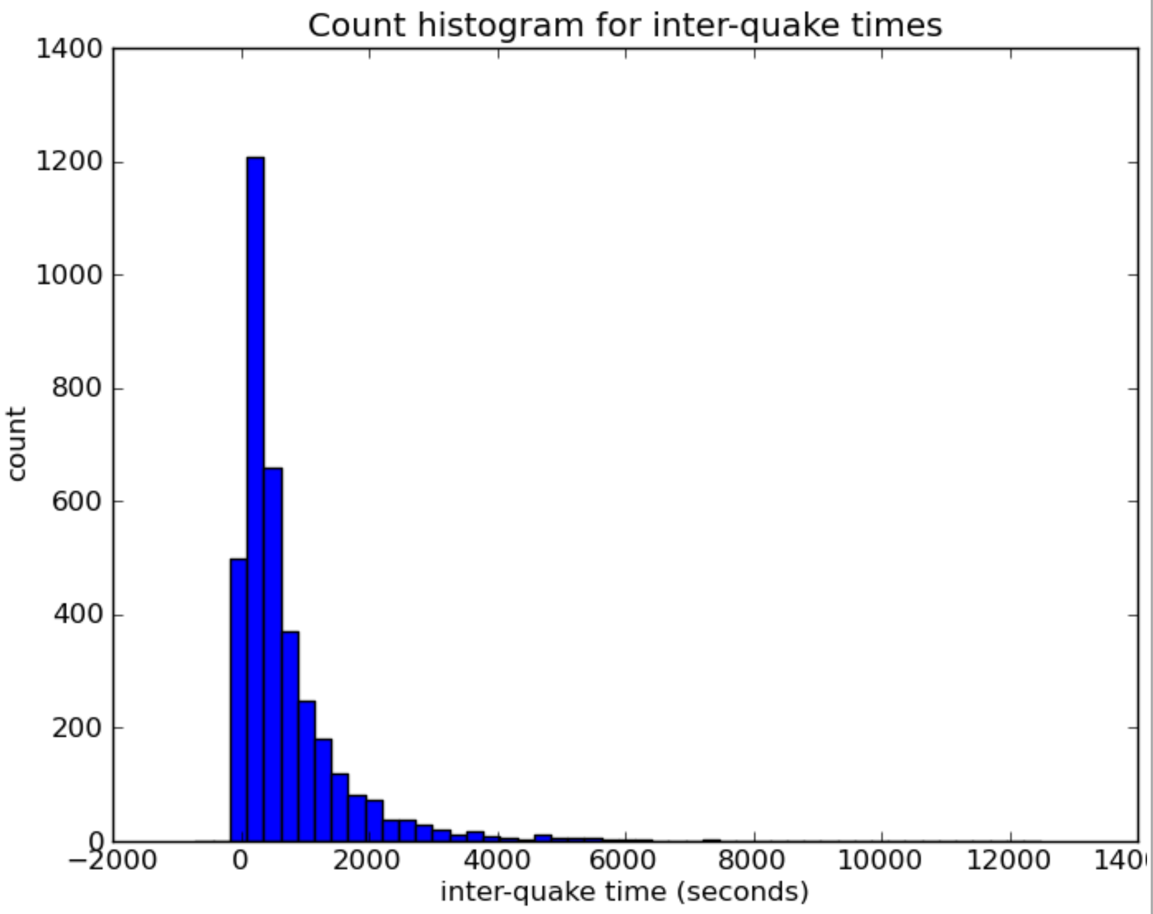
Explore some more recent earthquake data. We have two other files attached to this worksheet: `earthquakes_1Sept2010_30Sept2010.csv` and `earthquakes_1Jan2011_13Apr2011.csv`, or you could get and attach your own data (see Worksheet 5 for how to get and attach earthquake data).

jsMath

```
newFilename = 'earthquakes_1Sept2010_30Sept2010.csv'
newData = getData(newFilename,headerlines=1,sep=',')
newInterQuakesSecs = interQuakeTimes(newData, -50, -30, 150, 200)
```

```
pylab.mean(newInterQuakesSecs) # mean inter-earthquake times in seconds
```

```
pylab.clf() # clear current figure
n, bins, patches = pylab.hist(newInterQuakesSecs, 50) # make the histogram
(don't have to have n, bins, patches = ...)
pylab.xlabel('inter-quake time (seconds)') # use pyplot methods to set
labels, titles etc similar to as in matlab
pylab.ylabel('count')
pylab.title('Count histogram for inter-quake times')
pylab.savefig('myHist') # seem to need to have this to be able to actually
display the figure
pylab.show() # and finally show it
```



jsMath

```
n = len(newInterQuakesSecs)
tn = sum(newInterQuakesSecs)
newBestGuessSecs = n/tn
newBestGuessSecs
```

```
1/newBestGuessSecs
```

(end of You Try)

Optional you try

If you have time and you want to do more with Sage symbolic expressions, have a look at this optional section. You should only start it when you are happy with the essential material above.

Try solving a system of equations:

```
var('x y p q')
eq1 = p + q == 9
eq2 = q*y + p*x == -6
eq3 = q*y^2 + p*x^2 == 24
solve([eq1, eq2, eq3, p==1], p, q, x, y)
```

If you look at the documentation for the `solve(...)` function, you'll see that you have the option of returning a list of dictionaries of the solutions. Think about that: a *list* of *dictionaries* of the solutions. So, there can be more than one combination of variable values that can provide a solution, and a convenient way to indicate the values of each variable within each 'solving combination' is using a dictionary which maps a value to a variable (key).

```
help(solve)
```

docs-0.html

```
solns = solve([eq1, eq2, eq3, p==1], p, q, x, y, solution_dict = true)
solns
```

When we have the list of dictionaries of the solutions, we can have a look at the values. The following cell uses a list comprehension to take each dictionary in the list and find the values mapped to each variable.

```
[[s[p].n(digits=10), s[q].n(digits=10), s[x].n(digits=10),
s[y].n(digits=10)] for s in solns]
```

jsMath

Try some other systems of equations, or other features which catch your eye in the help page.

```
%hide
```

```
%hide
```

```
%hide
```

```
%hide
```

```
%hide
```

```
%hide
```

```
%hide
```

```
%hide
```

jsMath

jsMath