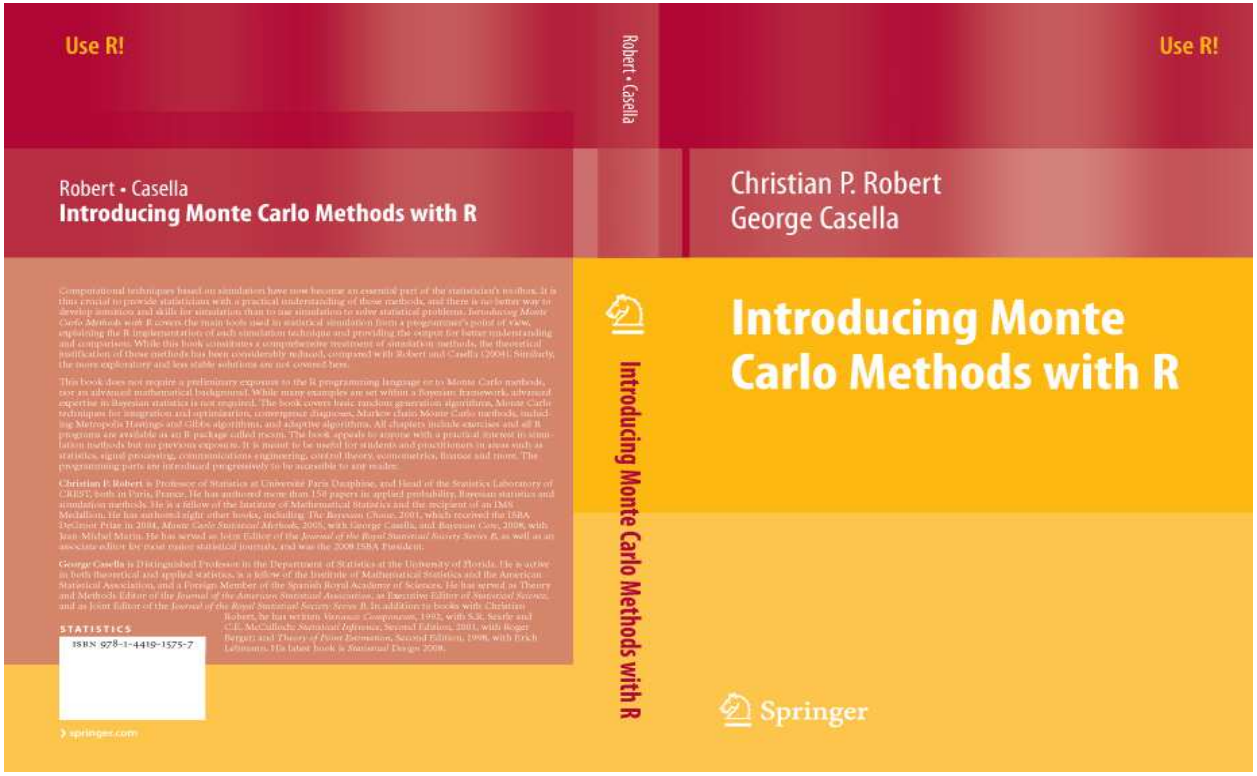
Introducing Monte Carlo Methods with R

Christian P. Robert
Université Paris Dauphine
xian@ceremade.dauphine.fr

George Casella
University of Florida
casella@ufl.edu

Based on

- Introducing Monte Carlo Methods with R, 2009, Springer-Verlag
- Data and R programs for the course available at http://www.stat.ufl.edu/ casella/IntroMonte/



Chapter 1: Basic R Programming

"You're missing the big picture," he told her. "A good album should be more than the sum of its parts."

Ian Rankin
Exit Music

This Chapter

- We introduce the programming language R
- Input and output, data structures, and basic programming commands
- The material is both crucial and unavoidably sketchy

Basic R Programming
Introduction

- This is a quick introduction to `R`
- There are entire books devoted to `R`
  - R Reference Card
  - available at http://cran.r-project.org/doc/contrib/Short-refcard.pdf
- Take Heart!
  - The syntax of `R` is simple and logical
  - The best, and in a sense the only, way to learn `R` is through trial-and-error
- Embedded help commands `help()` and `help.search()`
  - `help.start()` opens a Web browser linked to the local manual pages

Basic R Programming

Why R ?

- There exist other languages, most (all?) of them faster than R, like Matlab, and even free, like C or Python.
- The language combines a sufficiently high power (for an interpreted language) with a very clear syntax both for statistical computation and graphics.
- R is a flexible language that is *object-oriented* and thus allows the manipulation of complex data structures in a condensed and efficient manner.
- Its graphical abilities are also remarkable
  - ▷ Possible interfacing with LaTeXusing the package Sweave.

Basic R Programming
Why R ?

- R offers the additional advantages of being a free and open-source system
  - ▷ There is even an R newsletter, *R-News*
  - ▷ Numerous (free) Web-based tutorials and user's manuals
- It runs on all platforms: Mac, Windows, Linux and Unix
- R provides a powerful *interface*
  - ▷ Can integrate programs written in other languages
  - ▷ Such as C, C++, Fortran, Perl, Python, and Java.
- It is increasingly common to see people who develop new methodology simultaneously producing an R package
- Can interface with WinBugs

Basic R Programming
Getting started

- Type 'demo()' for some demos; `demo(image)` and `demo(graphics)`
- 'help()' for on-line help, or 'help.start()' for an HTML browser interface to help.
- Type 'q()' to quit R.
- Additional packages can be loaded via the `library` command, as in

```
> library(combinat) # combinatorics utilities
> library(datasets) # The R Datasets Package
```

  - There exist hundreds of packages available on the Web.

```
> install.package("mcsm")
```

- A library call is required each time R is launched

Basic R Programming
R objects

- R distinguishes between several types of *objects*
  - ▷ scalar, vector, matrix, time series, data frames, functions, or graphics.
  - ▷ An R object is mostly characterized by a *mode*
  - ▷ The different modes are
    - **null** (empty object),
    - **logical** (TRUE or FALSE),
    - **numeric** (such as 3, 0.14159, or 2+sqrt(3)),
    - **complex**, (such as 3-2i or complex(1,4,-2)), and
    - **character** (such as "Blue", "binomial", "male", or "y=a+bx"),
- The R function `str` applied to any R object will show its structure.

Basic R Programming
Interpreted

- R operates on those types as a regular function would operate on a scalar
- R is interpreted $\Rightarrow$ Slow
- Avoid loops in favor of matrix mainpulations

Basic R Programming – The vector class

`> a=c(5,5.6,1,4,-5)`	build the object `a` containing a numeric vector of dimension 5 with elements 5, 5.6, 1, 4, –5
`> a[1]`	display the first element of `a`
`> b=a[2:4]`	build the numeric vector `b` of dimension 3 with elements 5.6, 1, 4
`> d=a[c(1,3,5)]`	build the numeric vector `d` of dimension 3 with elements 5, 1, –5
`> 2*a`	multiply each element of `a` by 2 and display the result
`> b%%3`	provides each element of `b` modulo 3

Basic R Programming

More vector class

`> e=3/d`	build the numeric vector `e` of dimension 3 and elements 3/5, 3, –3/5
`> log(d*e)`	multiply the vectors `d` and `e` term by term and transform each term into its natural logarithm
`> sum(d)`	calculate the sum of `d`
`> length(d)`	display the length of `d`

Basic R Programming
Even more vector class

`> t(d)`	transpose `d`, the result is a row vector
`> t(d)*e`	elementwise product between two vectors with identical lengths
`> t(d)%*%e`	matrix product between two vectors with identical lengths
`> g=c(sqrt(2),log(10))`	build the numeric vector `g` of dimension 2 and elements $\sqrt{2}$, $\log(10)$
`> e[d==5]`	build the subvector of `e` that contains the components `e[i]` such that `d[i]=5`
`> a[-3]`	create the subvector of `a` that contains all components of `a` but the third.
`> is.vector(d)`	display the logical expression `TRUE` if a vector and `FALSE` else

Basic R Programming
Comments on the vector class

- The ability to apply scalar functions to vectors: Major Advantage of R.
  - `> lgamma(c(3,5,7))`
  - returns the vector with components $(\log\Gamma(3), \log\Gamma(5), \log\Gamma(7))$.
- Functions that are specially designed for vectors include

  `sample`, `permn`, `order`,`sort`, and `rank`

  - All manipulate the order in which the components of the vector occur.
  - `permn` is part of the combinat library
- The components of a vector can also be identified by names.
  - For a vector `x`, `names(x)` is a vector of characters of the same length as `x`

Basic R Programming
The matrix, array, and factor classes

- The matrix class provides the R representation of matrices.
- A typical entry is

  `> x=matrix(vec,nrow=n,ncol=p)`

  - Creates an $n \times p$ matrix whose elements are of the dimension np vector `vec`
- Some manipulations on matrices
  - The standard matrix product is denoted by `%*%`,
  - while `*` represents the term-by-term product.
  - `diag` gives the vector of the diagonal elements of a matrix
  - `crossprod` replaces the product `t(x)%*%y` on either vectors or matrices
    - `crossprod(x,y)` more efficient
  - `apply` is easy to use for functions operating on matrices by row or column

Basic R Programming

Some matrix commands

`> x1=matrix(1:20,nrow=5)`	build the numeric matrix `x1` of dimension 5×4 with first row 1, 6, 11, 16
`> x2=matrix(1:20,nrow=5,byrow=T)`	build the numeric matrix `x2` of dimension 5×4 with first row 1, 2, 3, 4
`> a=x1%*%t(x2)`	matrix product
`> c=x1*x2`	term-by-term product between `x1` and `x2`
`> dim(x1)`	display the dimensions of `x1`
`> b[,2]`	select the second column of `b`
`> b[c(3,4),]`	select the third and fourth rows of `b`
`> b[-2,]`	delete the second row of `b`
`> rbind(x1,x2)`	vertical merging of `x1` and `x2`rbind(*)`rbind`
`> cbind(x1,x2)`	horizontal merging of `x1` and `x2`rbind(*)`rbind`
`> apply(x1,1,sum)`	calculate the sum of each row of `x1`
`> as.matrix(1:10)`	turn the vector `1:10` into a 10×1 matrix

- Lots of other commands that we will see throughout the course

Basic R Programming
The list and data.frame classes
The Last One

- A list is a collection of arbitrary objects known as its *components*

  `> li=list(num=1:5,y="color",a=T)` create a list with three arguments

- The last class we briefly mention is the data frame
  - ▷ A list whose elements are possibly made of differing modes and attributes
  - ▷ But have the same length

`> v1=sample(1:12,30,rep=T)`	simulate 30 independent uniform $\{1, 2, \ldots, 12\}$
`> v2=sample(LETTERS[1:10],30,rep=T)`	simulate 30 independent uniform $\{a, b, ..., j\}$
`> v3=runif(30)`	simulate 30 independent uniform $[0, 1]$
`> v4=rnorm(30)`	simulate 30 independent standard normals
`> xx=data.frame(v1,v2,v3,v4)`	create a data frame

- R code

Probability distributions in R

- R , or the web, has about all probability distributions
- Prefixes: p, d,q, r

Distribution	Core	Parameters	Default Values
Beta	beta	shape1, shape2	
Binomial	binom	size, prob	
Cauchy	cauchy	location, scale	0, 1
Chi-square	chisq	df	
Exponential	exp	1/mean	1
F	f	df1, df2	
Gamma	gamma	shape,1/scale	NA, 1
Geometric	geom	prob	
Hypergeometric	hyper	m, n, k	
Log-normal	lnorm	mean, sd	0, 1
Logistic	logis	location, scale	0, 1
Normal	norm	mean, sd	0, 1
Poisson	pois	lambda	
Student	t	df	
Uniform	unif	min, max	0, 1
Weibull	weibull	shape	

Basic and not-so-basic statistics
t-test

- Testing equality of two means

```
> x=rnorm(25) #produces a N(0,1) sample of size 25
> t.test(x)

        One Sample t-test

data:  x
t = -0.8168, df = 24, p-value = 0.4220
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -0.4915103  0.2127705
sample estimates:
 mean of x
-0.1393699
```

Basic and not-so-basic statistics
Correlation

- Correlation

```
> attach(faithful) #resident dataset
> cor.test(faithful[,1],faithful[,2])

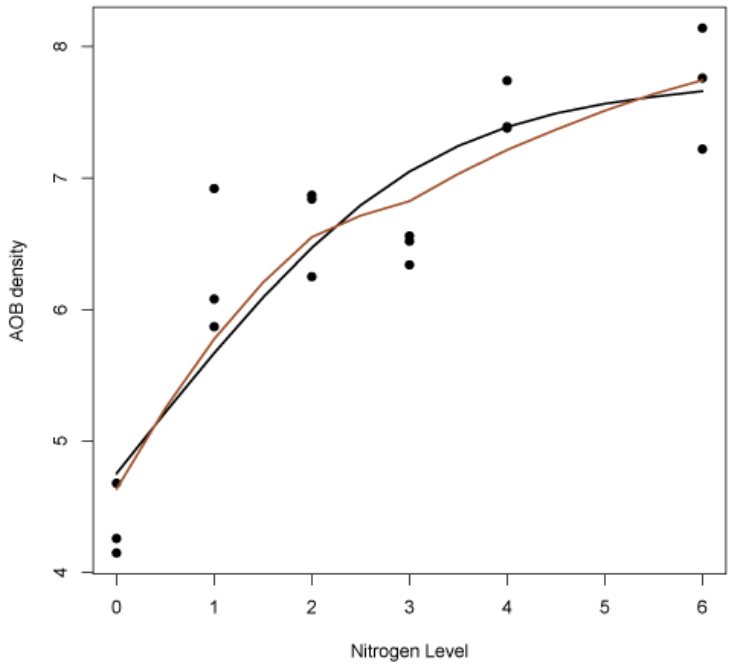
        Pearson's product-moment correlation

data:  faithful[, 1] and faithful[, 2]
t = 34.089, df = 270, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.8756964 0.9210652
sample estimates:
      cor
0.9008112
```

- R code

Basic and not-so-basic statistics
Splines

- Nonparametric regression with **loess** function or using *natural splines*
- Relationship between nitrogen level in soil and abundance of a bacteria `AOB`



- Natural spline fit *(dark)*
  - With `ns=2` (linear model)
- Loess fit *(brown)* with `span=1.25`
- `R` code

Basic and not-so-basic statistics
Generalized Linear Models

- Fitting a binomial (logistic) glm to the probability of suffering from diabetes for a woman within the Pima Indian population

```
> glm(formula = type ~ bmi + age, family = "binomial", data = Pima.tr)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.7935  -0.8368  -0.5033   1.0211   2.2531

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -6.49870    1.17459  -5.533 3.15e-08 ***
bmi          0.10519    0.02956   3.558 0.000373 ***
age          0.07104    0.01538   4.620 3.84e-06 ***
---
Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)
    Null deviance: 256.41  on 199  degrees of freedom
Residual deviance: 215.93  on 197  degrees of freedom
AIC: 221.93
Number of Fisher Scoring iterations: 4
```

Basic and not-so-basic statistics
Generalized Linear Models – Comments

- Concluding with the significance both of the body mass index `bmi` and the age
- Other generalized linear models can be defined by using a different `family` value

```
> glm(y ~x, family=quasi(var="mu^2", link="log"))
```

  - Quasi-Likelihood also
- Many many other procedures
  - Time series, anova,...
- One last one

Basic and not-so-basic statistics
Bootstrap

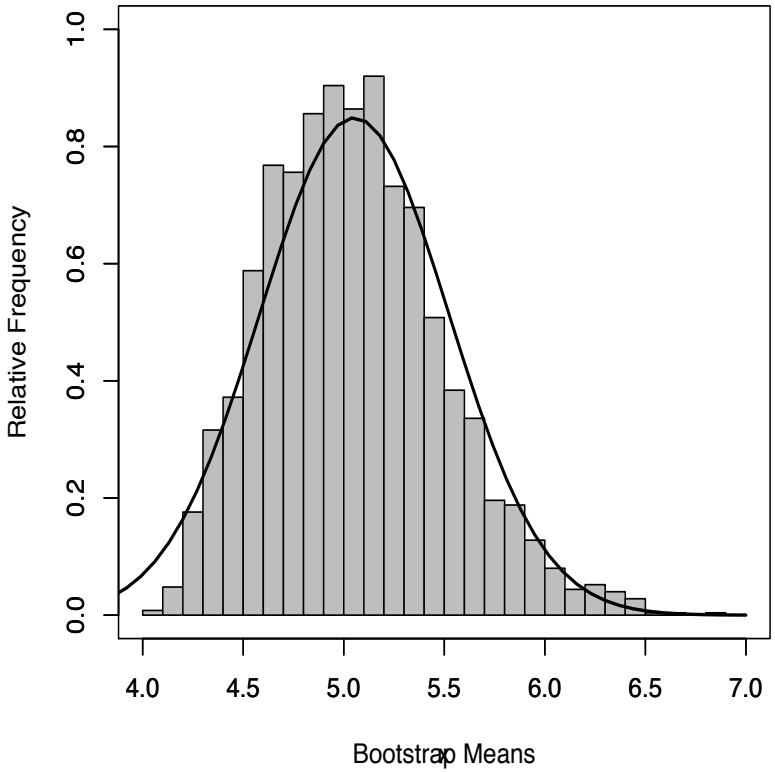
- The bootstrap procedure uses the empirical distribution as a substitute for the true distribution to construct variance estimates and confidence intervals.
  - A sample $X_1, \ldots, X_n$ is resampled with replacement
  - The empirical distribution has a finite but large support made of n^n points

- For example, with data y, we can create a bootstrap sample y^* using the code

```
> ystar=sample(y,replace=T)
```

  - For each resample, we can calculate a mean, variance, etc

Basic and not-so-basic statistics
Simple illustration of bootstrap



- A histogram of 2500 bootstrap means
- Along with the normal approximation
- Bootstrap shows some skewness
- R code

Basic and not-so-basic statistics
Bootstrapping Regression

- The bootstrap is not a panacea
  - ▷ Not always clear which quantity should be bootstrapped
  - ▷ In regression, bootstrapping the residuals is preferred
- Linear regression

$$Y_{ij} = \alpha + \beta x_i + \varepsilon_{ij},$$

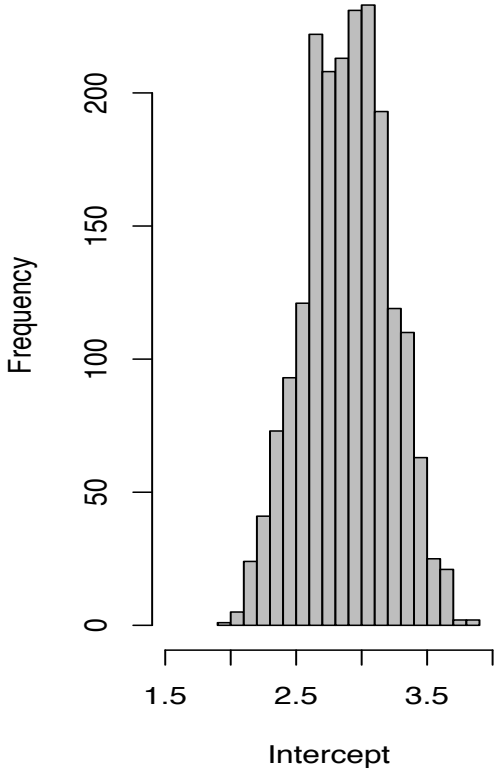
  α and β are the unknown intercept and slope, ε_{ij} are the iid normal errors
- The residuals from the least squares fit are given by

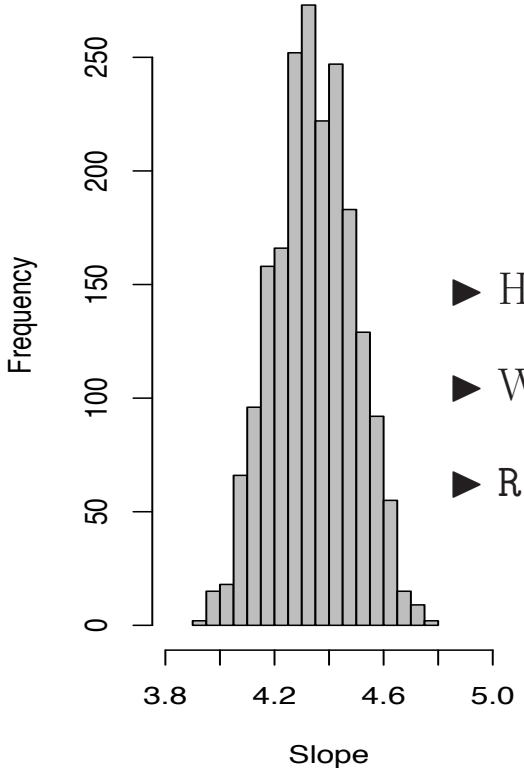
$$\hat{\varepsilon}_{ij} = y_{ij} - \hat{\alpha} - \hat{\beta} x_i,$$

  - ▷ We bootstrap the residuals
  - ▷ Produce a new sample $(\hat{\varepsilon}^*_{ij})_{ij}$ by resampling from the $\hat{\varepsilon}_{ij}$'s
  - ▷ The bootstrap samples are then $y^*_{ij} = y_{ij} + \hat{\varepsilon}^*_{ij}$

Basic and not-so-basic statistics

Bootstrapping Regression – 2





- Histogram of 2000 bootstrap samples
- We can also get confidence intervals
- R code

Basic R Programming
Some Other Stuff

- Graphical facilities
  - Can do a lot; see `plot` and `par`
- Writing new R functions
  - `h=function(x)(sin(x)^2+cos(x)^3)^(3/2)`
  - We will do this a lot
- Input and output in R
  - `write.table`, `read.table`, `scan`
- Don't forget the `mcsm` package

Chapter 2: Random Variable Generation

"It has long been an axiom of mine that the little things are infinitely the most important."

Arthur Conan Doyle
A Case of Identity

This Chapter

- We present practical techniques that can produce random variables
- From both standard and nonstandard distributions
- First: Transformation methods
- Next: Indirect Methods - Accept–Reject

Introduction

- Monte Carlo methods rely on
  - The possibility of producing a supposedly endless flow of random variables
  - For well-known or new distributions.
- Such a simulation is, in turn,
  - Based on the production of uniform random variables on the interval $(0, 1)$.
- We are not concerned with the details of producing uniform random variables
- We assume the existence of such a sequence

Introduction
Using the R Generators

R has a large number of functions that will generate the standard random variables

```
> rgamma(3,2.5,4.5)
```

produces three independent generations from a $\mathcal{G}(5/2, 9/2)$ distribution

- It is therefore,
  - Counter-productive
  - Inefficient
  - And even dangerous,
- To generate from those standard distributions
- If it is built into R , use it
- But....we will practice on these.
- The principles are essential to deal with distributions that are not built into R.

Uniform Simulation

- The uniform generator in `R` is the function `runif`
- The only required entry is the number of values to be generated.
- The other optional parameters are `min` and `max`, with `R` code

```
> runif(100, min=2, max=5)
```

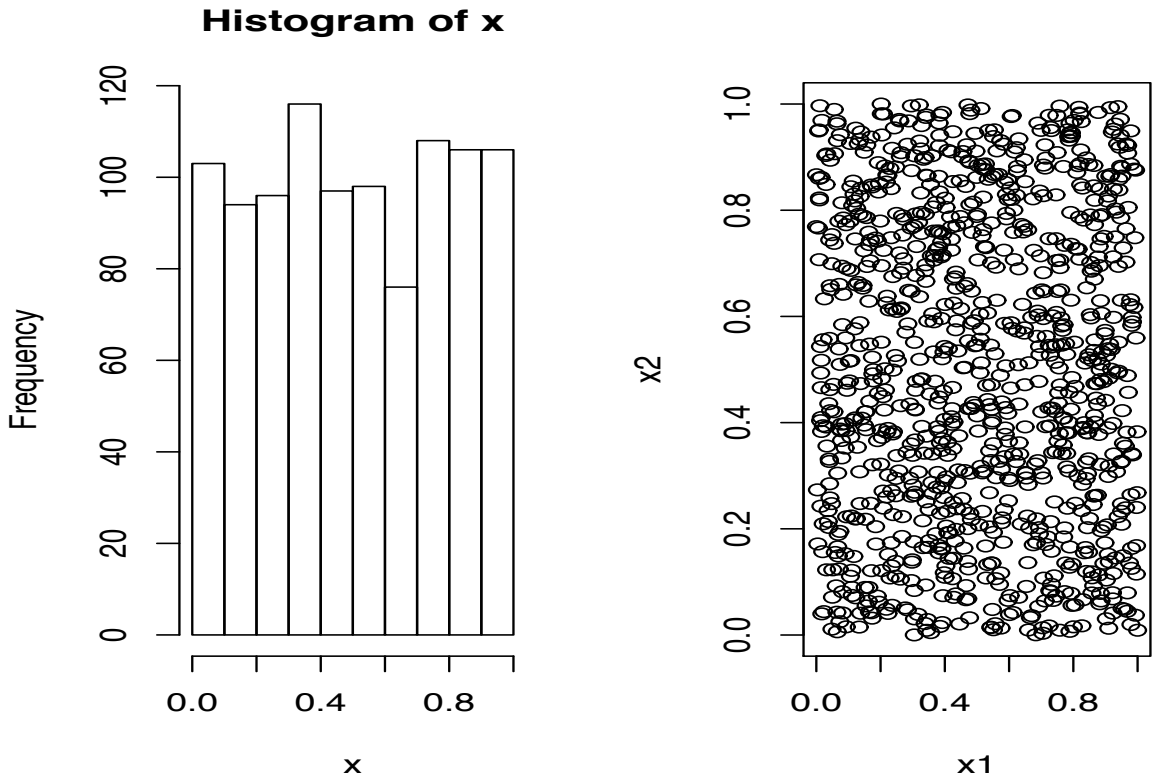
will produce 100 random variables $\mathcal{U}(2, 5)$.

Uniform Simulation
Checking the Generator

- A quick check on the properties of this uniform generator is to
  - Look at a histogram of the X_i's,
  - Plot the pairs (X_i, X_{i+1})
  - Look at the estimate autocorrelation function
- Look at the R code

```
> Nsim=10^4                  #number of random numbers
> x=runif(Nsim)
> x1=x[-Nsim]                #vectors to plot
> x2=x[-1]                   #adjacent pairs
> par(mfrow=c(1,3))
> hist(x)
> plot(x1,x2)
> acf(x)
```

Uniform Simulation
Plots from the Generator



▶ Histogram *(left)*, pairwise plot *(center)*, and estimated autocorrelation function *(right)* of a sequence of 10^4 uniform random numbers generated by **runif**.

Uniform Simulation
Some Comments

- Remember: `runif` does not involve randomness per se.
- It is a deterministic sequence based on a random starting point.
- The `R` function `set.seed` can produce the same sequence.

```
> set.seed(1)
> runif(5)
[1] 0.2655087 0.3721239 0.5728534 0.9082078 0.2016819
> set.seed(1)
> runif(5)
[1] 0.2655087 0.3721239 0.5728534 0.9082078 0.2016819
> set.seed(2)
> runif(5)
[1] 0.0693609 0.8177752 0.9426217 0.2693818 0.1693481
```

- Setting the seed determines all the subsequent values

The Inverse Transform

- The *Probability Integral Transform*
  - Allows us to transform a uniform into any random variable

- For example, if X has density f and cdf F, then we have the relation

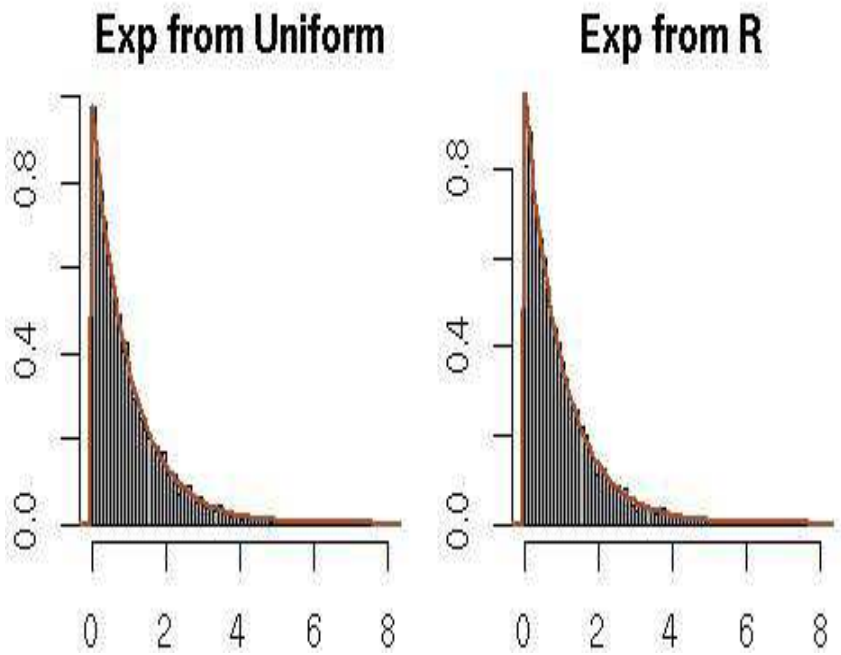
$$F(x) = \int_{-\infty}^{x} f(t)\,\mathrm{d}t,$$

  and we set $U = F(X)$ and solve for X

- Example 2.1
  - If $X \sim \mathcal{E}xp(1)$, then $F(x) = 1 - e^{-x}$
  - Solving for x in $u = 1 - e^{-x}$ gives $x = -\log(1-u)$

Generating Exponentials

```
> Nsim=10^4              #number of random variables
> U=runif(Nsim)
> X=-log(U)              #transforms of uniforms
> Y=rexp(Nsim)           #exponentials from R
> par(mfrow=c(1,2))      #plots
> hist(X,freq=F,main="Exp from Uniform")
> hist(Y,freq=F,main="Exp from R")
```



- Histograms of exponential random variables
  - Inverse transform *(right)*
  - R command `rexp` *(left)*
  - $\mathcal{E}xp(1)$ density on top

Generating Other Random Variables From Uniforms

- This method is useful for other probability distributions
  - Ones obtained as a transformation of uniform random variables

- Logistic pdf: $f(x) = \frac{1}{\beta}\frac{e^{-(x-\mu)/\beta}}{[1+e^{-(x-\mu)/\beta}]^2}$, cdf: $F(x) = \frac{1}{1+e^{-(x-\mu)/\beta}}$.
- Cauchy pdf: $f(x) = \frac{1}{\pi\sigma}\frac{1}{1+\left(\frac{x-\mu}{\sigma}\right)^2}$, cdf: $F(x) = \frac{1}{2} + \frac{1}{\pi}\arctan((x-\mu)/\sigma)$.

General Transformation Methods

- When a density f is linked in a relatively simple way
  - To another distribution easy to simulate
  - This relationship can be use to construct an algorithm to simulate from f
- If the X_i's are iid $\mathcal{E}xp(1)$ random variables,
  - Three standard distributions can be derived as

$$Y = 2\sum_{j=1}^{\nu} X_j \sim \chi^2_{2\nu}\,, \qquad \nu \in \mathbb{N}^*\,,$$

$$Y = \beta\sum_{j=1}^{a} X_j \sim \mathcal{G}(a,\beta)\,, \qquad a \in \mathbb{N}^*\,,$$

$$Y = \frac{\sum_{j=1}^{a} X_j}{\sum_{j=1}^{a+b} X_j} \sim \mathcal{B}e(a,b)\,, \qquad a,b \in \mathbb{N}^*\,,$$

where $\mathbb{N}^* = \{1,2,\ldots\}$.

General Transformation Methods
χ^2_6 Random Variables

- For example, to generate χ^2_6 random variables, we could use the R code

```
> U=runif(3*10^4)
> U=matrix(data=U,nrow=3) #matrix for sums
> X=-log(U)               #uniform to exponential
> X=2* apply(X,2,sum)     #sum up to get chi squares
```

- Not nearly as efficient as calling rchisq, as can be checked by the R code

```
> system.time(test1());system.time(test2())
   user  system elapsed
  0.104   0.000   0.107
   user  system elapsed
  0.004   0.000   0.004
```

- test1 corresponds to the R code above
- test2 corresponds to X=rchisq(10^4,df=6)

General Transformation Methods
Comments

- These transformations are quite simple and will be used in our illustrations.
- However, there are limits to their usefulness,
  - No odd degrees of freedom
  - No normals
- For any specific distribution, efficient algorithms have been developed.
- Thus, if R has a distribution built in, it is almost always worth using

General Transformation Methods
A Normal Generator

- Box–Muller algorithm - two normals from two uniforms
- If U_1 and U_2 are iid $\mathcal{U}_{[0,1]}$
- The variables X_1 and X_2

$$X_1 = \sqrt{-2\log(U_1)}\,\cos(2\pi U_2)\,, \qquad X_2 = \sqrt{-2\log(U_1)}\,\sin(2\pi U_2)\,,$$

- Are iid $\mathcal{N}(0,1)$ by virtue of a change of variable argument.
- The Box–Muller algorithm is exact, not a crude CLT-based approximation
- Note that this is *not* the generator implemented in R
  - ▷ It uses the probability inverse transform
  - ▷ With a very accurate representation of the normal cdf

General Transformation Methods
Multivariate Normals

- Can simulate a multivariate normal variable using univariate normals
  - Cholesky decomposition of $\Sigma = AA'$
  - $Y \sim \mathcal{N}_p(0, I) \Rightarrow AY \sim \mathcal{N}_p(0, \Sigma)$
- There is an R package that replicates those steps, called `rmnorm`
  - In the `mnormt` library
  - Can also calculate the probability of hypercubes with the function `sadmvn`

```
> sadmvn(low=c(1,2,3),upp=c(10,11,12),mean=rep(0,3),var=B)
[1] 9.012408e-05
attr(,"error")
[1] 1.729111e-08
```

- `B` is a positive-definite matrix
- This is quite useful since the analytic derivation of this probability is almost always impossible.

Discrete Distributions

- To generate discrete random variables we have an "all-purpose" algorithm.
- Based on the inverse transform principle
- To generate $X \sim P_\theta$, where P_θ is supported by the integers,
  - ▷ We can calculate—the probabilities
  - ▷ Once for all, assuming we can store them

$$p_0 = P_\theta(X \leq 0), \quad p_1 = P_\theta(X \leq 1), \quad p_2 = P_\theta(X \leq 2), \quad \ldots,$$

  - ▷ And then generate $U \sim \mathcal{U}_{[0,1]}$ and take

$$X = k \text{ if } p_{k-1} < U < p_k.$$

Discrete Distributions
Binomial

- Example To generate $X \sim \mathcal{B}in(10, .3)$
  - The probability values are obtained by `pbinom(k,10,.3)`

$$p_0 = 0.028, \quad p_1 = 0.149, \quad p_2 = 0.382, \ldots, p_{10} = 1\,,$$

  - And to generate $X \sim \mathcal{P}(7)$, take

$$p_0 = 0.0009, \quad p_1 = 0.0073, \quad p_2 = 0.0296, \ldots\,,$$

  - Stopping the sequence when it reaches 1 with a given number of decimals.
    - For instance, $p_{20} = 0.999985$.

- Check the R code

Discrete Distributions
Comments

- Specific algorithms are usually more efficient
- Improvement can come from a judicious choice of the probabilities first computed.

- For example, if we want to generate from a Poisson with $\lambda = 100$
  - The algorithm above is woefully inefficient
  - We expect most of our observations to be in the interval $\lambda \pm 3\sqrt{\lambda}$
  - For $\lambda = 100$ this interval is $(70, 130)$
  - Thus, starting at 0 is quite wasteful

- A first remedy is to "ignore" what is outside of a highly likely interval
  - In the current example $P(X < 70) + P(X > 130) = 0.00268$.

Discrete Distributions
Poisson R Code

- R code that can be used to generate Poisson random variables for large values of lambda.
- The sequence t contains the integer values in the range around the mean.

```
> Nsim=10^4; lambda=100
> spread=3*sqrt(lambda)
> t=round(seq(max(0,lambda-spread),lambda+spread,1))
> prob=ppois(t, lambda)
> X=rep(0,Nsim)
> for (i in 1:Nsim){
+    u=runif(1)
+    X[i]=t[1]+sum(prob<u)-1 }
```

- The last line of the program checks to see what interval the uniform random variable fell in and assigns the correct Poisson value to X.

Discrete Distributions
Comments

- Another remedy is to start the cumulative probabilities at the mode of the discrete distribution
- Then explore neighboring values until the cumulative probability is almost 1.

- Specific algorithms exist for almost any distribution and are often quite fast.
- So, if `R` has it, use it.
- But `R` does not handle every distribution that we will need,

Mixture Representations

- It is sometimes the case that a probability distribution can be naturally represented as a *mixture distribution*
- That is, we can write it in the form

$$f(x) = \int_{\mathcal{Y}} g(x|y)p(y)\ \mathrm{d}y \qquad \text{or} \qquad f(x) = \sum_{i \in \mathcal{Y}} p_i\ f_i(x)\ ,$$

  - The mixing distribution can be continuous or discrete.

- To generate a random variable X using such a representation,
  - we can first generate a variable Y from the mixing distribution
  - Then generate X from the selected conditional distribution

Mixture Representations
Generating the Mixture

- Continuous

$$f(x) = \int_{\mathcal{Y}} g(x|y)p(y)\ \mathrm{d}y \Rightarrow y \sim p(y) \text{ and } X \sim f(x|y),\ \text{then } X \sim f(x)$$

- Discrete

$$f(x) = \sum_{i \in \mathcal{Y}} p_i\ f_i(x) \Rightarrow i \sim p_i \text{ and } X \sim f_i(x),\ \text{then } X \sim f(x)$$

- Discrete Normal Mixture R code

  ▷ $p_1 * N(\mu_1, \sigma_1) + p_2 * N(\mu_2, \sigma_2) + p_3 * N(\mu_3, \sigma_3)$

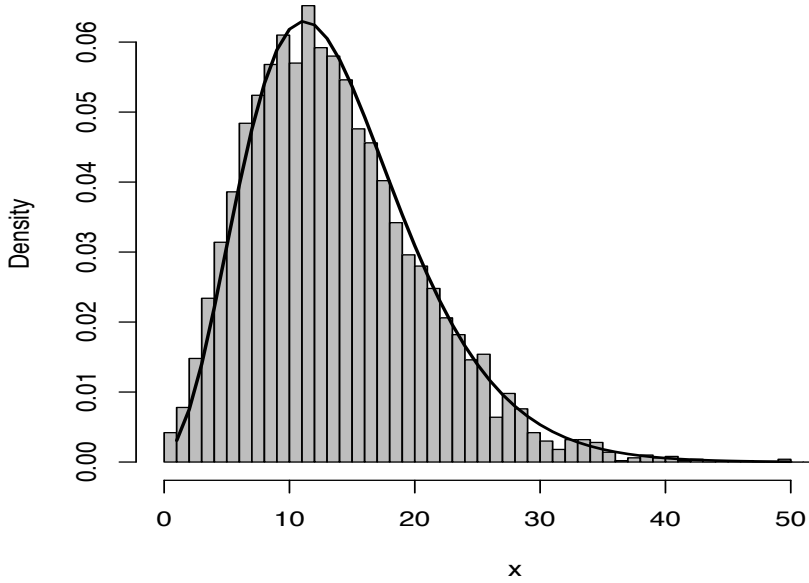
Mixture Representations
Continuous Mixtures

- Student's t density with ν degrees of freedom

$$X|y \sim \mathcal{N}(0, \nu/y) \quad \text{and} \quad Y \sim \chi^2_\nu.$$

  - Generate from a χ^2_ν then from the corresponding normal distribution
  - Obviously, using `rt` is slightly more efficient

- If X is negative binomial $X \sim \mathcal{N}eg(n, p)$
  - $X|y \sim \mathcal{P}(y)$ and $Y \sim \mathcal{G}(n, \beta)$,
  - `R` code generates from this mixture



Accept–Reject Methods
Introduction

- There are many distributions where transform methods fail
- For these cases, we must turn to *indirect* methods
  - We generate a candidate random variable
  - Only accept it subject to passing a test
- This class of methods is extremely powerful.
  - It will allow us to simulate from virtually any distribution.

- Accept–Reject Methods
  - Only require the functional form of the density f of interest
  - f = target, g=candidate
- Where it is simpler to simulate random variables from g

Accept–Reject Methods
Accept–Reject Algorithm

- The only constraints we impose on this candidate density g
  - f and g have compatible supports (i.e., $g(x) > 0$ when $f(x) > 0$).
  - There is a constant M with $f(x)/g(x) \leq M$ for all x.
- $X \sim f$ can be simulated as follows.
  - Generate $Y \sim g$ and, independently, generate $U \sim \mathcal{U}_{[0,1]}$.
  - If $U \leq \frac{1}{M}\frac{f(Y)}{g(Y)}$, set $X = Y$.
  - If the inequality is not satisfied, we then discard Y and U and start again.
- Note that $M = \sup_x \frac{f(x)}{g(x)}$
- $P(\text{ Accept }) = \frac{1}{M}$, Expected Waiting Time $= M$

Accept–Reject Algorithm
R Implementation

Succinctly, the Accept–Reject Algorithm is

Accept–Reject Method

1. Generate $Y \sim g$, $U \sim \mathcal{U}_{[0,1]}$;
2. Accept $X = Y$ if $U \leq f(Y)/Mg(Y)$;
3. Return to 1 otherwise.

▶ R implementation: If `randg` generates from g

```
> u=runif(1)*M
> y=randg(1)
> while (u>f(y)/g(y))
{
    u=runif(1)*M
    y=randg(1)
    }
```

▶ Produces a single generation y from f

Accept–Reject Algorithm
Normals from Double Exponentials

- Candidate $Y \sim \frac{1}{2}\exp(-|y|)$
- Target $X \sim \frac{1}{\sqrt{2\pi}}\exp(-x^2/2)$

$$\frac{\frac{1}{\sqrt{2\pi}}\exp(-y^2/2)}{\frac{1}{2}\exp(-|y|)} \leq \frac{2}{\sqrt{2\pi}}\exp(1/2)$$

  - ▷ Maximum at $y = 1$
- Accept Y if $U \leq \exp(-.5Y^2 + |Y| - .5)$
- Look at `R` code

Accept–Reject Algorithm
Theory

- Why does this method work?
- A straightforward probability calculation shows

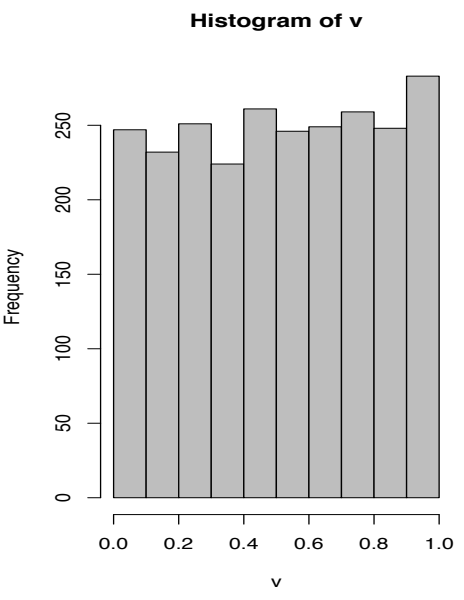
$$P(Y \le x| \text{ Accept }) = P\left(Y \le x | U \le \frac{f(Y)}{Mg(Y)}\right) = P(X \le x)$$

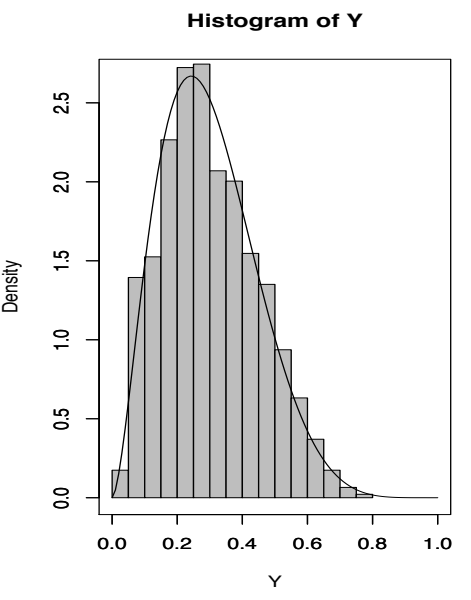
▷ Simulating from g, the output of this algorithm is exactly distributed from f.

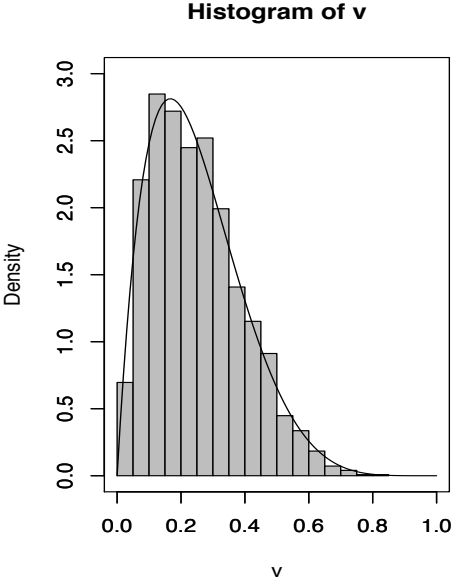
ϟ

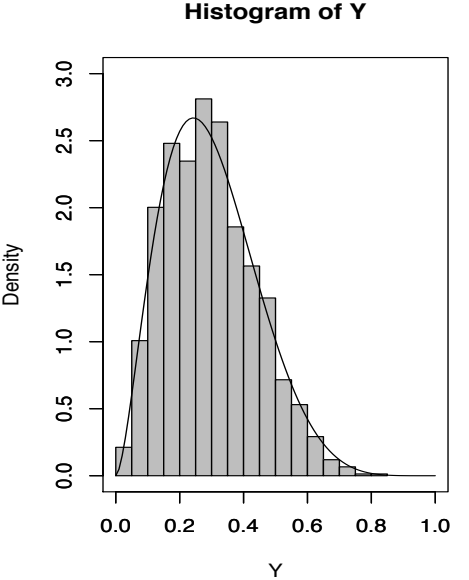
- The Accept–Reject method is applicable in any dimension
- As long as g is a density over the same space as f.

- Only need to know f/g up to a constant
- Only need an upper bound on M


Histogram of v
Frequency
0
50
100
150
200
250
0.0
0.2
0.4
0.6
0.8
1.0
v


Histogram of Y
Density
0.0
0.5
1.0
1.5
2.0
2.5
0.0
0.2
0.4
0.6
0.8
1.0
Y


Histogram of v
Density
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.0
0.2
0.4
0.6
0.8
1.0
v


Histogram of Y
Density
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.0
0.2
0.4
0.6
0.8
1.0
Y

Accept–Reject Algorithm
Betas from Betas-Details

- Beta density $\propto x^a(1-x)^b$
- Can generate if a and b integers
- If not, use candidate with a_1 and b_1 integers

$$\frac{y^a(1-y)^b}{y^{a_1}(1-y)^{b_1}} \text{ maximized at } y = \frac{a-a_1}{a-a_1+b-b_1}$$

  - ▷ Need $a_1 < a$ and $b_1 < b$

- Efficiency ↑ as the candidate gets closer to the target
- Look at `R` code

Accept–Reject Algorithm
Comments

ϟSome key properties of the Accept–Reject algorithm::

1. Only the ratio f/M is needed
   ▷ So the algorithm does not depend on the normalizing constant.
2. The bound $f \leq Mg$ need not be tight
   ▷ Accept–Reject is valid, but less efficient, if M is replaced with a larger constant.
3. The probability of acceptance is $1/M$
   ▷ So M should be as small as possible for a given computational effort.

Chapter 3: Monte Carlo Integration

"Every time I think I know what's going on, suddenly there's another layer of complications. I just want this damn thing solved."

John Scalzi
The Last Colony

This Chapter

- This chapter introduces the major concepts of Monte Carlo methods
- The validity of Monte Carlo approximations relies on the Law of Large Numbers
- The versatility of the representation of an integral as an expectation

Monte Carlo Integration
Introduction

- We will be concerned with evaluating integrals of the form

$$\int_{\mathcal{X}} h(x)\, f(x)\, \mathrm{d}x,$$

  - f is a density
  - We can produce an almost infinite number of random variables from f
- We apply probabilistic results
  - Law of Large Numbers
  - Central Limit Theorem
- The Alternative - Deterministic Numerical Integration
  - `R` functions `area` and `integrate`
  - OK in low (one) dimensions
  - Usually needs some knowledge of the function

Classical Monte Carlo Integration
The Monte Carlo Method

- The generic problem: Evaluate

$$\mathbb{E}_f[h(X)] = \int_{\mathcal{X}} h(x)\, f(x)\, \mathrm{d}x,$$

  - X takes its values in $\mathcal{X}$

- The Monte Carlo Method
  - Generate a sample $(X_1, \ldots, X_n)$ from the density f
  - Approximate the integral with

$$\overline{h}_n = \frac{1}{n} \sum_{j=1}^{n} h(x_j)\ ,$$

Classical Monte Carlo Integration
Validating the Monte Carlo Method

- The Convergence

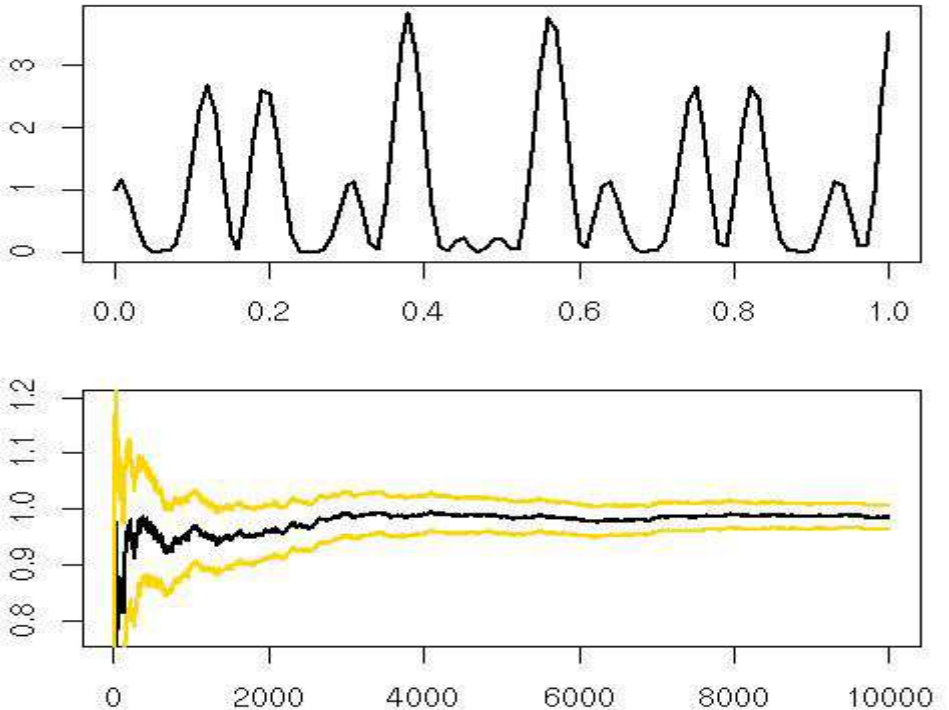
$$\overline{h}_n = \frac{1}{n} \sum_{j=1}^{n} h(x_j) \to \int_{\mathcal{X}} h(x)\, f(x)\, \mathrm{d}x = \mathbb{E}_f[h(X)]$$

  - Is valid by the Strong Law of Large Numbers

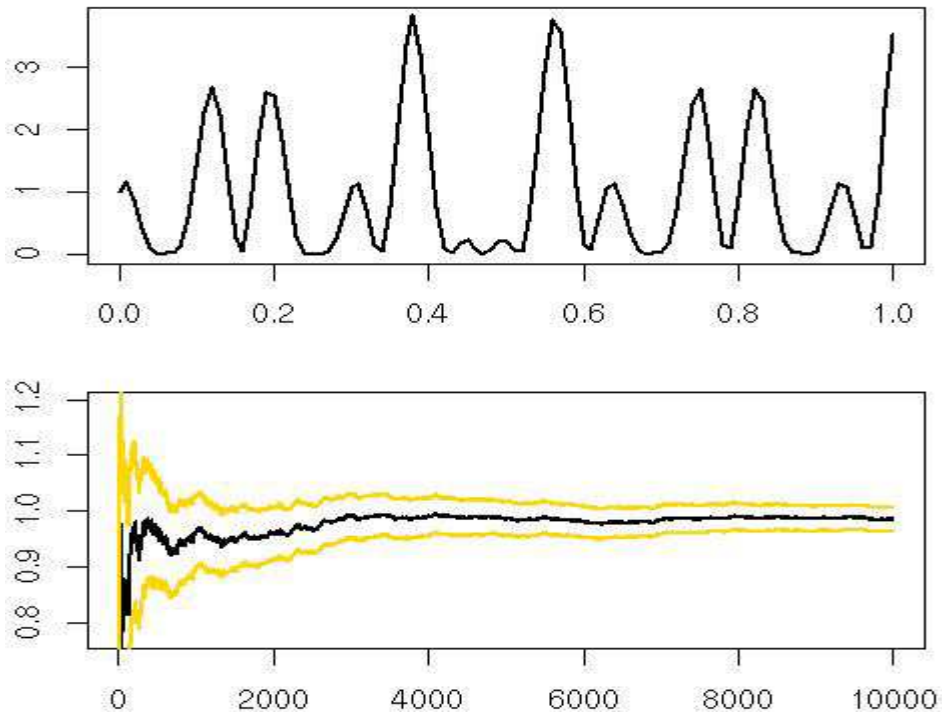
- When $h^2(X)$ has a finite expectation under f,

$$\frac{\overline{h}_n - \mathbb{E}_f[h(X)]}{\sqrt{v_n}} \to \mathcal{N}(0,1)$$

  - Follows from the Central Limit Theorem
  - $v_n = \frac{1}{n^2} \sum_{j=1}^{n} [h(x_j) - \overline{h}_n]^2$.


0
1
2
3
0.0
0.2
0.4
0.6
0.8
1.0
0.8
0.9
1.0
1.1
1.2
0
2000
4000
6000
8000
10000

Classical Monte Carlo Integration
A Caution



- The confidence band produced in this figure is not a 95% confidence band in the classical sense

- They are Confidence Intervals were you to stop at a chosen number of iterations

Classical Monte Carlo Integration
Comments

ϟ

- The evaluation of the Monte Carlo error is a bonus
- It assumes that v_n is a proper estimate of the variance of $\overline{h}_n$
- If v_n does not converge, converges too slowly, a CLT may not apply

Classical Monte Carlo Integration
Another Example

- Normal Probability

$$\hat{\Phi}(t) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}_{x_i \leq t} \to \Phi(t) = \int_{-\infty}^{t} \frac{1}{\sqrt{2\pi}} e^{-y^2/2} \mathrm{d}y$$

  - The exact variance $\Phi(t)[1 - \Phi(t)]/n$
  - Conservative: Var $\approx 1/4n$
  - For a precision of four decimals
    - Want $2 \times \sqrt{1/4n} \leq 10^{-4}$ simulations
    - Take $n = (10^4)^2 = 10^8$

- This method breaks down for tail probabilities

Importance Sampling
Introduction

▶ Importance sampling is based on an alternative formulation of the SLLN

$$\mathbb{E}_f[h(X)] = \int_{\mathcal{X}} h(x)\,\frac{f(x)}{g(x)}\,g(x)\,\mathrm{d}x = \mathbb{E}_g\left[\frac{h(X)f(X)}{g(X)}\right];$$

▷ f is the target density

▷ g is the candidate density

▷ Sound Familiar?

Importance Sampling
Introduction

- Importance sampling is based on an alternative formulation of the SLLN

$$\mathbb{E}_f[h(X)] = \int_{\mathcal{X}} h(x)\,\frac{f(x)}{g(x)}\,g(x)\,\mathrm{d}x = \mathbb{E}_g\left[\frac{h(X)f(X)}{g(X)}\right];$$

  - f is the target density
  - g is the candidate density
  - Sound Familiar? – Just like Accept–Reject

- So

$$\frac{1}{n}\sum_{j=1}^{n}\frac{f(X_j)}{g(X_j)}\,h(X_j) \to \mathbb{E}_f[h(X)]$$

- As long as
  - $\mathrm{Var}\,(h(X)f(X)/g(X)) < \infty$
  - $\mathrm{supp}(g) \supset \mathrm{supp}(h \times f)$

Importance Sampling
Revisiting Normal Tail Probabilities

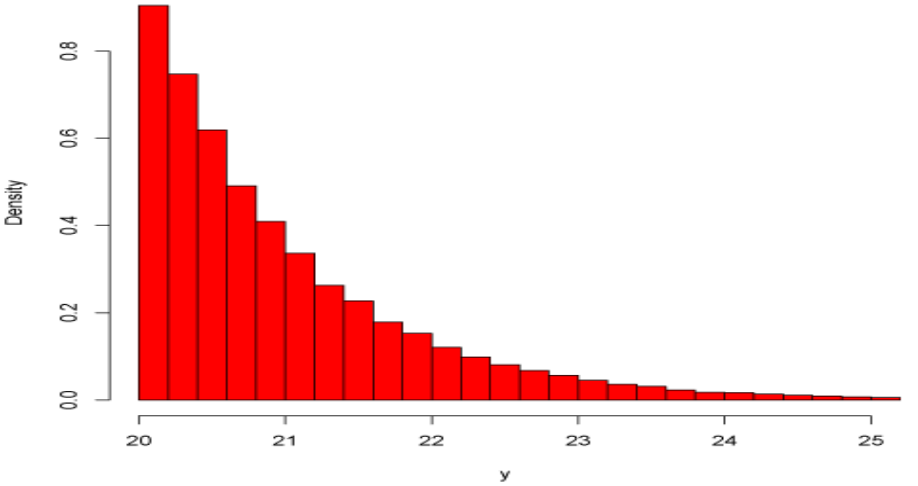
- $Z \sim \mathcal{N}(0,1)$ and we are interested in the probability $P(Z > 4.5)$
- ```
  > pnorm(-4.5,log=T)
  [1] -12.59242
  ```
- Simulating $Z^{(i)} \sim \mathcal{N}(0,1)$ only produces a hit once in about 3 million iterations!
  - ▷ Very rare event for the normal
  - ▷ Not-so-rare for a distribution sitting out there!
- Take $g = \mathcal{E}xp(1)$ truncated at 4.5:

$$g(y) = \frac{e^{-y}}{\int_{4.5}^{\infty} e^{-x}\mathrm{d}x} = e^{-(y-4.5)}\,,$$

- The IS estimator is

$$\frac{1}{n}\sum_{i=1}^{n}\frac{f(Y^{(i)})}{g(Y^{(i)})} = \frac{1}{n}\sum_{i=1}^{n}\frac{e^{-Y_i^2/2+Y_i-4.5}}{\sqrt{2\pi}}$$

R code
  ```


Density
0.0
0.2
0.4
0.6
0.8
20
21
22
23
24
25
y

Importance Sampling
Comments

ϟ Importance sampling has little restriction on the choice of the candidate

- g can be chosen from distributions that are easy to simulate
  - ▷ Or efficient in the approximation of the integral.
- Moreover, the same sample (generated from g) can be used repeatedly
  - ▷ Not only for different functions h but also for different densities f.

Importance Sampling
Easy Model - Difficult Distribution

Example: Beta posterior importance approximation

- Have an observation x from a beta $\mathcal{B}(\alpha, \beta)$ distribution,

$$x \sim \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\, x^{\alpha-1}(1-x)^{\beta-1}\, \mathbb{I}_{[0,1]}(x)$$

- There exists a family of conjugate priors on (α, β) of the form

$$\pi(\alpha,\beta) \propto \left\{\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\right\}^{\lambda} x_0^{\alpha} y_0^{\beta}\,,$$

  where λ, x_0, y_0 are hyperparameters,

- The posterior is then equal to

$$\pi(\alpha,\beta|x) \propto \left\{\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\right\}^{\lambda+1} [x x_0]^{\alpha}[(1-x)y_0]^{\beta}\,.$$

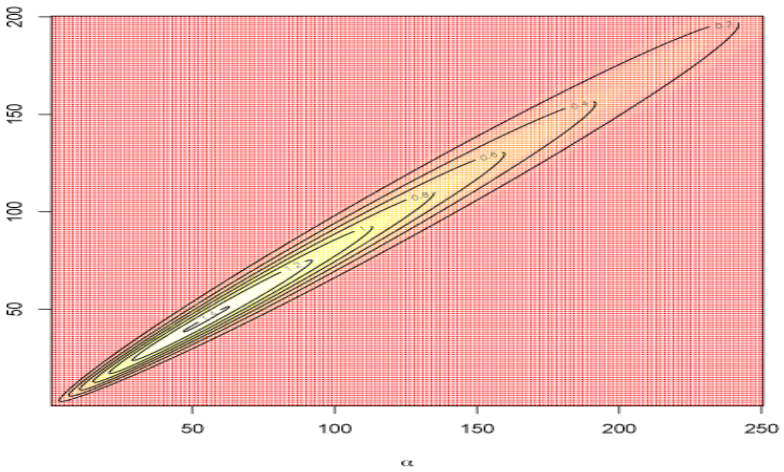
Importance Sampling
Easy Model - Difficult Distribution -2

- The posterior distribution is intractable

$$\pi(\alpha,\beta|x) \propto \left\{\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\right\}^{\lambda+1} [xx_0]^{\alpha}[(1-x)y_0]^{\beta}\,.$$

  - ▷ Difficult to deal with the gamma functions
  - ▷ Simulating directly from $\pi(\alpha,\beta|x)$ is impossible.

- What candidate to use?



- Contour Plot
- Suggest a candidate?
- R code

Importance Sampling
Easy Model - Difficult Distribution – 3

- Try a Bivariate Student's T (or Normal)
- Trial and error
  - Student's $\mathcal{T}(3, \mu, \Sigma)$ distribution with $\mu = (50, 45)$ and
$$\Sigma = \begin{pmatrix} 220 & 190 \\ 190 & 180 \end{pmatrix}$$
  - Produce a reasonable fit
  - R code
- Note that we are using the fact that
$$X \sim f(x) \Rightarrow \Sigma^{1/2} X + \mu \sim f\left((x-\mu)'\Sigma^{-1}(x-\mu)\right)$$

Importance Sampling
Easy Model - Difficult Distribution – Posterior Means

- The posterior mean of α is

$$\int\int \alpha\pi(\alpha,\beta|x)d\alpha d\beta = \int\int \left[\alpha\frac{\pi(\alpha,\beta|x)}{g(\alpha,\beta)}\right] g(\alpha,\beta)d\alpha d\beta \approx \frac{1}{M}\sum_{i=1}^{M}\alpha_i\frac{\pi(\alpha_i,\beta_i|x)}{g(\alpha_i,\beta_i)}$$

  where
  - $\pi(\alpha,\beta|x) \propto \left\{\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\right\}^{\lambda+1} [xx_0]^{\alpha}[(1-x)y_0]^{\beta}$
  - $g(\alpha,\beta) = \mathcal{T}(3,\mu,\Sigma)$

- Note that $\pi(\alpha,\beta|x)$ is not normalized, so we have to calculate

$$\frac{\int\int \alpha\pi(\alpha,\beta|x)d\alpha d\beta}{\int\int \pi(\alpha,\beta|x)d\alpha d\beta} \approx \frac{\sum_{i=1}^{M}\alpha_i\frac{\pi(\alpha_i,\beta_i|x)}{g(\alpha_i,\beta_i)}}{\sum_{i=1}^{M}\frac{\pi(\alpha_i,\beta_i|x)}{g(\alpha_i,\beta_i)}}$$

- The same samples can be used for every posterior expectation

- `R` code

Importance Sampling
Probit Analysis

Example: Probit posterior importance sampling approximation

- y are binary variables, and we have covariates $x \in \mathbb{R}^p$ such that
$$\Pr(y = 1|x) = 1 - \Pr(y = 0|x) = \Phi(x^{\mathrm{T}}\beta)\,, \quad \beta \in \mathbb{R}^p\,.$$
- We return to the dataset `Pima.tr`, x=BMI
- A GLM estimation of the model is (using centered x)

```
>glm(formula = y ~ x, family = binomial(link = "probit"))

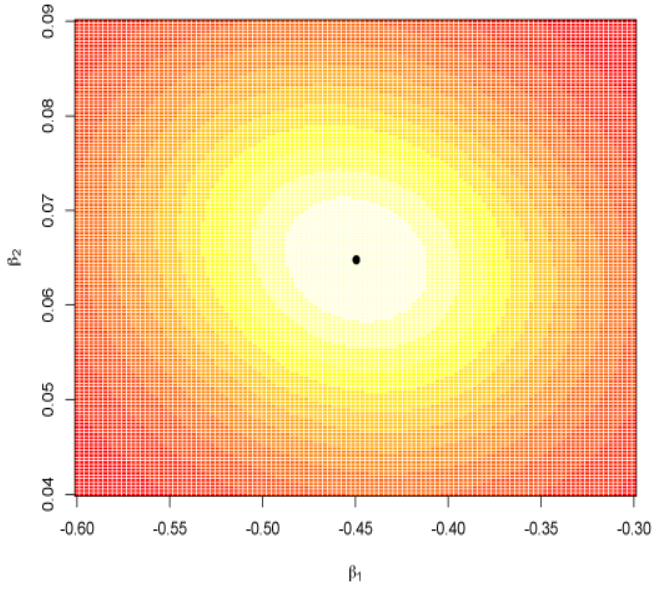
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.44957    0.09497  -4.734 2.20e-06 ***
x            0.06479    0.01615   4.011 6.05e-05 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

So BMI has a significant impact on the possible presence of diabetes.

Importance Sampling
Bayesian Probit Analysis

- From a Bayesian perspective, we use a vague prior
  - $\beta = (\beta_1, \beta_2)$, each having a $\mathcal{N}(0, 100)$ distribution
- With Φ the normal cdf, the posterior is proportional to

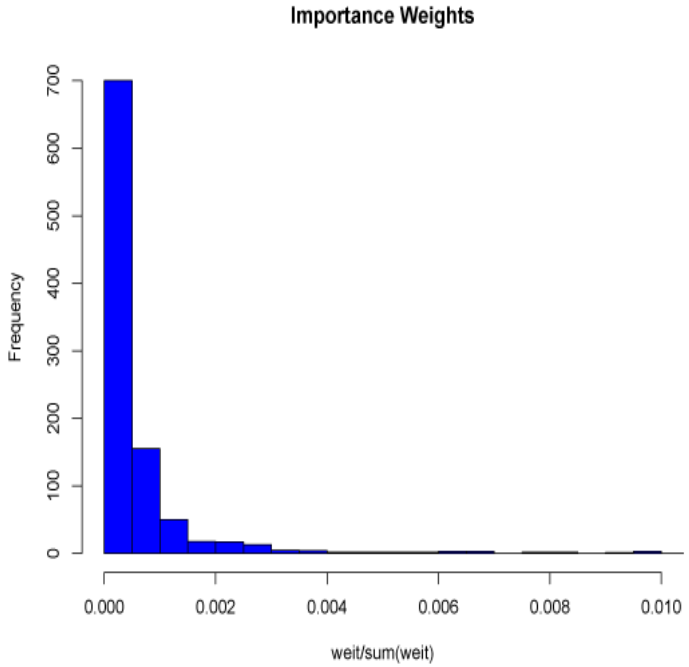
$$\prod_{i=1}^{n} [\Phi(\beta_1 + (x_i - \bar{x})\beta_2]^{y_i} [\Phi(-\beta_1 - (x_i - \bar{x})\beta_2]^{1-y_i} \times e^{-\frac{\beta_1^2+\beta_2^2}{2\times 100}}$$

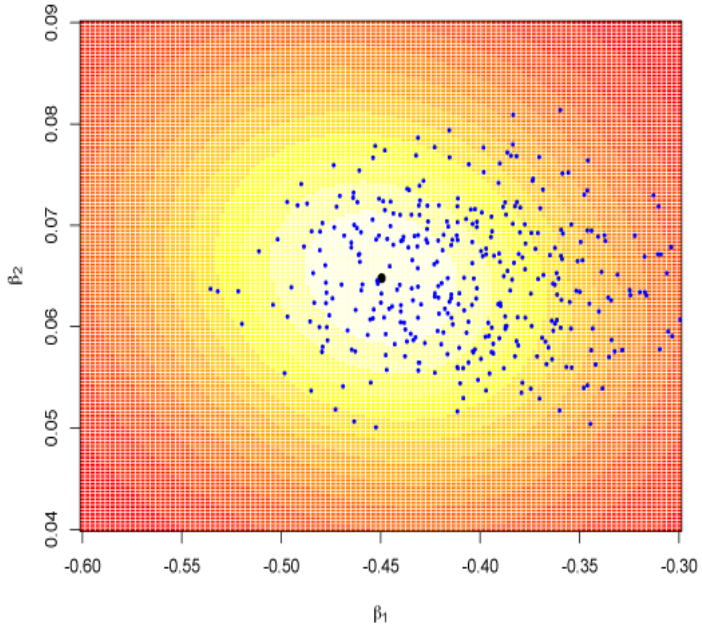


- Level curves of posterior
- MLE in the center
- R code

Importance Sampling
Probit Analysis Importance Weights

- Normal candidate centered at the MLE - no finite variance guarantee
- The importance weights are rather uneven, if not degenerate





- Right side = reweighted candidate sample (R code)
- Somewhat of a failure

Chapter 5: Monte Carlo Optimization

"He invented a game that allowed players to predict the outcome?"

Susanna Gregory
To Kill or Cure

This Chapter

- Two uses of computer-generated random variables to solve optimization problems.
- The first use is to produce stochastic search techniques
  - To reach the maximum (or minimum) of a function
  - Avoid being trapped in local maxima (or minima)
  - Are sufficiently attracted by the global maximum (or minimum).
- The second use of simulation is to approximate the function to be optimized.

Monte Carlo Optimization
Introduction

- Optimization problems can mostly be seen as one of two kinds:
  - Find the extrema of a function $h(\theta)$ over a domain Θ
  - Find the solution(s) to an implicit equation $g(\theta) = 0$ over a domain Θ.
- The problems are exchangeable
  - The second one is a minimization problem for a function like $h(\theta) = g^2(\theta)$
  - while the first one is equivalent to solving $\partial h(\theta)/\partial\theta = 0$
- We only focus on the maximization problem

Monte Carlo Optimization
Deterministic or Stochastic

- Similar to integration, optimization can be deterministic or stochastic
- Deterministic: performance dependent on properties of the function
  - such as convexity, boundedness, and smoothness
- Stochastic (simulation)
  - Properties of h play a lesser role in simulation-based approaches.
- Therefore, if h is complex or Θ is irregular, chose the stochastic approach.

Monte Carlo Optimization
Numerical Optimization

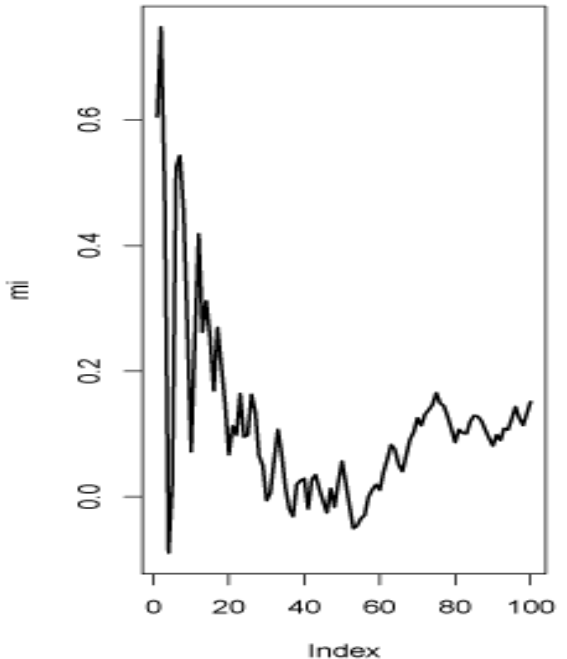
- R has several embedded functions to solve optimization problems
  - The simplest one is `optimize` (one dimensional)

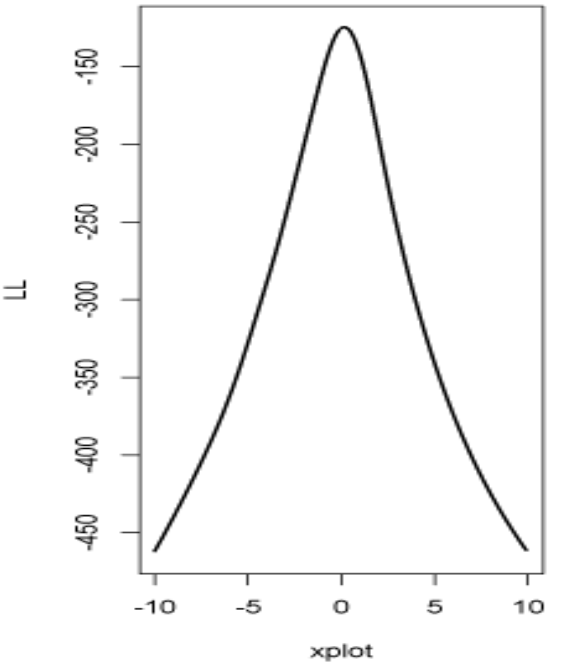
Example: Maximizing a Cauchy likelihood $\mathcal{C}(\theta, 1)$

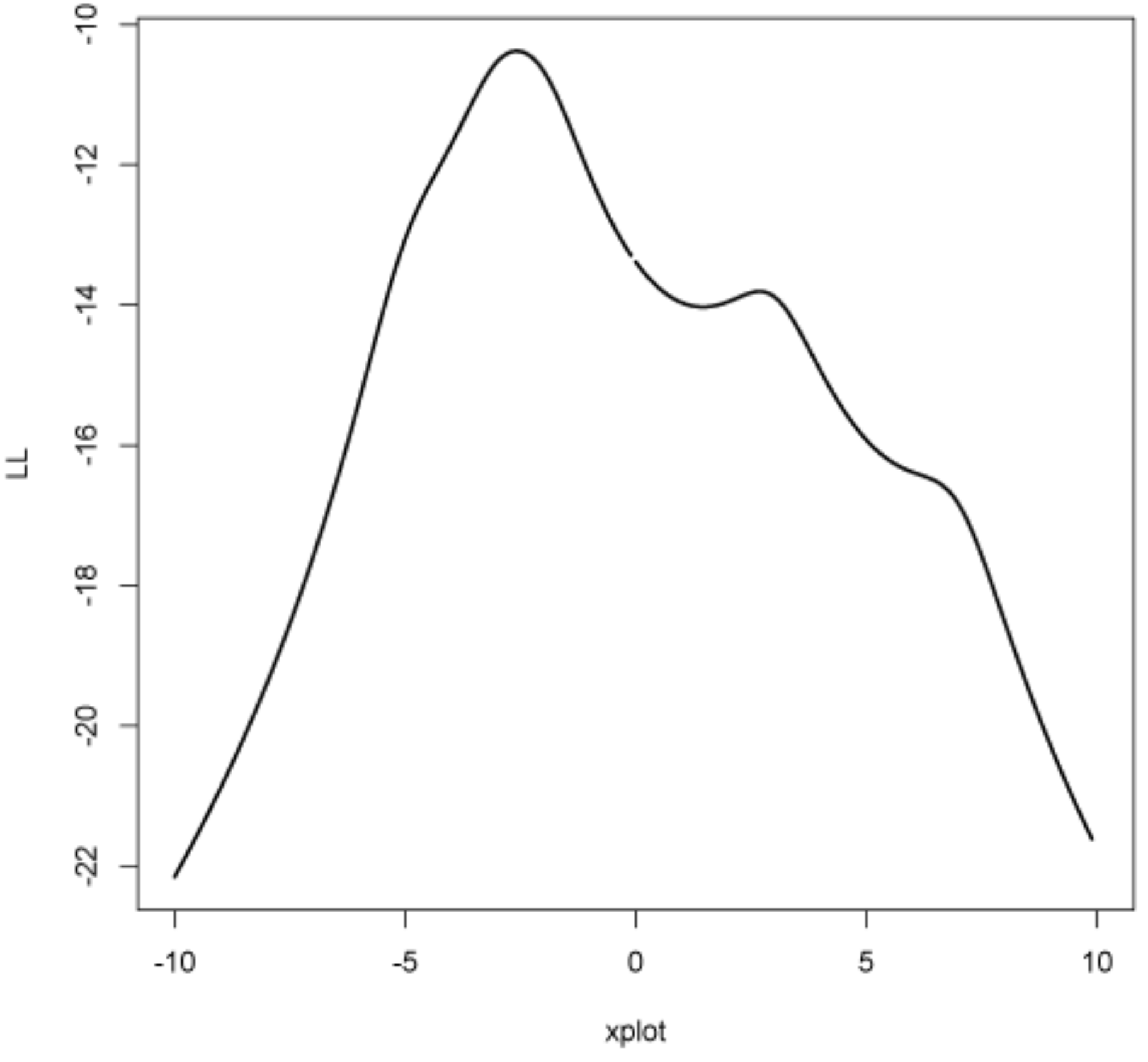
- When maximizing the likelihood of a Cauchy $\mathcal{C}(\theta, 1)$ sample,

$$\ell(\theta|x_1, \ldots, x_n) = \prod_{i=1}^{n} \frac{1}{1 + (x_i - \theta)^2},$$

- The sequence of maxima (MLEs) $\to \theta^* = 0$ when $n \to \infty$.
- But the journey is not a smooth one...


mi
0.0
0.2
0.4
0.6
0
20
40
60
80
100
Index


LL
-150
-200
-250
-300
-350
-400
-450
-10
-5
0
5
10
xplot


LL
-10
-12
-14
-16
-18
-20
-22
-10
-5
0
5
10
xplot

Monte Carlo Optimization
Newton-Raphson

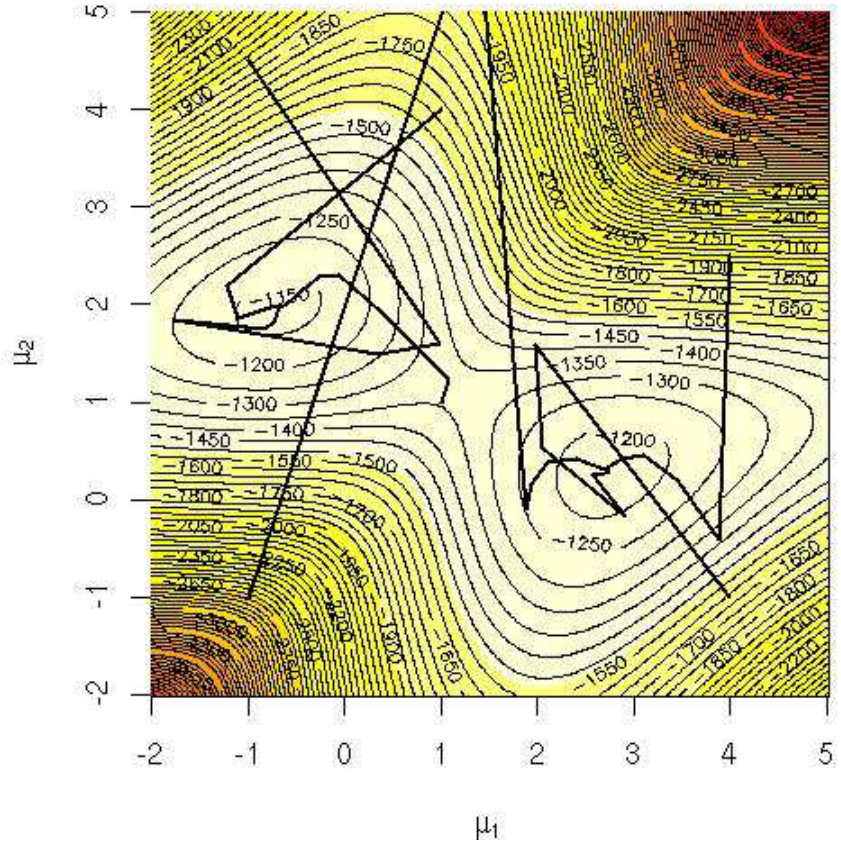
- Similarly, `nlm` is a generic `R` function using the Newton–Raphson method
- Based on the recurrence relation

$$\theta_{i+1} = \theta_i - \left[\frac{\partial^2 h}{\partial\theta\partial\theta^{\mathrm{T}}}(\theta_i)\right]^{-1} \frac{\partial h}{\partial\theta}(\theta_i)$$

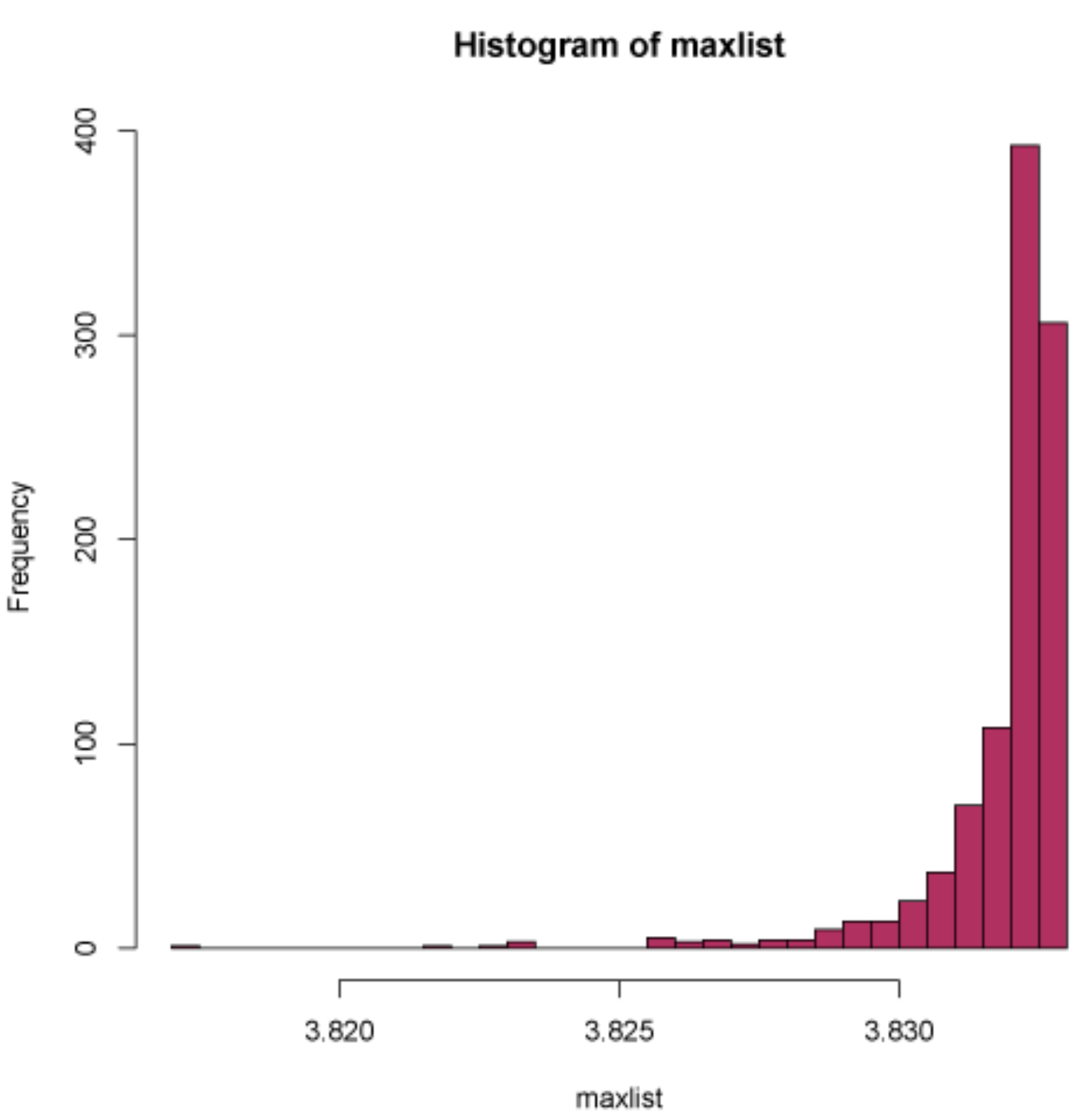
- Where the matrix of the second derivatives is called the *Hessian*
  - This method is perfect when h is quadratic
    - But may also deteriorate when h is highly nonlinear
  - It also obviously depends on the starting point θ_0 when h has several minima.

Monte Carlo Optimization
Newton-Raphson; Mixture Model Likelihood

- Bimodal Mixture Model Likelihood $\frac{1}{4}\mathcal{N}(\mu_1, 1) + \frac{3}{4}\mathcal{N}(\mu_2, 1)$



- Sequences go to the closest mode
- Starting point $(-1, -1)$ has a steep gradient
  - ▷ Bypasses the main mode $(-0.68, 1.98)$
  - ▷ Goes to other mode (lower likelihood)


Histogram of maxlist
Frequency
0
100
200
300
400
3.820
3.825
3.830
maxlist

Stochastic search
Stochastic Gradient Methods

- Generating direct simulations from the target can be difficult.
- Different stochastic approach to maximization
  - Explore the surface in a local manner.
  - Can use $\theta_{j+1} = \theta_j + \epsilon_j$
  - A Markov Chain
  - The random component ϵ_j can be arbitrary
- Can also use features of the function: Newton-Raphson Variation

$$\theta_{j+1} = \theta_j + \alpha_j \nabla h(\theta_j)\,, \qquad \alpha_j > 0\,,$$

  - Where $\nabla h(\theta_j)$ is the gradient
  - α_j the step size

Stochastic search
Stochastic Gradient Methods-2

- In difficult problems
  - The gradient sequence will most likely get stuck in a local extremum of h.
- Stochastic Variation

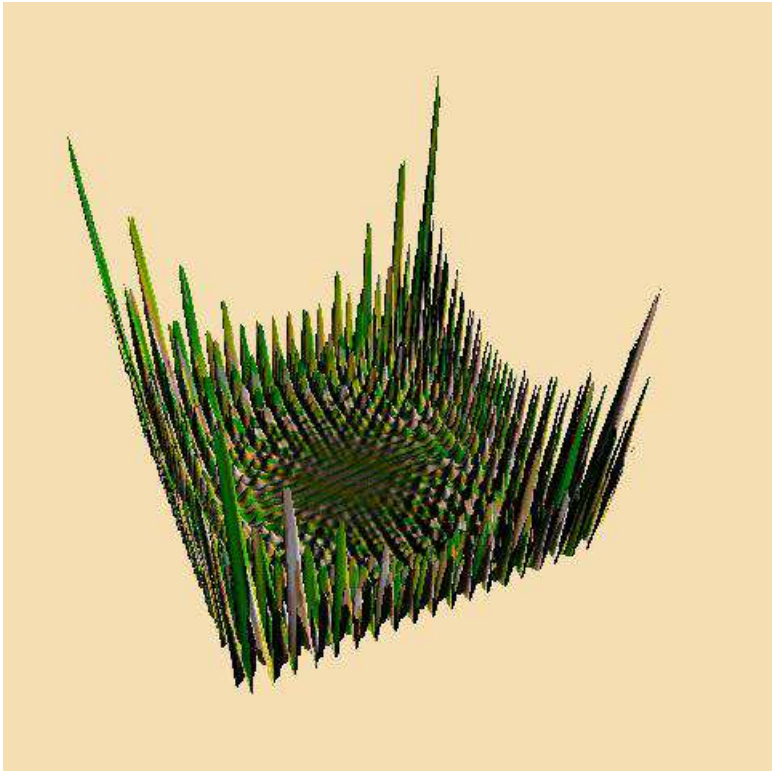
$$\nabla h(\theta_j) \approx \frac{h(\theta_j + \beta_j\zeta_j) - h(\theta_j + \beta_j\zeta_j)}{2\beta_j}\,\zeta_j = \frac{\Delta h(\theta_j, \beta_j\zeta_j)}{2\beta_j}\,\zeta_j\,,$$

  - (β_j) is a second decreasing sequence
  - ζ_j is uniform on the unit sphere $||\zeta|| = 1$.
- We then use

$$\theta_{j+1} = \theta_j + \frac{\alpha_j}{2\beta_j}\,\Delta h(\theta_j, \beta_j\zeta_j)\,\zeta_j$$

Stochastic Search

A Difficult Minimization

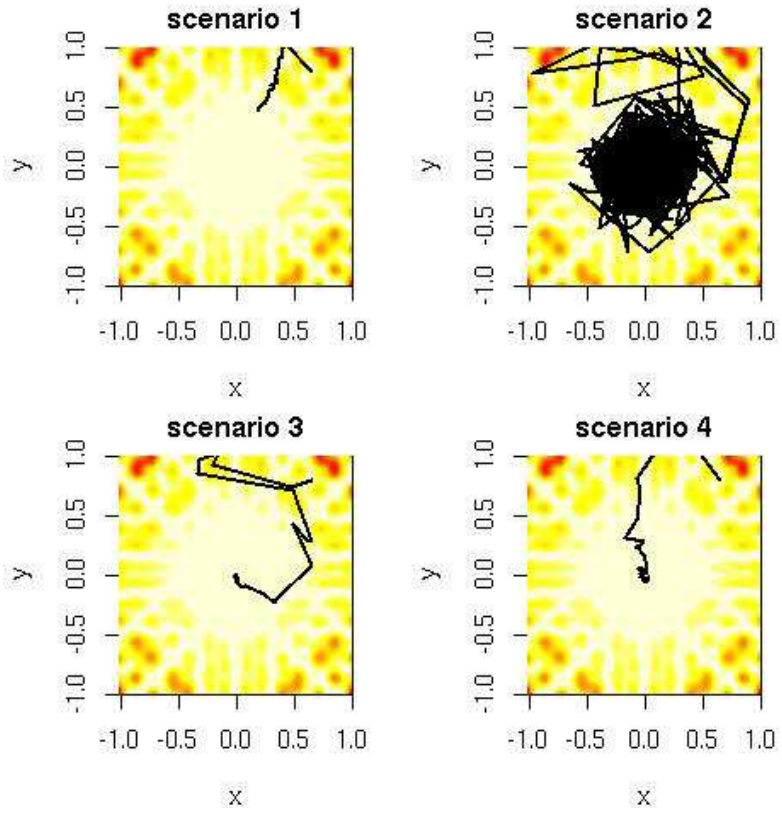


- Many Local Minima
- Global Min at $(0, 0)$
- Code in the text

Stochastic Search

A Difficult Minimization – 2

Scenario	1	2	3	4
α_j	$1/\log(j+1)$	$1/100\log(j+1)$	$1/(j+1)$	$1/(j+1)$
β_j	$1/\log(j+1)^{.1}$	$1/\log(j+1)^{.1}$	$1/(j+1)^{.5}$	$1/(j+1)^{.1}$



- $\alpha \downarrow 0$ slowly, $\sum_j \alpha_j = \infty$
- $\beta \downarrow 0$ more slowly, $\sum_j (\alpha_j/\beta_j)^2 < \infty$
- Scenarios 1-2: Not enough energy
- Scenarios 3-4: Good

Simulated Annealing
Introduction

- This name is borrowed from Metallurgy:
- A metal manufactured by a slow decrease of temperature (*annealing*)
  - Is stronger than a metal manufactured by a fast decrease of temperature.
- The fundamental idea of simulated annealing methods
  - A change of scale, or temperature
  - Allows for faster moves on the surface of the function h to maximize.
  - Rescaling partially avoids the trapping attraction of local maxima.
- As T decreases toward 0, the values simulated from this distribution become concentrated in a narrower and narrower neighborhood of the local maxima of h

Simulated Annealing

Metropolis Algorithm/Simulated Annealing

- Simulation method proposed by Metropolis *et al.* (1953)
- Starting from θ_0, ζ is generated from

$$\zeta \sim \text{Uniform in a neighborhood of } \theta_0.$$

- The new value of θ is generated as

$$\theta_1 = \begin{cases} \zeta & \text{with probability } \rho = \exp(\Delta h/T) \wedge 1 \\ \theta_0 & \text{with probability } 1-\rho, \end{cases}$$

  - $\Delta h = h(\zeta) - h(\theta_0)$
  - If $h(\zeta) \geq h(\theta_0)$, ζ is accepted
  - If $h(\zeta) < h(\theta_0)$, ζ may still be accepted
  - This allows escape from local maxima

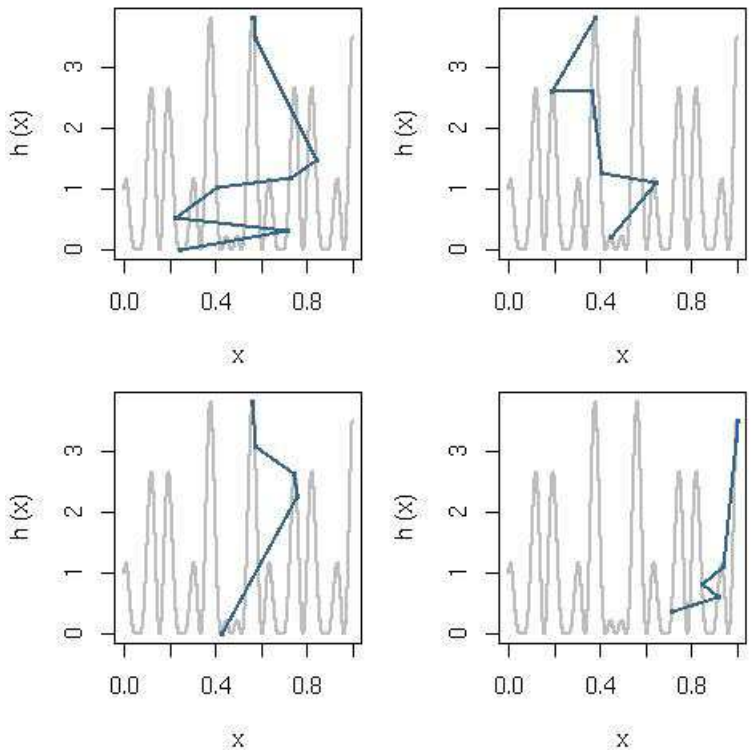
Simulated Annealing
Metropolis Algorithm - Comments

- Simulated annealing typically modifies the temperature T at each iteration
- It has the form

1. Simulate ζ from an instrumental distribution with density $g(|\zeta - \theta_i|)$;
2. Accept $\theta_{i+1} = \zeta$ with probability
$$\rho_i = \exp\{\Delta h_i / T_i\} \wedge 1;$$
take $\theta_{i+1} = \theta_i$ otherwise.
3. Update T_i to T_{i+1}.

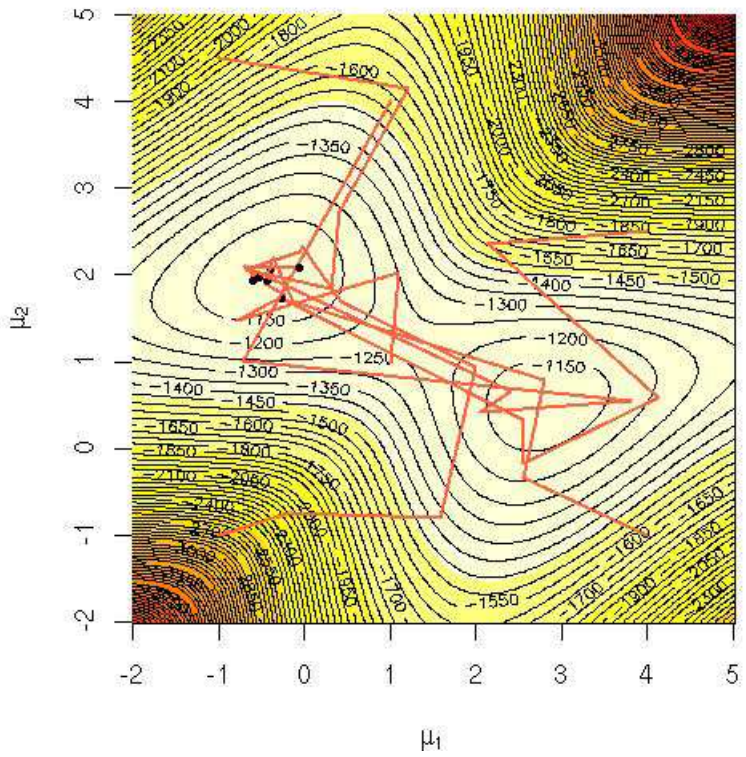
- All positive moves accepted
- As $T \downarrow 0$
  - Harder to accept downward moves
  - No big downward moves
- Not a Markov Chain - difficult to analyze

Simulated Annealing
Simple Example



- Trajectory: $T_i = \frac{1}{(1+i)^2}$
- Log trajectory also works
- Can Guarantee Finding Global Max
- R code

Simulated Annealing
Normal Mixture



- Previous normal mixture
- Most sequences find max
- They visit both modes

Stochastic Approximation
Introduction

- We now consider methods that work with the objective function h
  - Rather than being concerned with fast exploration of the domain Θ.
- Unfortunately, the use of those methods results in an additional level of error
  - Due to this approximation of h.
- But, the objective function in many statistical problems can be expressed as
  - $h(x) = \mathbb{E}[H(x, Z)]$
  - This is the setting of so-called missing-data models

Stochastic Approximation
Optimizing Monte Carlo Approximations

- If $h(x) = \mathbb{E}[H(x, Z)]$, a Monte Carlo approximation is

$$\hat{h}(x) = \frac{1}{m} \sum_{i=1}^{m} H(x, z_i),$$

  - Z_i's are generated from the conditional distribution $f(z|x)$.

- This approximation yields a convergent estimator of $h(x)$ for every value of x

  - This is a *pointwise convergent estimator*
  - Its use in optimization setups is not recommended
  - Changing sample of Z_i's $\Rightarrow$ unstable sequence of evaluations
  - And a rather noisy approximation to $\arg\max h(x)$

Stochastic Approximation
Bayesian Probit

Example: Bayesian analysis of a simple probit model

- $Y \in \{0, 1\}$ has a distribution depending on a covariate X:

$$P_\theta(Y = 1|X = x) = 1 - P_\theta(Y = 0|X = x) = \Phi(\theta_0 + \theta_1 x)\,,$$

  - Illustrate with `Pima.tr` dataset, Y= diabetes indicator, X=BMI

- Typically infer from the marginal posterior

$$\arg\max_{\theta_0} \int \prod_{i=1} \Phi(\theta_0 + \theta_1 x_n)^{y_i} \Phi(-\theta_0 - \theta_1 x_n)^{1-y_i}\, \mathrm{d}\theta_1 = \arg\max_{\theta_0} h(\theta_0)$$

  - For a flat prior on θ and a sample $(x_1, \ldots, x_n)$.

Stochastic Approximation
Bayesian Probit – Importance Sampling

- No analytic expression for h
- The conditional distribution of θ_1 given θ_0 is also nonstandard
  - Use importance sampling with a t distribution with 5 df
  - Take $\mu = 0.1$ and $\sigma = 0.03$ (MLEs)
- Importance Sampling Approximation

$$\widehat{h}_0(\theta_0) = \frac{1}{M}\sum_{m=1}^{M}\prod_{i=1}\Phi(\theta_0+\theta_1^m x_n)^{y_i}\Phi(-\theta_0-\theta_1^m x_n)^{1-y_i}\mathfrak{t}_5(\theta_1^m;\mu,\sigma)^{-1}\,,$$

Stochastic Approximation
Importance Sampling Evaluation

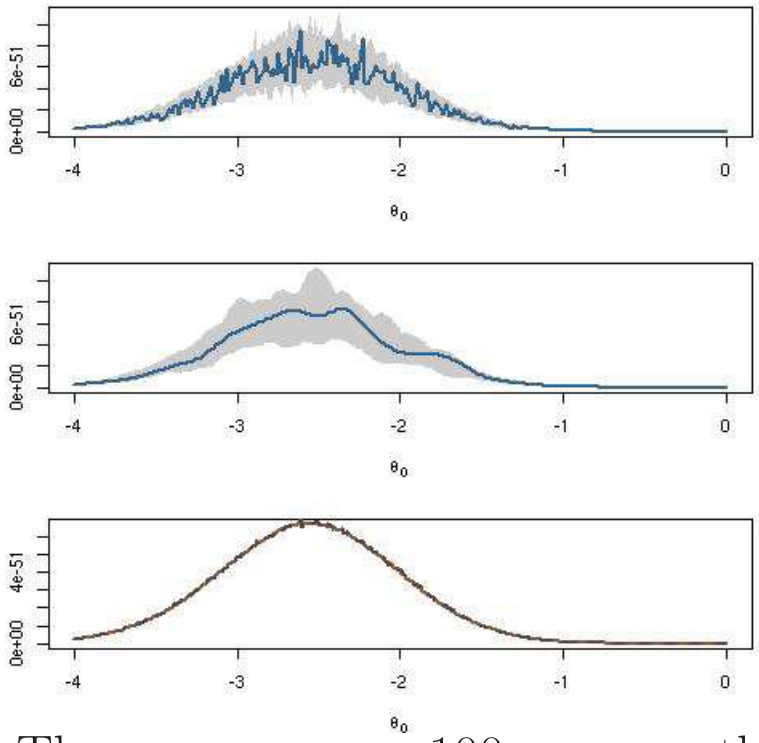
- Plotting this approximation of h with t samples simulated for each value of θ_0
  - The maximization of the represented $\widehat{h}$ function is not to be trusted as an approximation to the maximization of h.

- But, if we use the same t sample for all values of θ_0
  - We obtain a much smoother function

- We use importance sampling based on a *single* sample of Z_i's
  - Simulated from an importance function $g(z)$ for *all* values of x
  - Estimate h with

$$\hat{h}_m(x) = \frac{1}{m} \sum_{i=1}^{m} \frac{f(z_i|x)}{g(z_i)} H(x, z_i).$$

Stochastic Approximation
Importance Sampling Likelihood Representation



- Top: 100 runs, different samples
- Middle: 100 runs, same sample
- Bottom: averages over 100 runs

- The averages over 100 runs are the same - but we will not do 100 runs
- R code: Run `pimax(25)` from `mcsm`

Stochastic Approximation
Comments

- This approach is not absolutely fool-proof
  - The precision of $\hat{h}_m(x)$ has no reason to be independent of x
  - The number m of simulations has to reflect the most varying case.

- As in every importance sampling experiment
  - The choice of the candidate g is influential
  - In obtaining a good (or a disastrous) approximation of $h(x)$.

- Checking for the finite variance of the ratio $f(z_i|x)H(x,z_i)/g(z_i)$
  - Is a minimal requirement in the choice of g

Missing-Data Models and Demarginalization
Introduction

- Missing data models are special cases of the representation $h(x) = \mathbb{E}[H(x, Z)]$
- These are models where the density of the observations can be expressed as

$$g(x|\theta) = \int_{\mathcal{Z}} f(x, z|\theta)\, \mathrm{d}z\,.$$

- This representation occurs in many statistical settings
  - Censoring models and mixtures
  - Latent variable models (tobit, probit, arch, stochastic volatility, etc.)
  - Genetics: Missing SNP calls

Missing-Data Models and Demarginalization
Mixture Model

Example: Normal mixture model as a missing-data model

- Start with a sample $(x_1, \ldots, x_n)$
- Introduce a vector $(z_1, \ldots, z_n) \in \{1, 2\}^n$ such that

$$P_\theta(Z_i = 1) = 1 - P_\theta(Z_i = 2) = 1/4\,, \quad X_i | Z_i = z \sim \mathcal{N}(\mu_z, 1)\,,$$

- The (observed) likelihood is then obtained as $\mathbb{E}[H(\mathbf{x}, \mathbf{Z})]$ for

$$H(\mathbf{x}, \mathbf{z}) \propto \prod_{i;\, z_i=1} \frac{1}{4} \exp\left\{-(x_i - \mu_1)^2/2\right\} \prod_{i;\, z_i=2} \frac{3}{4} \exp\left\{-(x_i - \mu_2)^2/2\right\}\,,$$

- We recover the mixture model

$$\frac{1}{4}\mathcal{N}(\mu_1, 1) + \frac{3}{4}\mathcal{N}(\mu_2, 1)$$

  - As the marginal distribution of X_i.

Missing-Data Models and Demarginalization
Censored–Data Likelihood

Example: Censored–data likelihood

- Censored data may come from experiments
  - Where some potential observations are replaced with a lower bound
  - Because they take too long to observe.

- Suppose that we observe $Y_1, \ldots, Y_m$, iid, from $f(y-\theta)$
  - And the $(n-m)$ remaining $(Y_{m+1}, \ldots, Y_n)$ are censored at the threshold a.

- The corresponding likelihood function is

$$L(\theta|\mathbf{y}) = [1 - F(a-\theta)]^{n-m} \prod_{i=1}^{m} f(y_i - \theta),$$

  - F is the cdf associated with f

Missing-Data Models and Demarginalization
Recovering the Observed Data Likelihood

- If we had observed the last $n - m$ values
  - Say $\mathbf{z} = (z_{m+1}, \ldots, z_n)$, with $z_i \geq a$ $(i = m+1, \ldots, n)$,
  - We could have constructed the (complete data) likelihood

$$L^c(\theta|\mathbf{y}, \mathbf{z}) = \prod_{i=1}^{m} f(y_i - \theta) \prod_{i=m+1}^{n} f(z_i - \theta)\,.$$

- Note that

$$L(\theta|\mathbf{y}) = \mathbb{E}[L^c(\theta|\mathbf{y}, \mathbf{Z})] = \int_{\mathcal{Z}} L^c(\theta|\mathbf{y}, \mathbf{z}) k(\mathbf{z}|\mathbf{y}, \theta)\, \mathrm{d}\mathbf{z},$$

  - Where $k(\mathbf{z}|\mathbf{y}, \theta)$ is the density of the missing data
  - Conditional on the observed data
  - The product of the $f(z_i - \theta)/[1 - F(a - \theta)]$'s
  - $f(z - \theta)$ restricted to $(a, +\infty)$.

Missing-Data Models and Demarginalization
Comments

- When we have the relationship
$$g(x|\theta) = \int_{\mathcal{Z}} f(x, z|\theta)\, \mathrm{d}z\,.$$
  - $\mathbf{Z}$ merely serves to simplify calculations
  - it does not necessarily have a specific meaning
- We have the complete-data likelihood $L^c(\theta|\mathbf{x}, \mathbf{z})) = f(\mathbf{x}, \mathbf{z}|\theta)$
  - The likelihood we would obtain
  - Were we to observe $(\mathbf{x}, \mathbf{z})$,the complete data
- REMEMBER:
$$g(x|\theta) = \int_{\mathcal{Z}} f(x, z|\theta)\, \mathrm{d}z\,.$$

The EM Algorithm
Introduction

- The EM algorithm is a deterministic optimization technique
  - Dempster, Laird and Rubin 1977
- Takes advantage of the missing data representation
  - Builds a sequence of easier maximization problems
  - Whose limit is the answer to the original problem

- We assume that we observe $X_1, \ldots, X_n \sim g(\mathbf{x}|\theta)$ that satisfies

$$g(\mathbf{x}|\theta) = \int_{\mathcal{Z}} f(\mathbf{x}, \mathbf{z}|\theta)\, \mathrm{d}\mathbf{z},$$

  - And we want to compute $\hat{\theta} = \arg\max L(\theta|\mathbf{x}) = \arg\max g(\mathbf{x}|\theta)$.

The EM Algorithm
First Details

- With the relationship $g(\mathbf{x}|\theta) = \int_{\mathcal{Z}} f(\mathbf{x}, \mathbf{z}|\theta)\, \mathrm{d}\mathbf{z}$,
  - $(\mathbf{X}, \mathbf{Z}) \sim f(\mathbf{x}, \mathbf{z}|\theta)$
- The conditional distribution of the missing data $\mathbf{Z}$
  - Given the observed data $\mathbf{x}$ is

$$k(\mathbf{z}|\theta, \mathbf{x}) = f(\mathbf{x}, \mathbf{z}|\theta) \big/ g(\mathbf{x}|\theta)\,.$$

- Taking the logarithm of this expression leads to the following relationship

$$\underbrace{\log L(\theta|\mathbf{x})}_{\text{Obs. Data}} = \underbrace{\mathbb{E}_{\theta_0}[\log L^c(\theta|\mathbf{x}, \mathbf{Z})]}_{\text{Complete Data}} - \underbrace{\mathbb{E}_{\theta_0}[\log k(\mathbf{Z}|\theta, \mathbf{x})]}_{\text{Missing Data}},$$

- Where the expectation is with respect to $k(\mathbf{z}|\theta_0, \mathbf{x})$.
- In maximizing $\log L(\theta|\mathbf{x})$, we can ignore the last term

The EM Algorithm
Iterations

- Denoting
$$Q(\theta|\theta_0, \mathbf{x}) = \mathbb{E}_{\theta_0}[\log L^c(\theta|\mathbf{x}, \mathbf{Z})],$$
- EM algorithm indeed proceeds by maximizing $Q(\theta|\theta_0, \mathbf{x})$ at each iteration
  - If $\hat{\theta}_{(1)} = \text{argmax}Q(\theta|\theta_0, \mathbf{x})$, $\hat{\theta}_{(0)} \to \hat{\theta}_{(1)}$
- Sequence of estimators $\{\hat{\theta}_{(j)}\}$, where
$$\hat{\theta}_{(j)} = \text{argmax}Q(\theta|\hat{\theta}_{(j-1)})$$
- This iterative scheme
  - Contains both an expectation step
  - And a maximization step
  - Giving the algorithm its name.

The EM Algorithm

The Algorithm

Pick a starting value $\hat{\theta}_{(0)}$ and set $m = 0$.

Repeat

1. Compute (*the E-step*)

$$Q(\theta|\hat{\theta}_{(m)}, \mathbf{x}) = \mathbb{E}_{\hat{\theta}_{(m)}}[\log L^c(\theta|\mathbf{x}, \mathbf{Z})]\,,$$

where the expectation is with respect to $k(\mathbf{z}|\hat{\theta}_{(m)}, \mathbf{x})$.

2. Maximize $Q(\theta|\hat{\theta}_{(m)}, \mathbf{x})$ in θ and take (*the M-step*)

$$\hat{\theta}_{(m+1)} = \arg\max_{\theta} Q(\theta|\hat{\theta}_{(m)}, \mathbf{x})$$

and set $m = m + 1$

until a fixed point is reached; i.e., $\hat{\theta}_{(m+1)} = \hat{\theta}_{(m)}$.fixed point

The EM Algorithm
Properties

- Jensen's inequality $\Rightarrow$ The likelihood increases at each step of the EM algorithm

$$L(\hat{\theta}_{(j+1)}|\mathbf{x}) \geq L(\hat{\theta}_{(j)}|\mathbf{x}),$$

  - ▷ Equality holding if and only if $Q(\hat{\theta}_{(j+1)}|\hat{\theta}_{(j)}, \mathbf{x}) = Q(\hat{\theta}_{(j)}|\hat{\theta}_{(j)}, \mathbf{x})$.

- Every limit point of an EM sequence $\{\hat{\theta}_{(j)}\}$ is a stationary point of $L(\theta|\mathbf{x})$
  - ▷ Not necessarily the maximum likelihood estimator
  - ▷ In practice, we run EM several times with different starting points.

- Implementing the EM algorithm thus means being able to

  (a) Compute the function $Q(\theta'|\theta, \mathbf{x})$

  (b) Maximize this function.

The EM Algorithm
Censored Data Example

- The complete-data likelihood is

$$L^c(\theta|\mathbf{y},\mathbf{z}) \propto \prod_{i=1}^{m} \exp\{-(y_i-\theta)^2/2\} \prod_{i=m+1}^{n} \exp\{-(z_i-\theta)^2/2\}\,,$$

- With expected complete-data log-likelihood

$$Q(\theta|\theta_0,\mathbf{y}) = -\frac{1}{2}\sum_{i=1}^{m}(y_i-\theta)^2 - \frac{1}{2}\sum_{i=m+1}^{n}\mathbb{E}_{\theta_0}[(Z_i-\theta)^2]\,,$$

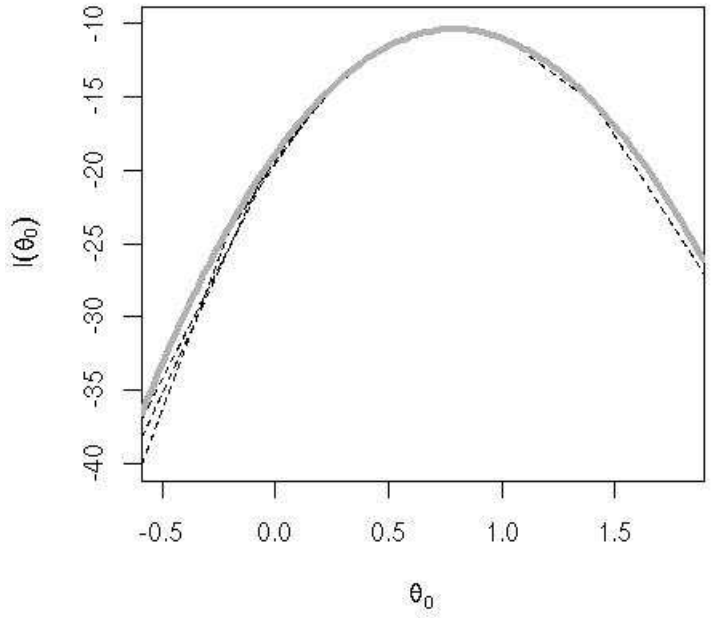
  ▷ the Z_i are distributed from a normal $\mathcal{N}(\theta,1)$ distribution truncated at a.

- M-step (differentiating $Q(\theta|\theta_0,\mathbf{y})$ in θ and setting it equal to 0 gives

$$\hat{\theta} = \frac{m\bar{y} + (n-m)\mathbb{E}_{\theta'}[Z_1]}{n}.$$

  ▷ With $\mathbb{E}_\theta[Z_1] = \theta + \frac{\varphi(a-\theta)}{1-\Phi(a-\theta)}$,

The EM Algorithm
Censored Data MLEs



- EM sequence

$$\hat{\theta}^{(j+1)} = \frac{m}{n}\bar{y} + \frac{n-m}{n}\left[\hat{\theta}^{(j)} + \frac{\varphi(a-\hat{\theta}^{(j)})}{1-\Phi(a-\hat{\theta}^{(j)})}\right]$$

- Climbing the Likelihood
- R code

The EM Algorithm
Normal Mixture

▶ Normal Mixture Bimodal Likelihood

$$Q(\theta'|\theta,\mathbf{x}) = -\frac{1}{2}\sum_{i=1}^{n}\mathbb{E}_\theta\left[Z_i(x_i-\mu_1)^2 + (1-Z_i)(x_i-\mu_2)^2\Big|\,\mathbf{x}\right].$$

Solving the M-step then provides the closed-form expressions

$$\mu_1' = \mathbb{E}_\theta\left[\sum_{i=1}^{n} Z_i x_i|\mathbf{x}\right] \Big/ \mathbb{E}_\theta\left[\sum_{i=1}^{n} Z_i|\mathbf{x}\right]$$

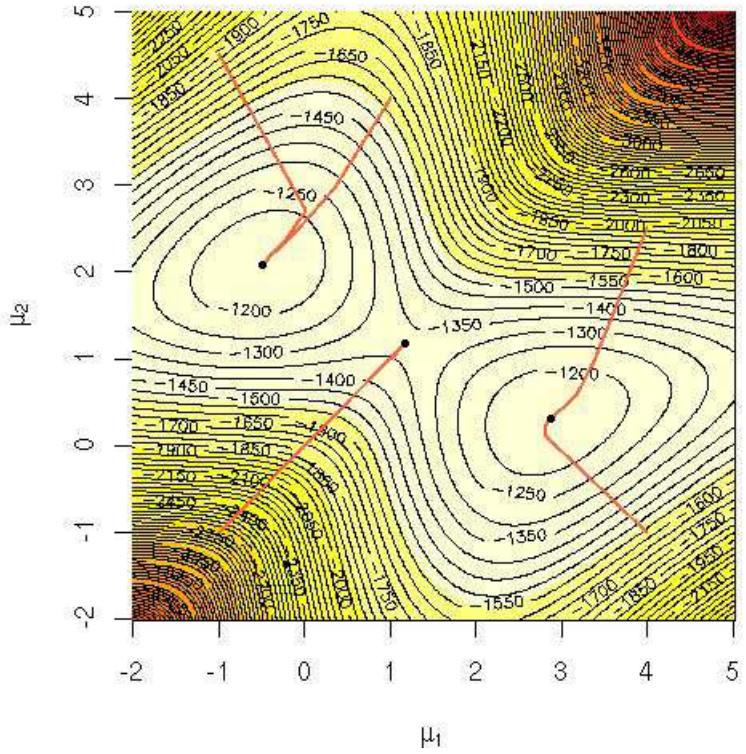
and

$$\mu_2' = \mathbb{E}_\theta\left[\sum_{i=1}^{n}(1-Z_i) x_i|\mathbf{x}\right] \Big/ \mathbb{E}_\theta\left[\sum_{i=1}^{n}(1-Z_i)|\mathbf{x}\right].$$

Since

$$\mathbb{E}_\theta\left[Z_i|\mathbf{x}\right] = \frac{\varphi(x_i-\mu_1)}{\varphi(x_i-\mu_1)+3\varphi(x_i-\mu_2)},$$

The EM Algorithm
Normal Mixture MLEs



- EM five times with various starting points
- Two out of five sequences → higher mode
- Others → lower mode

Monte Carlo EM
Introduction

- If computation $Q(\theta|\theta_0, \mathbf{x})$ is difficult, can use Monte Carlo
- For $\mathbf{Z}_1, \ldots, \mathbf{Z}_T \sim k(\mathbf{z}|\mathbf{x}, \hat{\theta}_{(m)})$, maximize

$$\hat{Q}(\theta|\theta_0, \mathbf{x}) = \frac{1}{T}\sum_{i=1}^{T} \log L^c(\theta|\mathbf{x}, \mathbf{z}_i)$$

- Better: Use importance sampling
  - Since

$$\arg\max_\theta L(\theta|\mathbf{x}) = \arg\max_\theta \log \frac{g(\mathbf{x}|\theta)}{g(\mathbf{x}|\theta_{(0)})} = \arg\max_\theta \log \mathbb{E}_{\theta_{(0)}}\left[\frac{f(\mathbf{x}, \mathbf{z}|\theta)}{f(\mathbf{x}, \mathbf{z}|\theta_{(0)})}\middle| \mathbf{x}\right],$$

  - Use the approximation to the log-likelihood

$$\log L(\theta|\mathbf{x}) \approx \frac{1}{T}\sum_{i=1}^{T} \frac{L^c(\theta|\mathbf{x}, \mathbf{z}_i)}{L^c(\theta_{(0)}|\mathbf{x}, \mathbf{z}_i)},$$

Monte Carlo EM
Genetics Data

Example: Genetic linkage.

- A classic example of the EM algorithm
- Observations (x_1, x_2, x_3, x_4) are gathered from the multinomial distribution

$$\mathcal{M}\left(n; \frac{1}{2} + \frac{\theta}{4}, \frac{1}{4}(1-\theta), \frac{1}{4}(1-\theta), \frac{\theta}{4}\right).$$

- Estimation is easier if the x_1 cell is split into two cells
  - We create the augmented model

$$(z_1, z_2, x_2, x_3, x_4) \sim \mathcal{M}\left(n; \frac{1}{2}, \frac{\theta}{4}, \frac{1}{4}(1-\theta), \frac{1}{4}(1-\theta), \frac{\theta}{4}\right)$$

  with $x_1 = z_1 + z_2$.

  - Complete-data likelihood: $\theta^{z_2+x_4}(1-\theta)^{x_2+x_3}$
  - Observed-data likelihood: $(2+\theta)^{x_1}\theta^{x_4}(1-\theta)^{x_2+x_3}$

Monte Carlo EM
Genetics Linkage Calculations

- The expected complete log-likelihood function is

$$\mathbb{E}_{\theta_0}[(Z_2 + x_4)\log\theta + (x_2 + x_3)\log(1-\theta)] = \left(\frac{\theta_0}{2+\theta_0}x_1 + x_4\right)\log\theta + (x_2 + x_3)\log(1-\theta),$$

  - which can easily be maximized in θ, leading to the EM step

$$\hat{\theta}_1 = \left\{\frac{\theta_0\, x_1}{2+\theta_0}\right\} \Big/ \left\{\frac{\theta_0\, x_1}{2+\theta_0} + x_2 + x_3 + x_4\right\}.$$

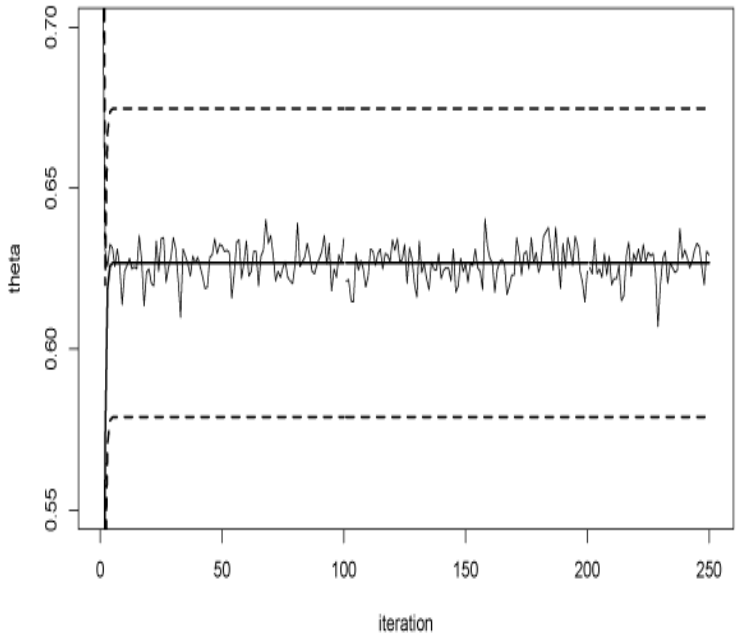
- Monte Carlo EM: Replace the expectation with
  - $\overline{z}_m = \frac{1}{m}\sum_{i=1}^{m} z_i$, $z_i \sim \mathcal{B}(x_1, \theta_0/(2+\theta_0))$

- The MCEM step would then be

$$\widehat{\widehat{\theta}}_1 = \frac{\overline{z}_m}{\overline{z}_m + x_2 + x_3 + x_4},$$

  which converges to $\hat{\theta}_1$ as m grows to infinity.

Monte Carlo EM
Genetics Linkage MLEs



- Note variation in MCEM sequence
- Can control with ↑ simulations
- R code

Monte Carlo EM
Random effect logit model

Example: Random effect logit model

- Random effect logit model,
  - y_{ij} is distributed conditionally on one covariate x_{ij} as a logit model

$$P(y_{ij} = 1|x_{ij}, u_i, \beta) = \frac{\exp\{\beta x_{ij} + u_i\}}{1 + \exp\{\beta x_{ij} + u_i\}},$$

  - $u_i \sim \mathcal{N}(0, \sigma^2)$ is an unobserved random effect.
  - $(U_1, \ldots, U_n)$ therefore corresponds to the missing data $\mathbf{Z}$

Monte Carlo EM
Random effect logit model likelihood

- For the complete data likelihood with $\theta = (\beta, \sigma)$,

$$\begin{aligned} Q(\theta'|\theta, \mathbf{x}, \mathbf{y}) = &\sum_{i,j} y_{ij} \mathbb{E}[\beta' x_{ij} + U_i | \beta, \sigma, \mathbf{x}, \mathbf{y}] \\ &- \sum_{i,j} \mathbb{E}[\log 1 + \exp\{\beta' x_{ij} + U_i\} | \beta, \sigma, \mathbf{x}, \mathbf{y}] \\ &- \sum_{i} \mathbb{E}[U_i^2 | \beta, \sigma, \mathbf{x}, \mathbf{y}]/2\sigma'^2 - n \log \sigma' \,, \end{aligned}$$

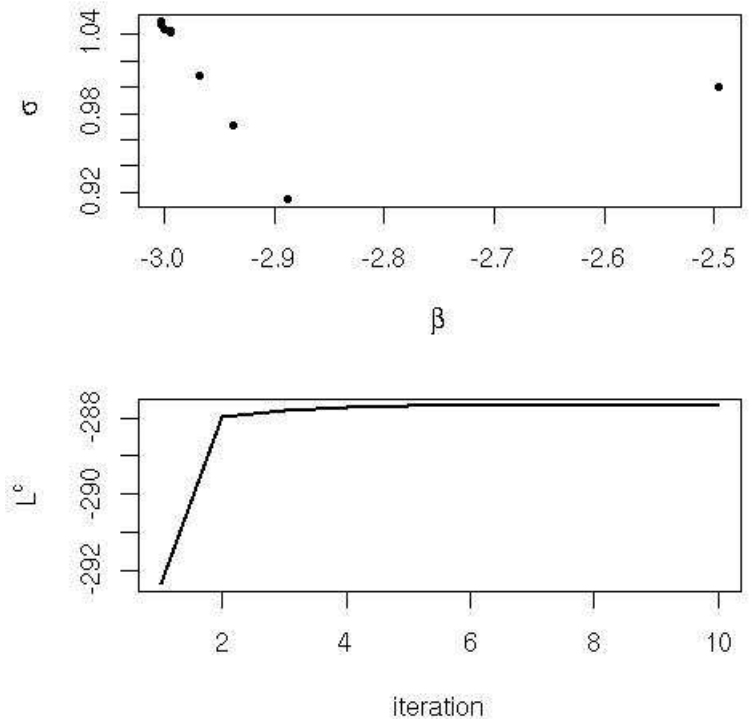
  - ▷ it is impossible to compute the expectations in U_i.

- Were those available, the M-step would be difficult but feasible

- MCEM: Simulate the U_i's conditional on $\beta, \sigma, \mathbf{x}, \mathbf{y}$ from

$$\pi(u_i|\beta, \sigma, \mathbf{x}, \mathbf{y}) \propto \frac{\exp\left\{\sum_j y_{ij} u_i - u_i^2/2\sigma^2\right\}}{\prod_j \left[1 + \exp\left\{\beta x_{ij} + u_i\right\}\right]}$$

Monte Carlo EM
Random effect logit MLEs



- Top: Sequence of β's from the MCEM algorithm
- Bottom: Sequence of completed likelihoods

- MCEM sequence
  - ▷ Increases the number of Monte Carlo steps at each iteration
- MCEM algorithm
  - ▷ Does not have EM monotonicity property