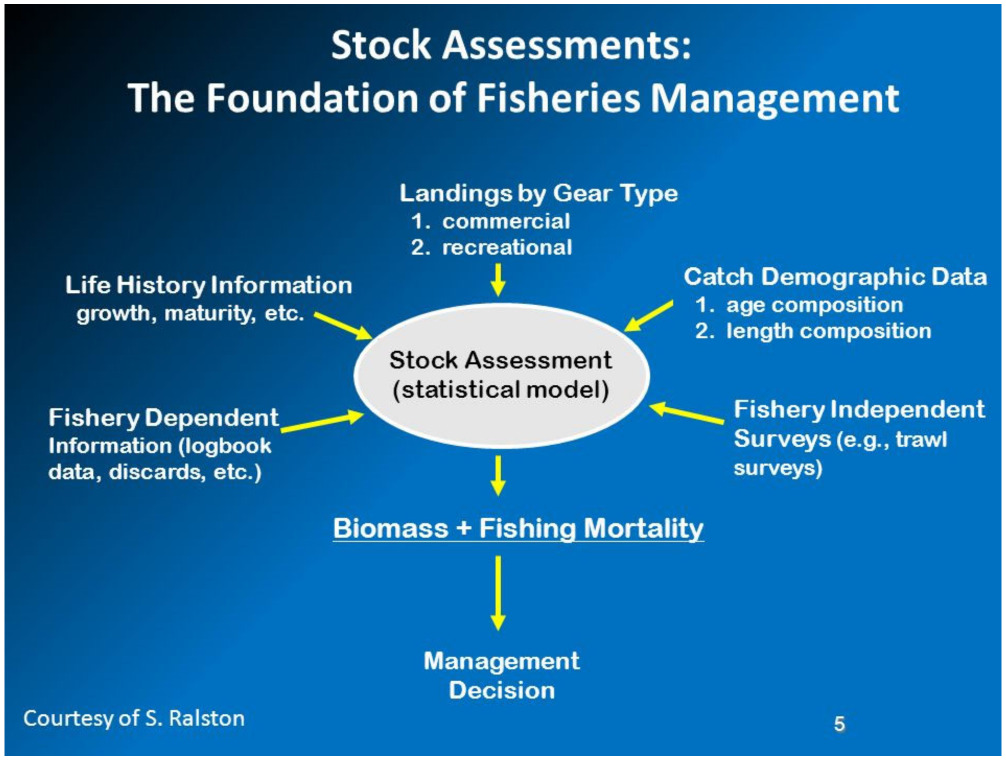
Marine Institute

Lecture 4+5: Models, Data, and Statistical Inference using R



Noel Cadigan

CFER

Centre for Fisheries Ecosystems Research

Marine Institute

Memorial University

F6004 Lecture 4 Outline

Models and data (parts of Haddon, chapter 3)

- Bringing models and data together, judging how well model predictions and the data "fit"
- Complexity in models, adding parameters, reality vs. simplicity
- Methods to judge how well models and data "fit"
- Least squares approach – residual error
- Non-linear estimation, using R to find minimum sum of residual error
- Numerical optimization and root finding.
- Estimating uncertainty about the model
- Methods based on parametric statistics, SD, SE, 95% CI, Variance…Normal distributions
- Bootstrap

Fitting models with data

- The models we have looked at so far (e.g. logistic, YPR) are simple mathematical representation of population dynamics
- that nevertheless give some fundamental insights about the potential productivity of a fish stock
- i.e. H_{MSY} or F_{MSY}, B_{MSY}, F_{MAX}

Parameters

- These simple models involve some fundamental parameters (e.g. r, K) that determine the behaviour of the model – our model of stock dynamics.
- We can simply assign plausible values to these parameters, and then examine the model output
- But what is plausible – it is subjective

Fitting models with data

- Remember that the clients of stock assessment prefer <u>objective</u> advice.
- So we need an objective means to specify values for the model parameters (e.g. *r*, *K*).
- However, the choice of model is in many cases somewhat subjective ……
- Advanced stock assessment looks at model uncertainty as well. We will too in F6005.

Fitting models with data

- Data can provide an objective means to estimate parameters.
- If we have some data from surveys or elsewhere that are measurements of the state of the stock over some period of time
- then we can find values of the model parameters that ensure that the model agrees as close as possible to the data

Fitting models with data

– We will start with very simple models and parameter estimation methods

– we will use these as “spring boards” to more complicate and practically relevant models

– so hang on,

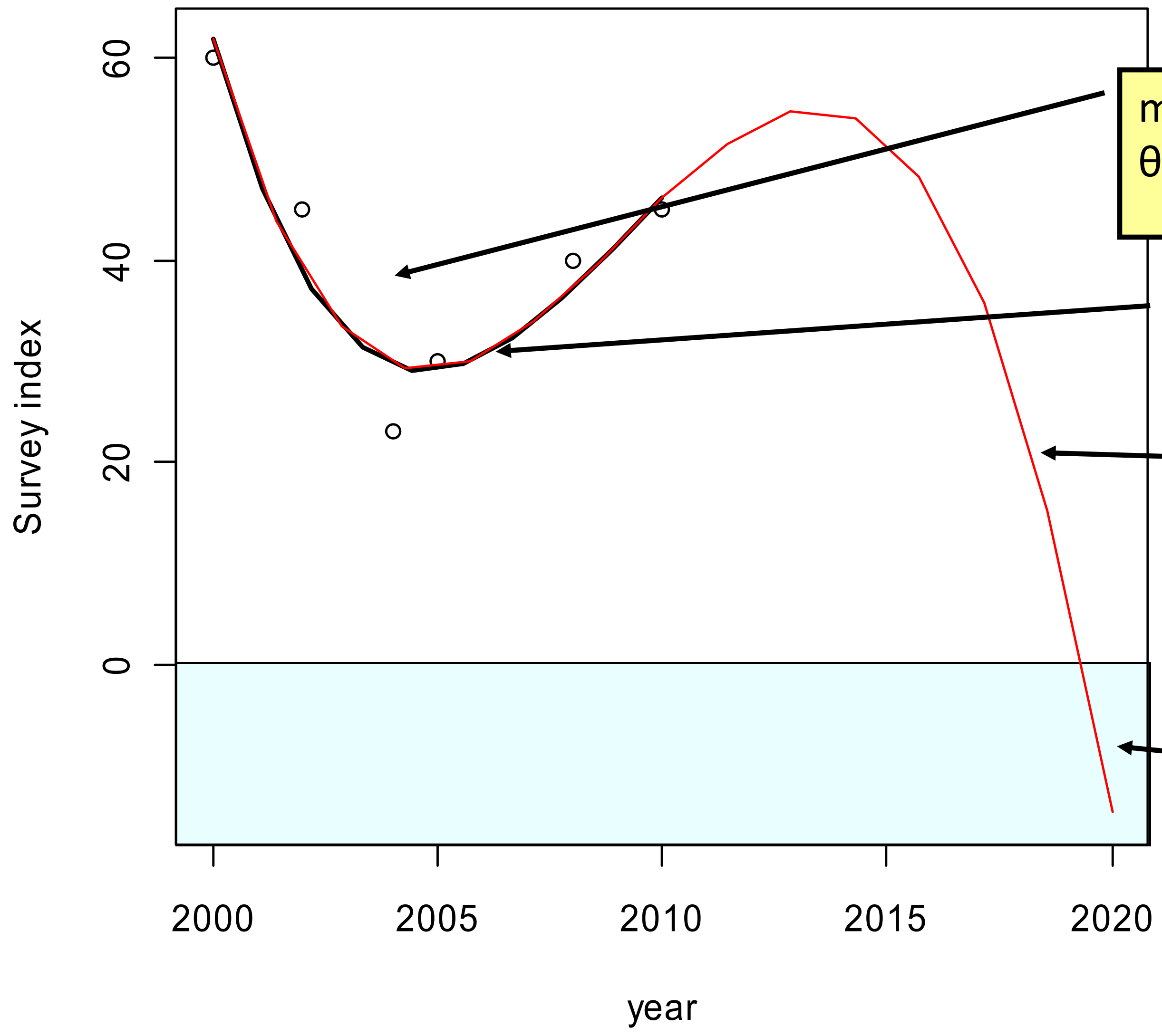
Fitting models with data: 3 requirements

- – A formal mathematical model of a system, which is capable of generating predictions about the observable world.
- – Data from the system to be used when estimating parameter values
- – A criterion (or objective function, or fit function) to judge the quality (or closeness) of the fit between model predictions and observed data.

Fitting models with data: 3 requirements

- However, simply fitting the data well does not mean a model is good
- The model must generate biologically sensible predictions, especially when predicting behaviour outside of the range of data (e.g. initial stock size, carrying capacity K, etc).
- There are many models (e.g. high-order polynomials) that can fit data well but give silly predictions

3rd order polynomial example



model: index = $\theta_o + \theta_1$year + $\theta_2$year^2 + $\theta_3$year^3

fits the data well

predictions are not reasonable

not possible

Fitting models with data: 3 requirements

– What is a biologically sensible model can be subjective

– However, there is a general principal that simple is usually better

– Principle of parsimony:

Principle of parsimony

From Wikipedia, the free encyclopedia

Occam's razor (or **Ockham's razor**)[1] often expressed in Latin as the ***lex parsimoniae***, translating to **law of parsimony**, **law of economy** or **law of succinctness**, is a principle that generally recommends, when faced with competing hypotheses that are equal in other respects, selecting the one that makes the fewest new assumptions.

Which comes first: the model or data

- Conceptually, model selection is independent of the data
- Once a model is chosen, optimum values for parameters are selected based on data
- In practise, a range of plausible models are available, and model selection occurs in steps, based partly on the ability to fit data, but also on the reliability of the model

Quality of fit vs parsimony vs reality

- – More complicated models may seem more realistic because they are more flexible (see production curve example in Haddon, Fig. 3.1)
- – If a more flexible model does not fit the data any better than a more simple model, then there is a tendency to go with the more simple model.
- – More complicated models are harder to estimate, and require more data – is the cost worth it??

Quality of fit vs parsimony vs reality

– More complicated models usually fit the data better, but other predictions may not be better.

– There are ways to assess model prediction error – i.e. cross validation, Akaike's Information Criterion (AIC) – described later

– Highly parameterized models can have some insidious biases –especially related to variance parameters which are required to account for uncertainty in the data

Uncertainty

- We have previously discussed survey indices of abundance that can be used to estimate stock assessment model parameters
- These indices are not exactly measurements of stock size
- They have sampling variability
- You will estimate this variance in assignment 1

Uncertainty

- – This sampling uncertainty (sometimes called measurement error) is one reason why our models will not fit our data exactly
- – The other main reason is model process error – the model is only a simplification of population dynamics; that is, M is oprobably not constant, r is probably not constant, etc.
- – We cannot expect perfect fits

Uncertainty

- Sometimes different values for parameters produce approximately the same fit to the data.
- i.e. there are no well defined optimal values
- and a range of values are consistent with our data
- this tends to be a problem with more complicated models
- referred to as parameter correlation

Uncertainty

- Accounting for these uncertainties is a vital part of population modelling and stock assessment
- We need to be able to characterize our uncertainty
- i.e. provide a range of parameter values that are plausible
- i.e. what is the range of realities that could produce our data
- This should be an objective of stock assessment, and not just to provide a single set of stock size estimates

Uncertainty

- First we focus on parameter estimation
- then we consider how to quantify uncertainty.

Least square parameter estimation

– The most commonly used estimation fit function.

– It involves minimizing the differences between data and model predictions

– Let *O* denote the observed value of some characteristic of a population, and let *P* denote a model predicted value of this characteristic.

– For example, the characteristic could be total stock size, and *P* could be the discrete logistic model prediction and *O* could be a survey estimate.

Models and data

- Another example is the relationship between the weight of a fish and its length.
- A linear model of weight vs. length *could [not should]* be used, $W = \theta_o + \theta_1 x L$, and the parameters θ_o and θ_1 could be estimated based on length and weight measurements from a sample of fish.
- I will try and use Greek symbols to denote parameters to estimate, to avoid confusion with variables that represent data or population characteristics

Models and data

- another example is the logistic model of the proportion *p* of the stock mature at age *a*,

$$p(a) = \frac{\exp(\theta_o + \theta_1 a)}{1 + \exp(\theta_o + \theta_1 a)}$$

- Note that *p(a)* is bounded between 0 and 1.
- This model could be estimated based on samples of fish – and counts of how many were mature and immature, and what their ages were.

Least square estimation

– Models are almost never perfect

– They predict nature with error

– Plus there is error in data

– We usually denote error terms using ε

– The basic estimation framework is $O = P + \varepsilon$.

– ε is the residual error term between the model predicted values (aka fitted or expected) and data.

Least square estimation

– Residual errors (ε's) can be positive or negative.

– We wish to estimate model parameters to make the errors as small as possible.

– Say we have observations $O_1, \ldots, O_n$, and a model that produces predictions $P_1, \ldots, P_n$.

n is the sample size

– The least squares approach is to choose parameter values that minimize $SSR = \Sigma_i (O_i - P_i)^2$ based on a set of data

Least square estimation

– $SSR = \Sigma_i \varepsilon_i^{\,2}$ is the sum of squared residuals,

$$\varepsilon_i = O_i - P_i.$$

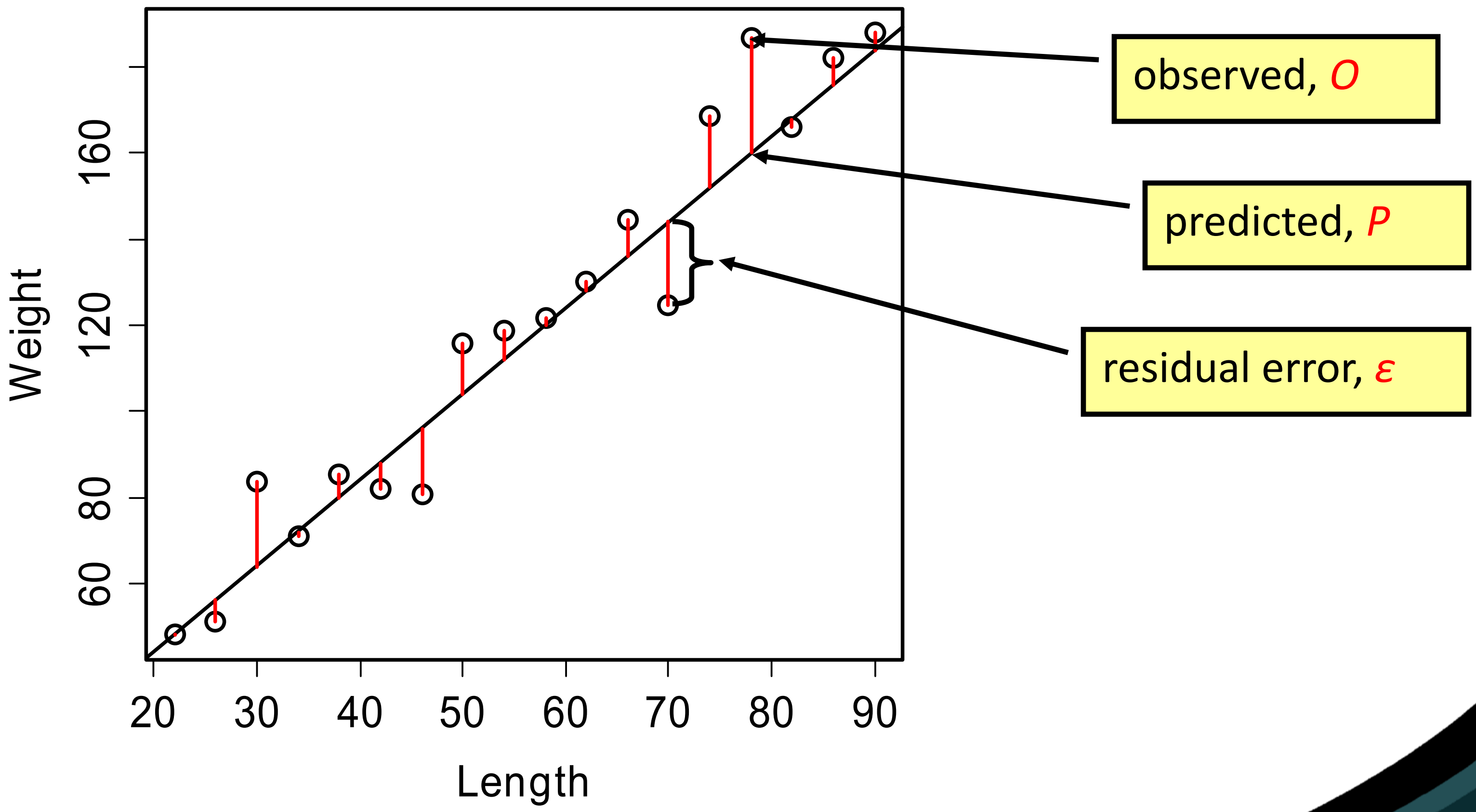
– For parameter estimation, we are not really concerned if residuals are positive or negative, just that they be as small as possible in absolute value

– This is why we square the residuals

– But there are many other fit functions ($SSR^* = \Sigma_i |O_i - P_i|$) ……

Least square estimation

– Least squares estimation minimized the overall distance between data and model predictions



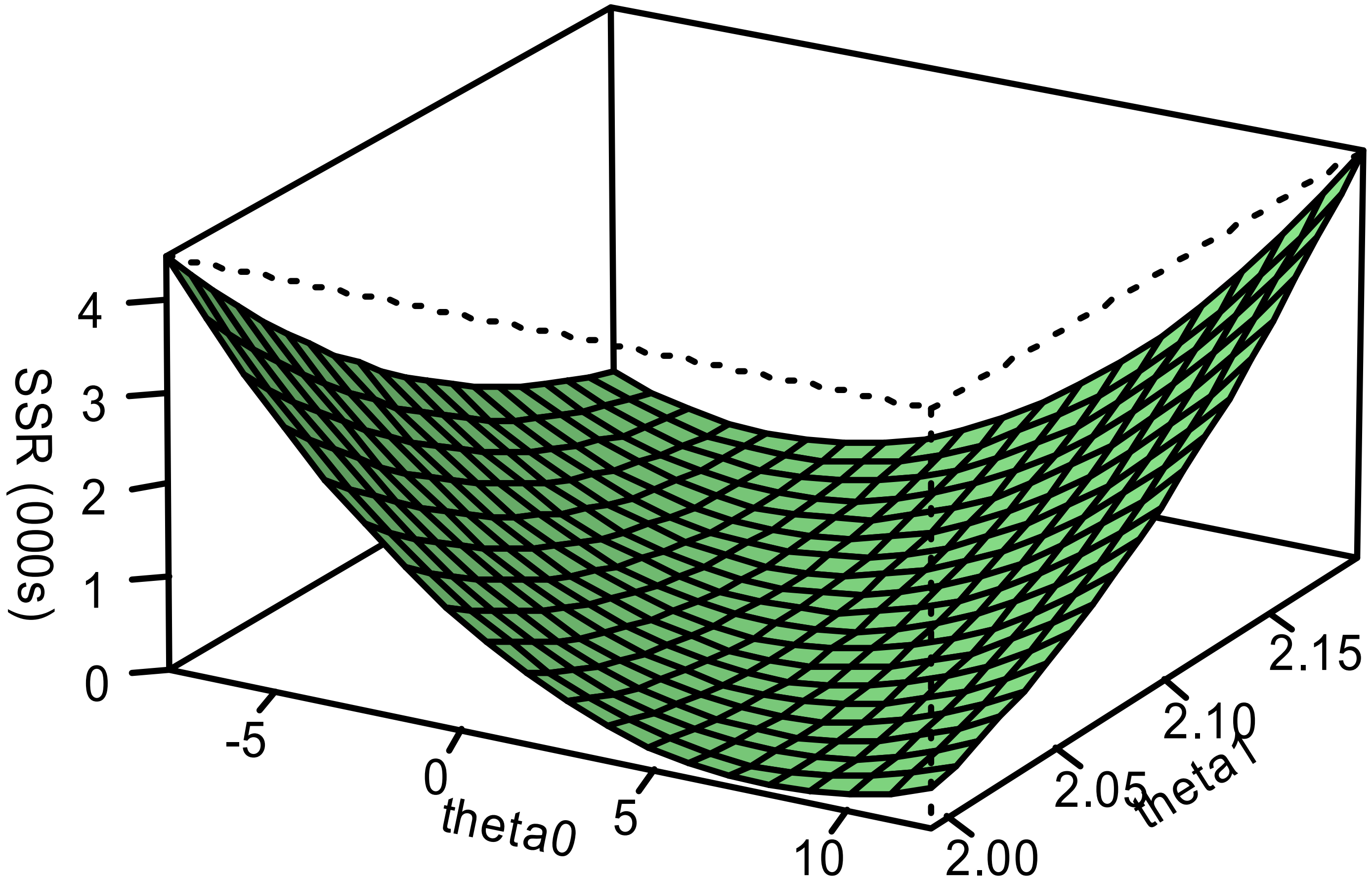
The length-weight data

– 18 data points

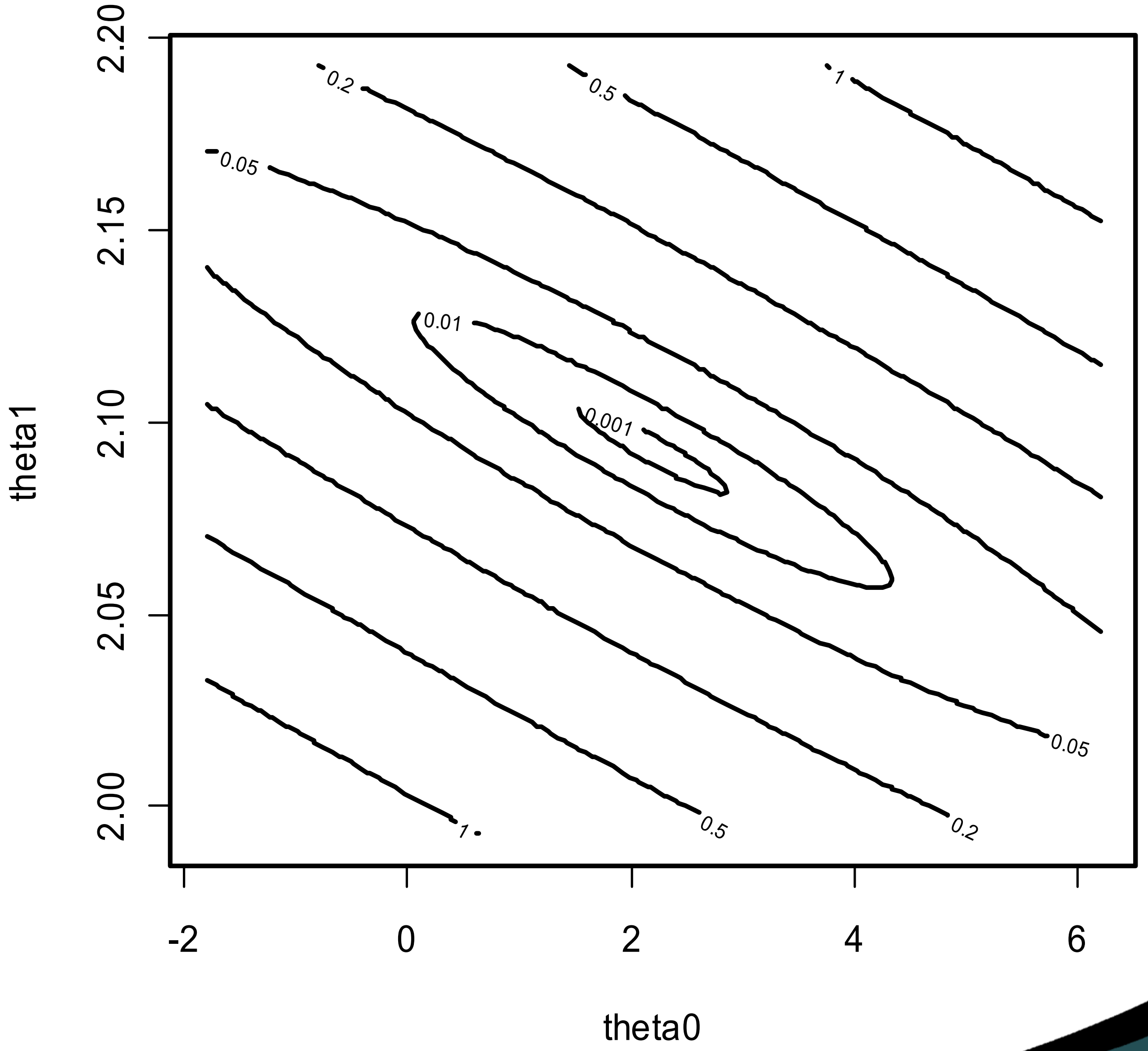
```
print(tdata,digits=4)
$length
 [1] 22 26 30 34 38 42 46 50 54 58 62 66 70 74 78 82 86 90

$weight
 [1]  48.01  50.94  83.51  70.98  85.21  81.90  80.87 115.83 118.94 121.65
[11] 130.17 144.51 124.78 168.76 186.87 166.13 182.15 187.93
```

- The model $W = \theta_o + \theta_1 {*} L + \varepsilon$

SSR Surface



SSR surface



How to minimize the fit function

- There are many ways to minimize the fit function, $G(\theta)$. Here θ represents all the parameters.
- The conceptually simplest method is to do a grid search over the range of parameter values.
- This is not numerically efficient
- There are a variety of numerical methods
- Many involve finding the root of the first derivative of the fit function

How to minimize the fit function

– Note that the slope at the minimum is zero

– and so the parameter values $\hat{\theta}$ that minimize the fit function also have the property

$$\left.\frac{dG(\theta)}{d\theta}\right|_{\theta=\hat{\theta}} = 0$$

– For the linear model, the solution to the least squares estimates of θ is known

Least squares estimates for the linear model

– The generic linear model is $Y = \theta_o + \theta_1 {}^* X + \varepsilon$

– The least squares estimators of both θ parameters, for a random sample of n data pairs $(X_1,Y_1),\ldots,(X_n,Y_n)$, are

$$\hat{\theta}_o = \bar{y} - \hat{\theta}_1 \bar{x}$$

$$\hat{\theta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

these are sample means

the ^ mean an estimate of… this is a statistic, calculated from data

How to minimize the fit function

- In R it is really easy to get least squares estimates for linear models,
- data.fit = lsfit(X,Y) returns an object with many elements, but data.fit$coef will give the parameter estimates
- For the length-weight data

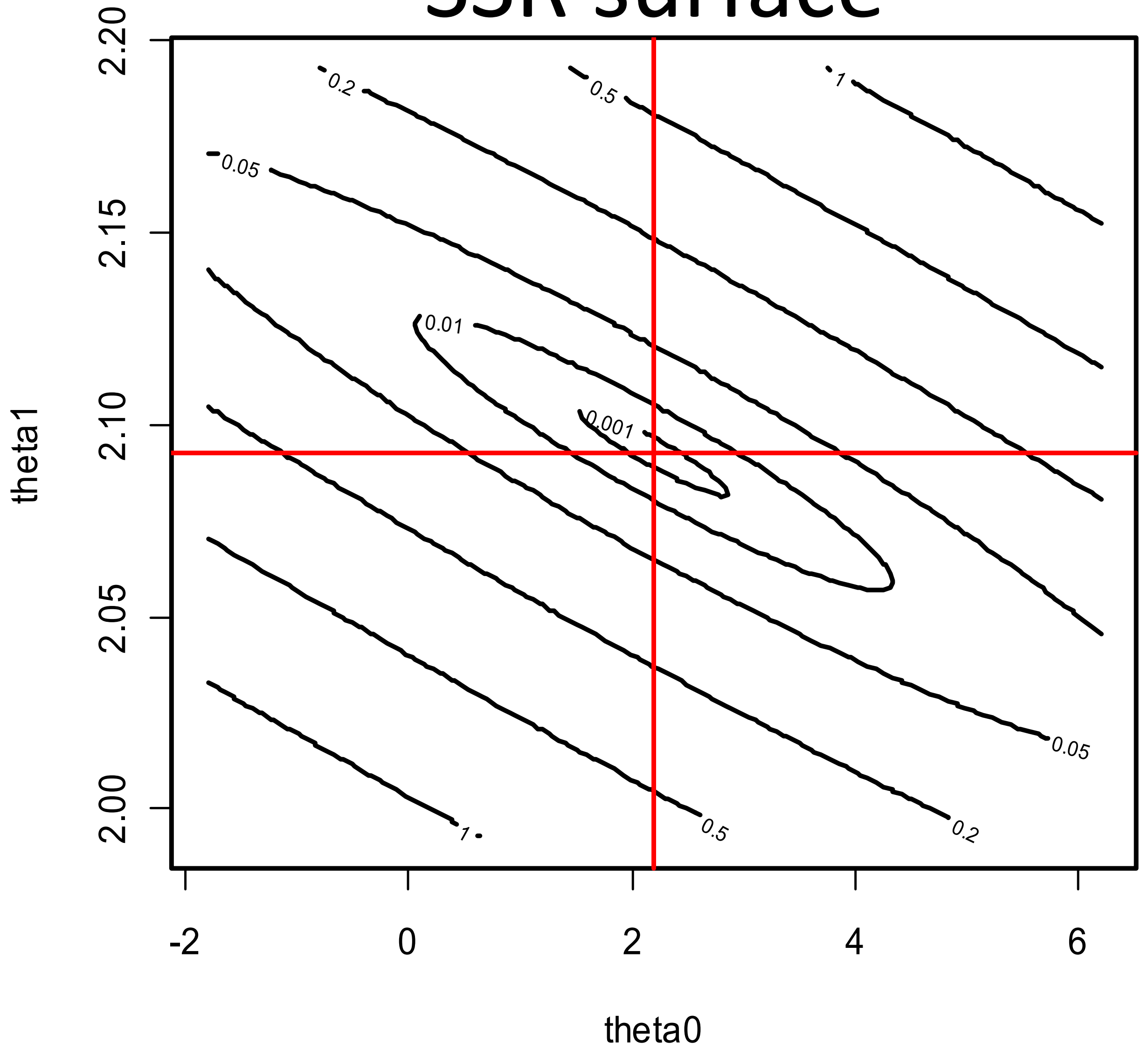
This is only for linear models

```
> data.fit = lsfit(L,W)
> print(data.fit$coef)
Intercept         X
 2.201257  2.092769
>
```

```
> names(data.fit)
[1] "coefficients" "residuals"    "intercept"    "qr"
```

SSR = 2.123

SSR surface



0.2
0.5
1
0.05
0.01
0.001
0.05
1
0.5
0.2
2.20
2.15
2.10
2.05
2.00
theta1
-2
0
2
4
6
theta0

How to minimize the fit function

– For nonlinear models you have to do more work.

– You have to write a function for the model and estimation, either as a single function or two functions

– For the length-weight data

```
LW_SSR = function(parm){
  Pred_W = parm[1] + parm[2]*L
  SSR = sum((W - Pred_W)^2)
  return(SSR)
}
```

The only argument for the function are the parameters (parm), everything else (i.e. L,W) are global.
Do not re-assign these global values within the function

How to minimize the fit function

– R has several function minimizers.

```
> parm.start = c(0,1)
> data.fit = optim(parm.start,LW_SSR)
> print(data.fit)
$par
[1] 2.214256 2.092548

$value
[1] 2123.409

$counts
function gradient
      83       NA

$convergence
[1] 0

$message
NULL
```

optim is an R function that finds the minimum of a fit function

And a fit function

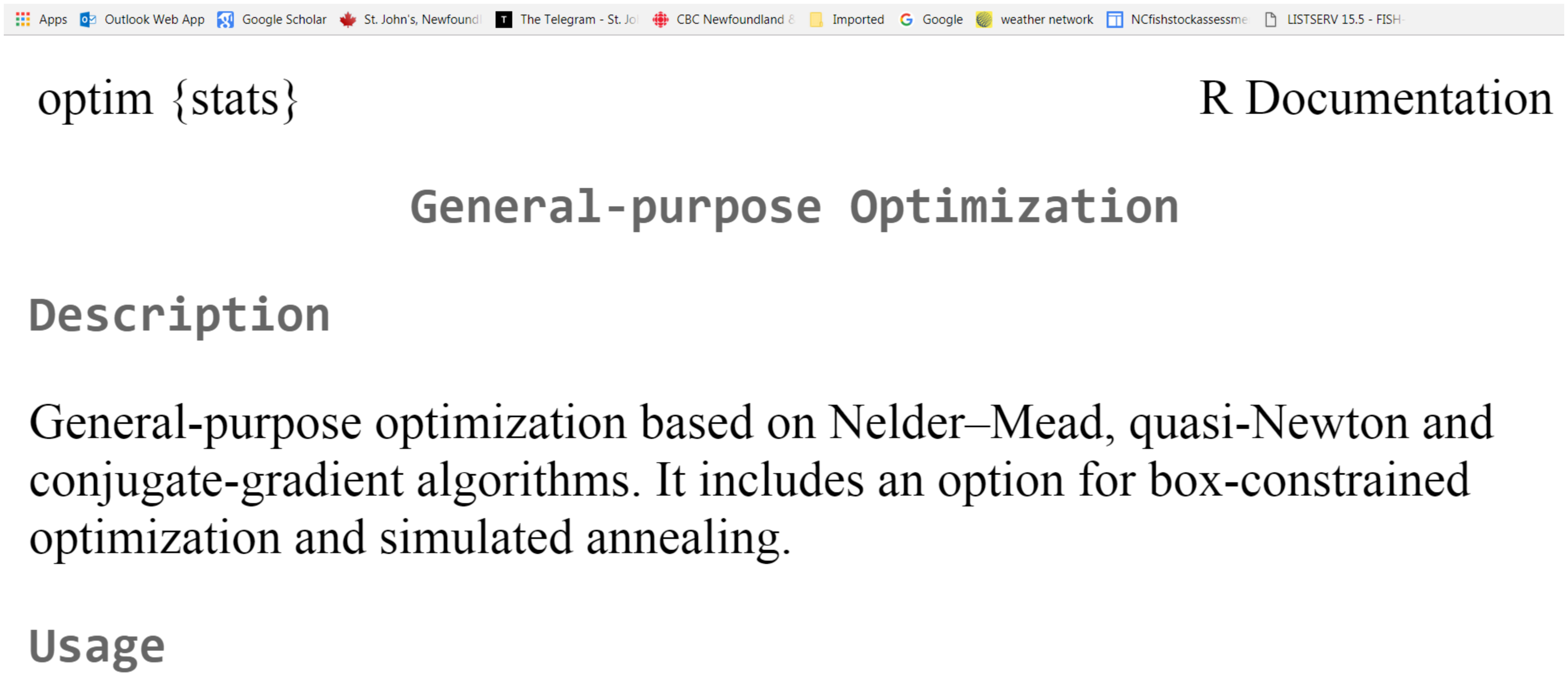
need to give optim starting values for the parameters

optim returns the estimates, which are sort of close to the exact lsfit results

```
lsfit results
Intercept        X
 2.201257  2.092769
```

0 indicates successful completion

Optim()


Apps Outlook Web App Google Scholar St. John's, Newfoundl The Telegram - St. Jo CBC Newfoundland & Imported Google weather network NCfishstockassessme LISTSERV 15.5 - FISH-

optim {stats} R Documentation

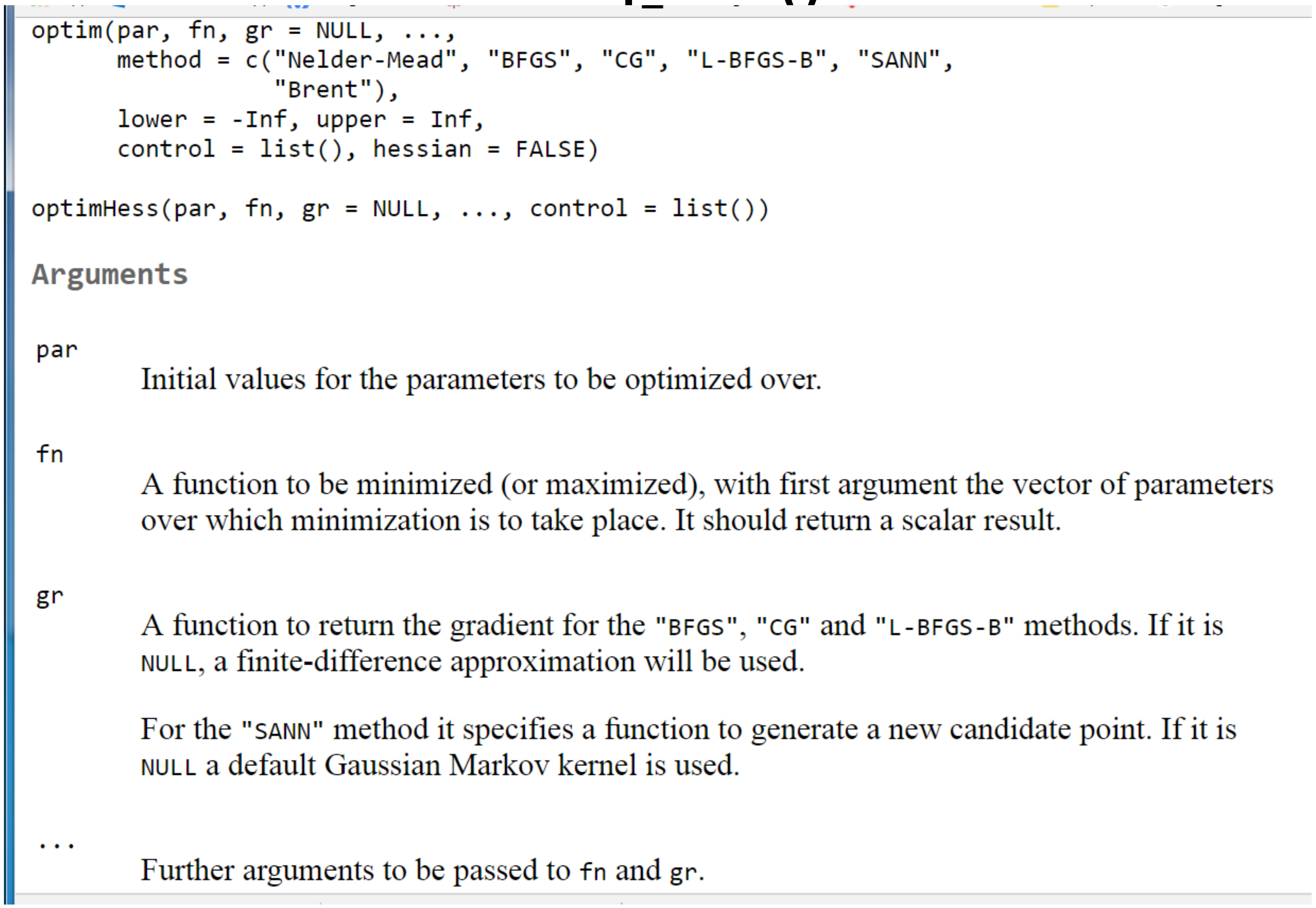
General-purpose Optimization

Description

General-purpose optimization based on Nelder–Mead, quasi-Newton and conjugate-gradient algorithms. It includes an option for box-constrained optimization and simulated annealing.

Usage

Optim()



```
optim(par, fn, gr = NULL, ...,
      method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN",
                 "Brent"),
      lower = -Inf, upper = Inf,
      control = list(), hessian = FALSE)

optimHess(par, fn, gr = NULL, ..., control = list())
```

Arguments

`par`
: Initial values for the parameters to be optimized over.

`fn`
: A function to be minimized (or maximized), with first argument the vector of parameters over which minimization is to take place. It should return a scalar result.

`gr`
: A function to return the gradient for the "`BFGS`", "`CG`" and "`L-BFGS-B`" methods. If it is `NULL`, a finite-difference approximation will be used.

For the "`SANN`" method it specifies a function to generate a new candidate point. If it is `NULL` a default Gaussian Markov kernel is used.

`...`
: Further arguments to be passed to `fn` and `gr`.

The default method is an implementation of that of Nelder and Mead (1965), that uses only function values and is robust but relatively slow. It will work reasonably well for non-differentiable functions.

Algorithm 1: Downhill simplex of [Nelder and Mead, 1965].

```
input  : the cost function $f : \mathbb{R}^n \to \mathbb{R}$
         $\{\mathbf{x}_i\}_{i=0}^{n}$ an initial simplex
output: $\mathbf{x}^\star$, a local minimum of the cost function $f$.
begin
    $k \leftarrow 0$ ;
    while STOP-CRIT and $(k < k_{max})$ do
        $h \leftarrow \arg\max_i f(\mathbf{x}_i)$ ;
        $l \leftarrow \arg\min_i f(\mathbf{x}_i)$ ;
        $\mathbf{x}' \leftarrow (1+\alpha)\bar{\mathbf{x}} - \alpha \mathbf{x}_h$ ;
            where $\alpha > 0$ is the reflection coefficient ;
        if $f(\mathbf{x}') < f(\mathbf{x}_l)$ then
            $\mathbf{x}'' \leftarrow (1+\gamma)\mathbf{x}' - \gamma\bar{\mathbf{x}}$ ;
                where $\gamma > 1$ is the expansion coefficient ;
            if $f(\mathbf{x}'') < f(\mathbf{x}_l)$ then
                $\mathbf{x}_h \leftarrow \mathbf{x}''$ ;                              /* expansion */
            else
                $\mathbf{x}_h \leftarrow \mathbf{x}'$ ;                               /* reflection */
        else if $f(\mathbf{x}') > f(\mathbf{x}_i), \forall i \neq h$ then
            if $f(\mathbf{x}') \leq f(\mathbf{x}_h)$ then
                $\mathbf{x}_h \leftarrow \mathbf{x}'$ ;                               /* reflection */
            $\mathbf{x}'' \leftarrow \beta \mathbf{x}_h + (1-\beta)\bar{\mathbf{x}}$ ;
                where $0 < \beta < 1$ is the contraction coefficient ;
            if $f(\mathbf{x}'') > f(\mathbf{x}_h)$ then
                $\mathbf{x}_i \leftarrow \frac{\mathbf{x}_i + \mathbf{x}_l}{2} \quad \forall i \in [\![0, n]\!]$ ;   /* multiple contraction */
            else
                $\mathbf{x}_h \leftarrow \mathbf{x}''$ ;                              /* contraction */
        else
            $\mathbf{x}_h \leftarrow \mathbf{x}'$ ;                                   /* reflexion */
        $k \leftarrow k + 1$ ;
    return $\mathbf{x}_l$
end
```

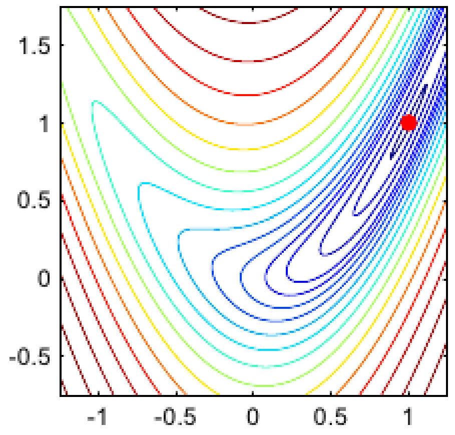


Figure: Contour plot of the Rosenbrock banana function.

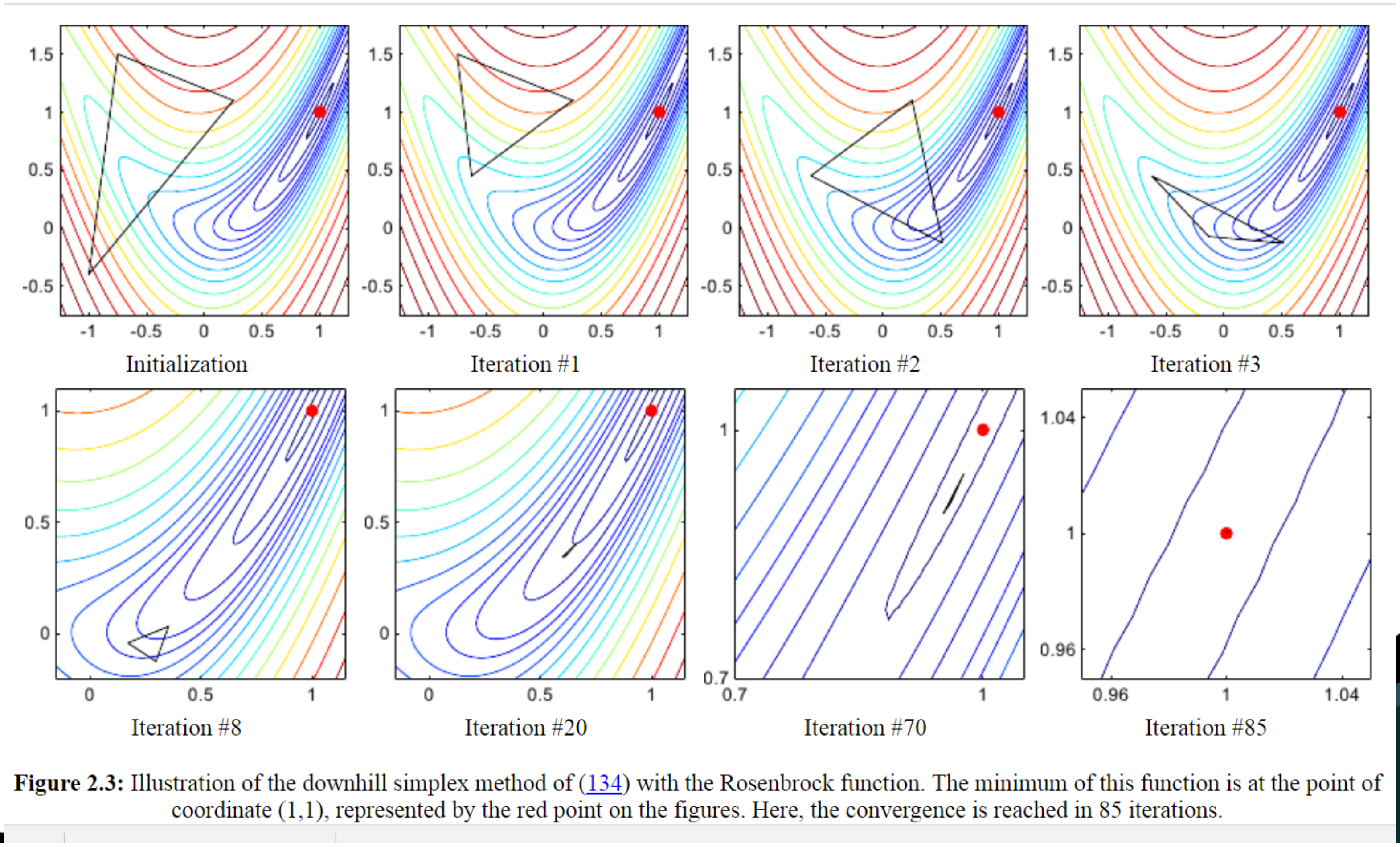
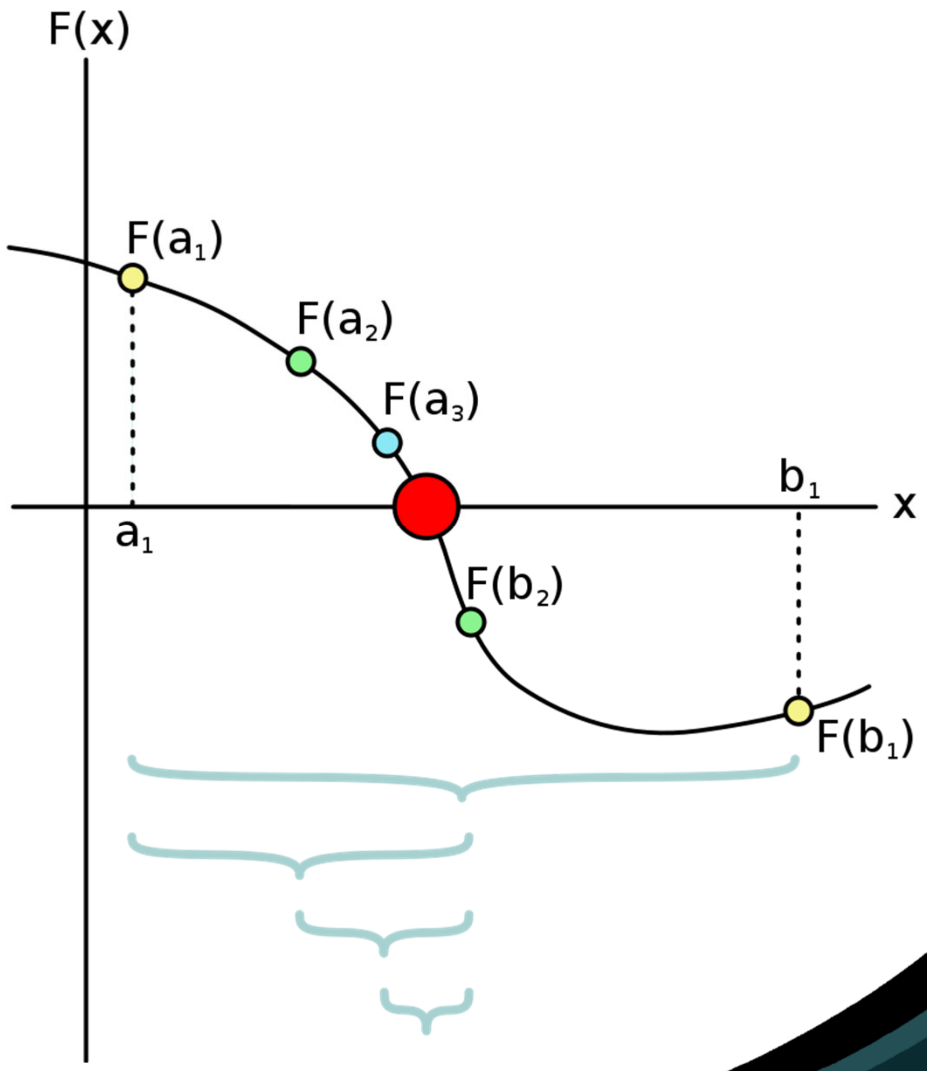


Figure 2.3: Illustration of the downhill simplex method of (134) with the Rosenbrock function. The minimum of this function is at the point of coordinate (1,1), represented by the red point on the figures. Here, the convergence is reached in 85 iterations.

Bisection method

The **bisection method** is a root-finding method that repeatedly bisects an interval and then selects a subinterval in which a root must lie for further processing. It is a very simple and robust method, but it is also relatively slow.



Function optimization: Newton’s Method

– Finding the parameters that minimize a fit statistic

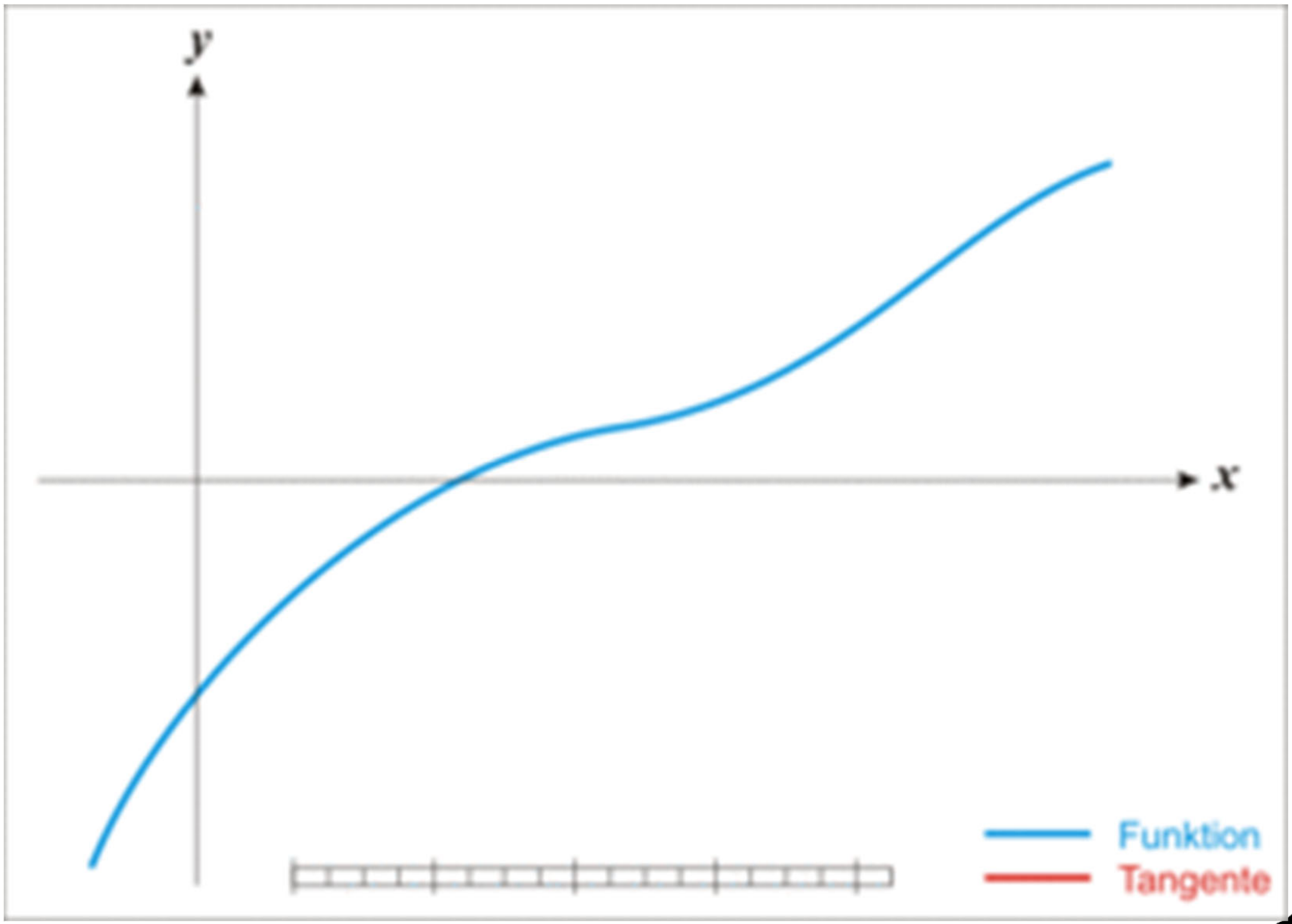
– By finding the θ root of the gradient function, $\dot{g}(\theta) = \partial G(\theta)/\partial\theta$

– If we have a reasonably good guess θ_j of the root value, then a first-order Taylors series expansion of $\dot{g}(\theta)$ about θ_j is $\dot{g}(\theta) = \dot{g}(\theta_j) + \ddot{g}(\theta_j)(\theta_j - \theta)$

– And we can improve our guess of the root using

$$\theta_{j+1} = \theta_j - \dot{g}(\theta_j)/\ddot{g}(\theta_j)$$

Newton’s root finding



Newton's method

– If θ is a vector valued parameter:

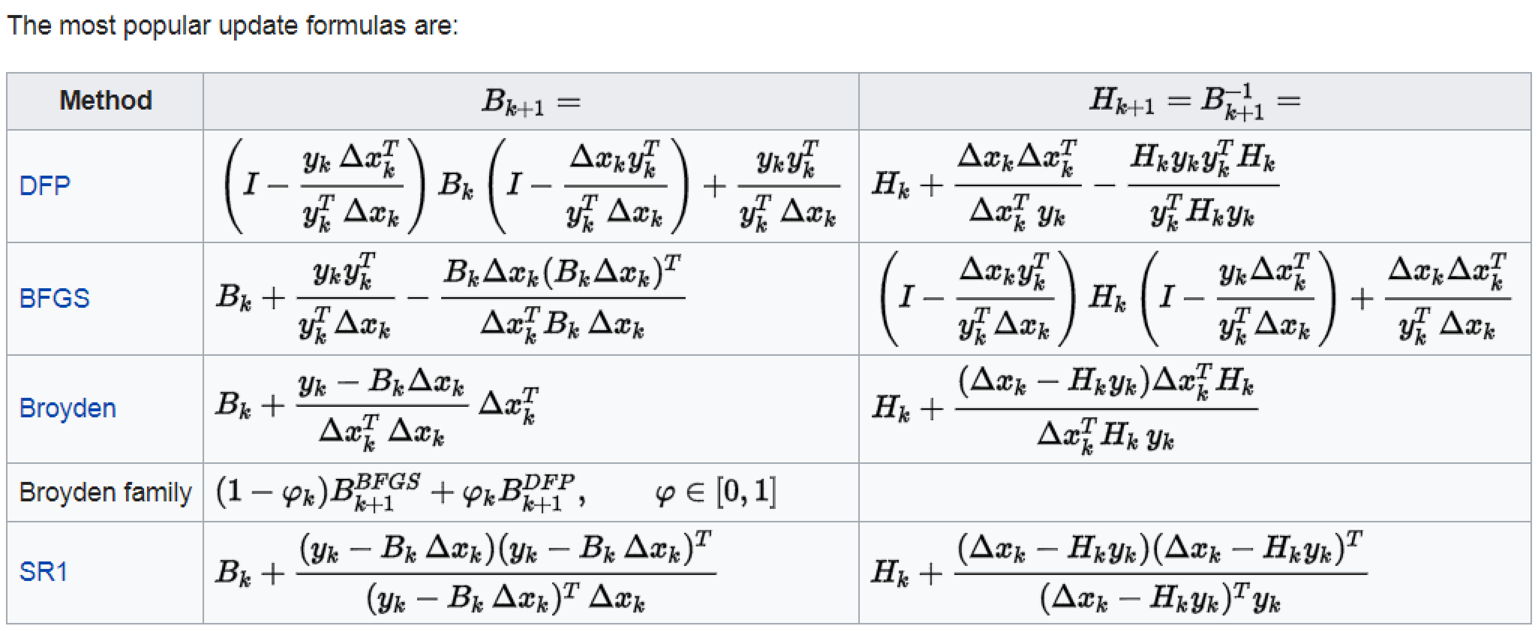
$$\theta_{j+1} = \theta_j - \ddot{g}^{-1}(\theta_j)\dot{g}(\theta_j)$$

– where $\dot{g}(\theta_j)$ is the gradient vector and $\ddot{g}(\theta_j)$ is the hessian matrix

Quasi-Newton method

– Quasi-Newton methods are used to either find zeroes or local maxima and minima of functions, as an alternative to Newton's method.

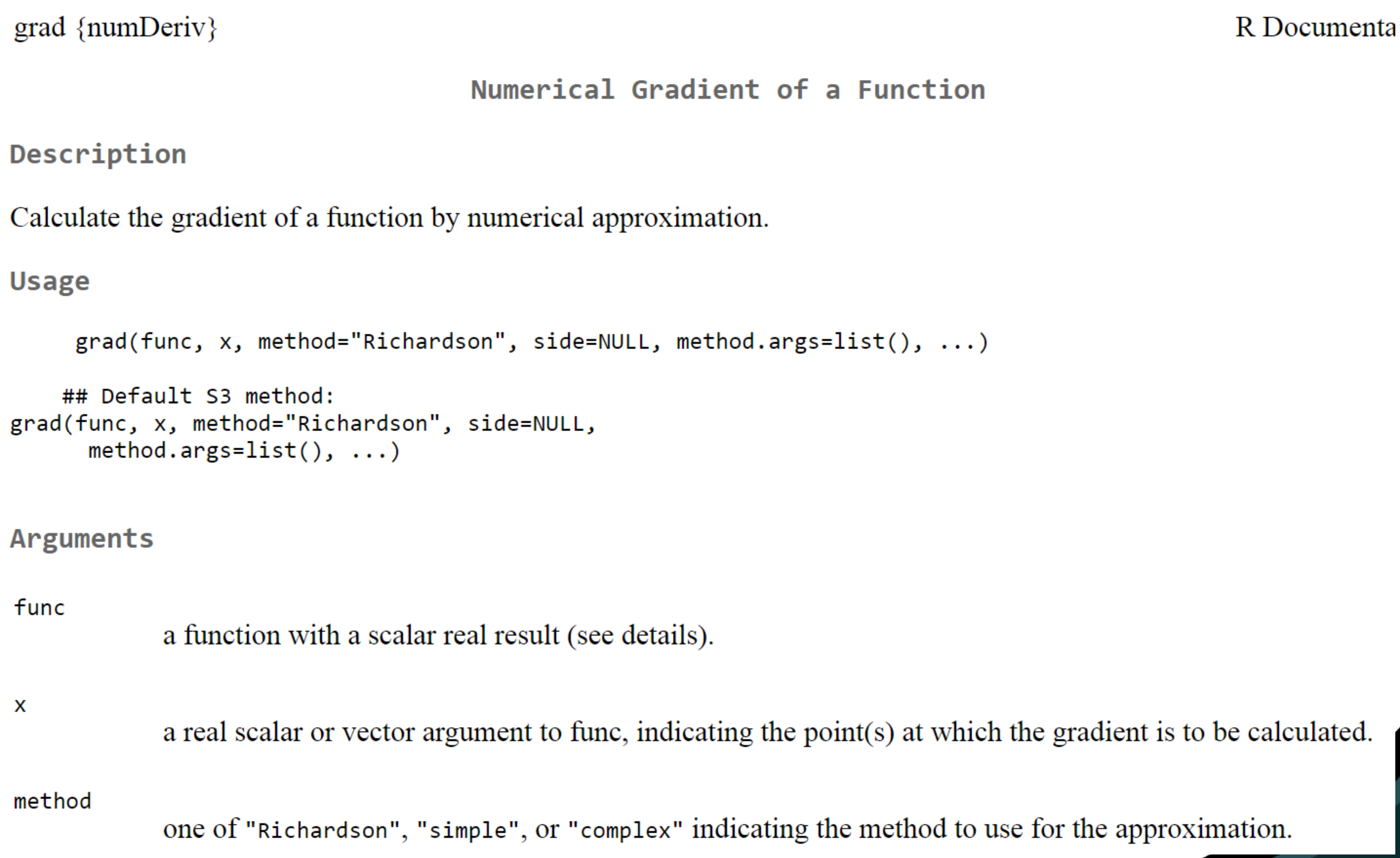
– They can be used if the Jacobian or Hessian is unavailable or is too expensive to compute at every iteration.



The most popular update formulas are:

Method	$B_{k+1} =$	$H_{k+1} = B_{k+1}^{-1} =$
DFP	$\left(I - \frac{y_k\,\Delta x_k^T}{y_k^T\,\Delta x_k}\right) B_k \left(I - \frac{\Delta x_k y_k^T}{y_k^T\,\Delta x_k}\right) + \frac{y_k y_k^T}{y_k^T\,\Delta x_k}$	$H_k + \frac{\Delta x_k \Delta x_k^T}{\Delta x_k^T\,y_k} - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k}$
BFGS	$B_k + \frac{y_k y_k^T}{y_k^T \Delta x_k} - \frac{B_k \Delta x_k (B_k \Delta x_k)^T}{\Delta x_k^T B_k\,\Delta x_k}$	$\left(I - \frac{\Delta x_k y_k^T}{y_k^T \Delta x_k}\right) H_k \left(I - \frac{y_k \Delta x_k^T}{y_k^T \Delta x_k}\right) + \frac{\Delta x_k \Delta x_k^T}{y_k^T\,\Delta x_k}$
Broyden	$B_k + \frac{y_k - B_k \Delta x_k}{\Delta x_k^T\,\Delta x_k}\,\Delta x_k^T$	$H_k + \frac{(\Delta x_k - H_k y_k)\Delta x_k^T H_k}{\Delta x_k^T H_k\,y_k}$
Broyden family	$(1-\varphi_k) B_{k+1}^{BFGS} + \varphi_k B_{k+1}^{DFP}, \qquad \varphi \in [0,1]$	
SR1	$B_k + \frac{(y_k - B_k\,\Delta x_k)(y_k - B_k\,\Delta x_k)^T}{(y_k - B_k\,\Delta x_k)^T\,\Delta x_k}$	$H_k + \frac{(\Delta x_k - H_k y_k)(\Delta x_k - H_k y_k)^T}{(\Delta x_k - H_k y_k)^T y_k}$

library(numDeriv)



grad {numDeriv} R Documenta

Numerical Gradient of a Function

Description

Calculate the gradient of a function by numerical approximation.

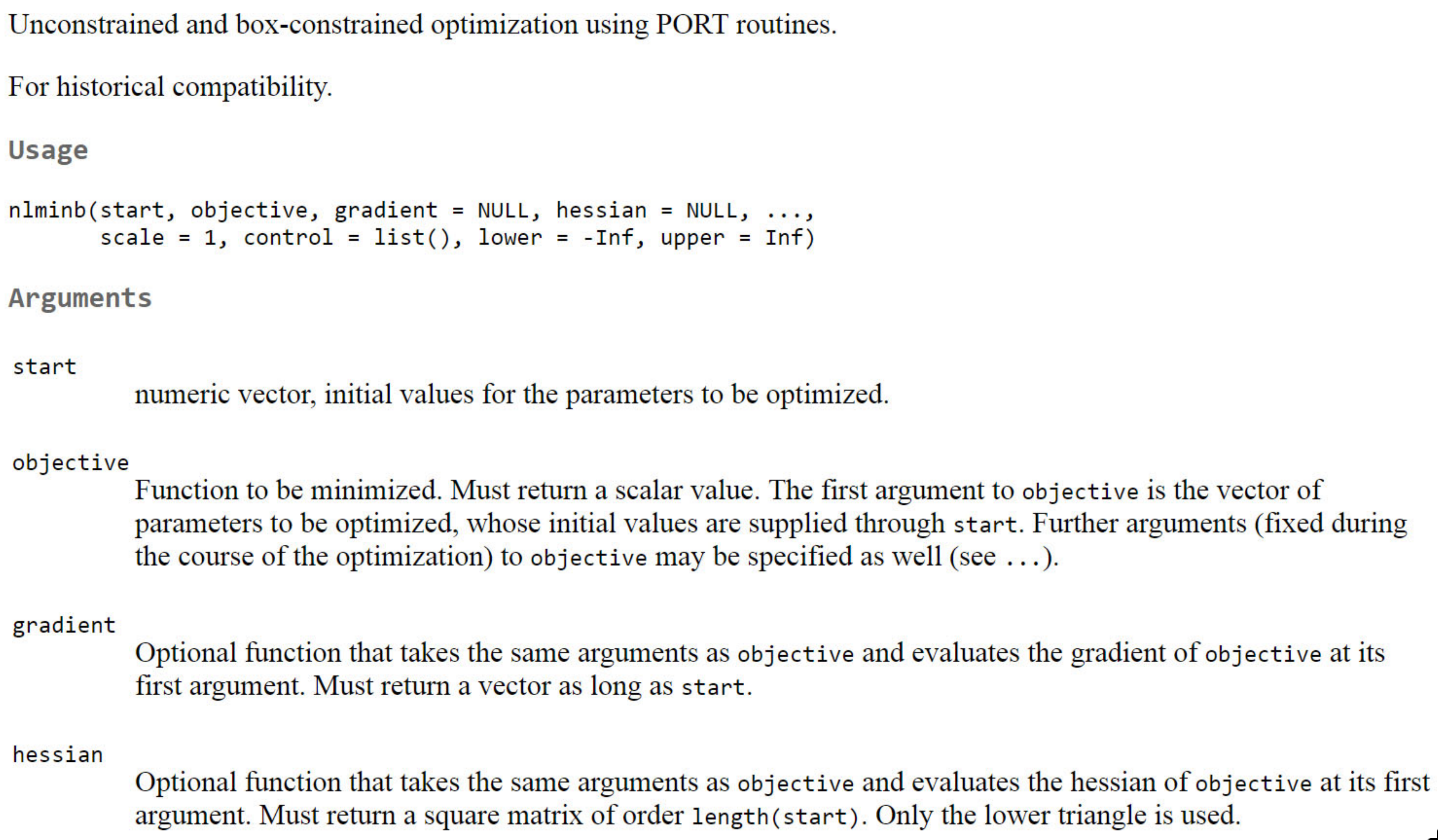
Usage

```
     grad(func, x, method="Richardson", side=NULL, method.args=list(), ...)

    ## Default S3 method:
grad(func, x, method="Richardson", side=NULL,
      method.args=list(), ...)
```

Arguments

`func`	a function with a scalar real result (see details).
`x`	a real scalar or vector argument to func, indicating the point(s) at which the gradient is to be calculated.
`method`	one of `"Richardson"`, `"simple"`, or `"complex"` indicating the method to use for the approximation.

nlminb()



Unconstrained and box-constrained optimization using PORT routines.

For historical compatibility.

Usage

```
nlminb(start, objective, gradient = NULL, hessian = NULL, ...,
       scale = 1, control = list(), lower = -Inf, upper = Inf)
```

Arguments

`start`
: numeric vector, initial values for the parameters to be optimized.

`objective`
: Function to be minimized. Must return a scalar value. The first argument to `objective` is the vector of parameters to be optimized, whose initial values are supplied through `start`. Further arguments (fixed during the course of the optimization) to `objective` may be specified as well (see `...`).

`gradient`
: Optional function that takes the same arguments as `objective` and evaluates the gradient of `objective` at its first argument. Must return a vector as long as `start`.

`hessian`
: Optional function that takes the same arguments as `objective` and evaluates the hessian of `objective` at its first argument. Must return a square matrix of order `length(start)`. Only the lower triangle is used.

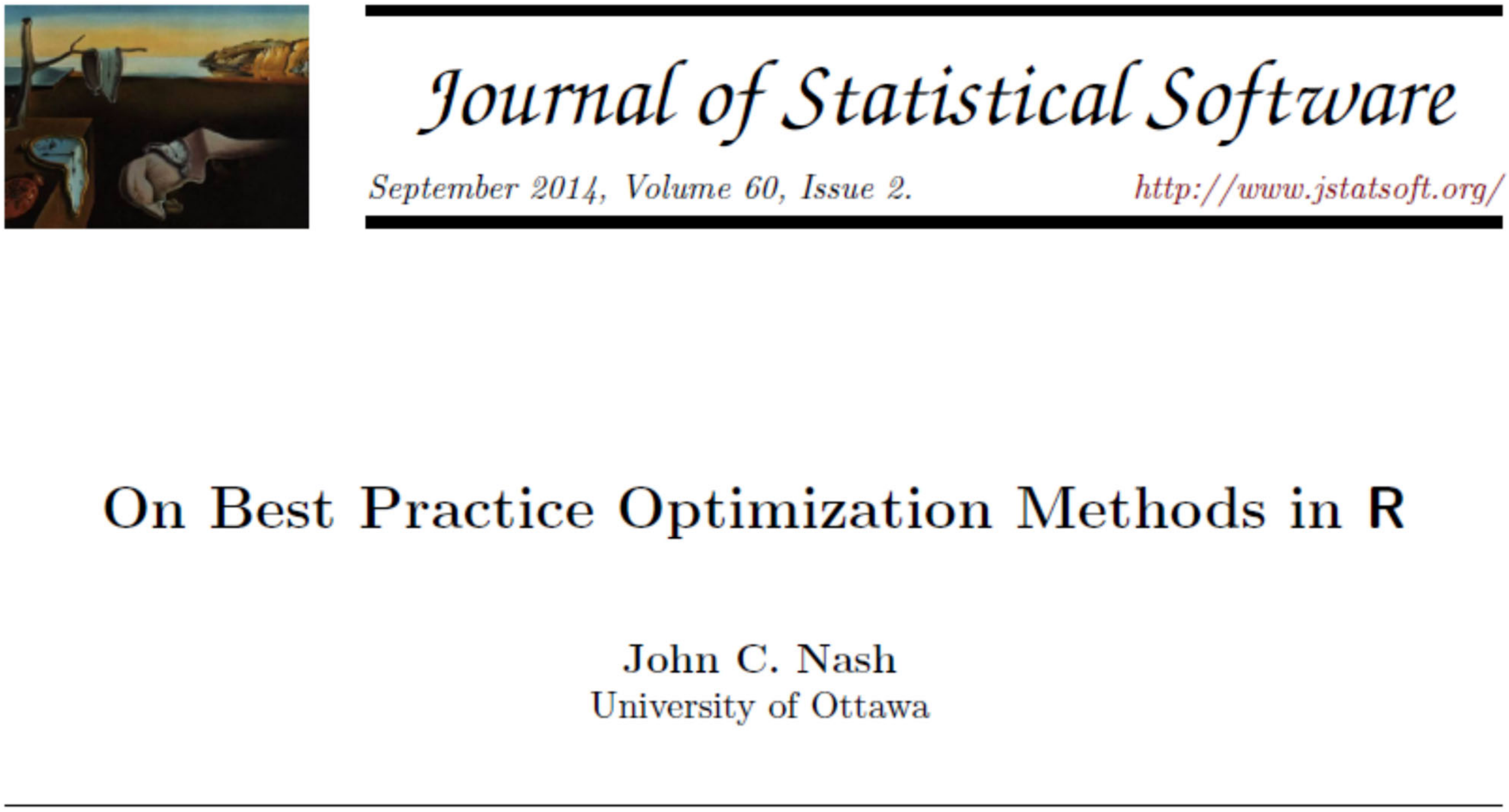
tend to find nlminb() is a better function minimzer

nlminb with length-weight data

```
> data.fit = nlminb(parm.start,LW_SSR)
> print(data.fit)
$par
[1] 2.201199 2.092770

$objective
[1] 2123.409

$convergence
[1] 0

$iterations
[1] 5

$evaluations
function gradient
      6       14

$message
[1] "relative convergence (4)"
```

```
Optim results
$par
[1] 2.214256 2.092548

$value
[1] 2123.409
```

```
lsfit results
Intercept        X
 2.201257  2.092769
```



Journal of Statistical Software

September 2014, Volume 60, Issue 2. http://www.jstatsoft.org/

On Best Practice Optimization Methods in R

John C. Nash
University of Ottawa

More general estimation

- The least squares method is more appropriate for situations where you feel the errors are additive, and the errors have approximately the same variation
- If you think your errors are multiplicative, then a log-transformation can make the errors additive,
- but your model may no longer be linear
- Log-transformations may not be good for count responses (i.e. 0’s? – can’t log), and proportions

More general estimation

– The method of maximum likelihood is more general, and suitable for almost any type of data and model

– Maximum likelihood estimates (MLE’s) of parameters are those values that maximize the likelihood of the data

– The likelihood is a fit function based on the probability distribution of the responses

– Covariates, like length, are not considered random.

Probability distribution

– Is a mathematical function that you feel well describes the random variability in your response.

– A coin toss: a probability distribution can give the probability (between 0 and 1) that you get 3 heads in 6 tosses, etc.

– A cumulative distribution function (CDF) gives the probability that your response lies within some range of values

– i.e. prob of getting 3 heads or more in 6 tosses

Probability density

– There are some technical distinctions to how we describe probability for count responses versus real valued measurements (i.e. 1.2345, etc)

– The probability of a specific value for a real valued measurement is technically zero, because there are uncountably many other potential values

– unlike count data where the possible responses are countable and therefore can have specific probabilities

– For real data, we examine probability for intervals

Probability density function

– That is,

$$\Pr(a \le Y \le b) = \int_a^b pdf(y)dy$$

- *pdf(y)* is the probability density function
- you can think of it as giving the probability of *y*, but technically it gives the probability of a very small interval about *y*.
- The $\int dy$ is integration, which is analogous to adding up over *pdf(y)* many small intervals about *y*.

Probability density function

- Note that *Pr(-∞ ≤ Y ≤ ∞) = 1*.
- The expected value, or mean, of a random variable *Y* is defined to be $E(Y)=\int_{-\infty}^{\infty} y\,pdf(y)dy$
- It is a measure of the central location of *Y*
- The variance of *Y* is $Var(Y)=\int_{-\infty}^{\infty}\{y-E(Y)\}^2\,pdf(y)dy$
- and is a measure of the spread of the possible values of the random variable *Y*

The normal pdf

- Many people have heard of the normal distribution
- It is commonly used to describe real valued measurements (length, weight, etc);
- Less useful for counts and proportions
- The normal *pdf* is $$pdf(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ \frac{-(y-\mu)^2}{2\sigma^2} \right\}$$
- The mean is μ, and the variance is σ^2.
- The standard deviation is σ = *sqrt{Var(Y)}*
- μ and σ^2 are parameters of the *pdf*, usually they are unknown

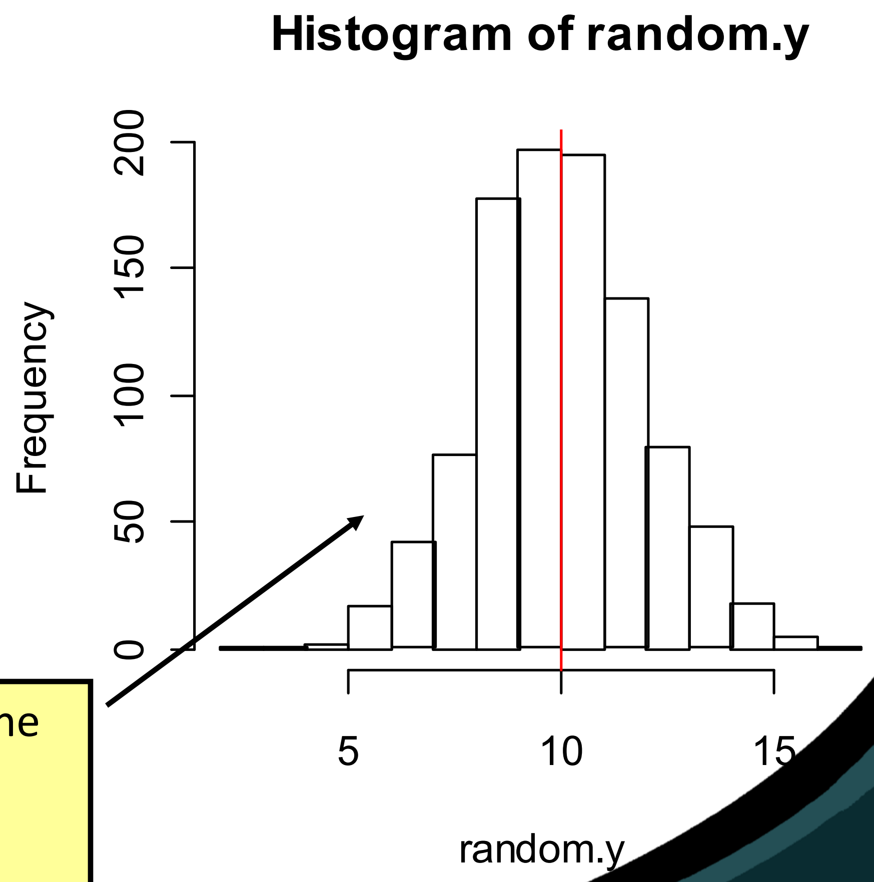
The normal pdf

- The normal distribution is symmetric about μ.
- It is really easy in R to look at this

```
random.y = rnorm(n=1000,mean=10,sd=2)
hist(random.y)
abline(v=10,col='red')
```

Produce a histogram

Normal random number generator



The histogram gives the number of y's (from the n=1000 generated) that are in each interval indicated on the x-axis; i.e. 200 y's between 9 and 10, etc

The normal pdf

- This was a large sample size, and

```
> mean(random.y)
[1] 9.993741
> var(random.y)
[1] 3.923442
> sqrt(var(random.y))
[1] 1.980768
```

True mean was 10, and true std. dev. was 2. A sample of size 1000 can give very precise estimates of population parameters

The normal pdf

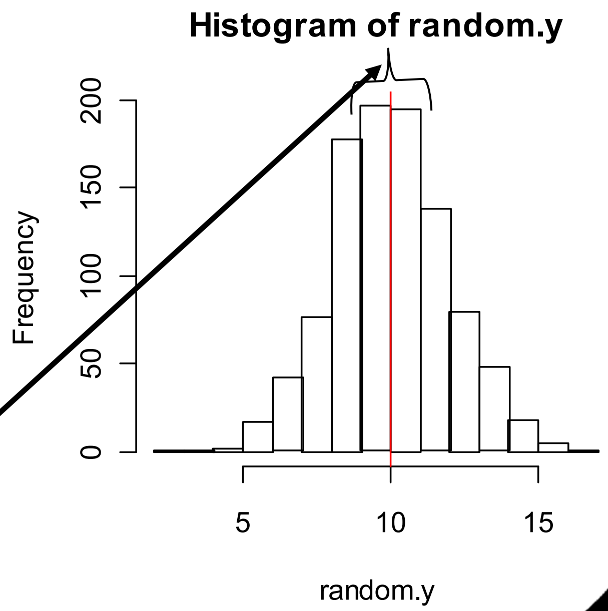
- Values close to the mean μ are more likely that values far from μ.

fraction of random y’s less than 7

fraction of random y’s >= 13

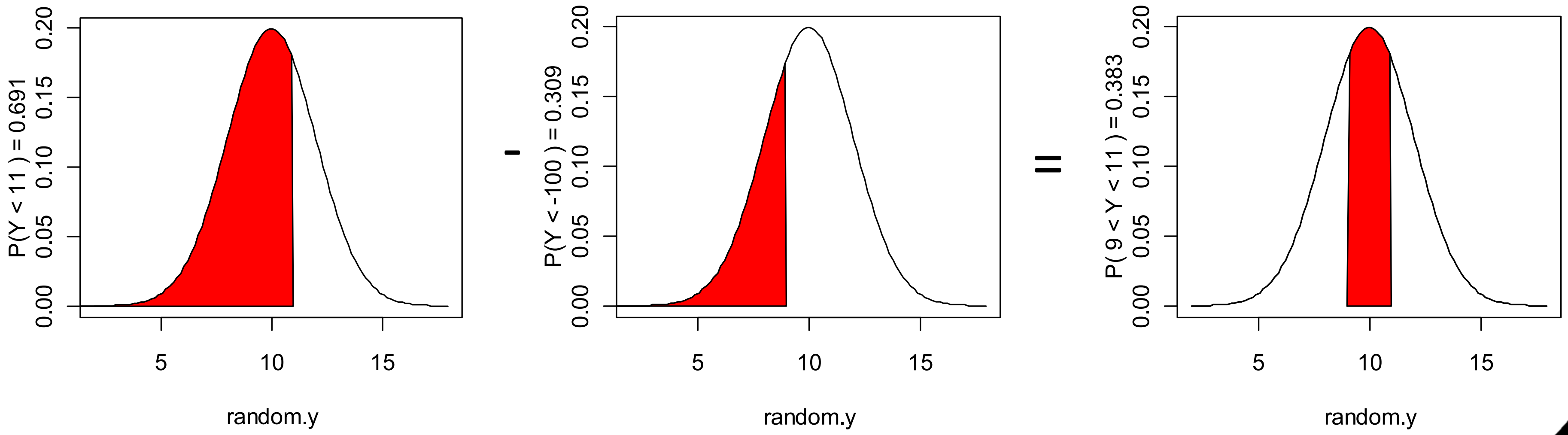
```
> n=1000
> length(random.y[random.y<=7])/n
[1] 0.063
> length(random.y[random.y>=13])/n
[1] 0.072
> length(random.y[(random.y>=9)&(random.y<=11)])/n
[1] 0.392
```

fraction of random y’s
>=9 and <= 11
i.e. 392/1000=0.392



The normal pdf

- It is easy to compute these probabilities exactly in R
- Pnorm(…) computes *$Pr(Y \leq q)$*
- Note that *$Pr(q_1 \leq Y \leq q_2) = Pr(Y \leq q_2) - Pr(Y \leq q_1)$*



```
pnorm(11, mean=10, sd=2) - pnorm(9, mean=10, sd=2)
[1] 0.3829249
```

close to 0.392 for the 1000 random y's

The normal pdf

- Quantiles are also easy
- A quantile is the value y_α such that $\Pr(Y \le y_\alpha) = \alpha$

```
> qnorm(0.95, mean=10, sd=2)
[1] 13.28971
> qnorm(0.05, mean=10, sd=2)
[1] 6.710293
>
```

95% of values are less than 13.29, and 5% greater

5% of values are less than 6.71, and 95% are greater

- 90% of responses will be less than 13.29 but greater than 6.71; that is, $Pr(6.71 \le Y \le 13.29) = 0.9$

```
length(random.y[(random.y>=6.71)&(random.y<=13.29)])/n
[1] 0.902
```

Other pdf’s

For the beta distribution see dbeta.

For the binomial (including Bernoulli) distribution see dbinom.

For the Cauchy distribution see dcauchy.

For the chi-squared distribution see dchisq.

For the exponential distribution see dexp.

For the F distribution see df.

For the gamma distribution see dgamma.

For the geometric distribution see dgeom. (This is also a special case of the negative binomial.)

For the hypergeometric distribution see dhyper.

For the log-normal distribution see dlnorm.

For the multinomial distribution see dmultinom.

For the negative binomial distribution see dnbinom.

For the normal distribution see dnorm.

For the Poisson distribution see dpois.

For the Student's t distribution see dt.

For the uniform distribution see dunif.

For the Weibull distribution see dweibull.

The likelihood

- For discrete data, the *pdf* gives the probability of observing the particular values in our data, given values for the parameters of the *pdf*.
- For continuous data, the *pdf* gives the probability of observing the particular values in our data (really a very small range around the data values), given values for the parameters of the *pdf*.
- For example, given values for the mean is μ, and the variance is σ^2, we can compute the normal *pdf*.

The likelihood

- The likelihood is a function used to describe what values of the *pdf* parameters are more likely given the data we observe. It is a fit function.
- Recall that we denote the set of parameters as θ.
- Given an observed value of a random variable Y, the likelihood function for θ is $L(\theta|y_{obs}) = pdf(Y=y_{obs}| \theta)$

The | means given

- We denote random variables with upper case, and observations of the random variables with lower case

The likelihood

- If the have *n* independent observations then the likelihood is the product,

$$L(\theta \mid y_1,\ldots,y_n) = pdf(y_1 \mid \theta) \times pdf(y_2 \mid \theta) \times \ldots \times pdf(y_n \mid \theta)$$

$$= \prod_{i=1}^{n} pdf(y_i \mid \theta)$$

- In general, the probability of independent events is the product of their individual probabilities.

The Normal likelihood

- For *n* independent observations from a normal distribution with parameters μ and σ^2 for the mean and variance, the likelihood is

$$L(\theta = \{\mu, \sigma^2\} \mid y_1, \ldots, y_n) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ \frac{-(y_i - \mu)^2}{2\sigma^2} \right\}$$

$$= \left(2\pi\sigma^2\right)^{-n/2} \exp\left\{ \frac{-\sum_{i=1}^{n}(y_i - \mu)^2}{2\sigma^2} \right\}$$

The likelihood

- The likelihood function $L(\theta|y_{obs})$ is a fit function, and we will use it to estimate values for θ that are "most likely" with the data we observe.
- We will call these estimates the maximum likelihood estimates (MLE's) of θ.
- They will maximize $L(\theta|y_{obs})$.
- A useful fact: maximizing $L(\theta|y_{obs})$ is the same as maximizing the log of $L(\theta|y_{obs})$.

The likelihood

- We will denote *$\log\{L(\theta|y_{obs})\}$* as *$l(\theta|y_{obs})$*.
- so lower case *l* means log of the likelihood, which is shortened to log-likelihood.
- Many optimizers minimize by default, and this is no problem because the MLE of *θ* is simply the value of *θ* that minimizes the negative log-likelihood.
- I will also use *nll* for negative log-likelihood

The Normal log-likelihood

- For *n* observations from a normal distribution with parameters *μ* and *σ*² for the mean and variance, the log-likelihood is

$$l(\{\mu,\sigma^2\}\mid y_1,\dots,y_n)=\frac{-n}{2}\log\left(2\pi\sigma^2\right)-\frac{\sum_{i=1}^{n}(y_i-\mu)^2}{2\sigma^2}$$

- I get tired of writing $y_1,\dots,y_n$ so lets drop this, and remember that the likelihood depends on data

- this can also be written as

$$-l(\{\mu,\sigma^2\})=\frac{n}{2}\log\left(2\pi\sigma^2\right)+\frac{\sum_{i=1}^{n}(y_i-\bar{y})^2}{2\sigma^2}+\frac{n(\bar{y}-\mu)^2}{2\sigma^2}$$

The negative log-likelihood, nll

MLE for μ

$$nll(\mu,\sigma^2) = \frac{n}{2}\log(2\pi\sigma^2) + \frac{\sum_{i=1}^{n}(y_i - \bar{y})^2}{2\sigma^2} + \boxed{\frac{n(\bar{y}-\mu)^2}{2\sigma^2}}$$

- Let $\hat{\mu}_{mle}$ denote the MLE for μ
- For any value of σ^2, the *nll* is minimized when $\hat{\mu}_{mle} = \bar{y}$
- This is obvious by inspection of the *nll*, but you could also take the derivative with respect to μ and then find the value of μ that makes the derivative 0. This will be $\hat{\mu}_{mle} = \bar{y}$. Try this if you remember how to do derivatives

MLE for σ^2

- It is not hard using the derivative method to show that

$$\hat{\sigma}^2_{mle} = \frac{\sum_{i=1}^{n} (y_i - \bar{y})^2}{n}$$

- this estimator is a slight under-estimate of σ^2. An unbiased estimator is

$$\hat{\sigma}^2_{bc} = \frac{\sum_{i=1}^{n} (y_i - \bar{y})^2}{n-1}$$

bc means bias-corrected

MLE’s in R

- This is straight-forward.
- Need to write a function whose arguments are the parameters,
- and the function returns the *nll* for the parameters

Normal MLE's in R

```
> n=20
> mu=5
> s2=1;
> y = rnorm(n=n,mean=mu,sd=sqrt(s2))
> ybar=mean(y)
> yvar = sum((y-ybar)^2)/n
> print(y,digits=3)
 [1] 6.02 6.36 2.80 5.34 3.84 3.96 3.89 6.52 5.48 5.33
[11] 5.46 5.47 5.62 5.71 3.89 4.14 4.33 4.38 4.90 6.23
> print(ybar)
[1] 4.98454
> print(yvar)
[1] 0.989681
```

$\mu = 5, \sigma^2 = 1$

MLE of μ

MLE of σ^2

n=20 random observations from a Normal distribution with *$\mu = 5, \sigma^2 = 1$*

Minimize nll in R

the function to minimize

```
normal.nll <- function(parm){
  mu = parm[1]
  s2 = parm[2]
  prob = dnorm(y,mean=mu,sd=sqrt(s2))
  nll = -sum(log(prob))
  return(nll)
}
```

Minimize nll in R

```
> parm.start = c(0,1)
> mle.fit = optim(parm.start,normal.nll)
Warning messages:
1: In sqrt(s2) : NaNs produced
2: In sqrt(s2) : NaNs produced
> print(mle.fit)
$par
[1] 4.9843782 0.9901456

$value
[1] 28.27505

$counts
function gradient
      95       NA

$convergence
[1] 0

$message
NULL
```

need to give optim the name of the fit function you created

need to give optim starting values for the parameters

It works. You can check that these are close to the MLE's of μ and σ^2 that we computed earlier using exact equations

Better Approach to estimate σ^2

```
normal.nll <- function(parm){
  mu = parm[1]
  s2 = exp(parm[2])
  prob = dnorm(y,mean=mu,sd=sqrt(s2))
  nll = -sum(log(prob))
  return(nll)
}
```

estimate $\log(\sigma^2)$ and
$\sigma^2 = \exp(\log(\sigma^2))$

parm[2] can be + or -,
but s2 always +

Better Approach

```
> parm.start = c(3,0)
> mle.fit = optim(parm.start,normal.nll)
> print(mle.fit)
$par
[1]  4.98448207 -0.01014594

$value
[1] 28.27505

$counts
function gradient
      63       NA

$convergence
[1] 0

$message
NULL

> exp(mle.fit$par[2])
[1] 0.9899054
```

estimate of *$\log(\sigma^2)$*

estimate of σ^2, which is close to the result we got before

Normal MLE’s

- It turns out that MLE’s of regression parameters are the same as least squares estimates when the data are normally distributed with the same variance.
- Lets check this with the growth data example.

Normal MLE's for growth data example

```
fname = c("E:\\FRM6002_2011\\lecture5\\LW.dat")
tdata = read.table(fname,header=TRUE)

normal.nll <- function(parm){
  mu= parm[1] + parm[2]*tdata$length
  s2 = exp(parm[3])
  prob = dnorm(tdata$weight,mean=mu,sd=sqrt(s2))
  nll = -sum(log(prob))
  return(nll)
}
```

returns pdf for weight, with mean mu and st. dev. sqrt(s2)

has to be either a scalar (single value) or a vector of length n

Normal MLE's for growth data example

```
> parm.start = c(2,2,2)
> normal.nll(parm.start)
[1] 218.2105
> mle.fit =
optim(parm.start,normal.nll,method = "BFGS",
+ control=list(maxit=1000,trace=1))
initial  value 218.210452
iter  10 value 68.506295
iter  20 value 68.474572
final  value 68.474551
converged
```

more efficient method

print some results during minimization

Needed to increase number of iterations

Normal MLE's for growth data example

lsfit results
Intercept X
2.201257 2.092769

```
> print(mle.fit)
$par
[1] 2.202959 2.092741 4.770448

$value
[1] 68.47455

$counts
function gradient
      48       22

$convergence
[1] 0
```

Also get an estimate of error variance

you can check that these are really close to what we computed earlier

0 is good – means it converged

```
> mle.fit = nlminb(parm.start,normal.nll)
> mle.fit
$par
[1] 2.201260 2.092769 4.770407
```

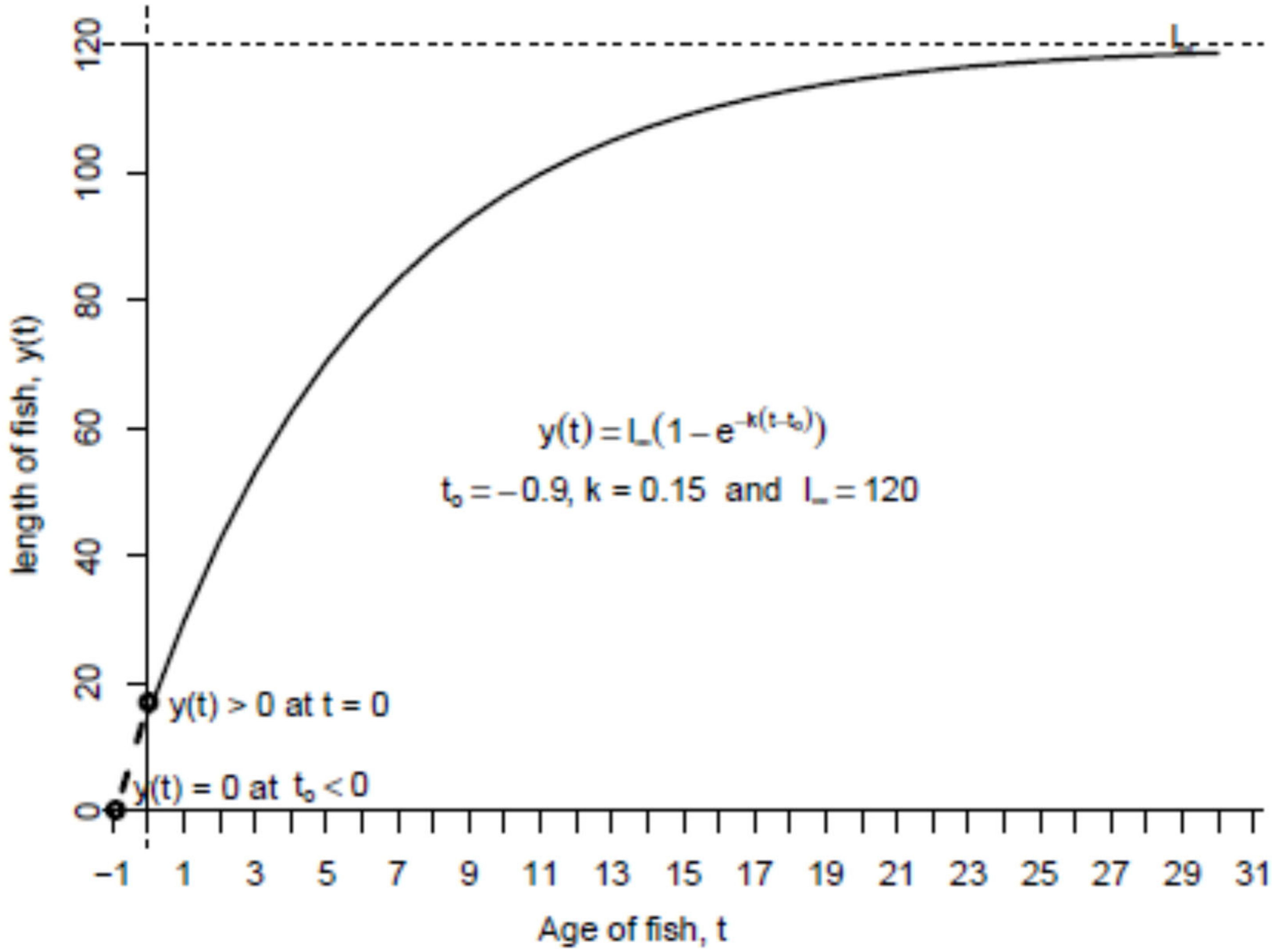
MLE’s

- We have looked at R code to get MLE’s of the parameters of the linear model $Y = \theta_o + \theta_1 * X + \varepsilon$
- We are now in good shape to get MLE’s for many types of models
- Including the ones in Haddon
- i.e. Lognormal, Gamma, Binomial, Poisson, ……
- We don’t need to worry about writing computer code for pdf’s

Von Bertalanffy growth curve

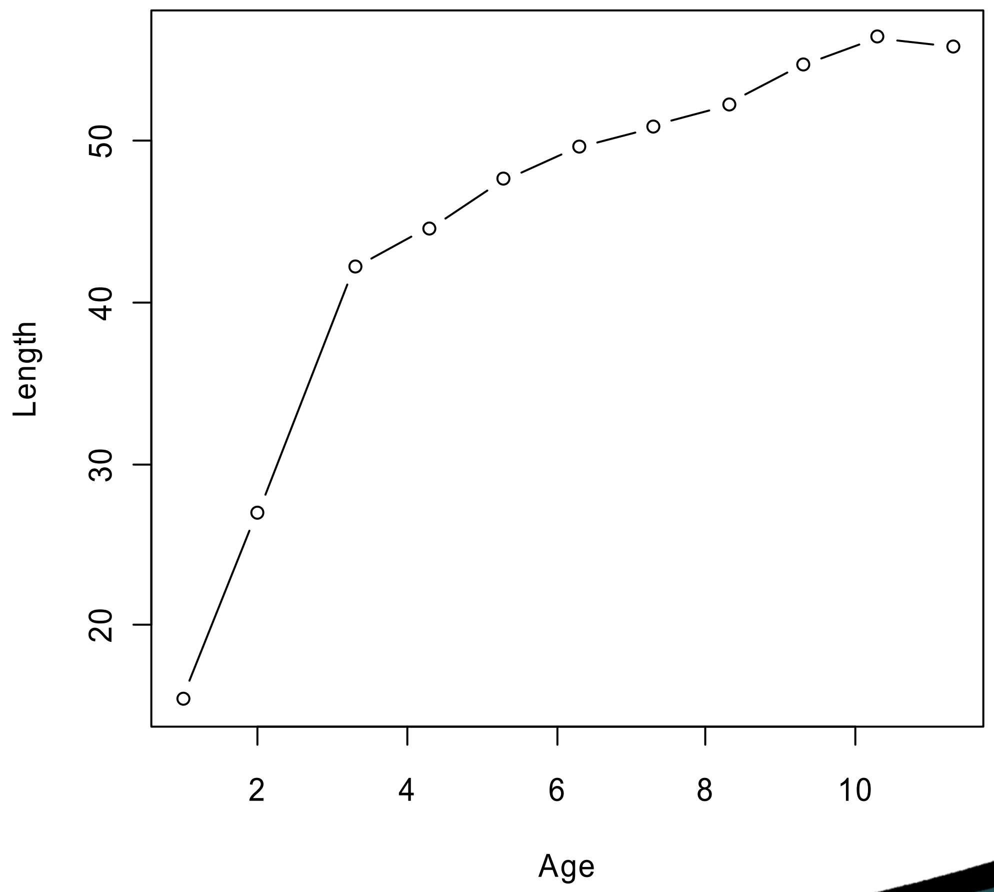
- The vonB growth curve (length vs age) is
- $L = L_\infty\{1- \exp(-K(age-a_o)\} + \varepsilon$
- The data are the length L at age
- The parameters are
  - L_∞ - maximum size,
  - K - growth rate parameter,
  - a_o - hypothetical age at zero length, and
  - ε is a random error term.

Von Bertalanffy growth curve



R estimation of vonB for ex 3.6 in Haddon

```
 print(age)
 [1]  1.0  2.0  3.3  4.3  5.3  6.3  7.3  8.3  9.3 10.3 11.3
> print(len)
 [1] 15.40 26.93 42.23 44.59 47.63 49.67 50.87 52.30 54.77 56.43 55.88
```



R estimation of vonB for ex 3.6 in Haddon

```
normal.nll <- function(parm){
  Linf = exp(parm[1])
  K = exp(parm[2])
  ao = exp(parm[3])
  s2 = exp(parm[4])
  mu = Linf*(1 - exp(-K*(age-ao)))
  prob = dnorm(len,mean=mu,sd=sqrt(s2))
  nll = -sum(log(prob))
  return(nll)
}
```

a 4 x1 vector of parameters, all logged

R estimation of vonB for ex 3.6 in Haddon

```
> parm.start = c(4,-1,-10,0)
> mle.fit = optim(parm.start,normal.nll,method="BFGS",
> control=list(maxit=1000,trace=1))
initial  value 29.733522
iter  10 value 19.485344
final  value 19.485292
converged
> print(mle.fit)
$par
[1]  4.0376957 -1.0510839 -9.9973615  0.7048937

$value
[1] 19.48529


$counts
function gradient
      44       14

$convergence
[1] 0

$message
NULL

> print(exp(mle.fit$par),digits=3)
[1] 5.67e+01 3.50e-01 4.55e-05 2.02e+00
```

Lognormal Distribution

- Is commonly used in biology, for continuous valued measurements
- It is appropriate when errors (ε's) are multiplicative
- Common to assume $Y = \mu \cdot exp(\varepsilon)$, where ε has a normal distribution with mean zero and variance σ^2.
- In this case the mean of Y is $\mu \cdot exp(\sigma^2/2)$, the median is μ, and the variance is complicated

Lognormal Distribution

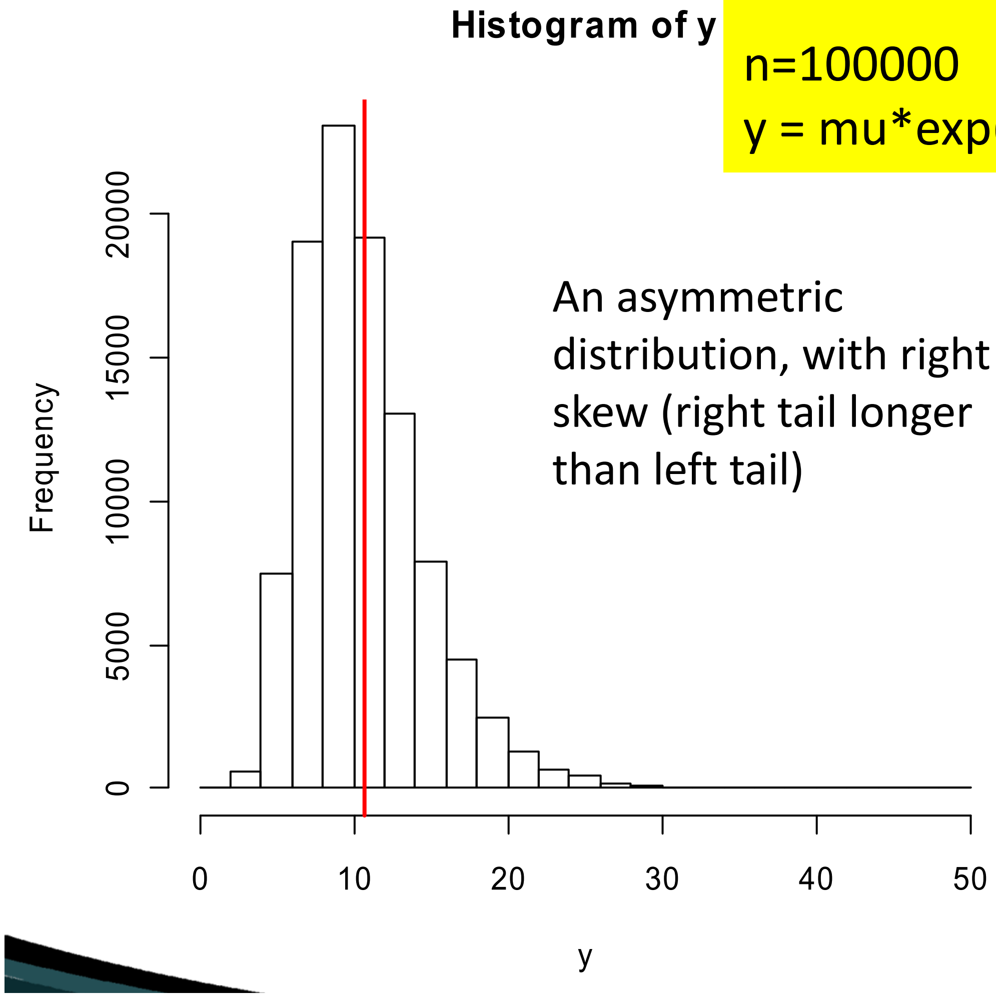
- The variance is $\varphi^2\mu^2$
- where $\varphi^2 = exp(\sigma^2)\{exp(\sigma^2) - 1\}$.
- The coefficient of variation is defined as the standard error divided by the mean $CV(Y) = \sqrt{Var(Y)}/_{E(Y)}$
- For the Lognormal, $CV = \{exp(\sigma^2) - 1\}^{1/2} \approx \sigma$ when σ is small.
- The Lognormal is sometimes called a constant CV distribution

Lognormal in R

Solve for in σ^2 in $\varphi^2 = \exp(\sigma^2)\{\exp(\sigma^2) - 1\}$

Hint. Let $X = \exp(\sigma^2)$ and complete the square: $X^2 - X$

```
mu=10
phi=0.4
sigma2 = log(0.5 + sqrt(phi**2 + 0.25))

n=100000
y = mu*exp(rnorm(n,mean=0,sd=sqrt(sigma2)))
```



An asymmetric distribution, with right skew (right tail longer than left tail)

Lognormal in R

```
> EY = mu*exp(sigma2/2)
> VY = (phi**2)*(mu**2)
> print(EY)
[1] 10.67854
> print(mean(y))
[1] 10.69479
> print(median(y))
[1] 10.03076
> print(VY)
[1] 16
> print(var(y))
[1] 15.89633
```

true values

μ = 10, which is very close to the median. The mean is somewhat greater than μ.

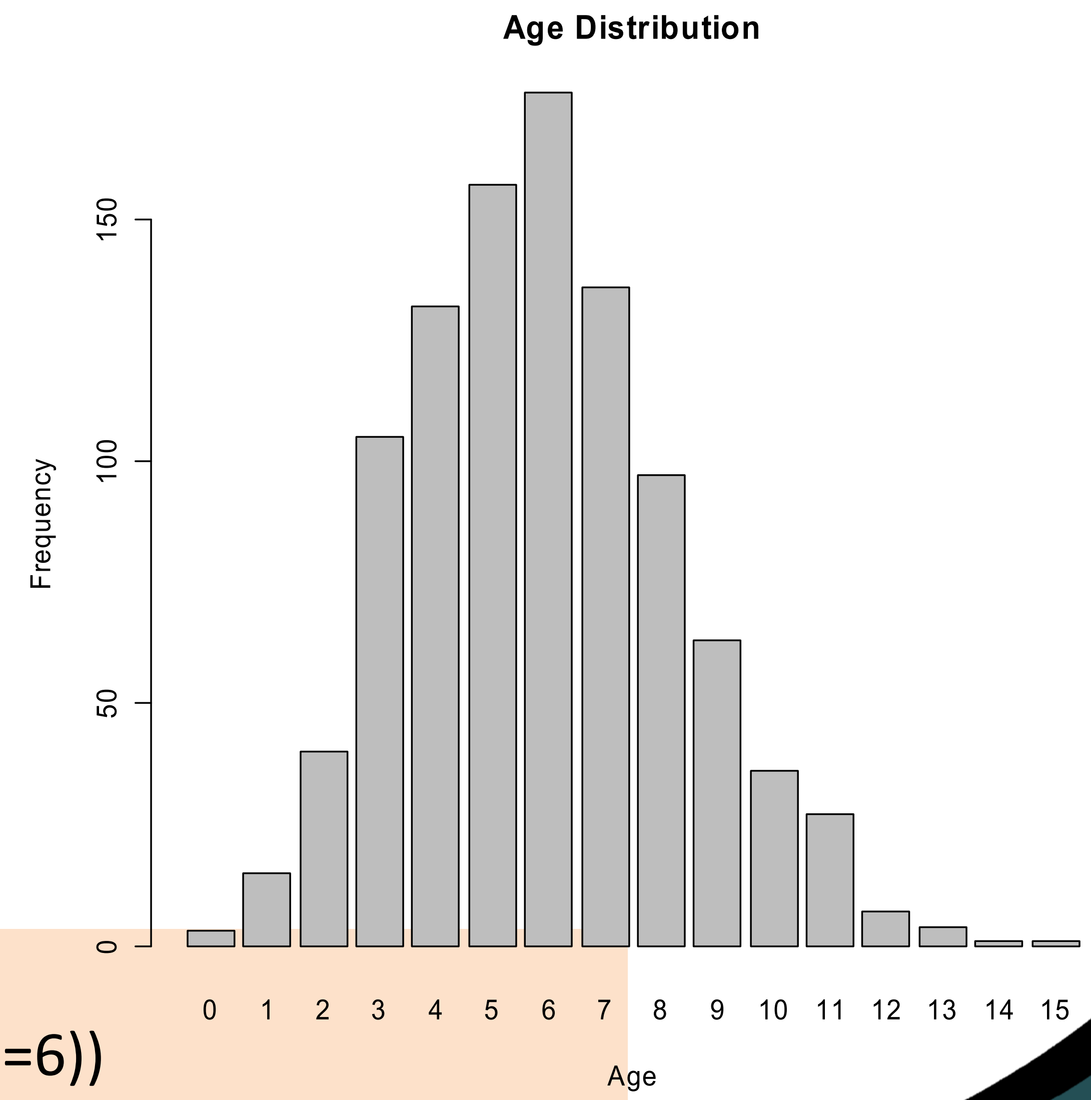
Lognormal variation increases with the mean

- In the regression setting, when the mean is a function of covariates, then the lognormal distribution is often used when it seems that the error variance increases with the regression mean.

- Lets look at an example of this, using the vonB growth curve.

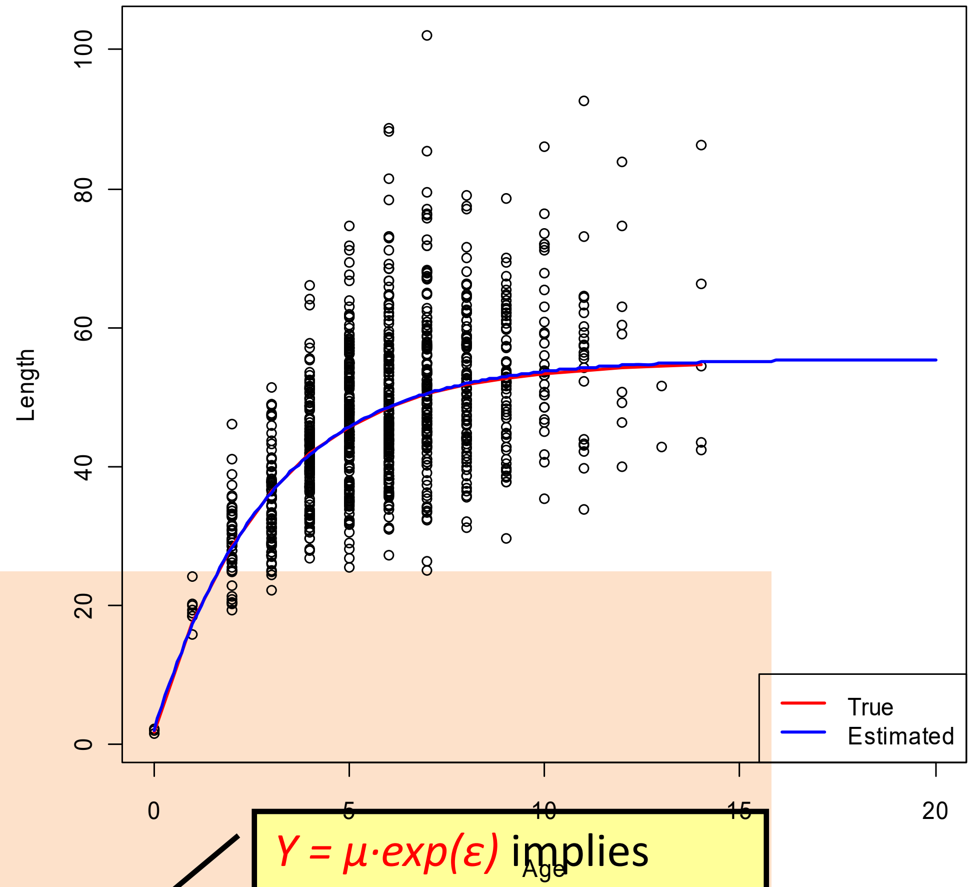
- Lets pretend we have ages and lengths for a 1000 fish – and the true vonB parameters are $L_\infty = 55$, $K = 0.35$, and $a_o = 0$.

Lognormal VonB data – the ages



```
n=1000
age = sort(rpois(n,lambda=6))
barplot(table(age),xlab='Age',ylab='Frequency',
main='Age Distribution')
```

Lognormal VonB data – the lengths



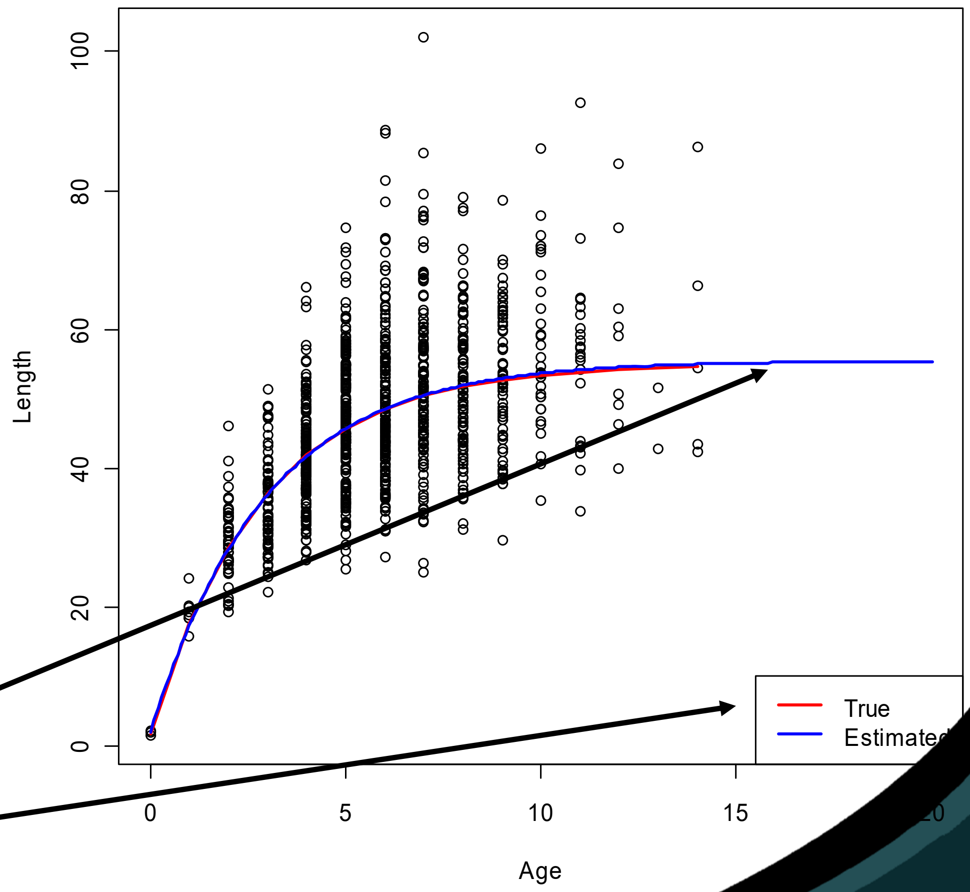
```
Linf=55
K=0.35
ao=-0.1
sigma=0.2
mu = Linf*(1 - exp(-K*(age-ao)))
len = rlnorm(n, meanlog = log(mu), sdlog = sigma)
plot(age,len,xlab='Age',ylab="Length",type='p',xlim=c(0,20))
lines(age,mu,lty=1,lwd=2,col='red')
```

Y = μ·exp(ε) implies
log(Y) = log(μ) + ε,

Lognormal VonB data – the fit

```
vonb <- function(Linf,K,ao,age){
  mu = Linf*(1 - exp(-K*(age-ao)))
  return(mu)
}

lognormal.nll <- function(parm){
  Linf = exp(parm[1])
  K = exp(parm[2])
  ao = parm[3]
  sigma = exp(parm[4])
  mu = vonb(Linf,K,ao,age)
  prob = dlnorm(len,mean=log(mu),sd=sigma)
  nll = -sum(log(prob))
  return(nll)
}

parm.start = c(3,-1,-0.1,-1)
mle.fit = optim(parm.start,lognormal.nll)

pred.age = seq(0,20,by=0.1)
Linf.hat = exp(mle.fit$par[1])
K.hat = exp(mle.fit$par[2])
ao.hat = mle.fit$par[3]
pred.mu = vonb(Linf.hat,K.hat,ao.hat,pred.age)
lines(pred.age,pred.mu,lty=1,lwd=2,col='blue')

legend("bottomright",c("True","Estimated"),
lty=1,col=c('red','blue'),lwd=2)
```

Better practise to have different functions for (1) your model of the mean, and (2) fit function



Lognormal vonB for ex 3.6 in Haddon

```
 print(age)
 [1]  1.0  2.0  3.3  4.3  5.3  6.3  7.3  8.3  9.3 10.3 11.3
> print(len)
 [1] 15.40 26.93 42.23 44.59 47.63 49.67 50.87 52.30 54.77 56.43 55.88
> parm.start = c(3,-1,-0.1,-1)
> mle.fit = optim(parm.start,lognormal.nll)
> print(mle.fit$par)
[1]  4.0265506 -0.9627808  0.1660191 -3.4528217
> print(mle.fit$converge)
[1] 0
> vonb.parms = c(exp(mle.fit$par[1]),exp(mle.fit$par[2]),
+ mle.fit$par[3])
> names(vonb.parms) = c('Linf','K','ao')
> print(vonb.parms,digits=2)
 Linf     K    ao
56.07  0.38  0.17
> sigma = exp(mle.fit$par[4])
> print(sigma)
[1] 0.03165618
```

```
$par
[1]  4.0376957 -1.0510839 -9.9973615  0.7048937
```

Gamma Distribution

- Can be used in situations where you feel a constant CV model is appropriate
- That is, when the variance is proportional to the square of the mean
- It may be a more robust distribution than lognormal
- Usually gives similar results, unless the CV is large

Gamma vonB for ex 3.6 in Haddon

```
gamma.nll <- function(parm){
 Linf = exp(parm[1])
 K = exp(parm[2])
 ao = parm[3]
 sigma = exp(parm[4])
 mu = vonb(Linf,K,ao,age)
 var = (sigma*mu)**2
 s = var/mu
 a = mu/s;
 prob = dgamma(len,shape=a,scale=s)
 nll = -sum(log(prob))
 return(nll)
}
```

get R help on dgamma, to see how shape and scale are related to mean and variance

Gamma vonB for ex 3.6 in Haddon

```
> parm.start = c(5,-1,-0.1,-1)
> mle.fit = optim(parm.start,gamma.nll)
> print(mle.fit$par)
[1]  4.0262464 -0.9588169  0.1689758 -3.4481118
> print(mle.fit$converge)
[1] 0
> vonb.parms = c(exp(mle.fit$par[1]),exp(mle.fit$par[2]),
+ mle.fit$par[3])
> names(vonb.parms) = c('Linf','K','ao')
> print(vonb.parms,digits=2)
 Linf     K    ao
56.05  0.38  0.17
> sigma = exp(mle.fit$par[4])
> print(sigma)
[1] 0.03180564
```

estimates are really similar to lognormal

Binomial distribution

- Is a distribution used for counts
- where you are interested in the number of times an “event” occurs in n trials or observations
- i.e. number of mature fish in a sample of size n
- number of fish caught from n that contact the fishing gear
- the number of tagged fish caught from n releases

Binomial distribution

- If the probability of the event happening, referred to as a “success”, is *p*,
- and if this probability is the same for each of the *n* trials or observation events
- and if *Y* is the number of successes in *n* trials, then

$$\Pr(Y = y \mid n, p) = \binom{n}{y} p^{y} (1-p)^{n-y} \qquad \binom{n}{y} = \frac{n!}{y!(n-y)!}$$

- where *x! = x*(x-1)*(x-2)*…*2*1*
- *E(Y)=np, Var(Y) = np(1-p)*

Binomial distribution – Maturity example

```
> matdata = read.table(fname,header=TRUE)
> print(matdata)
   sex year age    prop  n cohort  y
1    5 1995   2 0.00000  2   1993  0
2    5 1996   3 0.00000 73   1993  0
3    5 1997   4 0.23471 14   1993  3
4    5 1998   5 0.36244 18   1993  7
5    5 1999   6 0.78855 43   1993 34
6    5 2000   7 0.94920 33   1993 31
7    5 2001   8 1.00000 21   1993 21
8    5 2002   9 1.00000 15   1993 15
9    5 2003  10 1.00000  4   1993  4
10   5 2004  11 1.00000  5   1993  5
11   5 2005  12 1.00000  3   1993  3
```

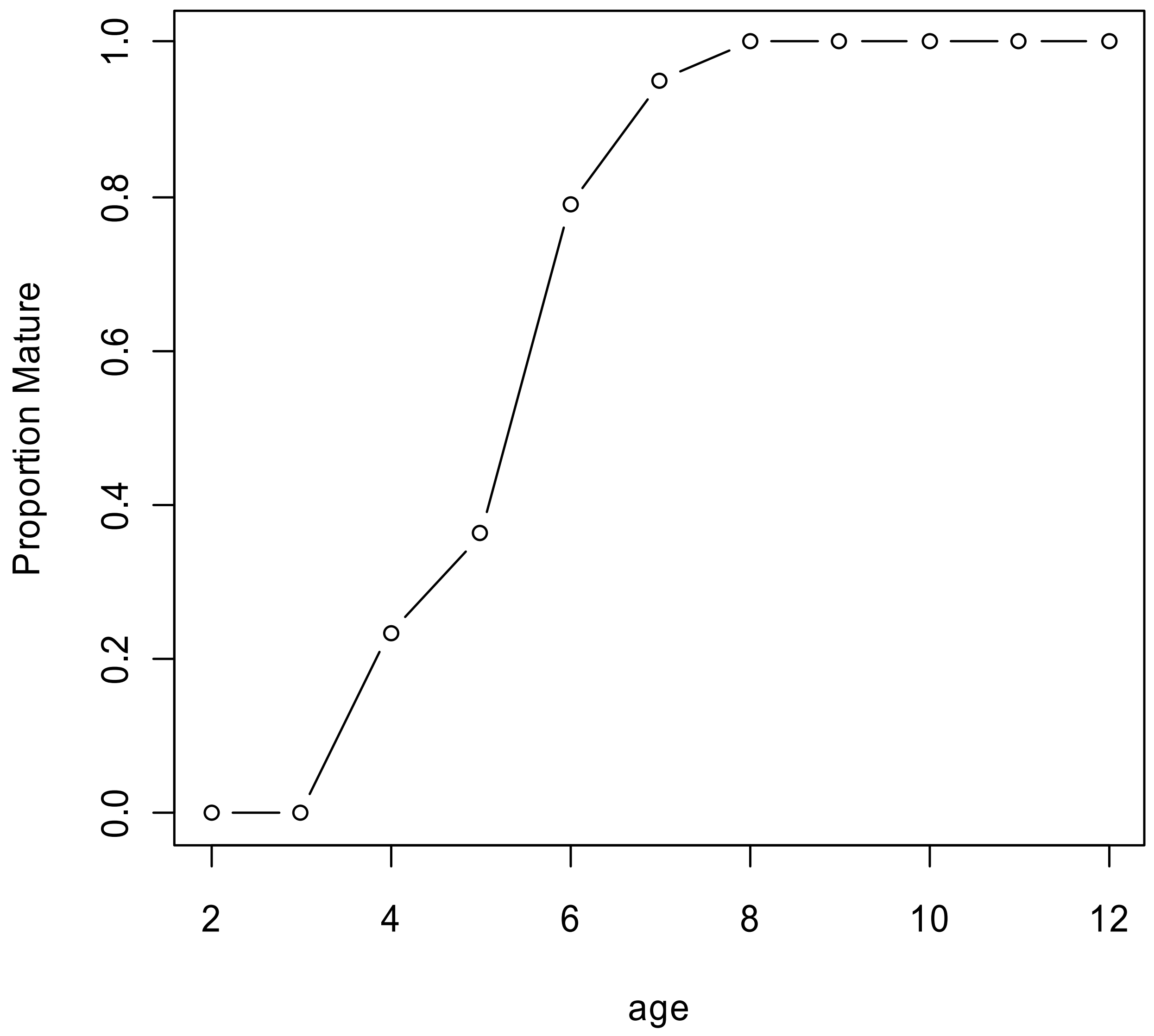
Binomial logistic regression

- Maturity should increase with age for a cohort
- ie once fish become mature they cannot become immature again
- Common to model the proportion mature at age using a logistic function

$$p(age) = \frac{\exp(\theta_o + \theta_1 age)}{1 + \exp(\theta_o + \theta_1 age)}$$

- *p(age)* increases smoothly as a function of age

Proportion mature



Binomial logistic regression - maturity

```
pmat <- function(parm,age){
  logitp = parm[1] + parm[2]*age
  prob.mat = exp(logitp)/(1+exp(logitp))
  return(prob.mat)
}

binomial.nll <- function(parm){
  prob.mat = pmat(parm,age);
  prob = dbinom(y,n,prob.mat)
  nll = -sum(log(prob))
  return(nll)
}

y = matdata$y
n = matdata$n
age = matdata$age
```

logistic R function

Binomial nll R function

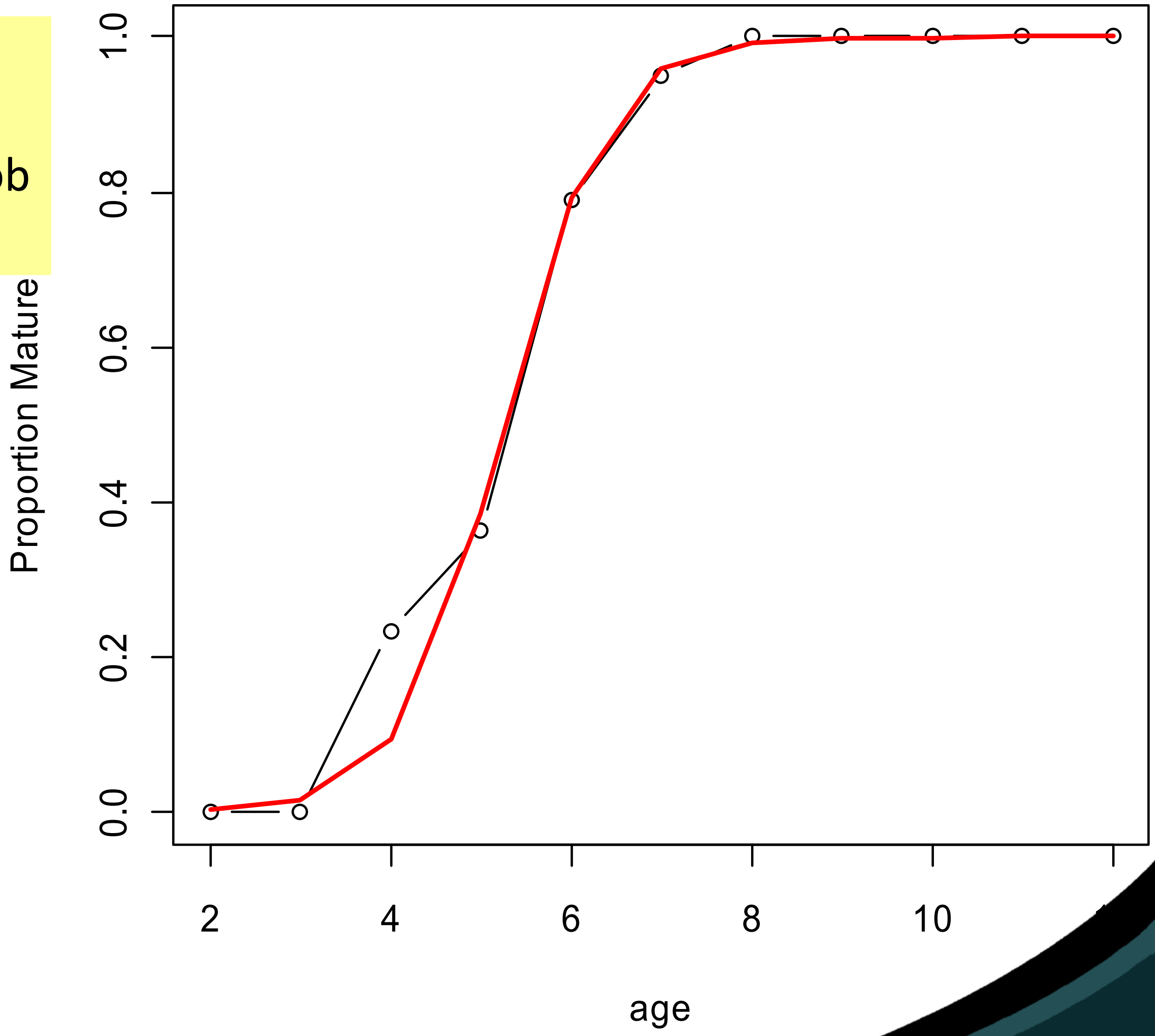
required for Binomial nll

R code to estimate the parameters for the logistic maturity model

```
> parm.start = c(-4,1)
> mle.fit =
optim(parm.start,binomial.nll)
> print(mle.fit$par)
[1] -9.474728  1.801821
> print(mle.fit$converge)
[1] 0
```

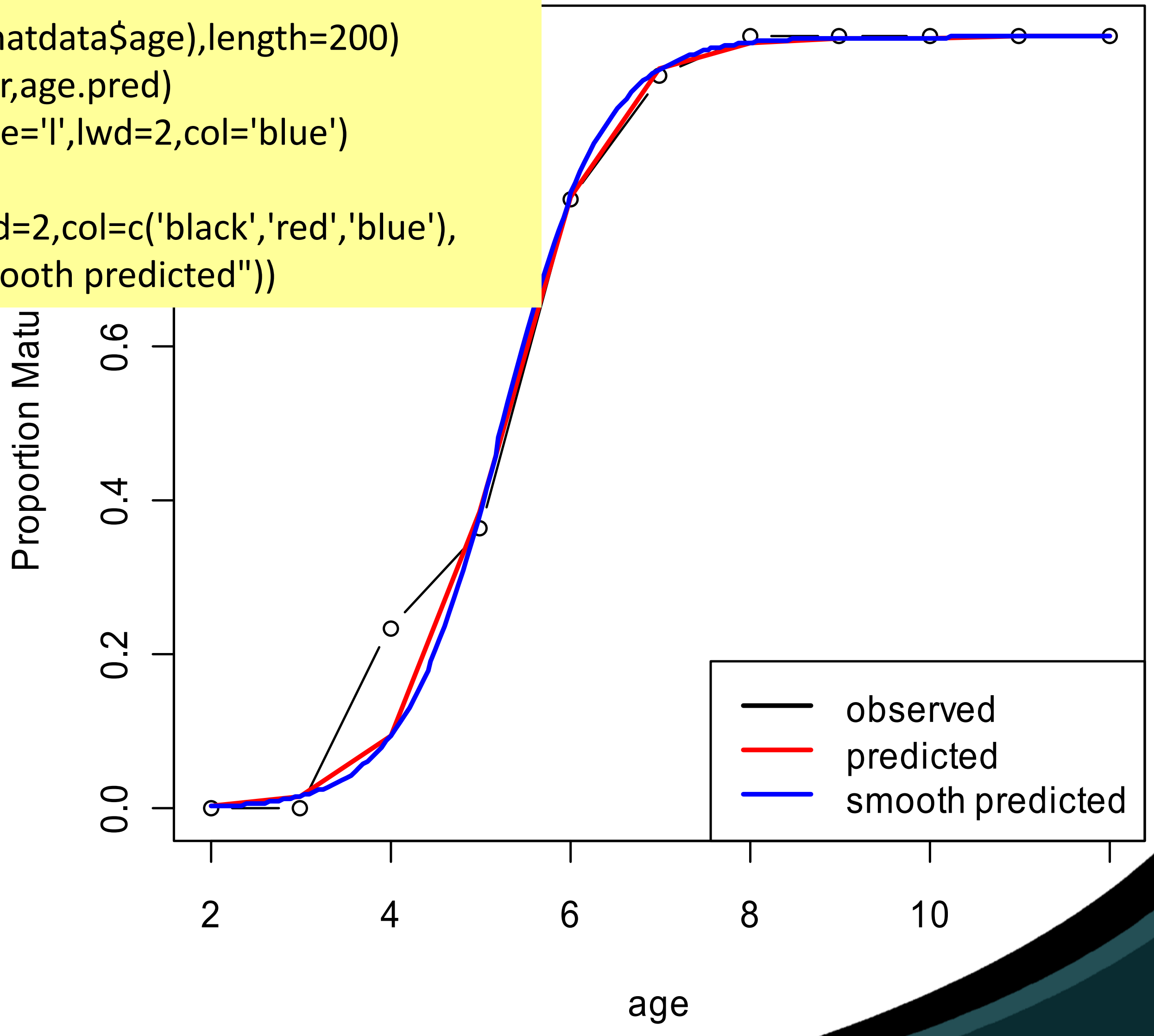
Plot the regression results

```
pred.prob =
pmat(mle.fit$par,age)
lines(matdata$age,pred.prob
,type='l',lwd=2,col='red')
```



Better plot of regression results

```
age.pred =
seq(min(matdata$age),max(matdata$age),length=200)
pred.prob1 = pmat(mle.fit$par,age.pred)
lines(age.pred,pred.prob1,type='l',lwd=2,col='blue')

legend('bottomright',lty=1,lwd=2,col=c('black','red','blue'),
c("observed","predicted","smooth predicted"))
```



Binomial logistic regression

- This model falls in the class of generalized linear models (GLIMs)
- A linear regression with covariates (e.g. age, etc) that is related to responses via a link function
- Can handle many types of distributions
- including the Binomial
- R has the GLM package for this
- Gives statistics (confidence intervals, etc)

Binomial logistic regression

$$p(age) = \frac{\exp(\theta_o + \theta_1 age)}{1 + \exp(\theta_o + \theta_1 age)}$$

$$\Rightarrow \log\left\{\frac{p(age)}{1 - p(age)}\right\} = \theta_o + \theta_1 age$$

the linear part

the logit "link" function that links *p* to *age*

Binomial logistic regression - maturity

```
Binfit <- glm(cbind(y,n-y) ~
age,family=binomial(link = "logit"),data=matdata)

> temp = summary(Binfit)
> print(temp$coefficients)

              Estimate   Std. Error   z value  Pr(>|z|)
(Intercept)   -9.475      1.3359     -7.092   1.319e-12
age            1.802      0.2413      7.468   8.132e-14
```

Pr(Z>|z value|) +
Pr(Z<-|z value|)
=2Pr(Z<-|z value|)

```
> 2*pnorm(-7.092)
[1] 1.321875e-12
```

square root of the variance of the estimate

estimate/std.error – gives an indication how different the estimate is from 0

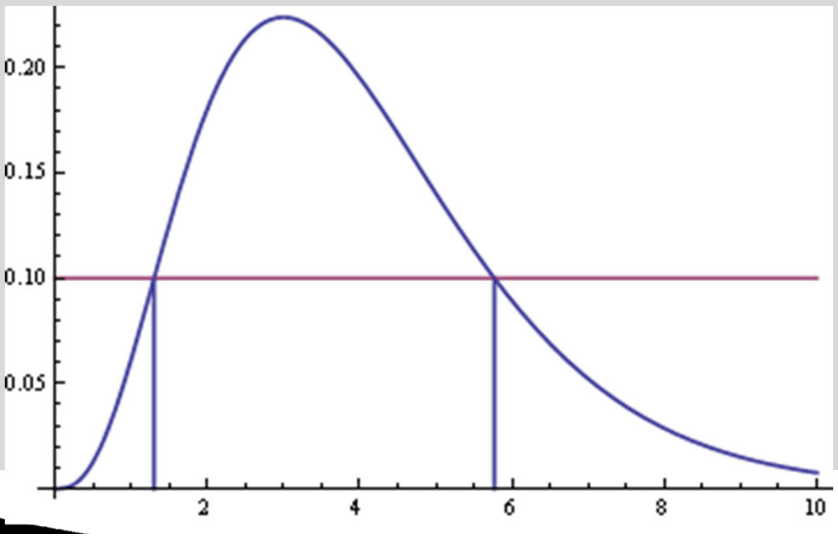
```
> print(mle.fit$par)
[1] -9.474728  1.801821
```

confidence interval (CI)

- Is a statistic – a function of the data
- A random interval that contains the true value of a parameter with a specific probability
- If θ^* is the true but unknown value of θ
- Then a $100*(1-2\alpha)$ CI for θ, (L,U), has the property
- $Pr(L < \theta^* < U) = 1- 2\alpha$
- so if $\alpha=0.025$, a 95% CI has the property
- $Pr(L < \theta^* < U) = 0.95$

confidence interval (CI)

- $Pr(L < \theta^* < U) = 1- \alpha$ is a two-sided interval
- A one-sided interval: $Pr(-\infty < \theta^* < U) = 1- \alpha$ or $Pr(L< \theta^* < \infty)$
- An equal-tailed two-sided CI:
- $Pr(L < \theta^* < U) = 1- \alpha$, $Pr(\theta^* < L) = \alpha/2$, $Pr(\theta^* >U) = \alpha/2$



Shortest CI may not be equal-tailed for asymentric distributions

confidence interval (CI)

- Often a good approximate *100*(1-2α)* CI is
- estimate ± std.err x $Z_{1-\alpha}$ – normal approximation CI
- where $Z_{1-\alpha}$ is the (*1-α*)th percentile from a normal distribution

```
> qnorm(0.975)
[1] 1.959964
> qnorm(0.95)
[1] 1.644854
> qnorm(0.995)
[1] 2.575829
```

$Z_{1-\alpha}$ for a 95% CI

$Z_{1-\alpha}$ for a 90% CI

$Z_{1-\alpha}$ for a 99% CI

Binomial logistic regression - maturity

```
Binfit <- glm(cbind(y,n-y) ~ age,family=binomial(link =
"logit"),data=matdata)

> temp = summary(Binfit)
> print(temp$coefficients)

             Estimate   Std. Error   z value  Pr(>|z|)
(Intercept)   -9.475      1.3359    -7.092   1.319e-12
age            1.802      0.2413     7.468   8.132e-14
```

95% CI : -9.475 ± 1.3359x1.96 = (-12.093, -6.857)

CI in R

An R function that returns a normal approximation CI for some regression stored in an R object (e.g. Binfit)

```
> confint.default(Binfit)
                2.5 %    97.5 %
(Intercept) -12.092945 -6.856308
age           1.328946  2.274693
```

```
> confint.default(Binfit, level=0.99)
                0.5 %    99.5 %
(Intercept) -12.915681 -6.033573
age           1.180358  2.423281
```

```
> confint.default(Binfit, level=0.9)
                  5 %      95 %
(Intercept) -11.671989 -7.277265
age           1.404972  2.198667
```

Profile likelihood CI

- Consider the *i*th parameter, θ_i, of $\theta_1 ,\dots, \theta_p$
- The other parameters are $\{\theta_{j\neq i}\}$
- The profile likelihood involves examining how the likelihood changes for different values of θ_i.
- each time the likelihood is maximized for $\{\theta_{j\neq i}\}$ with θ_i fixed at a specific value
- Let $nll(\theta_i = x; \{\hat{\theta}_{j\neq i}\})$ denote the value of the negative loglikelihood when θ_i is fixed at some value x but all the other parameters are estimated

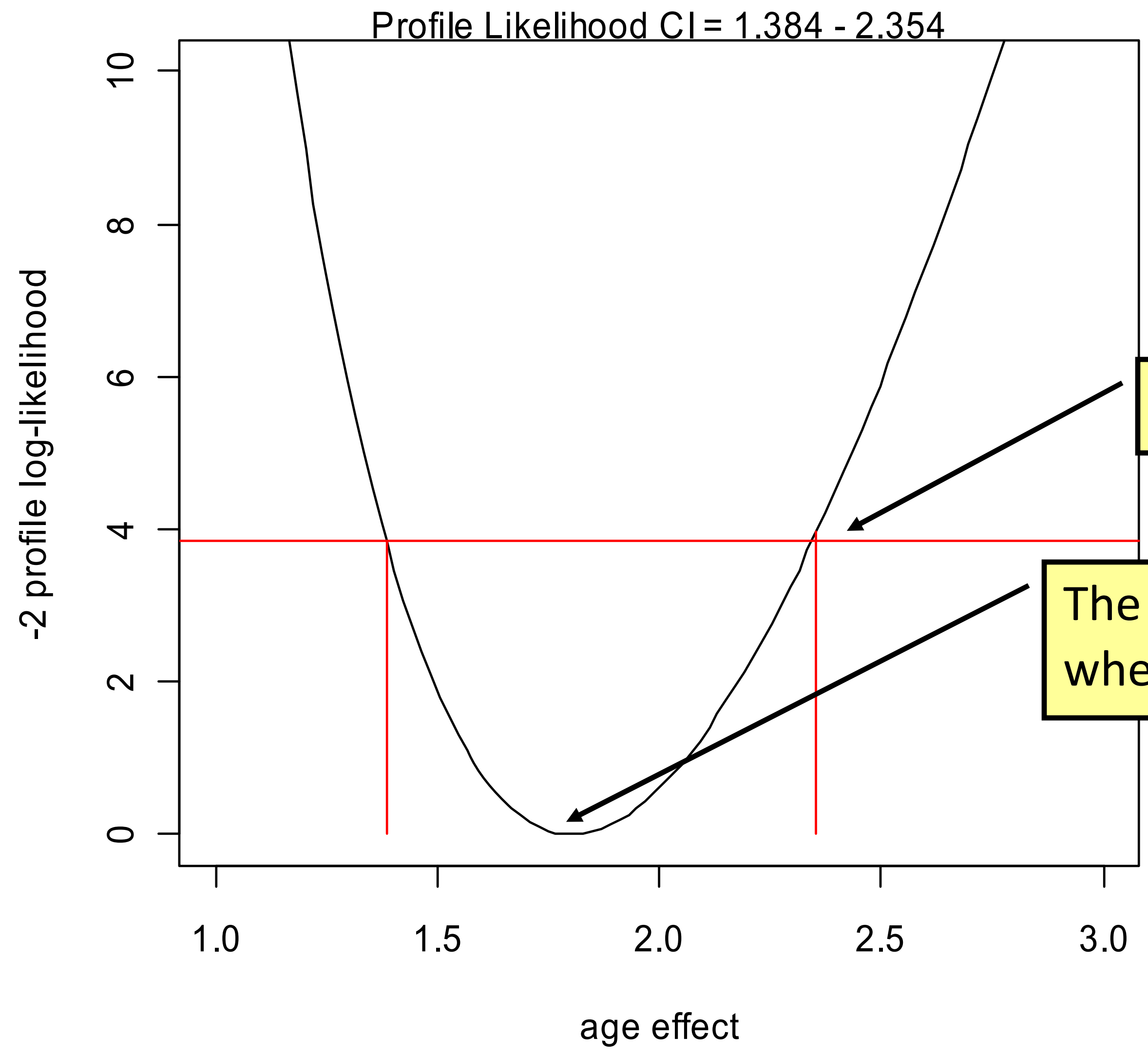
Profile likelihood CI

- Let *nll* denote the value of the negative loglikelihood when θ_i is estimated with the rest of the parameters (i.e. the main estimation), $nll = nll(\hat{\theta}_1, \dots, \hat{\theta}_p)$
- Profile likelihood (PL) CI is based on the result
- $2[nll(\theta_i = x; \{\hat{\theta}_{j \neq i}\}) - nll]$ has an approximate chisquare distribution with one degree of freedom

$$\Pr\left(\chi_1^2 < 3.841\right) = 0.95$$

```
> qchisq(0.95,1)
[1] 3.841459
```

PL for the maturity data slope



qchisq(0.95,1)

The minimum is 0, and occurs when θ_i = the mle of θ_i, $\hat{\theta}_i$

in R

```
> confint(Binfit)
Waiting for profiling to be done...
                 2.5 %    97.5 %
(Intercept) -12.495493 -7.183837
age           1.384860  2.343543
```

```
> confint.default(Binfit)
                 2.5 %    97.5 %
(Intercept) -12.092945 -6.856308
age           1.328946  2.274693
```

```
> confint(Binfit, level=0.99)
Waiting for profiling to be done...
                 0.5 %    99.5 %
(Intercept) -13.628814 -6.580419
age           1.273764  2.545387
```

```
> confint(Binfit, level=0.9)
Waiting for profiling to be done...
                   5 %      95 %
(Intercept) -11.951473 -7.512638
age           1.445114  2.246418
```

Bayesian Probability

- Read about this in Haddon
- It is a different basis of statistical inference in which parameters are considered to be random, and the data are fixed
- It involves specifying “prior distributions” for the parameters, that are modified or updated based on the data collected
- the use of priors is controversial

Bayesian Probability

- Priors are a difficulty, especially multivariate priors
- We should be robust to subjectivity
- Potential for false precision
- But sometimes I find a small number of <u>objective</u> priors helpful
- Objective priors are just a likelihood component for me (see F6005)

Bayesian Probability

- My inference is still frequentist
- You can simulation test frequentist inference, so there is an objective interpretation
- Bayesian probability: is interpreted either as 1) reasonable expectation representing a state of knowledge, or 2) as quantification of a personal belief (Wikipedia)
- It is not completely honest to explain Bayesian probability using frequentist concepts.

Foundations of Frequentist statistical inference (not Bayesian)

- lets go back to the basics to understand what is going on with CI's
- The basis of frequentist statistical inference is the distribution generated by applying your model and parameter estimation method to many, many, data-sets
- We imagine these data-sets – they are not real

Foundations of Frequentist inference

- In Lab 1 you had an example of this in the survey sampling context. You looked at all possible samples from a small finite population,
- and verified that the mean from a random sample of size n was an unbiased estimator for the mean of the entire population (of size $N >> n$)
- You also verified that the variance formula gave the variance of means generated from all possible samples

Foundations of Frequentist inference

- Lets say that our problem involves estimating some parameter θ,
- Assume we have a sample of n observations from a population with a very large size, so that we can assume that we are sampling from an infinite population
- Let s denote the set of observations,
- $s = (y_1, \ldots, y_n)$

Foundations of Frequentist inference

- We estimate θ as some function of the observations s.
- This function may be very complex, and we may only be able to say that it minimizes a fit function
- In any event, let $\hat{\theta}$ denote the estimate of θ.
- Imagine estimating θ many times by applying the estimation function to the different data-sets we generate in our mind,
- $s_1 \rightarrow \hat{\theta}_1, s_2 \rightarrow \hat{\theta}_2, s_3 \rightarrow \hat{\theta}_3 \ldots$

Foundations of Frequentist inference

- We say $\hat{\theta}$ is unbiased for θ if

$$\lim_{B\to\infty}\left(\frac{\sum_{i=1}^{B}\hat{\theta}_i}{B}\right)=\theta$$

- We also write this as $\mathrm{E}(\hat{\theta})=\theta$

- The $\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_B$ have a distribution, and the amount of spread in the distribution is measured using

$$Var(\hat{\theta})=\lim_{B\to\infty}\left(\frac{\sum_{i=1}^{B}\left\{\hat{\theta}_i-E(\theta)\right\}^2}{B}\right)$$

simple example

- If *θ=p* for a Binomial distribution with size *n=30* and true probability of success *p=0.42*
- Statistical theory tells us that the mean and variance of this Binomial distribution is

$$E(Y) = \sum_{y=0}^{n} y\,pdf(y) = \sum_{y=0}^{n} y \binom{n}{y} p^{y}(1-p)^{n-y} = np$$

$$Var(Y) = \sum_{y=0}^{n} \{y - E(Y)\}^{2}\, pdf(y) =$$

$$= \sum_{y=0}^{n} (y - np)^{2} \binom{n}{y} p^{y}(1-p)^{n-y} = np(1-p)$$

simple example

- So the mean is *12.6* and the variance is *7.308*

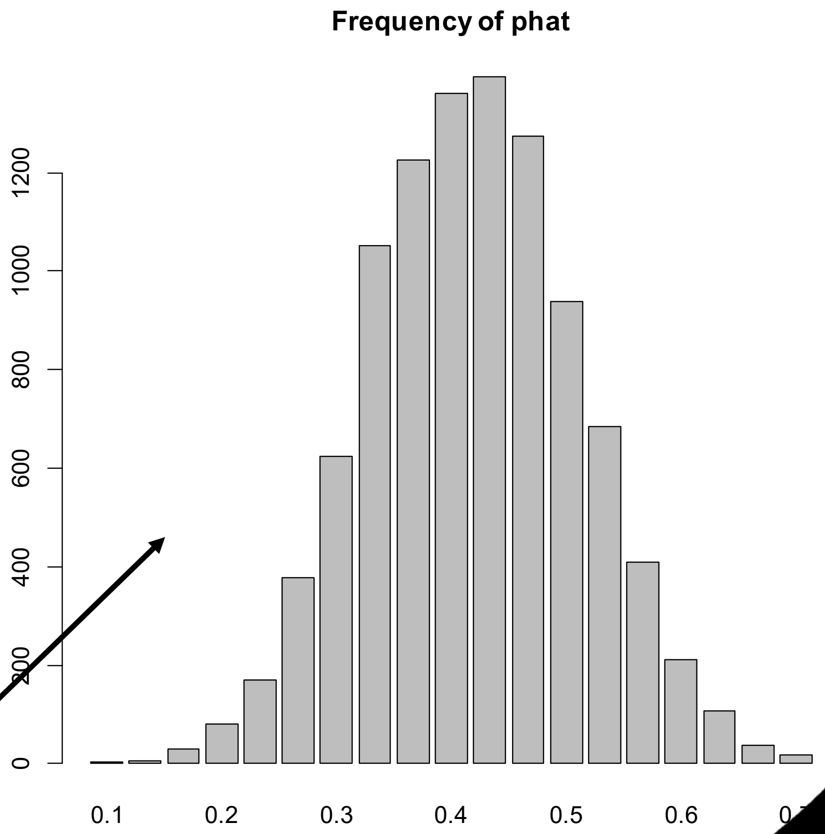
- Statistical theory also tells us that if we estimate p using $\hat{p} = y/n$ then this estimator is unbiased and has variance

$$Var(\hat{p}) = \frac{Var(Y)}{n^2} = \frac{p(1-p)}{n} = 0.00812$$

simple example results

```
> y = rbinom(10000,30,0.42)
> phat = y/30
> temp= table(phat)
> barplot(temp,main='Frequency
of phat')
> E.phat = mean(phat)
> var.phat = var(phat)
> print(E.phat)
[1] 0.41892
> print(var.phat)
[1] 0.008015946
```

10000 estimates of p



looks like a normal distribution

simple example

- We can see that the sampling distribution of $\hat{p}$ looks Normal with mean *p* and variance *p(1-p)/n*
- If we subtract off the mean and divide by square root of the variance then the sampling distribution should look Normal with mean 0 and variance 1
- That is, $Z(p) = \dfrac{\hat{p} - p}{\sqrt{p(1-p)/n}} \sim N(0,1)$

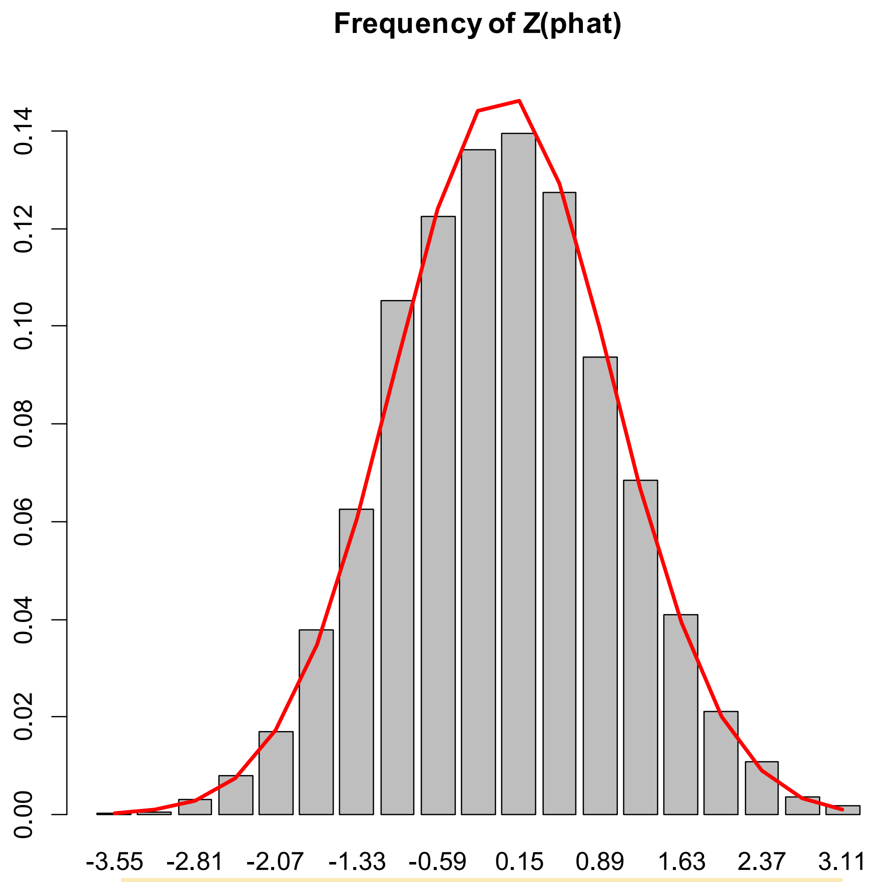
simple example

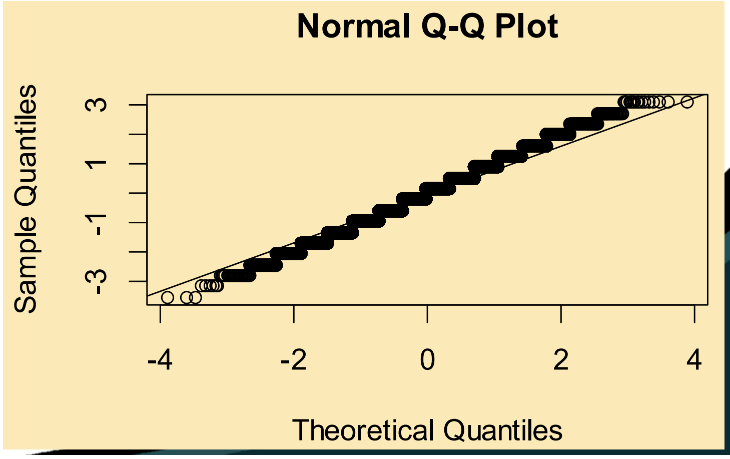
- We can use the normal approximation to produce a confidence interval for p,
- The upper confidence interval, $P.U$, is given by $Z(P.U) = Z_{\alpha/2} = - Z_{1-\alpha/2}$, and the lower interval is given by $Z(P.L) = Z_{1-\alpha/2}$

$$Z(p) = \frac{\hat{p} - p}{\sqrt{p(1-p)/n}} \sim N(0,1)$$

simple example

```
p=0.42;n=30
Zphat = (as.numeric(names(temp))-
p)/sqrt(p*(1-p)/n)
temp2=barplot(temp/10000,main='Frequenc
y of (phat)',names.arg=round(Zphat,digits=2),
ylim=c(0,0.15))

pt = dnorm(Zphat,0,1)
lines(x=temp2,pt/sum(pt),col='red',lwd=3)

Zphat = (phat-p)/sqrt(p*(1-p)/n)
qqnorm(Zphat)
qqline(Zphat)
```





simple example

- This can be solved algebraically, but it is a pain

```
> n=30
> y = rbinom(1,n,0.42)
> print(y)
[1] 10
> phat = y/n
> print(phat)
[1] 0.3333333
>
> Zp = function(p,Z,n){
+   sd =sqrt(p*(1-p)/n)
+   ret = (phat - p)/sd - Z
+   return(ret)
+ }
```

```
> alpha = 0.05
> Za = qnorm(alpha/2)
> ciroot=uniroot(Zp,c(0.01,0.99),Z=Za,n=30)
> pU.95 = ciroot$root
> ciroot=uniroot(Zp,c(0.01,0.99),Z=-Za,n=30)
> pL.95 = ciroot$root
> ci = c(pL.95,pU.95)
> print(ci)
[1] 0.1923031 0.5121984
```

bounds on the root of Zp

other inputs to Zp

a 95% CI for *p*

uniroot finds the value of p that makes Zp return 0

simple example

- Another less accurate approach is to estimate the variance in the CI statistic, $Z(p) = \dfrac{\hat{p} - p}{\sqrt{\hat{p}(1-\hat{p})/n}} \sim N(0,1)$
- A *(1-α)100%* CI for *p* is given by

$$CI(p) = \hat{p} \pm Z_{1-\alpha/2}\sqrt{\hat{p}(1-\hat{p})/n}$$

- When *y=10* and *n=30*, *phat=1/3* and the CI is

```
0.1646465 0.5020202
```

- This is close to the CI on the previous slide

```
> print(ci)
[1] 0.1923031 0.5121984
```

simple example simulation

- I simulated 30 CI’s using the same R code

these 2 CI’s may not contain *$p=0.42$*

```
      [,1] [,2]
 [1,] 0.27 0.61
 [2,] 0.42 0.75
 [3,] 0.30 0.64
 [4,] 0.30 0.64
 [5,] 0.25 0.58
 [6,] 0.30 0.64
 [7,] 0.17 0.48
 [8,] 0.36 0.70
 [9,] 0.27 0.61
[10,] 0.19 0.51
```

```
      [,1] [,2]
 [1,] 0.17 0.48
 [2,] 0.27 0.61
 [3,] 0.27 0.61
 [4,] 0.42 0.75
 [5,] 0.36 0.70
 [6,] 0.17 0.48
 [7,] 0.27 0.61
 [8,] 0.30 0.64
 [9,] 0.27 0.61
[10,] 0.30 0.64
```

```
      [,1] [,2]
 [1,] 0.25 0.58
 [2,] 0.25 0.58
 [3,] 0.27 0.61
 [4,] 0.25 0.58
 [5,] 0.30 0.64
 [6,] 0.22 0.54
 [7,] 0.22 0.54
 [8,] 0.17 0.48
 [9,] 0.19 0.51
[10,] 0.27 0.61
```

simple example 1000 simulation

- I ran the simulation 1000 times.
- The fraction that exceeded the 1st CI method was 0.028 for the lower bound, and 0.021 for the upper bound.
- This is very close to the intended 0.025 level
- The exact same result for the 2nd CI method.
- So both intervals, although they are different, have the same coverage probability

simple example 1000 simulation

- when the *true p = 0.2* the
- The probability of exceeding the 1st CI method was 0.03 for the lower bound, and 0.012 for the upper bound. Not bad.
- The probability of exceeding the 2nd CI method was 0.01 for the lower bound, and 0.05 for the upper bound.
- The total coverage error is OK, but the one-sided errors are considerably different from 0.025

Computer intensive methods

- There are many methods for statistical inference (i.e. confidence intervals, hypothesis tests) that rely on computer intensive simulation methods.
- They tend to mimic in various ways the values of estimators we would get with repeated sampling.

Randomization methods

- Haddon discusses these methods in more detail than most stats books do
- They are more applicable to hypothesis testing in which you formulate a null hypothesis, and some appropriate test statistic that is large when the null hypothesis is false
- The hypothesis is usually related to a change in some characteristic across groups in your population.

Randomization methods

- A randomization test is derived by randomly re-allocating the data to groups, and computing the test statistic
- This is done many times, with different re-allocations each time,
- and this generates a distribution for the test statistic that you can use to decide if the statistic based on the actual observations is large enough to reject the null hypothesis.

Randomization methods

- I rarely see randomization methods used, so I won't say much about them
- or consider examples
- Randomization methods are not straight-forward for the regression case, or for CI's for parameters
- You can read Haddon for more information

Bootstrap methods

- These are widely used methods for generating estimates of the distribution of a parameter estimator, or the distribution of a hypothesis test statistic or a confidence interval statistic
- They are computer intensive, and involve various strategies to resample based only on the observed data
- There are books on bootstrap, and I will just give an overview for some simple cases, for now

One sample mean

- Consider the single sample situation where you have a sample $s = (y_1, \ldots, y_n)$ of size n from a population
- Assume $Y \sim N(\mu, \sigma^2)$ – Y is normal with mean μ and variance σ^2.
- If σ^2 is known the "best" statistic to get CI's for μ is

$$Z(\mu) = \frac{\bar{y} - \mu}{\sqrt{\sigma^2 / n}} \sim N(0,1) \longleftarrow$$

an exact result for $N(\mu,\sigma^2)$ data, and a good approach otherwise when n is large

- Similar to before, CI = $\bar{y} \pm Z_{1-\alpha/2} \sqrt{\sigma^2 / n}$

One sample mean

- You usually won't know σ^2 and the "best" approach is to replace it by the estimate $\hat{\sigma}^2_{bc} = \frac{\sum_{i=1}^{n}(y_i - \bar{y})^2}{n-1}$

- The CI statistic is then $T(\mu) = \frac{\bar{y}-\mu}{\sqrt{\hat{\sigma}^2_{bc}/n}} \sim T_{n-1}$

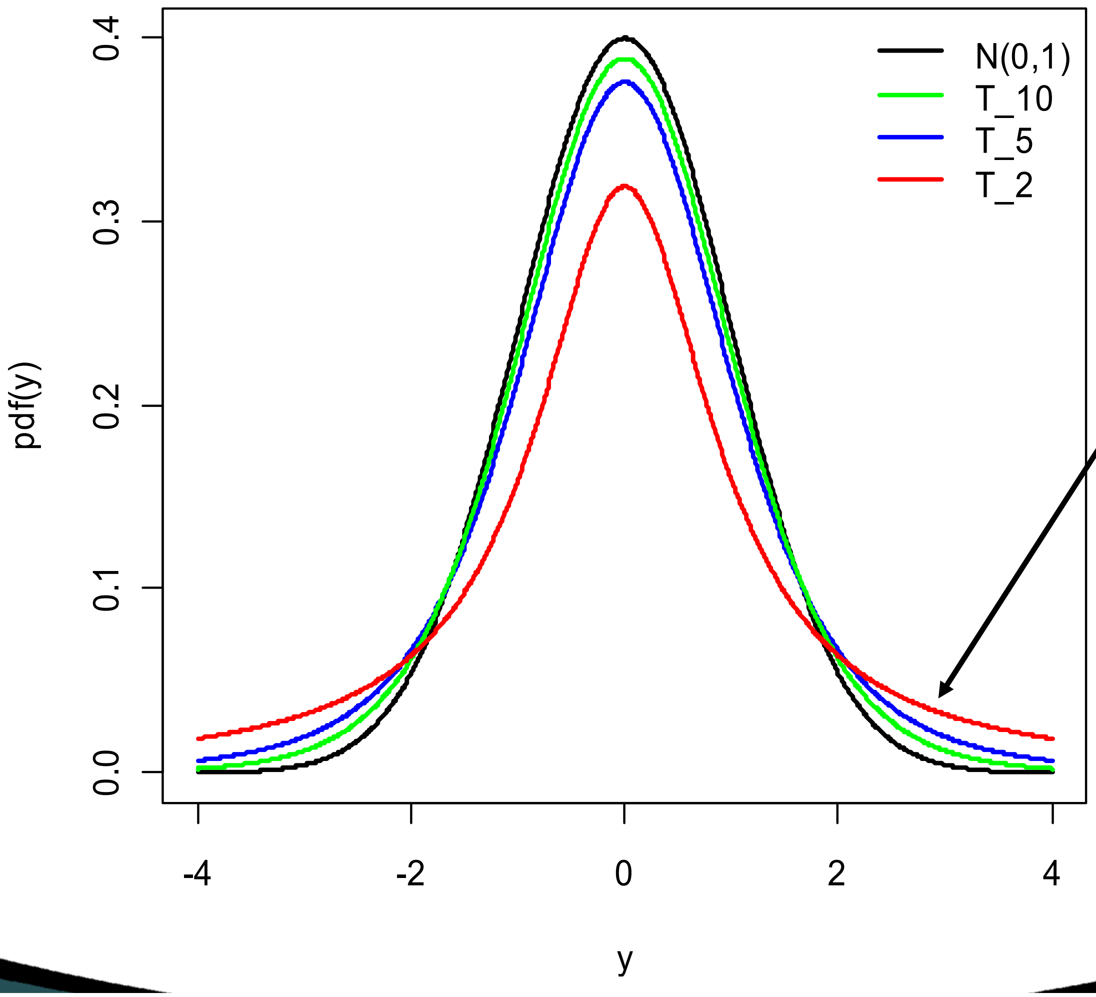
- where T_{n-1} is the t-distribution with n-1 degree's of freedom

an exact result for $N(\mu,\sigma^2)$ data, and a good approach otherwise when n is large

- CI: $\bar{y} \pm T_{1-\alpha/2,n-1}\sqrt{\hat{\sigma}^2_{bc}/n}$

One sample mean

- The t-distribution is similar to a Normal distribution, with wider tails when *n* is small that accounts for the additional uncertainty in the T-statistic because of estimating σ^2.



tails get wider when n is smaller

Bootstrap

- What do you do when you don't know the distribution of *Y*, or your sample size is not large enough to assume normality for the CI Z statistic?
- The bootstrap is a re-sampling method that you can use to estimate the distribution of the T-statistic.
- Draw a random sample with replacement of size *n* from $s = (y_1, \ldots, y_n)$. Call the new sample s_b.
- Compute the sample mean ($\bar{y}_b$) and variance ($\hat{\sigma}_b^2$) from s_b.

Bootstrap

- compute $$T_b = \frac{\bar{y}_b - \bar{y}}{\sqrt{\hat{\sigma}_b{}^2 / n}}$$

- Note that the population you are bootstrap sampling from has a known mean ($\bar{y}$) and you are mimicking the T statistic $$T(\mu) = \frac{\bar{y} - \mu}{\sqrt{\hat{\sigma}^2 / n}}$$

- Repeat the bootstrapping procedure many times (*B*), and generate a bootstrapped distribution T^*

Bootstrap

- Bootstrap-t CI's:
- Approximate the T-statistic distribution using

$$T(\mu) = \frac{\bar{y} - \mu}{\sqrt{\hat{\sigma}^2 / n}} \sim T^*$$

If *α=0.05* then this is the 0.975 quantile of T*

$$L_\alpha = \bar{y} - T^*_{1-\alpha/2} \sqrt{\hat{\sigma}^2 / n},$$

$$U_\alpha = \bar{y} - T^*_{\alpha/2} \sqrt{\hat{\sigma}^2 / n}$$

If *α=0.05* then this is the 0.025 quantile of T*

One sample bootstrap in R

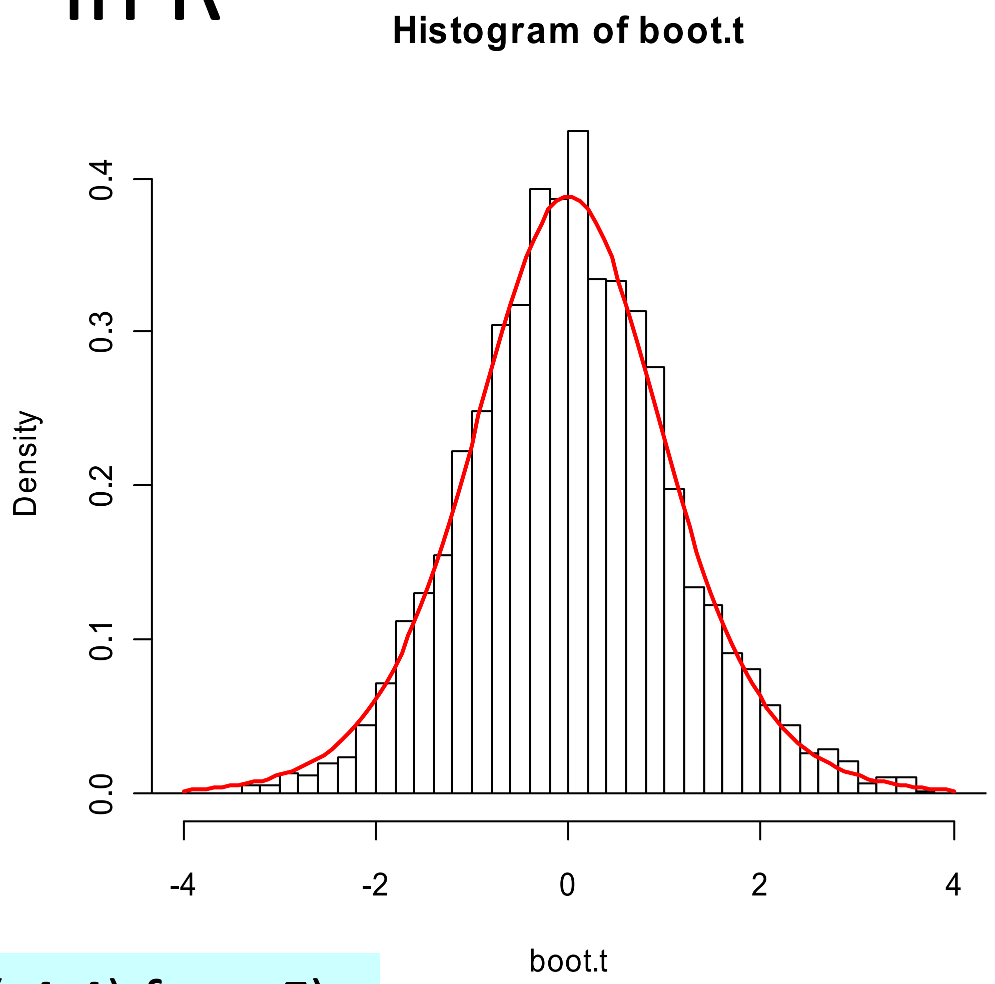
```
mu=10;
sigma2=1
n=10
y = rnorm(n,mu,sd=sqrt(sigma2))
ybar=mean(y)
s2 = var(y)
```

```
B=5000
booty = lapply(1:B, function(i) sample(y, replace = T))
boot.mean = sapply(booty,mean)
boot.var = sapply(booty,var)
boot.t = (boot.mean - ybar)/sqrt(boot.var/n)
```

```
quant.boot.t = quantile(boot.t,probs=c(0.025,0.975))
```

Store the quantiles of the bootstrapT distribution in quant.boot.t.

in R

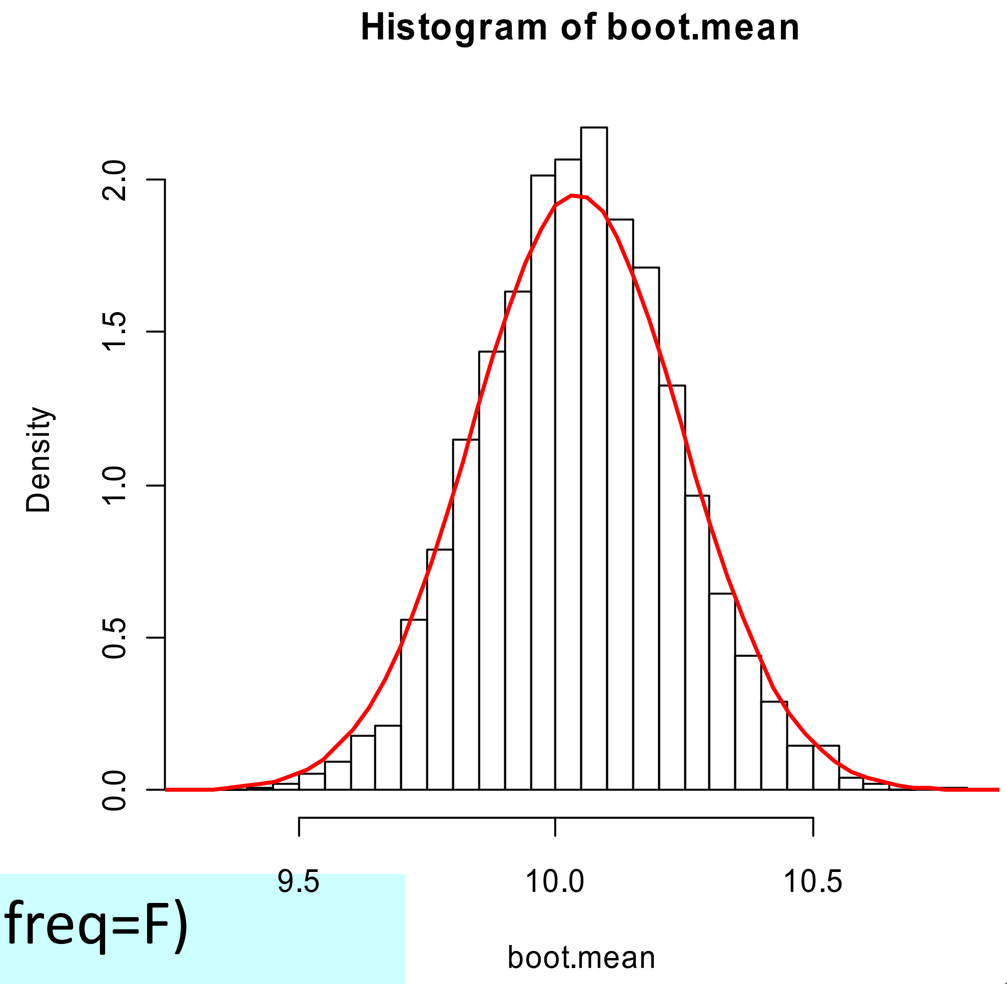


```
hist(boot.t,breaks=50,xlim=c(-4,4),freq=F)
x=seq(-4,4,length=100)
pt = dt(x,n-1)
lines(x,pt,lwd=2,col='red',lty=1)
```

Bootstrap

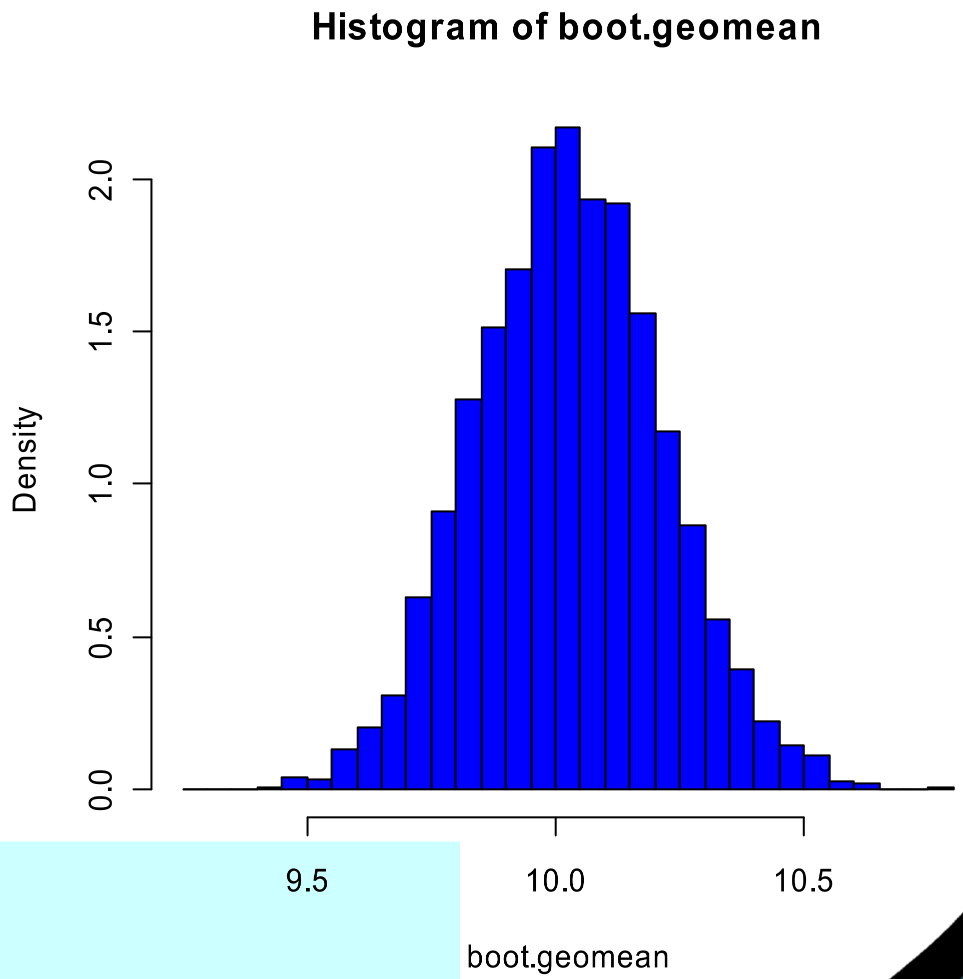
- What is the point, because we know the distribution of T anyway?
- We can compute the bootstrap t-distribution of any statistic, as long as we know its std. err.
- If fact, even if we don’t know the std. err, we can get an approximate CI directly from the bootstrapped distribution of the estimate

Bootstrapped mean in R



```
hist(boot.mean,breaks=50,freq=F)
x=seq(8,11,length=100)
pz = dnorm(x,mean=ybar,sd=sqrt(s2/n))
lines(x,pz,lwd=2,col='red',lty=1)
```

Bootstrapped geometric mean in R

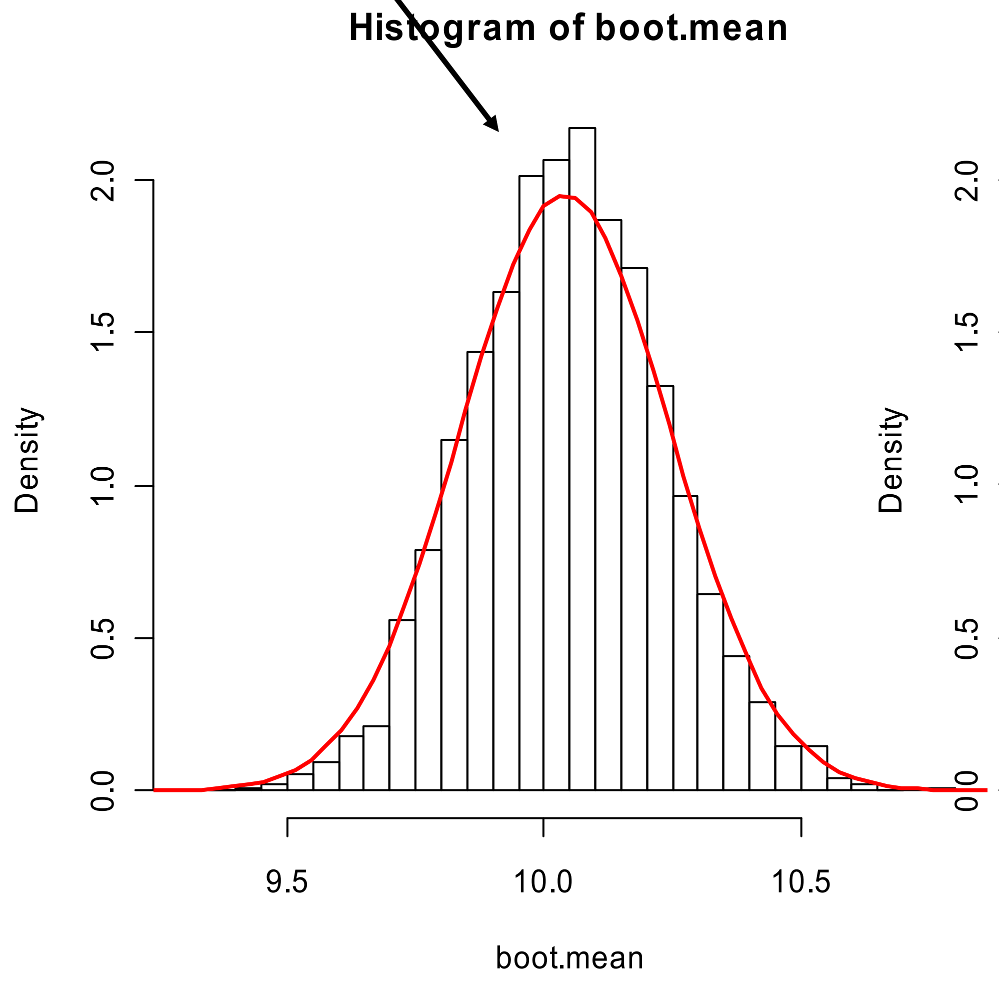


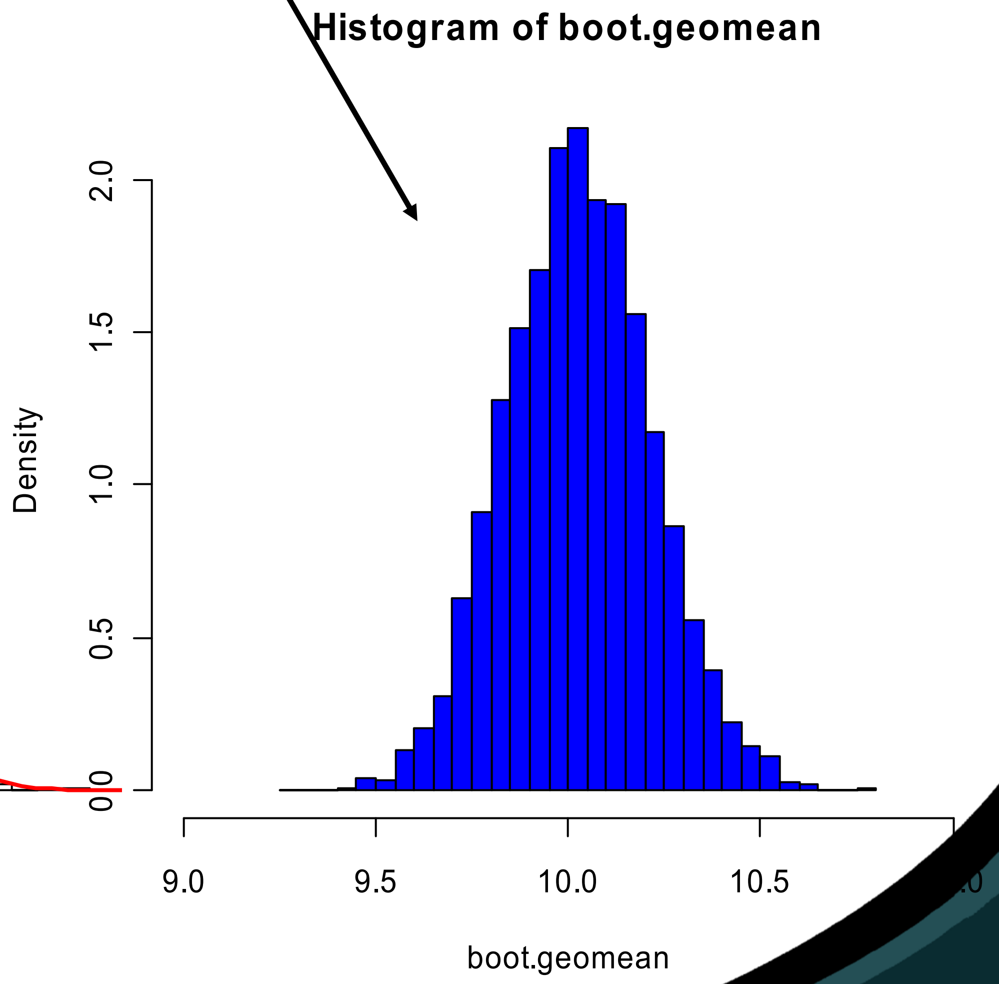
```
install.packages("psych")
library("psych")
boot.geomean = sapply(booty,geometric.mean)
hist(boot.geomean,breaks=30,freq=F)
```

Bootstrapped geometric mean in R

sd=0.189

sd=0.190





Bootstrap

- There are other versions of the bootstrap that are designed for the regression situation
- and there are several ways of getting bootstrap CI’s once you decide how to resample your data
- But that’s enough for now. I will introduce some of the other methods later in the course

Jackknife

- The jackknife procedure is used to estimate the variance of the statistic (e.g. $\hat{\theta}$ - estimator of θ).
- It is based on deleting an observation and re-estimating θ.
- You do this for every observation, to get n $\hat{\theta}$ ‘s.
- you can estimate the bias in $\hat{\theta}$, if any, and the std. err. based on the n jackknifed estimates
- see Haddon. The bootstrap is usually preferred