

Computational Statistics with Application to Bioinformatics

Prof. William H. Press
Spring Term, 2008
The University of Texas at Austin

Unit 7: Fitting Models to Data and Estimating Errors in
Model-Derived Quantities

Unit 7: Fitting Models to Data and Estimating Errors in Model-Derived Quantities (Summary)

- “Classic” nonlinear least squares (n.l.s.) fits
 - here, to binned event data
 - but could be (x_i, y_i, s_i) data just as easily
 - fit one and two Gaussians
 - neither has “acceptable” goodness of fit
 - but two Gaussians capture most qualitative features of the data
 - Summarize n.l.s. fits
 - a complete job consists of finding model parameters, their errors (or covariance matrix), and a goodness-of-fit p-value.
 - if you expect the model to be exact, then take the p-value seriously.
- How to find the distribution of a value derived from fitted parameters
 - e.g., ratio of areas of the two Gaussians
 - Method 1: linear propagation of errors using Jacobian and covariance matrix
 - do example using symbolic computation in Matlab
 - Method 2: sample the posterior distribution of the parameters
 - actually, we cheat and use a multivariate Normal distribution for the parameters
 - but this illustrates the concept
 - and is even correct in this case, because the CLT applies
 - Sampling the posterior is more general than linear propagation of errors
 - and will be even better when we learn about MCMC
 - Method 3: bootstrap resampling
 - a very general way of assessing accuracy
 - frequentist (sampling the posterior is Bayesian)
 - applies to almost any analysis “black box”
 - but can be hugely computationally expensive

Let's fit a model to the exon-length data.

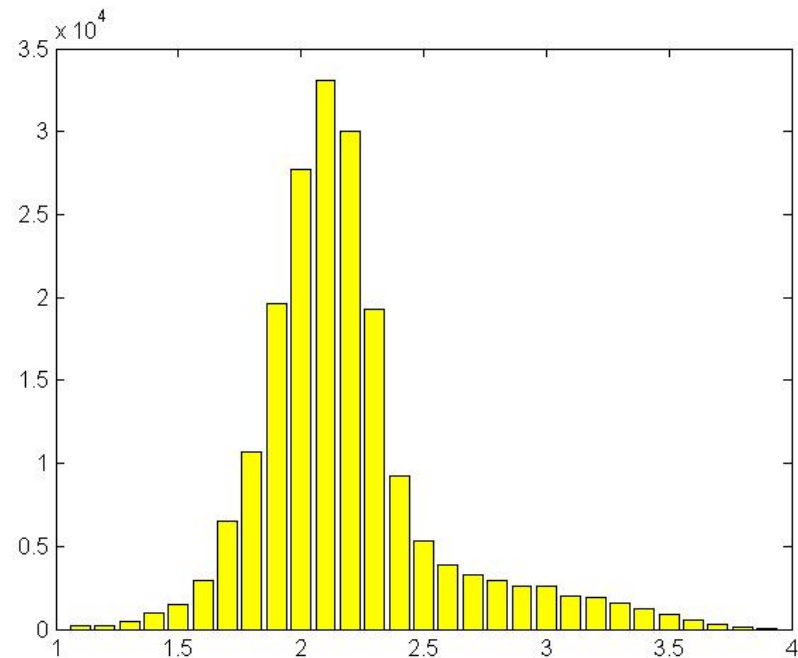
Why?

First, the “classic” approach: bin the data, do weighted nonlinear least squares fit (This then applies to x,y data points in general, not just counts, the example here.)

binning:

```
exloglen = log10(cell2mat(g.exonlen));  
[count cbin] = hist(exloglen, (1:1:4));  
count = count(2:end-1); % trim ends, which have overflow counts  
cbin = cbin(2:end-1);  
ecount = sqrt(count+1);  
bar(cbin, count, 'y')  
hold on;
```

for general x,y data, this would be the standard deviation or “error bar”



Inexplicably, Matlab lacks a weighted nonlinear fit function, so we have to make one out of their unweighted function nlinfit (they have a help page telling how to do this):

```
function [beta r J Covar mse] = nlinfitw(x, y, sig, model, guess)  
yw = y./sig;  
modelw = @(b, x) model(b, x) ./ sig;  
[beta r J Covar mse] = nlinfit(x, yw, modelw, guess);  
Covar = Covar ./ mse; % undo Matlab's well-intentioned scaling
```

more on this later!

Fit a single Gaussian (of course, it's in log space)

```

model oneg = @(b, x) b(1). *exp(-0.5. *((x-b(2)). /b(3)). ^2);
guess = [3.5e4 2.1 .3];
[bfit r J Covar mse] = nlinfitw(cbin, count, ecount, model oneg, guess);
bfit, Covar, mse
stderr = sqrt(diag(Covar))
plot(cbin, model oneg(bfit, cbin), 'b')

```

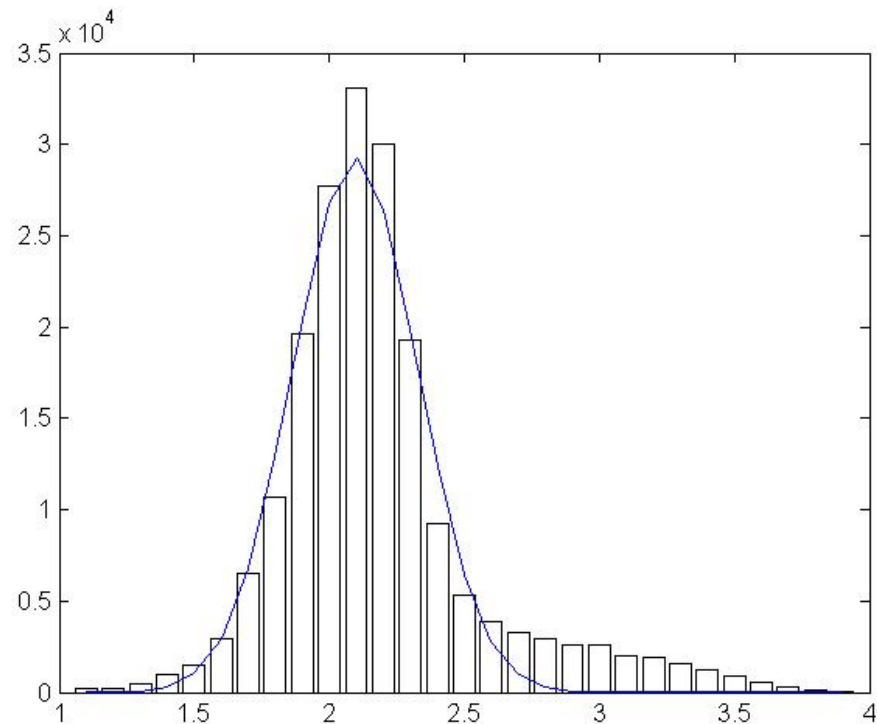
```

bfit =
    29219         2.0966         0.23196
Covar =
    8513    -0.0012396    -0.02769
   -0.0012396    3.1723e-007    9.833e-009
   -0.02769    9.833e-009    2.1986e-007
mse =
    849.37
stderr =
    92.266
    0.00056323
    0.0004689

```

“mean square error”
relative to our scaled
errors, so it should be < 1
for a good fit

$$\text{mse} \approx \frac{\text{pts} - \text{params}}{\text{pts}}$$



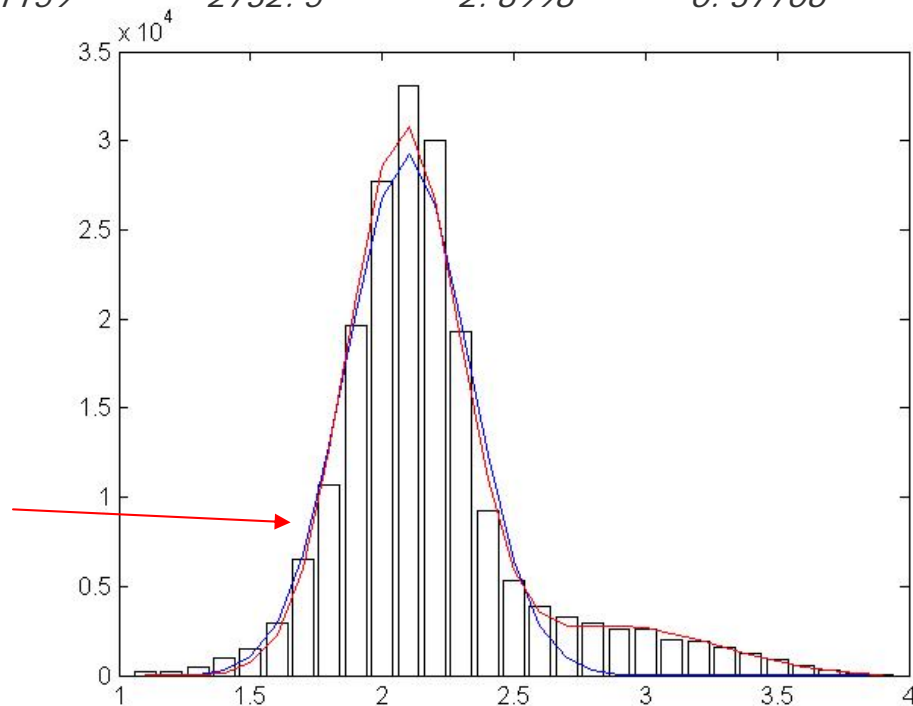
Fit sum of two Gaussians:

This time, we'll put the model function into an external file:

```
function y = model twog(b, x)
y = b(1). *exp(-0.5. *(x-b(2)). /b(3)). ^2) + ...
    b(4). *exp(-0.5. *(x-b(5)). /b(6)). ^2);
```

```
guess2 = [guess 3000 3. 0.5];
[bfi t2 r2 J2 Covar2 mse2] = nli nfi tw(cbi n, count, ecount, @model twog, guess2);
bfi t2, sqrt(di ag(Covar2)), mse2
pl ot(cbi n, model twog(bfi t2, cbi n), ' r' )
hol d off
bfi t2 =
    30633      2. 0823      0. 21159      2732. 5      2. 8998      0. 37706
ans =
    99. 609
    0. 00069061
    0. 00056174
    23. 667
    0. 0069429
    0. 0041877
mse2 =
    163. 44
```

Although it seems to capture the data qualitatively, this is still a bad fit. Whether this should worry you or not depends completely on whether you believe the model should be “exact”



We keep getting these big mse's!

Let's demonstrate that $mse \sim 1$ is what you should get for a perfect model:

```
perfect = 2.0 .* randn(10000,1) + 4. ;
[count cbin] = hist(perfect, (-1:.2:9));
count = count(2:end-1);
cbin = cbin(2:end-1);
ecount = sqrt(count+1);
[bfit r J Sigma mse] = nlinfitw(cbin, count, ecount, model oneg, guess);
bfit, Sigma, mse
bfit =
    393.27    4.0159    2.0201
Sigma =
    25.784    -0.0005501    -0.057248
   -0.0005501    0.00046937    3.5555e-006
   -0.057248    3.5555e-006    0.00032997
mse =
    0.95507
chisq = numel(count)*mse
df = numel(count)-3;
pvalue = chi2cdf(chisq, df)
chisq =
    46.799
pvalue =
    0.56051
```

Let's see if 0.955 is actually good enough:

by definition, mse times number of bins equals chi-square

three fitted parameters

yep, good enough!

Summary of "classic" n.l.s fits:

A complete job consists of finding model parameters, their errors (or covariance matrix), and a goodness-of-fit p-value.

If you expect the model to be exact, then take the p-value seriously.

What is the uncertainty in quantities other than the fitted coefficients:

For example, what if we want the ratio of areas in the two Gaussians?

Method 1: Linearized propagation of errors

$$f \equiv f(\mathbf{b}) = f(\mathbf{b}_0) + \mathbf{f}' \cdot \mathbf{b}_1$$

$$\langle f \rangle = \langle f(\mathbf{b}_0) \rangle + \mathbf{f}' \cdot \langle \mathbf{b}_1 \rangle = f(\mathbf{b}_0)$$

$$\begin{aligned} \langle f^2 \rangle - \langle f \rangle^2 &= 2f(\mathbf{b}_0)(\mathbf{f}' \cdot \langle \mathbf{b}_1 \rangle) + \langle (\mathbf{f}' \cdot \mathbf{b}_1)^2 \rangle \\ &= \mathbf{f}' \cdot \langle \mathbf{b}_1 \otimes \mathbf{b}_1 \rangle \cdot \mathbf{f}' \\ &= \mathbf{f}' \cdot \Sigma \cdot \mathbf{f}' \end{aligned}$$

Recall the meaning of the b's:

```
function y = model twog(b, x)
y = b(1) .* exp(-0.5 .* ((x-b(2)). /b(3)). ^2) + ...
    b(4) .* exp(-0.5 .* ((x-b(5)). /b(6)). ^2);
```

Do whole thing in Matlab, including the symbolic derivatives:

```
syms b1 b2 b3 b4 b5 b6 real
func = (b1*b3)/(b4*b6);
b = [b1 b2 b3 b4 b5 b6];
symgrad = jacobian(func, b)'
symgrad =
    b3/b4/b6
         0
    b1/b4/b6
 -b1*b3/b4^2/b6
         0
 -b1*b3/b4/b6^2
grad = double(subs(symgrad, b, bfit2))
grad =
    0.00020537
         0
    29.732
 -0.0023024
         0
    -16.685
mu = double(subs(func, b, bfit2))
sigma = sqrt(grad' * Covar2 * grad)
mu =
    6.2911
sigma =
    0.096158
```

ratio of the areas

its standard error

Method 2: Sample from the posterior distribution (sort of*)

What makes this Bayesian is that it holds the data fixed, but samples over a space of hypotheses (the fitted parameters).

```
samp = mvnrnd(bfit2, Covar2, 1000);
```

```
samp(1:5, :)
```

```
ans =
```

30430	2.082	0.21203	2754.6	2.8975	0.37636
30421	2.0829	0.21213	2701.1	2.9026	0.37738
30645	2.0815	0.21125	2775.3	2.8969	0.37969
30548	2.0822	0.21229	2714.7	2.9011	0.37712
30607	2.0826	0.21175	2718	2.9016	0.37779

multivariate Normal random generator

```
funcsam = (samp(:, 1) .* samp(:, 3)) ./ (samp(:, 4) .* samp(:, 6));
```

```
funcsam(1:5)
```

```
ans =
```

```
6.2234
6.3307
6.1437
6.3346
6.3116
```

```
hist(funcsam, [5.9:0.025:6.7]);
```

```
mu = mean(funcsam)
```

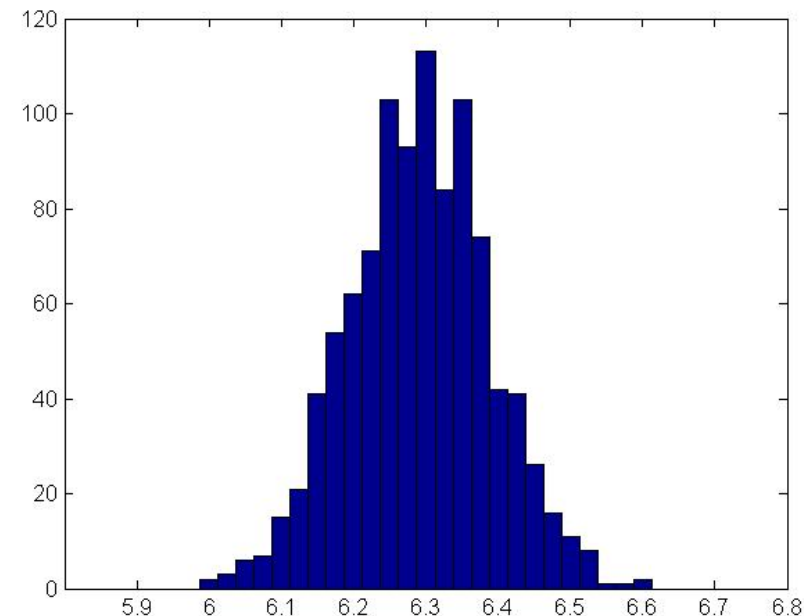
```
sigma = std(funcsam)
```

```
mu =
```

```
6.2911
```

```
sigma =
```

```
0.096832
```



*This is not really sampling from the posterior, because we are assuming, based on vague CLT-ology, that the b's are multivariate normal. Actually, they have their own multivariate posterior based on the data. Later, we'll learn how to sample it. But using multivariate normal for now lets us explore the concept, and is here not a bad approximation.

It's important that we use the full covariance matrix, not just its diagonal (the individual parameter variances):

wrongcov = diag(diag(Covar2))

wrongcov =

9921.9	0	0	0	0	0	0
0	4.7694e-007	0	0	0	0	0
0	0	3.1555e-007	0	0	0	0
0	0	0	0	560.14	0	0
0	0	0	0	0	4.8203e-005	0
0	0	0	0	0	0	1.7537e-005

sigma = sqrt(grad' * wrongcov * grad)

sigma = sqrt(grad' * Covar2 * grad)

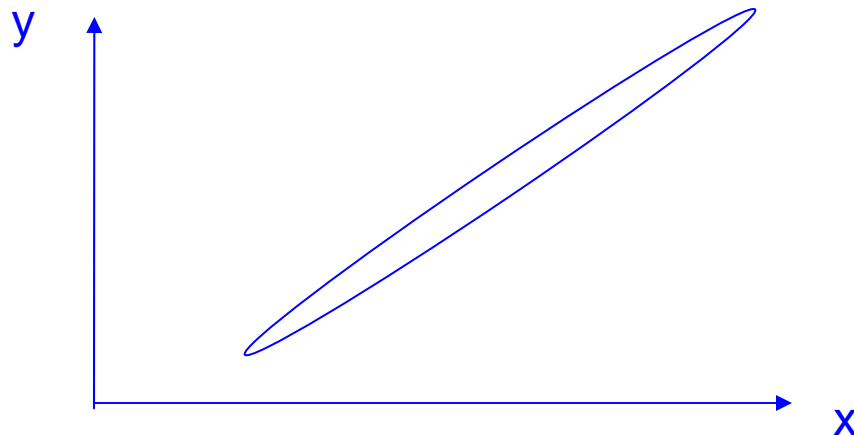
sigma =

0.092457

sigma =

0.096158

Here, there's not much difference. But if some of the b's were highly correlated in the fit, there could be a huge difference. The correct error estimate can be larger or smaller than the wrong one using only the diagonal (pos or neg correlations).



Example:

x and y have comparable (large) variances

x+y also has large variance

but x-y has a much smaller variance

Compare linear propagation of errors to sampling the posterior

- Note that even with lots of data, so that the distribution of the b 's really \rightarrow multivariate normal, a derived quantity might be very non-Normal.
 - In this case, sampling the posterior is a good idea!
- For example, the ratio of two normals of zero mean is Cauchy
 - which is very non-Normal!.
- So, sampling the posterior is a more powerful method than linear propagation of errors.
 - even when optimistically (or in ignorance) assuming multivariate Gaussian for the fitted parameters

Crito [or Doubting Thomas)] says:

“But there are so many places that you may have made an error!

For example, the fit was never actually good, so how do I know I don't have to scale the final uncertainty in the ratio of errors accordingly? (I've heard of people doing this!)

And how do I know that your mysterious 'correction' of Matlab's Covar was right? Aren't the MathSoft people smarter than you are?

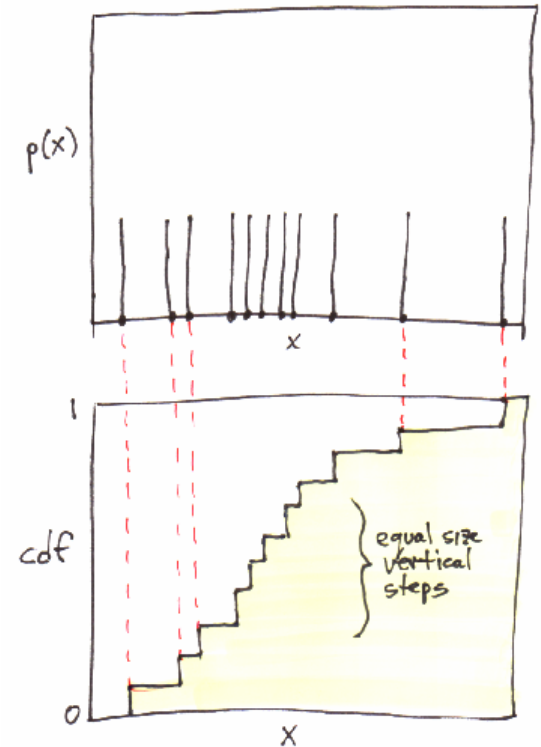
Oh my, oh my, what to do!”

You should always be thinking about how to check that the formalism that you are using is giving a sensible answer. Thus may you hope to avoid embarrassing errors!

Now is the time to learn about bootstrap resampling!

Bootstrap resampling (very important!)

- We applied some end-to-end process to a data set and got a number out
 - the ratio of areas
- The data set was drawn from a population
 - which we don't get to see, unfortunately
 - we see only a sample of the population
- We'd like to draw new data sets from the population, reapply the process, and see the distribution of answers
 - this would tell us how accurate the original answer was
 - but we can't: we don't have access to the population
- However, the data set itself is an estimate of the population pdf!
 - in fact, it's the only estimate we've got!
- We draw – with replacement – from the data set and carry out the proposed program
 - Bootstrap theorem [glossing over technical assumptions]: **The distribution of any resampled quantity around its full-data-set value estimates the distribution of the data set value around the population value.**



Do it in Matlab:

```
function mu = areasboot(data)
samp = randsample(data, numel(data), true);
[count cbin] = hist(samp, (1: .1: 4));
count = count(2:end-1);
cbin = cbin(2:end-1);
ecount = sqrt(count+1);
guess = [3.5e4 2.1 .3 3000 3. 0.5];
[bfit r J Covar mse] = nlinfitw(cbin, count, ecount, @model twog, guess);
mu = (bfit(1)*bfit(3))/(bfit(4)*bfit(6));
```

```
areas = arrayfun(@(x) areasboot(exloglen), (1:1000));
```

takes about 1 min on my machine

```
mean(areas)
```

```
std(areas)
```

```
ans =
```

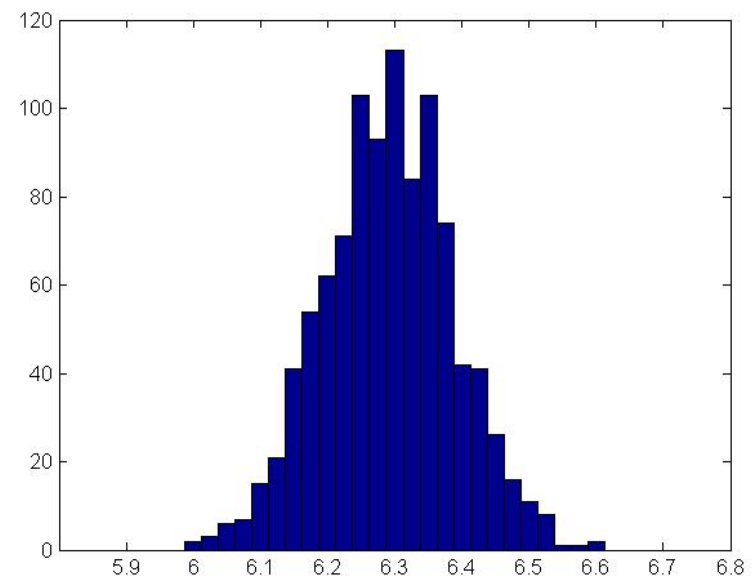
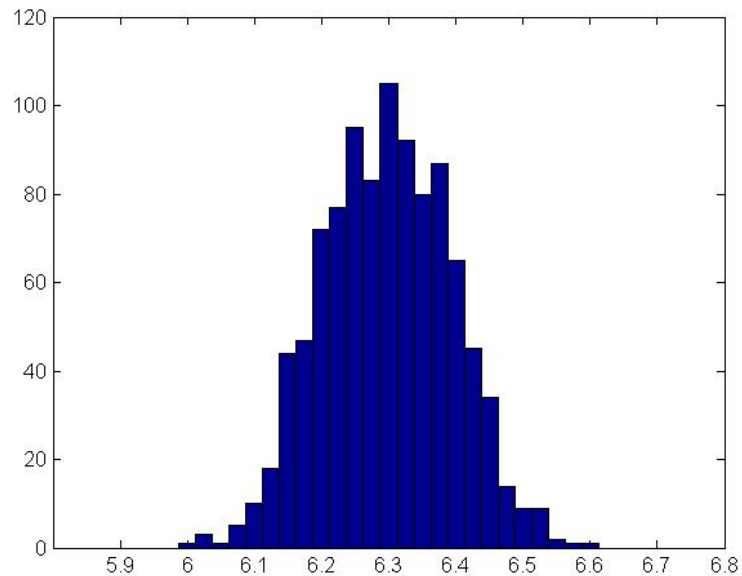
```
6.2974
```

```
ans =
```

```
0.095206
```

```
hist(areas, [5.9:0.025:6.7]);
```

recall, sampling from the posterior gave:



Compare and contrast bootstrap resampling and sampling from the posterior!

Both have same goal: Estimate the accuracy of fitted parameters.

- **Bootstrap** is frequentist in outlook
 - draw from “the population”
 - even if we have only an estimate of it (the data set)
- Easy to code but computationally intensive
 - great for getting your bearings
 - must repeat your basic fitting calculation over all the data 100 or 1000 times
- Applies to both model fitting and descriptive statistics
- Fails completely for some statistics
 - e.g. (extreme example) “harmonic mean of distance between consecutive points”
 - how can you be sure that your statistic is OK (without proving theorems)?
- Doesn't generalize much
 - take it or leave it!
- **Sampling from the posterior** is Bayesian in outlook
 - there is only one data set and it is never varied
 - what varies from sample to sample is the goodness of fit
 - we don't just sit on the (frequentist's) ML estimate, we explore around
- In general harder to implement
 - we haven't learned how yet, except in the simple case of an assumed multivariate normal posterior
 - will come back to this when we do Markov Chain Monte Carlo (MCMC)
 - may or may not be computationally intensive (depending on whether there are shortcuts possible in computing the posterior)
- Rich set of variations and generalizations are possible

(patients not polyps)