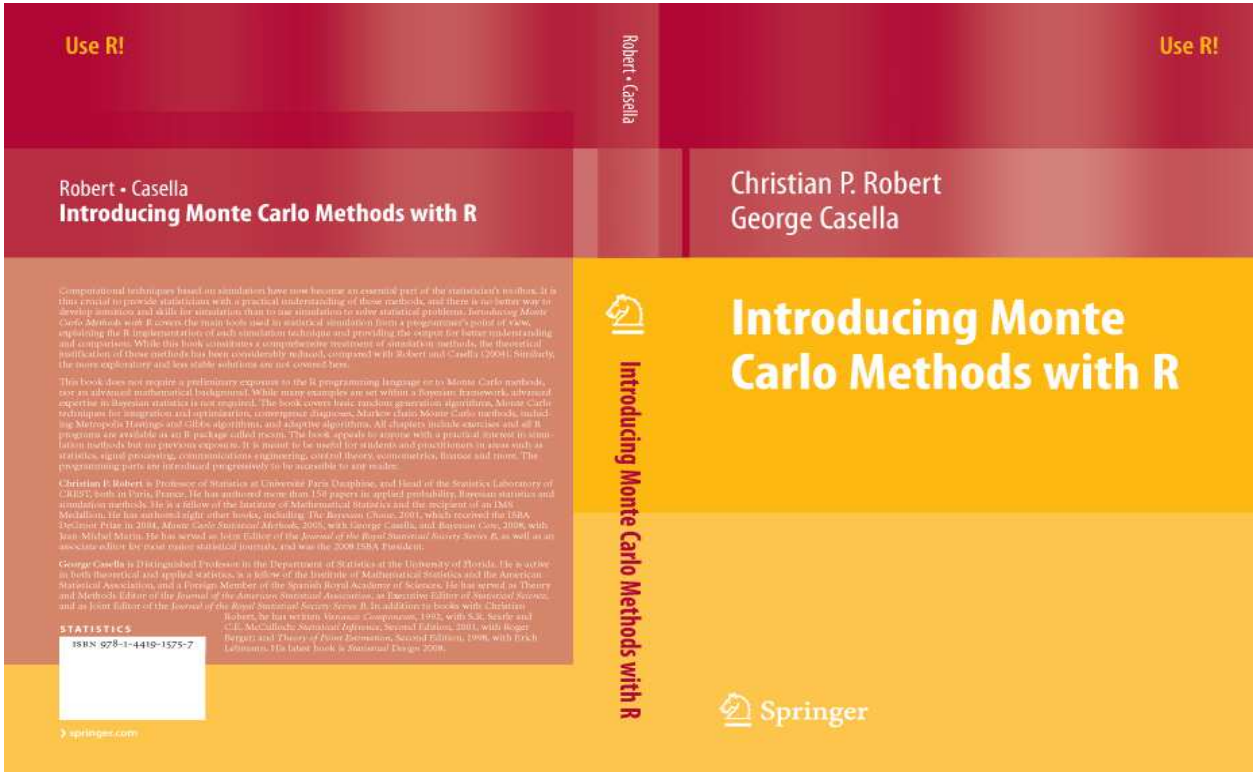
Introducing Monte Carlo Methods with R

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])

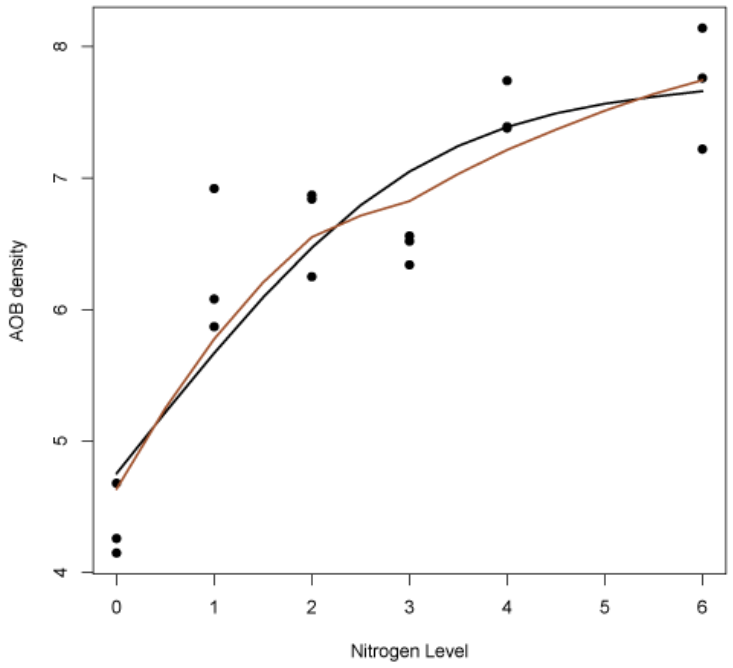
        Pearson's product-moment correlation

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

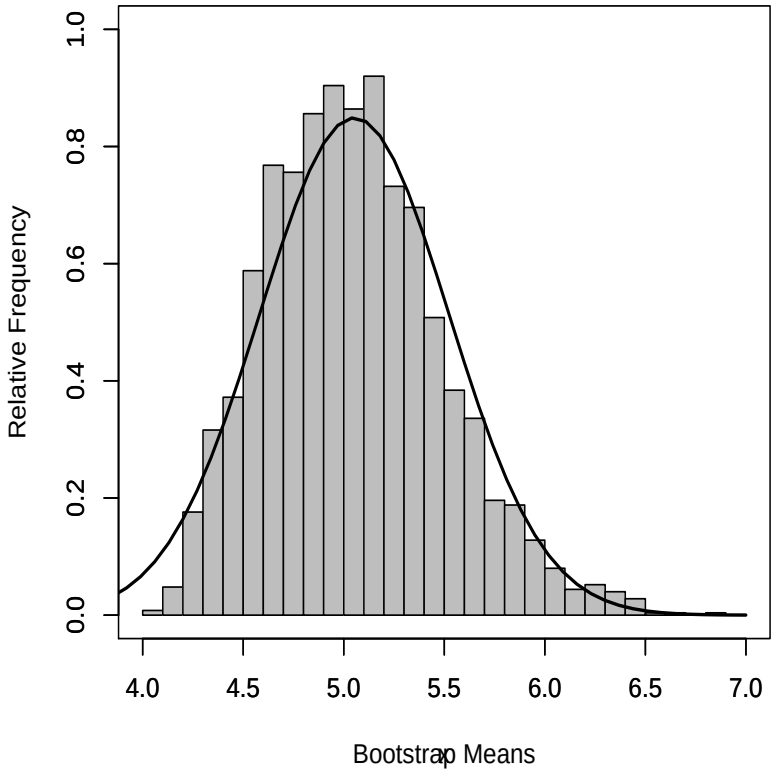
- The bootstrap procedure uses the empirical distribution as a substitute for the true distribution to construct variance estimates and confidence intervals.
  - A sample $X_1, \ldots, X_n$ is resampled with replacement
  - The empirical distribution has a finite but large support made of n^n points

- 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



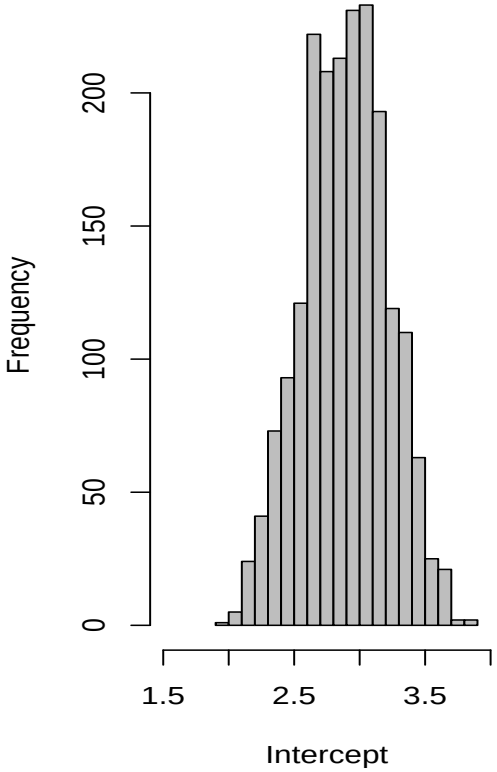
- A histogram of 2500 bootstrap means
- Along with the normal approximation
- Bootstrap shows some skewness
- R code

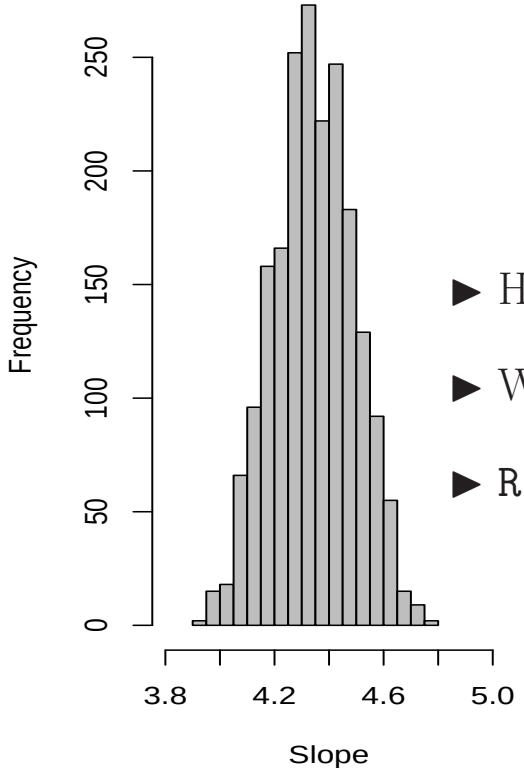
Basic and not-so-basic statistics
Bootstrapping Regression

- 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)
```

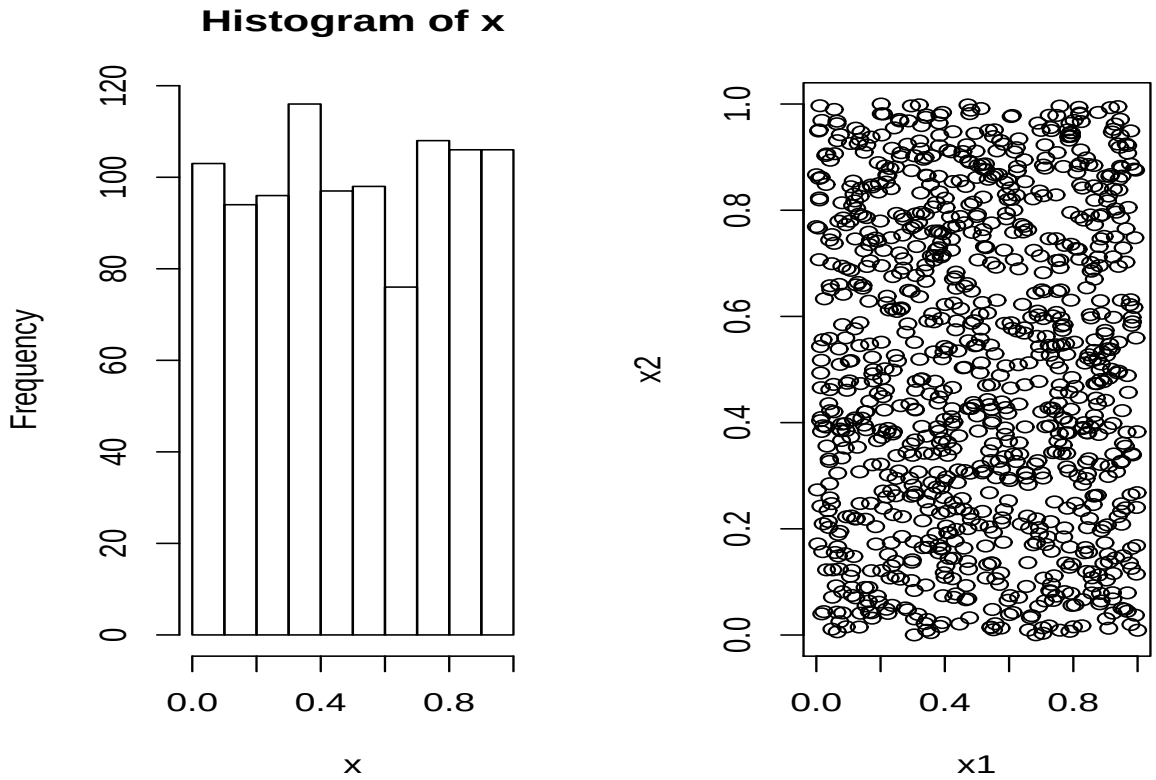
will produce 100 random variables $\mathcal{U}(2, 5)$.

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

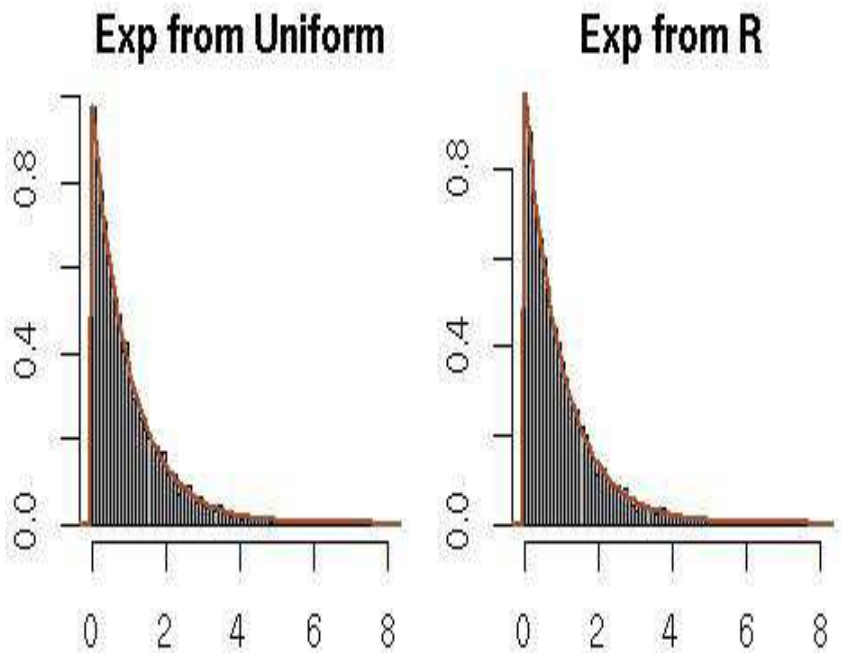
$$F(x) = \int_{-\infty}^{x} f(t)\,\mathrm{d}t,$$

  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

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

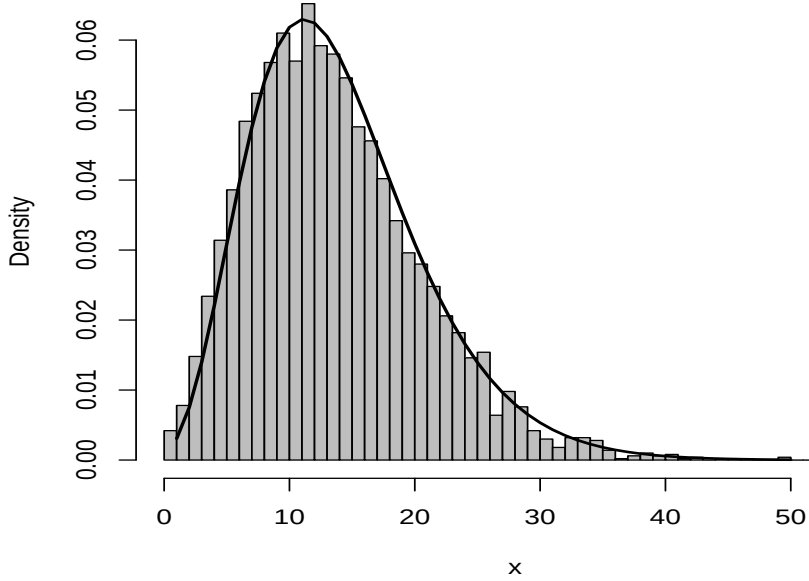
Mixture Representations
Continuous Mixtures

- 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 $X \sim \mathcal{N}eg(n, p)$
  - $X|y \sim \mathcal{P}(y) \quad \text{and} \quad Y \sim \mathcal{G}(n, \beta)$,
  - `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

$$P(Y \leq x| \text{ Accept }) = P(Y \leq x|U \leq f(Y)/\{Mg(Y)\}) = P(X \leq x)$$

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

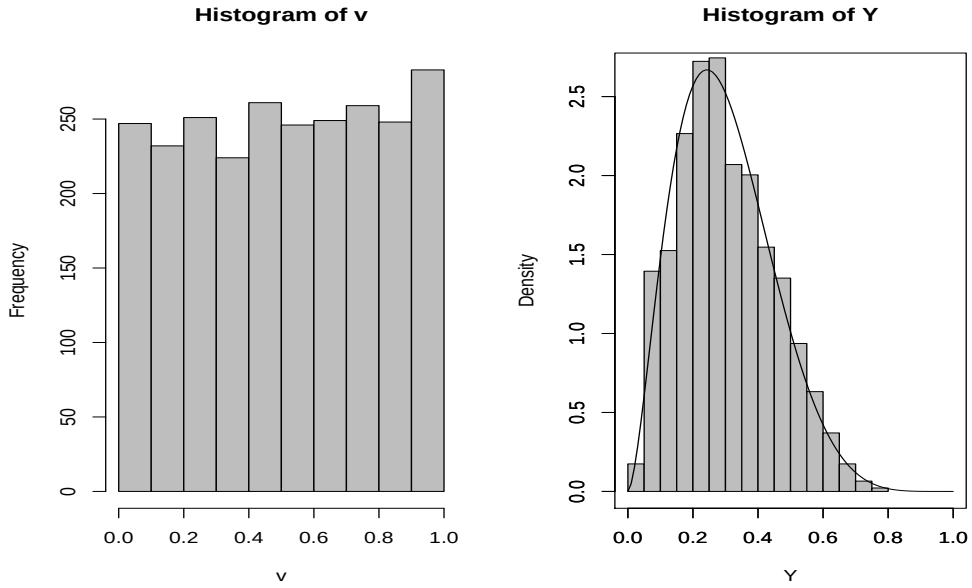
ϟ

- 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

Accept–Reject Algorithm
Betas from Uniforms

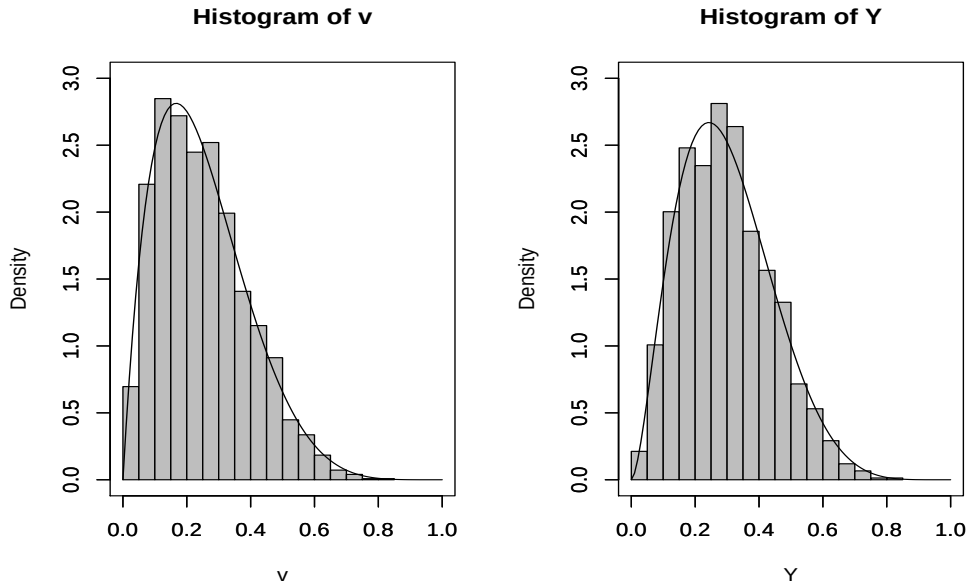
- Generate $X \sim \text{beta}(a, b)$.
- No direct method if a and b are not integers.
- Use a uniform candidate
- For $a = 2.7$ and $b = 6.3$



▶ Acceptance Rate =37%

Accept–Reject Algorithm
Betas from Betas

- Generate $X \sim \text{beta}(a, b)$.
- No direct method if a and b are not integers.
- Use a beta candidate
- For $a = 2.7$ and $b = 6.3$, $Y \sim \text{beta}(2, 6)$



▶ Acceptance Rate =60%

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

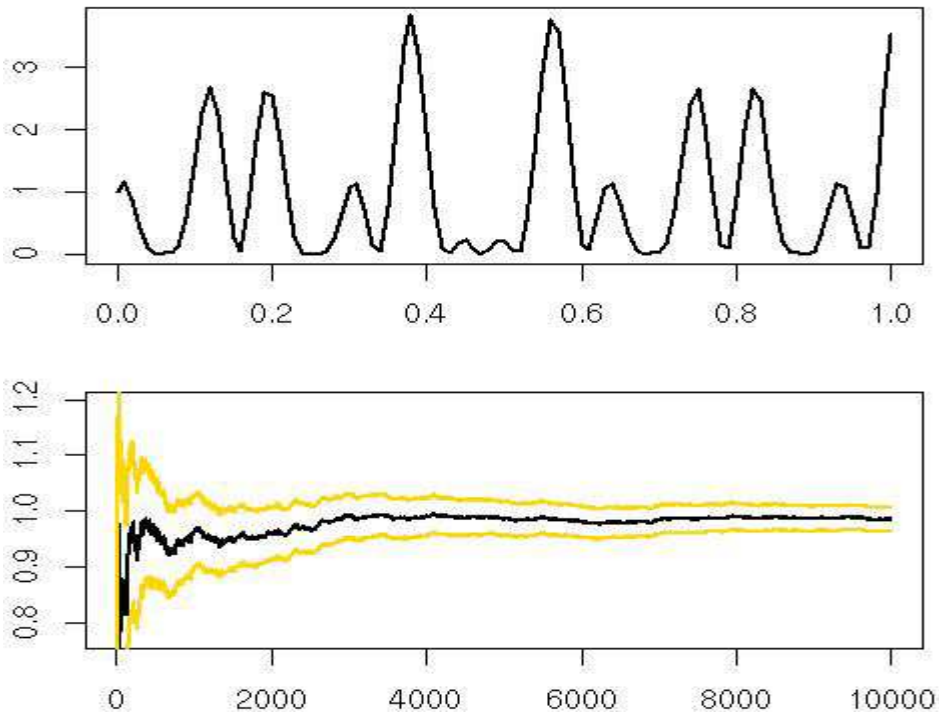
- When $h^2(X)$ has a finite expectation under f,

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

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

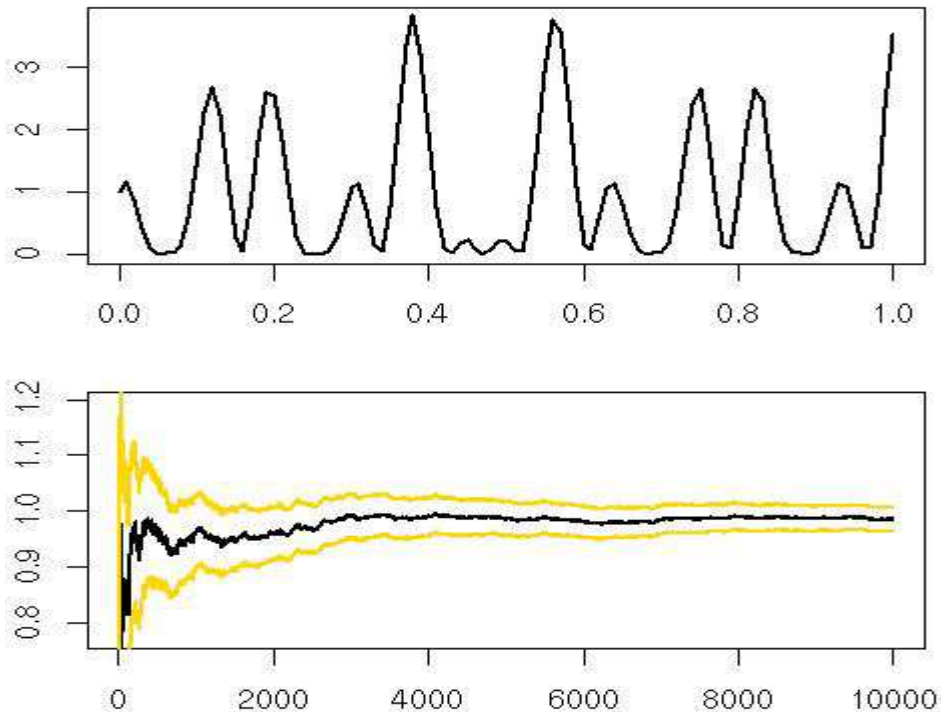
Classical Monte Carlo Integration
A First Example

- Look at the function



- $h(x) = [\cos(50x) + \sin(20x)]^2$
- Monitoring Convergence
- `R` code

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 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

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

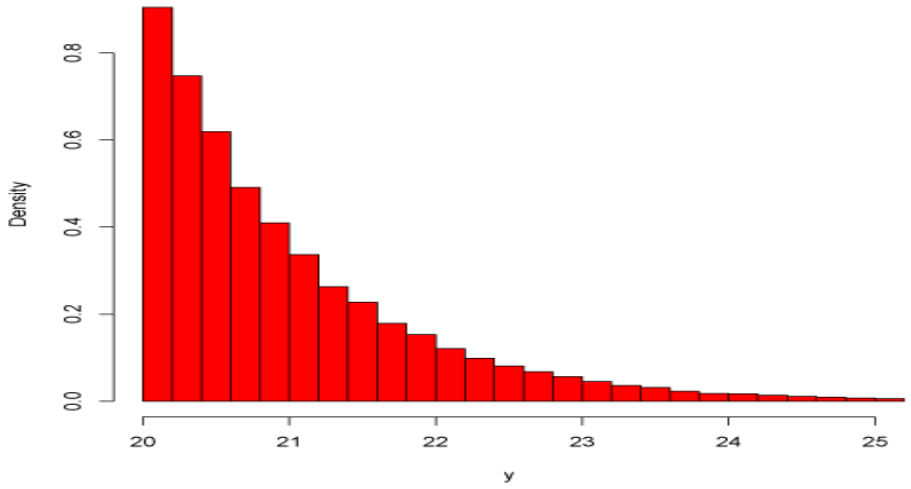
R code
  ```

Importance Sampling
Normal Tail Variables

- The Importance sampler does not give us a sample ⇒ Can use Accept–Reject
- Sample $Z \sim \mathcal{N}(0,1)$, $Z > a \Rightarrow$ Use Exponential Candidate

$$\frac{\frac{1}{\sqrt{2\pi}}\exp(-.5x^2)}{\exp(-(x-a))} = \frac{1}{\sqrt{2\pi}}\exp(-.5x^2+x+a) \leq \frac{1}{\sqrt{2\pi}}\exp(-.5a^{*2}+a^*+a)$$

  - Where $a^* = max\{a, 1\}$



- Normals > 20
- The Twilight Zone
- R code

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

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

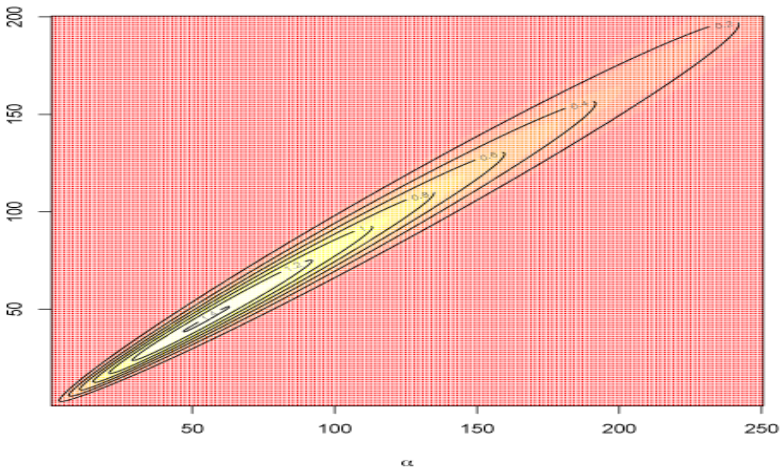
Importance Sampling
Easy Model - Difficult Distribution -2

- 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"))

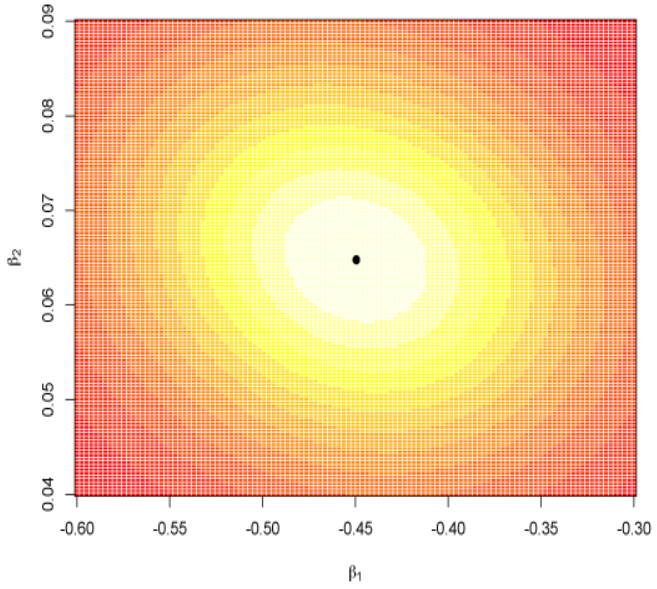
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.44957    0.09497  -4.734 2.20e-06 ***
x            0.06479    0.01615   4.011 6.05e-05 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

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

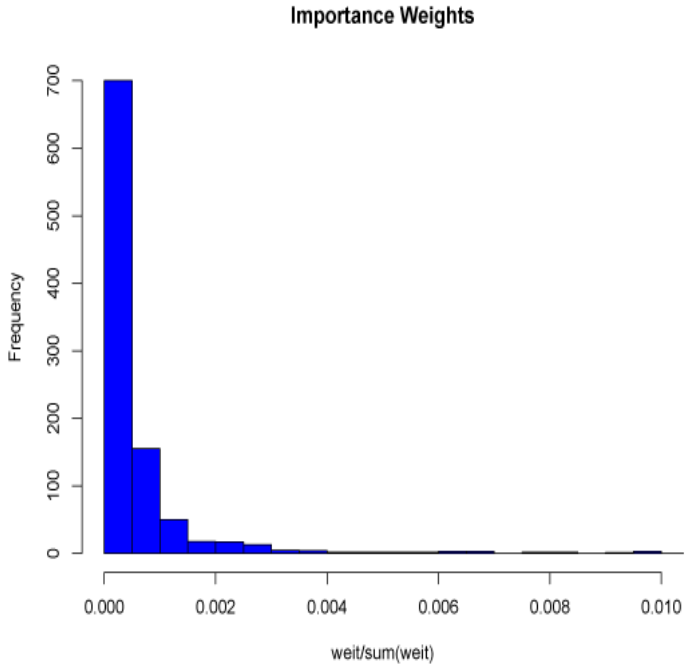
$$\prod_{i=1}^{n} [\Phi(\beta_1 + (x_i - \bar{x})\beta_2]^{y_i} [\Phi(-\beta_1 - (x_i - \bar{x})\beta_2]^{1-y_i} \times e^{-\frac{\beta_1^2+\beta_2^2}{2\times 100}}$$

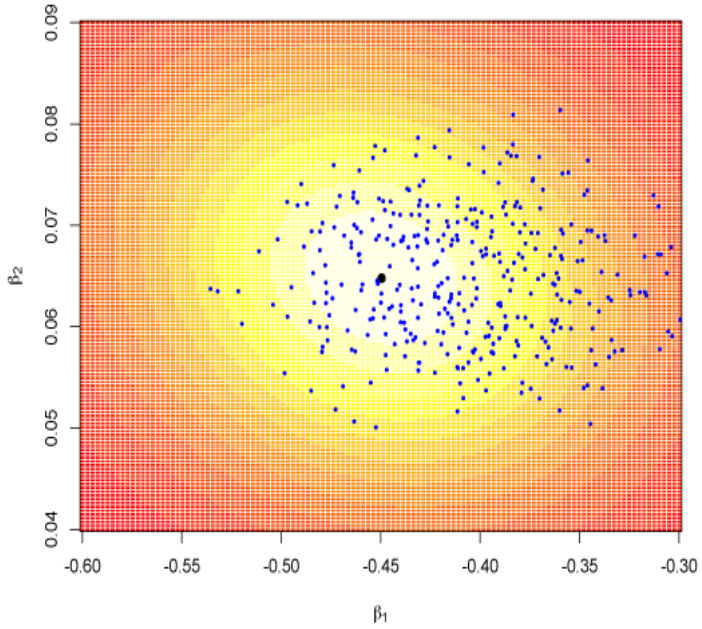


- Level curves of posterior
- MLE in the center
- R code

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.