A Short Introduction to Monte Carlo Methods with R[†]

Sepideh Mosaferi smosafer@umd.edu

November 2, 2016

1. What Is Simulation?

Simulation is a way to model random events, such that simulated outcomes closely match real-world outcomes. By observing simulated outcomes, researchers gain insight on the real world.

2. How to Conduct a Simulation?

The steps required to produce a useful simulation are presented below:

1. Describe the possible outcomes.
2. Link each outcome to one or more random numbers.
3. Choose a source of random numbers.
4. Choose a random number.
5. Based on the random number, note the "simulated" outcome.
6. Repeat steps 4 and 5 multiple times; preferably, until the outcomes show a stable pattern.
7. Analyze the simulated outcomes and report results.

[†]General definitions are extracted from standard books in R and Wikipedia. Main ref.: Introducing Monte Carlo Methods with R by Christian P. Robert & George Casella

3. Simulation in Statistical Word

In statistics, simulation is used to assess the performance of a method, typically when there is a lack of theoretical background. With simulations, the statistician knows and controls the truth.

Simulation is used advantageously in a number of situations. This includes providing the empirical estimation of sampling distributions, studying the misspecification of assumptions in statistical procedures, determining the power in hypothesis tests, etc.

Simulation studies should be designed with lots of rigour. Burton et al. (2006) gave a very nice overview in their paper 'The design of simulation studies in medical statistics'. Simulation studies conducted in a wide variety of situations may be found in the references.

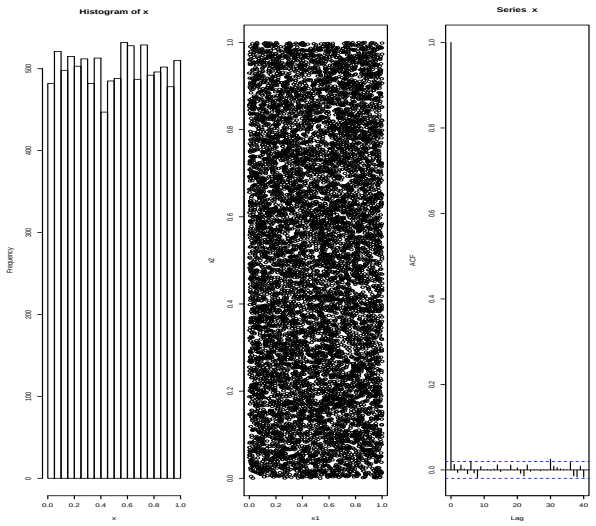
Here, we use R software for simulation study.

4. R Software

R is an implementation of a functional programming language called S. It has been developed and is maintained by a core of statistical programmers, with the support of a large community of users. Unlike S-plus, the other currently available implementation of S, R is free. It is most widely used for statistical computing and graphics, but is a fully functional programming language well suited to scientific programming in general.

4.1 Installing R

You can download R from one of the many mirror sites of the Comprehensive R Archive Network (CRAN) https://cran.r-project.org/. In the first instance it will be sufficient to obtain the base distribution. Advice on downloading and



installing R is available from the FAQs provided on the CRAN site.

Part I. Simple Simulation

5. Generating Random Variables

Practical techniques that can produce random variables from standard or nonstandard distributions.

Example

```
> Nsim <- 10^4
> x <- runif(Nsim) #Uniform(0,1)
> x1 <- x[-Nsim]
> x2 <- x[-1]
> par(mfrow=c(1,3))
> hist(x)
> plot(x1,x2)
> acf(x)
```

We need to 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. Setting the seed determines all the subsequent values.

Example

```
> set.seed(1)
> runif(5)
[1] 0.2655 0.3721 0.5729 0.9082 0.2017
> set.seed(1)
> runif(5)
[1] 0.2655 0.3721 0.5729 0.9082 0.2017
> set.seed(2)
> runif(5)
[1] 0.1849 0.7024 0.5733 0.1681 0.9438
```

Remark: Most of the time the purpose of simulation is evaluating a new methodology, estimator, or a hypothesis when collecting the data set is expensive. In the simulation study we have the capability of controling the conditions via a case study to artificially produce different situations regarding our study. Another great advantage of simulation is the availabilty of truth which permits us to evaluate the results coming from the simulation.

6. Simulating Population

Generally speaking, it is beneficial to create an artificial population based on a special technique to have the truth for the evaluation parts and generate samples from the created population to study our methodology or hypothesis from each sample and provide empirical results (average over the replications) that satisfies the ergodicity, which will be defined later.

Here we focus on three methods of generating population as follows:

- Model based simulation

- Design based simulation
- Model-Design based simulation

6.1. Model Based Simulation

In this situation, we generate samples from the super population following a model (or models) based on the independency assumption. For example assume that we have the fay-Herriot model

$$y_i = X_i^T\beta + v_i + e_i \quad (i = 1, ..., m)$$

, where i stands for the i-th area, v_i and e_i are mutually independent random variables with mean 0 for all i. It is commonly assumed that the model error $v_i \sim \mathcal{N}(0, \sigma_v^2)$ and the design error $e_i \sim \mathcal{N}(0, \psi_i)$. Commonly, β, σ_v^2 are not known.

Then we are interested to provide an optimal estimator for the quantity of interest $Y_i = X_i^T\beta + v_i$, assumed to be the population mean of area i. We could propose different estimators but how we could evaluate them beyond of analytical expressions via an experimental study?

To do so, we generate x_i from a $\mathcal{N}(5, 9)$ distribution and ψ_i from a gamma distribution with shape parameter 4.5 and scale parameter 2. Then for each iteration, we generate $Y_i = 1 + 3x_i + v_i$, $y_i = Y_i + e_i$ where v_i and e_i are independent normal variates with mean 0 and respective variances σ_v^2 and ψ_i.

Assume that we have 50 areas and the proposed estimator is $\hat{Y}_i = x_i\hat{\beta}$ where only β is unknown and model error is ignored. The R code is as follows and we want to evaluate the estimator based on the empirical MSE. If I propose another estimator for Y_i (let's call it $\tilde{Y}$) in the similar ground, I can follow the same method and compare $\hat{Y}_i$ and $\tilde{Y}$ to see which one outperforms.

```
# R code
> empiricalMSE=rep(0,50)
> for(i in 1:50)
+ {
+  x=rnorm(1000,5,3) #var=9
+  psi=rgamma(1000,4.5,scale=2)
+  v=rnorm(1000,0,2) #var=4
+  e=rnorm(1000,0,sqrt(psi))
+  Y=1+3*x+v
+  fit=lm(Y~x)
+  X=cbind(rep(1,1000),x)
+  beta_hat=c(coef(fit)[1],coef(fit)[2])
+  Y_hat=X%*%beta_hat
+  empiricalMSE[i]=sum((Y_hat-Y)^2)/1000
+}
> empiricalMSE
> mean(empiricalMSE)
```

Results: $EMSE = (4.143085, 3.749997, ..., 3.949147, 4.014750)^t$.

6.2. Design Based Simulation

This simulation is based on a real dataset. In a way that we draw samples repeatedly for e.g. based on stratified random sample without replacement, with proportional allocation to the size of strata from a real finite population. The finite population could be assumed to be constructed from the 2012 Census of Governments: Employment, where strata are U.S. states, and the parameter of interest is the average number of full-time employees for the U.S. states. If we want to estimate it from the direct estimator $\hat{\bar{Y}}_i$, we have the following R code:

```
> one.rep <- function(){
+  ## Sample selection
```

```
+  SMP.IDs <- strata(data=Employment,
+  stratanames="STATE", size=nh,method="srswor")
+  #Output of selected sample:
+  SAMPLE <- getdata(Employment,SMP.IDs)
+  SAMPLEMEAN <- tapply(SAMPLE$TOTEMP12rep,
+  INDEX=SAMPLE$STATE,FUN=mean)
+  SAMPLESD <- tapply(SAMPLE$TOTEMP12rep,
+  INDEX=SAMPLE$STATE,FUN=sd)
+  SAMPLEVAR <- (1-(nh/Nh))*(SAMPLESD^2/nh)
+}
> ## Replicate 1000 sample:
> R <- 1000
> many.reps <- replicate(n=R,one.rep())
```

6.3. Model-Design Based Simulation

This situation can be used when we do not access a real finite population, but we would take care of design because of our particular study. For example assume that we are interested to propose a new estimator for the variance of 2 primary sampling units (PSU's) per stratum design instead of the regular one. Therefore based on the study, we need to create strata and generate samples from them. But when we do not have a real finite population with strata such as the one mentioned in the former subsection, what should we do?

Illustration We can generate a finite population of size $N_T = 20,000$ units by drawing a random sample of size $20,000$ from a bivariate super-population characterized by two-dimensional random vector (x, y), where the variable x has a gamma distribution with shape 2 and scale 5, $f(x) = .04xexp(-x/5)$, and the variable y (conditional on x) has a gamma distribution with density function $g(y|x) = (1/b^{\Gamma}(c))y^{c-1}exp(-y/b)$, where $c = .04x^{-3/2}(8 + 5x)^2$ and $b = 1.25x^{3/2}(8 + 5x)^{-1}$. This realized population is a realistic approximate representation of a finite population used by Hansen et al. (1983). Strata are formed based on the quantiles of x in an increasing order.

Then after creation of strata, we could draw simple random samples (2 PSUs) if PSU is assumed to be the single element. Then after sample draws from each stratum, we could estimate the quantity of interest and its corresponding measure of uncertainty (the variance) based on our proposed method and compare it with the regular design-based variance estimate to study which one outperforms based on our criteria. The part of R code is as follows:

```
## Model-Design based simulation
> require(PracTools); require(sampling)
> require(pscl); require(MCMCpack)
# Population:
> x <- rgamma(2e4,shape=2,scale=5)
> b <- 1.25*(x^(3/2))*((8+5*x)^(-1))
> c <- 0.04*(x^(-3/2))*(8+5*x)^2
#Our realistic population:
> y <- rgamma(2e4,shape=c,scale=b)
> range(y);summary(y); summary(x)
> cor(x,y)
> FRAME <- data.frame(cbind(y,x))
> head(FRAME)
> H <- 10 #total number of strata
> N <- nrow(FRAME) #population size
# Create strata
> STRATUM <- cut(x,breaks=c(-Inf,quantile(x,
```

```
+ seq(0.1,0.9,by=0.1)),Inf))
> UPDATEFRAME <- data.frame(FRAME,STRATUM)
> UPDATEFRAME <- UPDATEFRAME[order(STRATUM),]
> one.rep <- function(){
+  #Sample selection (selecting 1 PSU)
+  SMP.IDs <- strata(data=UPDATEFRAME,
+  stratanames="STRATUM",size=ng,method="srswor")
+  # Output of selected sample:
+  SAMPLE <- getdata(UPDATEFRAME,SMP.IDs)
+ }
> ## Replicate 1000 sample:
> R <- 1000
> many.reps <- replicate(n=R,one.rep())
```

Part II. More Advanced Simulation

7. Probability Integral Transform

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

$$F(x) = \int_{-\infty}^{x} f(t)dt$$

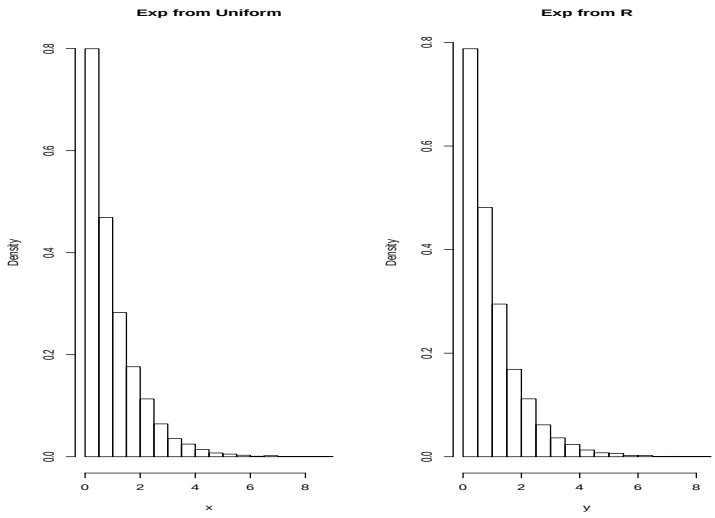
and we set $U = F(x)$ and solve for X.

Example

If X $Exp(1)$, then $F(x) = 1 - e^{-x}$.

Solving for x in $u = 1 - e^{-x}$ gives $x = -log(1-u)$.

```
## R code
> u <- runif(Nsim,0,1)
> x <- -log(1-u)
> y <- rexp(Nsim)
> par(mfrow=c(1,2))
```




```
> hist(x,freq=F,main="Exp from Uniform")
> hist(y,freq=F,main="Exp from R")
```

But there is a difference in the system time.

```
> system.time(test1()) #Exp from Uniform
   user  system elapsed
  0.012   0.000   0.012
> system.time(test2()) #Exp from R
   user  system elapsed
  0.000   0.000   0.002
```

8. General Transformation Method

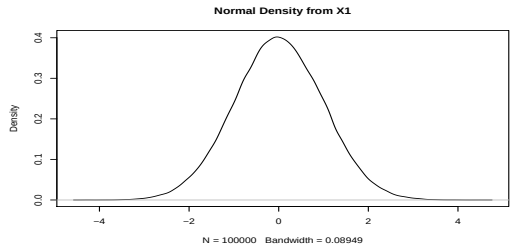
Sometimes generating a random variable x with the density f is easier to be simulated from another distribution by finding out the relationship between variables.

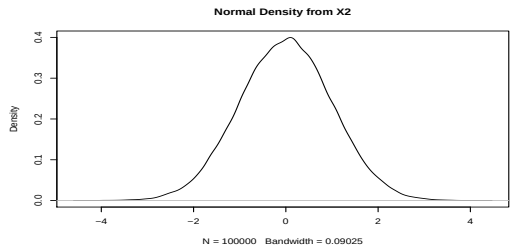
Example Box-Muller Algorithm: Two Normals from two Uniforms

If U_1 and U_2 are *i.i.d* $\mathcal{U}[0,1]$, then the variables X_1 and X_2 are

$$X_1 = \sqrt{(-2log(U_1))}cos(2\pi U_2) \sim \mathcal{N}(0,1)$$

$$X_2 = \sqrt{(-2log(U_1))}sin(2\pi U_2) \sim \mathcal{N}(0,1)$$





The Box-Muller algorithm is exact, not a crude CLT-based approximation.

```
## R code
> size = 100000
> U1 <- runif(size)
> U2 <- runif(size)
> X1 <- rep(0,size)
> X2 <- rep(0,size)
> for (i in 1:size){
+  X1[i] <- sqrt(-2*log(U1[i]))*cos(2*pi*U2[i])
+  X2[i] <- sqrt(-2*log(U1[i]))*sin(2*pi*U2[i])
+}
```

9. Generating 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

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

- And then generate $U \sim U[0,1]$ and take

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

The above method might be tedious for distributions such as poisson with very large λ where $X \sim Poisson(\lambda)$.

```
> 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)
+}
```

If R has the built-in distribution use it, but R does not handle every distribution that we need.

Mixture Representation

- Continuous (To generate x)

$$f(x) = \int_y g(x|y)p(y)dy \Rightarrow y \sim p(y) \text{and} X \sim f(x|y), \text{then} X \sim f(x)$$

- Discrete (To generate x)

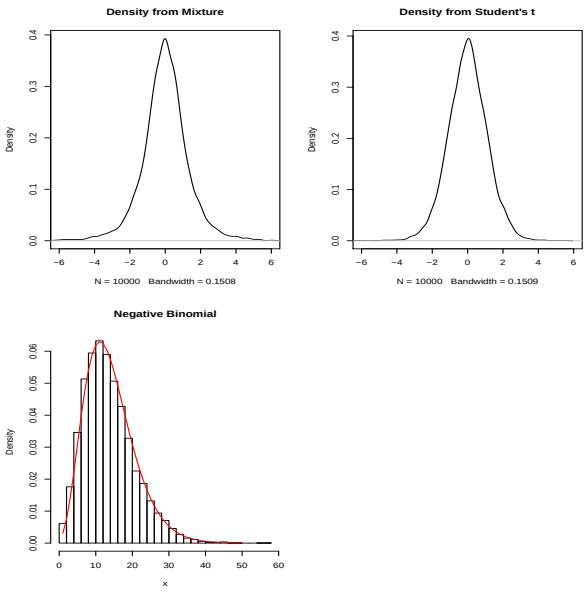
$$f(x) = \sum_{i \in y} p_i f_i(x) \Rightarrow i \sim p_i \text{and} X \sim f_i(x), \text{then} X \sim f(x)$$

- Discrete Normal Mixture

$$p_1 \times \mathcal{N}(\mu_1, \sigma_1) + p_2 \times \mathcal{N}(\mu_2, \sigma_2) + p_3 \times \mathcal{N}(\mu_3, \sigma_3)$$

Example Generating Student's t $X|y \sim \mathcal{N}(0, \nu/y), Y \sim \chi^2_\nu$; Negative Binomial $X|y \sim \mathcal{P}(y), Y \sim \mathcal{G}(n, \beta)$ where $\beta = (1-p)/p$.

```
## Student's t density (df=10): Continuous
> df <- 10
> y <- rchisq(Nsim,df,ncp=0)
> x <- rnorm(Nsim,0,df/y)
> par(mfrow=c(2,2))
> plot(density(x),main="Density from Mixture")
> plot(density(rt(Nsim,df,ncp=0)),
     main="Density from Student's t")
## Negative Binomial: Discrete
> n <- 6
> p <- 0.3
> y <- rgamma(Nsim,n,rate=p/(1-p))
> x <- rpois(Nsim,y)
> hist(x,freq=FALSE,breaks = 40,
      main="Negative Binomial")
> lines(1:50,dnbinom(1:50,n,p),col="red")
```



10. Accept Reject Algorithm

There are many distributions where transform methods fail. For these cases, we must turn to indirect methods. We generate a candidate random variable and 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.

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.

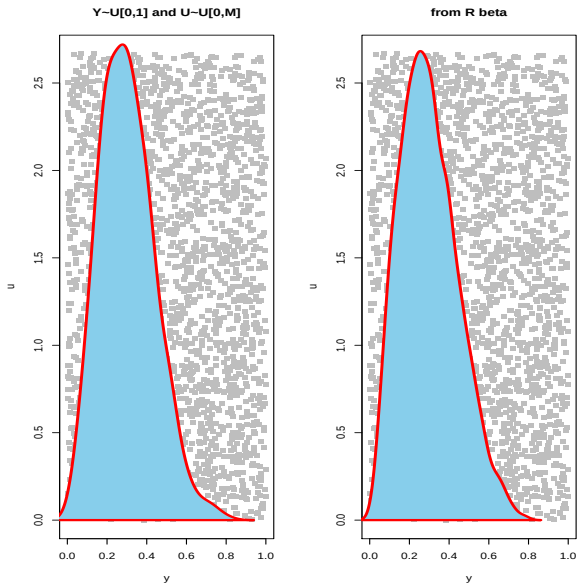
$X \sim f$ can be simulated as follows:

- 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(Accept)=$\frac{1}{M}$, Expected Waiting Time=M

So M should be as small as possible for a given computational effort.

Example Simulating $\mathcal{B}e(\alpha, \beta)$ when $\alpha = 2.7$ and $\beta = 6.3$.

```
## Accept-Reject Algorithm (Betas from Uniforms)
> M <- 2.669744 #Maximum of U[0,M]
> Nsim <- 2500
> a <- 2.7;b <- 6.3
> par(mfrow=c(1,2))
> y <- runif(Nsim) #generated from g (U[0,1])
> u <- runif(Nsim,max=M) #generated from U[0,M]
> x <- y[u<dbeta(y,a,b)] #accepted subsample
> plot(y,u,pch=15,col="grey",main="Y~U[0,1]
      and U~U[0,M]")
> polygon(density(x,bw=0.04), col="skyblue",
       border="red",lwd=4)
#the acceptance area as shown in the skyblue
#   is 1/M=1/2.669=%36
> rb=rbeta(Nsim,2.7,6.3)
> d<-density(rb)
> plot(y,u,pch=15,col="grey",
    main="from R beta")
> polygon(d,col="skyblue",
                    border="red",lwd=4)
```



11. Monte Carlo Integration

The validity of Monte Carlo approximations relies on the Law of Large Numbers. The versatility of the representation of an integral as an expectation as a Lebesgue measure We will be concerned with evaluating integrals of the form

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

- 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

The generic problem: Evaluate

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

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

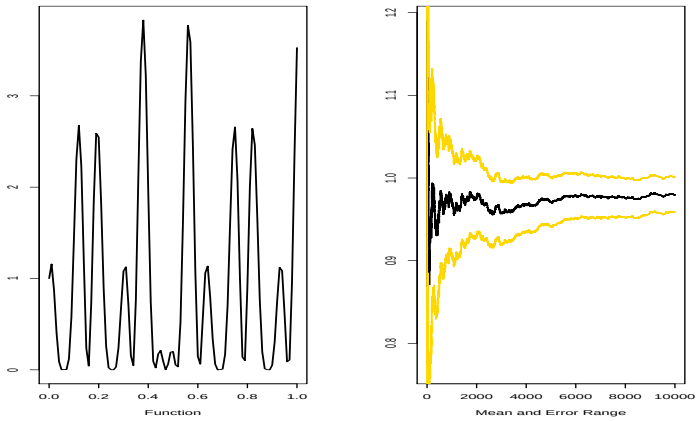
The Monte Carlo Method

- Generate a sample $(X_1, ..., X_n)$ from the density f
- Approximate the integral with

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

Example Integrating the function of $h(x) = [cos(50x) + sin(20x)]^2$ which is integrable analytically. We are looking to find its integral over $[0, 1]$ from a uniform expectation by generating $U_1, U_2, ..., U_n \quad iid \quad \mathcal{U}(0, 1)$ as follows.

```
> par(mfrow=c(1,2))
> h <- function(x){(cos(50*x)+sin(20*x))^2}
> curve(h,xlab="Function",ylab="",lwd=2)
> integrate(h,0,1)
> x <- h(runif(Nsim))
> estint <- cumsum(x)/(1:Nsim)
> esterr <- sqrt(cumsum((x-estint)^2))/(1:Nsim)
> plot(estint,xlab="Mean and Error Range",type="l",
+   lwd=2, ylim=mean(x)+20*c(-esterr[Nsim],
+   esterr[Nsim]),ylab="")
> lines(estint+2*esterr,col="gold",lwd=2)
> lines(estint-2*esterr,col="gold",lwd=2)
```



11.1. Importance Sampling

Importance sampling is based on the formula just we mentioned where f is the target density and g is the candidate density.

$$\mathbb{E}_f[h(X)] = \int_{\mathcal{X}} h(x)\frac{f(x)}{g(x)}g(x)dx = \mathbb{E}_g[\frac{h(X)f(X)}{g(X)}]$$

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

- $Var(h(X)f(X)/g(X)) < \infty$
- $supp(g) \supset supp(h \times f)$

Example: Beta posterior importance approximation

Assume $x \sim \mathcal{B}(\alpha, \beta)$ 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

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

where λ, x_0, y_0 are hyperparameters, and the posterior is then equal to

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

This is intractable because of gamma functions.

The quantity of interest is marginal likelihood as

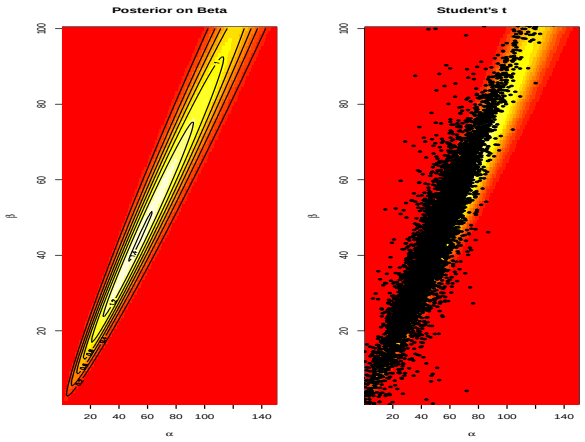
$$\begin{aligned} m(x) &= \int_{\mathbb{R}_+^2} f(x|\alpha,\beta)\pi(\alpha,\beta)d\alpha d\beta \\ &= \frac{\int_{\mathbb{R}_+^2}\{\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\}^{\lambda+1}[xx_0]^{\alpha}[(1-x)y_0]^{\beta}d\beta}{\int_{\mathbb{R}_+^2}\{\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\}x_0^{\alpha}y_0^{\beta}d\alpha d\beta} \end{aligned}$$

We can approximate the both integrals by t-distribution (see the Figure). Therefore we have

$$\hat{m}(x) = \sum_{i=1}^{n}\{\frac{\Gamma(\alpha_i+\beta_i)}{\Gamma(\alpha_i)\Gamma(\beta_i)}\}^{\lambda+1}[xx_0]^{\alpha_i}[(1-x)y_0]^{\beta_i}/g(\alpha_i,\beta_i)/$$

$$\sum_{i=1}^{n}\{\frac{\Gamma(\alpha_i+\beta_i)}{\Gamma(\alpha_i)\Gamma(\beta_i)}\}^{\lambda} x_0^{\alpha_i} y_0^{\beta_i}/g(\alpha_i,\beta_i),$$

where g follows student's t. The given figure is related to the situation where $\lambda = 1, x_0 = y_0 = 0.5$, and $x = 0.6$. The t-distribution has 3 $d.f.$ with parameters $\mu \equiv (50,45)^t$ and $\Sigma \equiv ((220,190)^t,(190,180)^t)$.




```
#### R code

## Marginal distribution related to the

## Beta distribution

#we consider the exp(log(f))=f.

#By looking at the image of f,

#we can figure out a better function for

#simulation rather than the true

#posterior which is difficult for simulation.

#lambda=1;x0=y0=0.5;x=0.6

> f=function(a,b){

+  exp(2*(lgamma(a+b)-lgamma(a)-

+  lgamma(b))+a*log(.3)+b*log(.2))}

> aa=1:150 #alpha grid for image

> bb=1:100 #beta grid for image

> post=outer(aa,bb,f)

> image(aa,bb,post,xlab=expression(alpha),

+ ylab=expression(beta),main="Posterior on Beta")

#data image "It basically makes a frame for outr plot"

> contour(aa,bb,post,add=T)  #create a counter plot

#From the plot,we can consider t-student distribution.

> x=matrix(rt(2*10^4,3),ncol=2)#T sample (df=3)

> E=matrix(c(220,190,190,180),ncol=2)

> image(aa,bb,post,xlab=expression(alpha),
```

```
+  ylab=expression(beta),main="Student's t")
##interested simulated variable:
> y=t(t(chol(E))%*%t(x)+c(50,45))
#y=(x-mu)/sigma.  x=sigam*y+mu f(y)=fx(y*sigma+mu)
> points(y,cex=.6,pch=19)
#Finding out the marginal likelihood of
# (m(x))while g(a,b)=t-student
> x=matrix(rt(2*10^4,3),ncol=2)
> E=matrix(c(220,190,190,180),ncol=2)
> y=t(t(chol(E))%*%t(x)+c(50,45))
#Hint:As a and b must essentially be positive
#but t distribution brings negative
#values for them, we have to eliminate them
#by using y=y[ine>0]
> ine=apply(y,1,min);ine
> y=y[ine>0,] ;y
> normx=sqrt(x[,1]^2+x[,2]^2)
> f=function(y){
+  exp(2*(lgamma(y[,1]+y[,2])-lgamma(y[,1])-
+   lgamma(y[,2]))+y[,1]*log(.3)+y[,2]*log(.2))}
> h=function(y){
+  exp(lgamma(y[,1]+y[,2])-lgamma(y[,1])-
+  lgamma(y[,2])+y[,1]*log(.5)+y[,2]*(log(.5)))}
> den=dt(normx,3) #density of t-distribution as g
#answer of integral:
> estimator=mean(f(y)/den)/mean(h(y)/den);estimator
#finding the posterior expectations of
#the parameters
> f=function(y){
+  exp(2*(lgamma(y[,1]+y[,2])-lgamma(y[,1])-
+  lgamma(y[,2]))+y[,1]*log(.3)+y[,2]*log(.2))}
> mean(y[,1]*f(y)/den)/mean(h(y)/den) #mean of a
> mean(y[,2]*f(y)/den)/mean(h(y)/den) #mean of b
```

12. Monte Carlo Optimization

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.

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.

- 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)

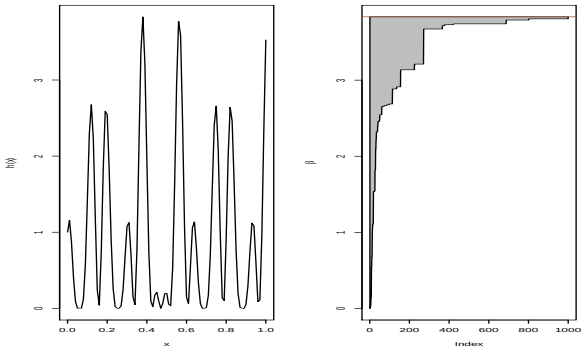
Properties of h play a lesser role in simulation-based approaches.

Therefore, if h is complex or Θ is irregular, chose the stochastic approach.

Example 1.

Recall the simple but highly variable function $h(x) = [cos(50x) + sin(20x)]^2$ defined on $[0, 1]$. A call to optimise provides an identification of the maximum at $x^\star = 0.379$ with a value of $h(x^\star) = 3.8325$. For assessing the variability of a uniform sampler we have the following R codes.

```
#Using optimize function:
> h=function(x){
+  (cos(50*x)+sin(20*x))^2}
> optimize(h,lower=0,upper=1,maximum=TRUE)
#According to the above code, xmax=0.379 &
#h(xmax)=3.8325.
#Using Uniform sampler (proposed by myself):
> set.seed(1)
> Nsim=10^4
> x=runif(Nsim)
> h=function(x){
+  (cos(50*x)+sin(20*x))^2}
> curve(h)
> xnew=sort(x)
> optimize(h,xnew,maximum=TRUE)
#Using Uniform sampler (Assess Variability of Uniform)
> rangom=h(matrix(runif(10^6),ncol=10^3))
> monitor=t(apply(rangom,1,cummax))
> plot(monitor[1,],type="l",col="white")
> polygon(c(1:10^3,10^3:1),c(apply(monitor,2,max),
> rev(apply(monitor,2,min))),col="grey")
> abline(h=optimise(h,int=c(0,1),maximum=TRUE)$ob,
+ col="sienna",lwd=2)
```

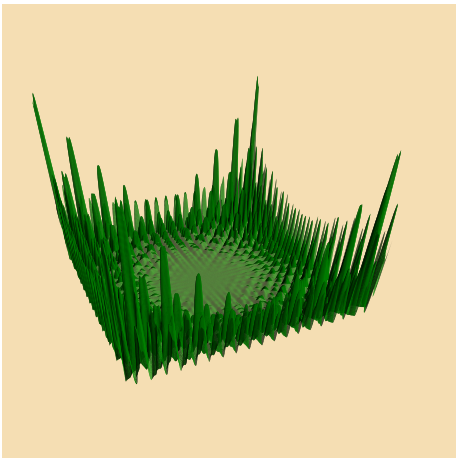


Example 2: Considering minimizing the (artificially constructed) function in $\mathbb{R}^2$

$$h(x,y) = [xsin(20y) + ysin(20x)]^2 cosh(sin(10x)x)$$
$$+ [xcos(10y) - ysin(10x)]^2 cosh(cos(20y)y),$$

whose global minimum is 0 at $(x,y) = (0,0)$. Since this function has many local minima, as shown in the Figure, we have:

```
## R code
## Artificical Mini
> h=function(x,y){(x*sin(20*y)+y*sin(20*x))^2
+ *cosh(sin(10*x)*x)+(x*cos(10*y)-y*sin(10*x))^2
+ *cosh(cos(20*y)*y)}
> x=y=seq(-3,3,le=435) #defines a grid for persp
> z=outer(x,y,h)
> par(bg="wheat",mar=c(1,1,1,1)) #bg stands background
> persp(x,y,z,theta=155,phi=30,col="green4",
+    ltheta=-120,shade=.75,border=NA,box=FALSE)
> min(z);max(z)
```



Practice: Try to find the minimum by Accept-Reject Algorithm based on the uniform distribution.

Ergodicity

In recurrent chains, the stationary distribution is also a *limiting distribution*

If f is the limiting distribution

$X(t) \to X \sim f$ for any initial value $X(0)$

This property is based on the Law of Large Numbers also called Ergodic Theorem.

For integrable functions h, the standard average

$$\frac{1}{T}\sum_{t=1}^{T} h(X^{(t)}) \to \mathbb{E}_f[h(X)]$$

13. Metropolis-Hastings Algorithm

The Metropolis-Hastings algorithm is one of the most general MCMC algorithms and one of the simplest. We now make a fundamental shift in the choice of our simulation strategy. Up to now we have typically generated iid variables. The Metropolis-Hastings algorithm generates correlated variables from a Markov chain

Algorithm

Metropolis-Hastings Given $x^{(t)}$,

1. Generate $Y_t \sim q(y|x^{(t)})$.

2. Take

$$X^{(t+1)} = \begin{cases} Y_t & \text{with probability } \rho(x^{(t)}, Y_t), \\ x^{(t)} & \text{with probability } 1 - \rho(x^{(t)}, Y_t) \end{cases}$$

where

$$\rho(x,y) = min\left\{\frac{f(y)}{f(x)}\frac{q(x|y)}{q(y|x)}, 1\right\}$$

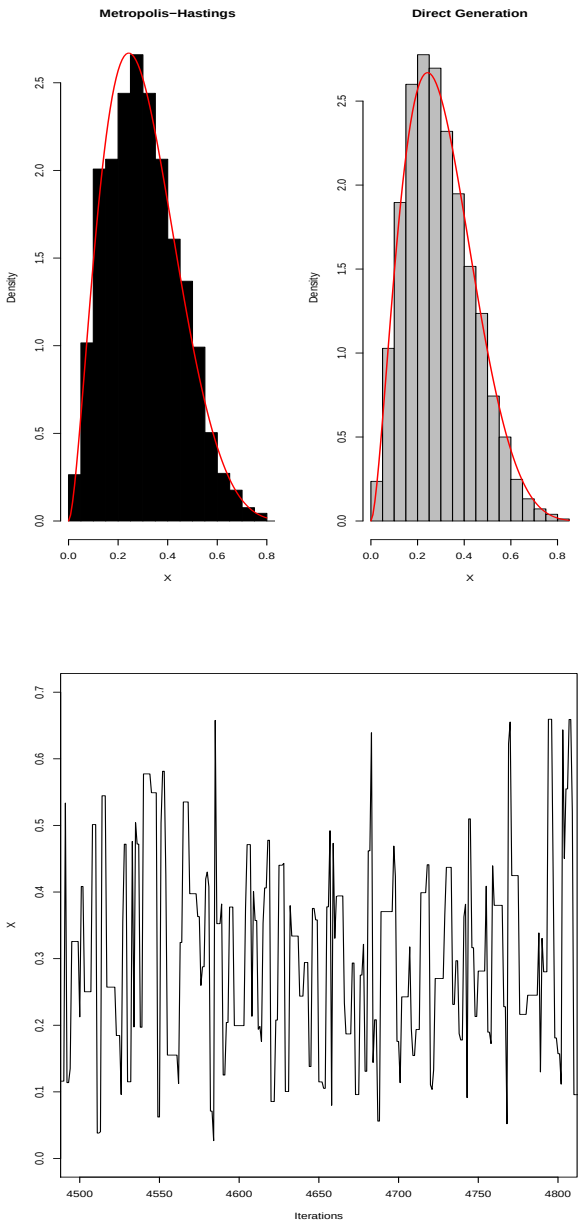
- q is called the instrumental or proposal or candidate distribution.
- $\rho(x,y)$ is the Metropolis-Hastings acceptance probability.
- Looks like *Simulated Annealing*- but constant temprature.

Metropolis-Hastings explores rather than maximizes.

Example 1: Target density f is the $\mathcal{B}e(2.7, 6.3)$ and candidate q is uniform. Look at the graph and the repeats. Repeats must be kept!

```
## R code
## Metropolis-Hastings Beta distribution
> a=2.7;b=6.3;c=2.669 #initial values
> Nsim=5000
> X=rep(runif(1),Nsim)
> for (i in 2:Nsim){
+  Y=runif(1)
+  rho=dbeta(Y,a,b)/dbeta(X[i-1],a,b)
+  X[i]=X[i-1]+(Y-X[i-1])*(runif(1)<rho)
+}
> print(X)
> par(mfrow=c(1,3))
> hist(X,col="black",freq=F,
+ main="Metropolis-Hastings", xlim=c(0,0.8))
> curve(dbeta(x,a,b),col="red",lwd=2,add=TRUE)
> hist(rbeta(5000,a,b),nclass=20,col="gray",
+  main="Direct Generation",freq=F,xlab="X")
> curve(dbeta(x,a,b),col="red",lwd=2,add=TRUE)
#Checking by Kolmogrove-Smirnov test:
> ks.test(jitter(X),rbeta(5000,a,b))
#Checking by moments:
 #For Beta distribution:
> a/(a+b);a*b/(((a+b)^2)*(a+b+1))
#For Metropolis-Hasting:
> xbar=mean(X);xvar=sd(X)^2;xbar;xvar
#Checking fordifferent iteration:
> Nsim=4500
> X1=rep(runif(1),Nsim)
> for (i in 2:Nsim){
+  Y=runif(1)
+  rho=dbeta(Y,a,b)/dbeta(X1[i-1],a,b)
+  X1[i]=X1[i-1]+(Y-X1[i-1])*(runif(1)<rho)
+ }
> print(X1)
> Nsim=4550
> X2=rep(runif(1),Nsim)
> for (i in 2:Nsim){
+  Y=runif(1)
+  rho=dbeta(Y,a,b)/dbeta(X2[i-1],a,b)
+  X2[i]=X2[i-1]+(Y-X2[i-1])*(runif(1)<rho)
+ }
> print(X2)
> Nsim=4600
> X3=rep(runif(1),Nsim)
> for (i in 2:Nsim){
+  Y=runif(1)
+  rho=dbeta(Y,a,b)/dbeta(X3[i-1],a,b)
+  X3[i]=X3[i-1]+(Y-X3[i-1])*(runif(1)<rho)
+ }
> print(X3)
> Nsim=4650
> X4=rep(runif(1),Nsim)
> for (i in 2:Nsim){
+  Y=runif(1)
+  rho=dbeta(Y,a,b)/dbeta(X4[i-1],a,b)
+  X4[i]=X4[i-1]+(Y-X4[i-1])*(runif(1)<rho)
```

```
+ }
> print(X4)
> Nsim=4700
> X5=rep(runif(1),Nsim)
> for (i in 2:Nsim){
+  Y=runif(1)
+  rho=dbeta(Y,a,b)/dbeta(X5[i-1],a,b)
+  X5[i]=X5[i-1]+(Y-X5[i-1])*(runif(1)<rho)
+ }
> print(X5)
> Nsim=4750
> X6=rep(runif(1),Nsim)
> for (i in 2:Nsim){
+  Y=runif(1)
+  rho=dbeta(Y,a,b)/dbeta(X6[i-1],a,b)
+  X6[i]=X6[i-1]+(Y-X6[i-1])*(runif(1)<rho)
+ }
> print(X6)
> Nsim=4800
> X7=rep(runif(1),Nsim)
> for (i in 2:Nsim){
+  Y=runif(1)
+  rho=dbeta(Y,a,b)/dbeta(X7[i-1],a,b)
+  X7[i]=X7[i-1]+(Y-X7[i-1])*(runif(1)<rho)
+ }
> print(X7)
> X=c(X1,X2,X3,X4,X5,X6,X7)
#Iteration=c(4500,4550,4600,4650,4700,4750,4800)
> plot(X,xlim=c(4500,4800),type="l",xlab="Iterations",
+  ylim=c(0,0.7))
```

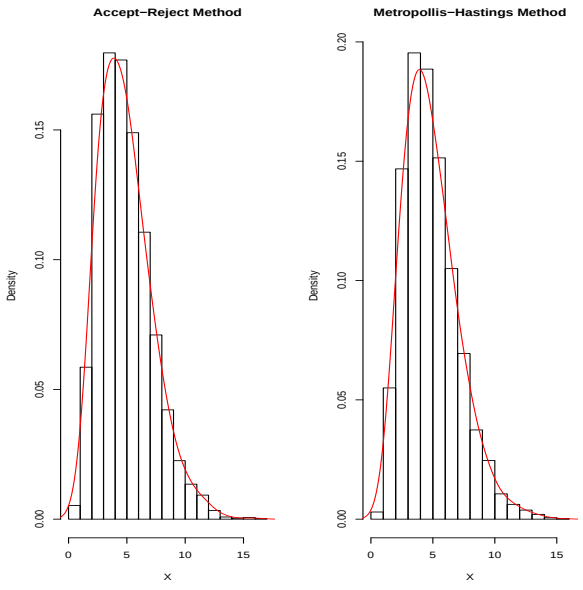


Example 2: Comparison of Accept-Reject Algorithm and Metropolis-Hasting Algorithm

We are interested to generate the $\mathcal{G}(4.85, 1)$ while the candidate distribution is $\mathcal{G}(4, 4/4.85)$. The R code is

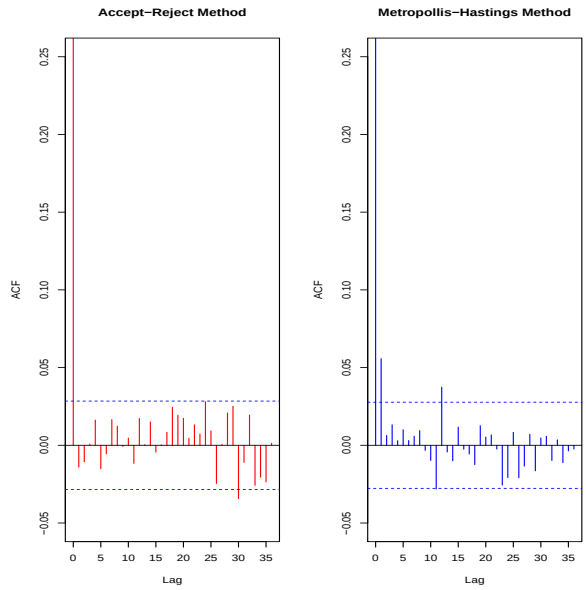
```
#Comparison of Accept-Reject & Metropolis-Hastings
> 4/4.85=0.8247423
> Nsim=5000
> par(mfrow=c(1,2))
> hist(rgamma(Nsim,4.85,1))
> hist(rgamma(Nsim,4,0.8247423))
```

```
> f=function(x){dgamma(x,4.85,1)}
> g=function(x){dgamma(x,4,0.8247423)}
> par(mfrow=c(1,2))
> plot(f, from=0, to=15, col="blue",
+  ylab="")
> plot(g, from=0, to=15, col="red",add=T)
#Trial and Error method to get a good M:
> g=function(x) {1*dgamma(x,4,0.8247423)}
#M=1 suits well.
> f=function(x) {dgamma(x,4.85,1)}
> plot(f, from=0, to=15, ylim=c(0,1),col="blue",
+  ylab="")
> plot(g, from=0, to=15, col="red",add=T)
#Accept-Reject algorithm:
> Nsim=5000
> x=rgamma(Nsim,4,0.8247423)
> u=runif(Nsim,0,1)
> ratio=dgamma(x,4.85,1)/dgamma(x,4,0.8247423)
> ind=I(u<ratio)
> sim<-x[ind==1]
> length(sim)
> par(mfrow=c(1,2))
> hist(sim,freq=F,nclass=20,xlab="X",ylab="Density",
+  main="Accept-Reject Method")
> lines(density(sim,adjust=2),col="red")
> Nsim=5000
#initiate value from target:
> X=rep(rgamma(1,4,0.8247423),Nsim)
> for (i in 2:Nsim){
+  Y=rgamma(1,4,0.8247423) #Candidate
+  rho=dgamma(Y,4.85,1)*dgamma(X[i-1],4,0.8247423)/
+  (dgamma(X[i-1],4.85,1)*dgamma(Y,4,0.8247423))
+  X[i]=X[i-1]+(Y-X[i-1])*(runif(1)<rho)
+  print(rho)
+ }
> print(X);length(X)
> hist(X,freq=F,nclass=20,xlab="X",
+ ylab="Density",main="Metropollis-Hastings Method")
> lines(density(X,adjust=2),col="red")
> mean(sim);sd(sim)^2
> mean(X);sd(X)^2
> par(mfrow=c(1,2))
#To assess correlation in both samples:
> acf(sim,ylim=c(-0.05,0.25),
+ main="Accept-Reject Method", col="red")
> acf(X,ylim=c(-0.05,0.25),
+ main="Metropollis-Hastings Method", col="blue")
```



Remark 1. In the symmetric case $q(x|y) = q(y|x)$,

$$\rho = min\Big\{\frac{f(y_t)}{f(x_t)}, 1\Big\}$$



so the acceptance probability is independent of q (see Example 1 in this section).

Remark 2. The Accept-Reject sample is i.i.d but the Metropolis-Hastings is not. The Accept-Reject acceptance step requires calculating M, so Metropolis-Hastings is Accept-Reject "for the lazy person".

14. Gibbs Samplers

We cover both the two-stage and the multistage Gibbs samplers.

- The two-stage sampler has superior convergence properties
- The multistage Gibbs sampler is the workhorse of the MCMC world
- Gibbs samplers can also be used for missing data and models with latent variables, but we cannot cover them for this short Workshop.
- And, of course, hierarchical models

More Explanation: Gibbs samplers has a huge applications for generating full conditional distributions in a hierarchical models when the distributions have known closed forms. Otherwise, if the distributions are not known the Metropolis Hastings Algorithm can be combined with the Gibbs samplers.

Some properties of Gibbs Samplers:

- Gibbs samplers gather most of their calibration from the target density
- They break complex problems (high dimensional) into a series of easier problems
- May be impossible to build random walk Metropolis-Hastings algorithm
- The sequence of simple problems may take a long time to converge
- But Gibbs sampling is an interesting and useful algorithm.

Historical Points: Gibbs sampling is from the landmark paper by Geman and Geman (1984). The Gibbs sampler is a special case of Metropolis-Hastings. Gelfand and Smith (1990) sparked new interest. In Bayesian methods and statistical computing. They solved problems that were previously unsolvable.

Two-Stage Gibbs Samplers

- △ Creates a Markov chain from a joint distribution
- △ If two random variables X and Y have joint density $f(x, y)$
- △ With corresponding conditional densities $f_{Y|X}$ and $f_{X|Y}$
- △ Generates a Markov chain (X_t, Y_t) according to the following steps

Algorithm

Take $X_0 = x_0$

For $t = 1, 2, ...,$ *generate*

1. $Y_t \sim f_{Y|X}(.|x_{t-1})$;
2. $X_t \sim f_{X|Y}(.|y_t)$.

Multi-stage Gibbs Samplers

- △ There is a natural extension from the two-stage to the multistage Gibbs sampler
- △ For $p > 1$ write $\mathcal{X} = X = (X_1, ..., X_p)^t$
- △ Suppose that we can simulate from the full conditional densities

$$X_i|x_1, x_2, ..., x_{i-1}, x_{i+1}, ..., x_p \sim f_i(x_i|x_1, x_2, ..., x_{i-1}, x_{i+1}, x_p)$$

- △ The multistage Gibbs sampler has the following transition from $X^{(t)}$ to $X^{(t+1)}$:

Algorithm

At iteration $t = 1, 2, ...,$ *given* $x^{(t)} = (x_1^{(t)}, ..., x_p^{(t)})$, *generate*

1. $X_1^{(t+1)} \sim f_1(x_1|x_2^{(t)}, ..., x_p^{(t)})$;
2. $X_2^{(t+1)} \sim f_2(x_2|x_1^{(t+1)}, x_3^{(t)}, ..., x_p^{(t)})$;

...

p. $X_p^{(t+1)} \sim f_p(x_p|x_1^{(t+1)}, ..., x_{p-1}^{(t+1)})$.

Remark: Do not forget to check the convergency of chains for full conditional distributions in hierarchical models (The Rule of checking the propriety of posterior!!)

Example Hierarchical specification for the normal model is the one-way random effects model. There are different ways to parameterize this model, but a possibility is as follows:

$$X_{ij} \sim \mathcal{N}(\theta_i, \sigma^2), \quad i = 1, ..., k, \quad j = 1, ..., n_i,$$

$$\theta_i \sim \mathcal{N}(\mu, \tau^2), \quad i = 1, ..., k,$$

$$\mu \sim \mathcal{N}(\mu_0, \sigma_\mu^2), \quad \sigma^2 \sim \mathcal{IG}(a_1, b_1)$$

$$\tau^2 \sim \mathcal{IG}(a_2, b_2), \quad \sigma_\mu^2 \sim \mathcal{IG}(a_3, b_3).$$

We can derive the set of full conditionals as follows

$$\theta_i \sim \mathcal{N}(\frac{\sigma^2}{\sigma^2 + n_i\tau^2}\mu + \frac{n_i\tau^2}{\sigma^2 + n_i\tau^2}\bar{X}_i, \frac{\sigma^2\tau^2}{\sigma^2 + n_i\tau^2}), \quad i = 1, ..., k,$$

$$\mu \sim \mathcal{N}(\frac{\tau^2}{\tau^2 + k\sigma_\mu^2}\mu_0 + \frac{k\sigma_\mu^2}{\tau^2 + k\sigma_\mu^2}\bar{\theta}, \frac{\sigma_\mu^2\tau^2}{\tau^2 + k\sigma_\mu^2}),$$

$$\sigma^2 \sim \mathcal{IG}(n/2 + a_1, (1/2)\sum_{ij}(X_{ij} - \theta_i)^2 + b_1),$$

$$\tau^2 \sim \mathcal{IG}(k/2 + a_2, (1/2)\sum_{i}(\theta_i - \mu)^2 + b_2),$$

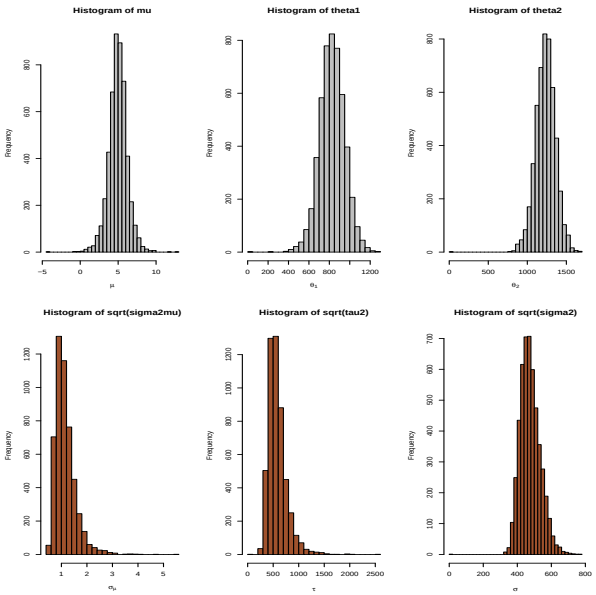
$$\sigma_\mu^2 \sim \mathcal{IG}(1/2 + a_3, (1/2)(\mu - \mu_0)^2 + b_3),$$

where $n = \sum_i n_i$ and $\bar{\theta} = \sum_i n_i\theta_i/n$.

R code and Histograms of marginal posterior distributions from the Gibbs samplers based on 5000 iterations are as follows.

```
## R code

## Gibbs Samplers

> library(MASS) #Required packages for dataset Energy

> library(coda)

> library(lattice)

> library(mcsm)

> Energy
```

```
> x=c(91,504,557,609,693,727,764,803,857,929,970,1043,
+    1089,1195,1384,1713,457,645,736,790,899,991,1104,
+    1154,1203,1320,1417,1569,1685,1843,2296,2710)
> x1=c(91,504,557,609,693,727,764,803,857,929,970,1043,
+     1089,1195,1384,1713) #For Girls
> x2=c(457,645,736,790,899,991,1104,1154,1203,1320,1417,
+     1569,1685,1843,2296,2710) #For Boys
#1 for girls and 2 for boys:
> xbar1=mean(x1);xbar2=mean(x2)
> Nsim=5000
> a1=a2=a3=b1=b2=b3=3
> n1=n2=16;n=32;k=2;mu0=5
> sh1=(n/2)+a1;sh2=(k/2)+a2;sh3=(1/2)+a3
#init arrays
> theta1=theta2=mu=sigma2=tau2=sigma2mu=rep(0.5,Nsim)
> for(i in 2:Nsim){
+  B1=sigma2[i-1]/(sigma2[i-1]+(n1*tau2[i-1]))
+  B2=sigma2[i-1]/(sigma2[i-1]+(n2*tau2[i-1]))
+  B=tau2[i-1]/(tau2[i-1]+(k*sigma2mu[i-1]))
+  thetabar=((n1*theta1[i-1])+(n2*theta2[i-1]))/n
+  theta1[i]=rnorm(1,m=B1*mu[i-1]+(1-B1)*xbar1,
+  sd=sqrt(B1*tau2[i-1]))
+  theta2[i]=rnorm(1,m=B2*mu[i-1]+(1-B2)*xbar2,
+  sd=sqrt(B2*tau2[i-1]))
+  mu[i]=rnorm(1,m=B*mu0+(1-B)*thetabar,
+  sd=sqrt(B*sigma2mu[i-1]))
+  ra1=0.5*(sum((x1-theta1[i])^2)+
+  sum((x2-theta2[i])^2))+b1
+  ra2=0.5*(((theta1[i]-mu[i])^2)+
+  ((theta2[i]-mu[i])^2))+b2
+  ra3=0.5*((mu[i]-mu0)^2)+b3
+  sigma2[i]=1/rgamma(1,shape=sh1,rate=ra1)
+  tau2[i]=1/rgamma(1,shape=sh2,rate=ra2)
+  sigma2mu[i]=1/rgamma(1,shape=sh3,rate=ra3)
+  par(mfrow=c(2,3))
+  hist(mu,xlab=expression(mu),nclass=30,col='gray')
+  hist(theta1,xlab=expression(theta[1]),nclass=30,
+  col='gray')
+  hist(theta2,xlab=expression(theta[2]),nclass=30,
+  col='gray')
+  hist(sqrt(sigma2mu),xlab=expression(sigma[mu]),
+  nclass=30,col='sienna')
+  hist(sqrt(tau2),xlab=expression(tau),nclass=30,
+  col='sienna')
+  hist(sqrt(sigma2),xlab=expression(sigma),
+  nclass=30,col='sienna')
+ }
```



15. Monitoring Convergence

- We look at different diagnostics to check the convergence

of an MCMC algorithm

- To answer to question: "When do we stop our MCMC algorithm?"
- We distinguish between two separate notions of convergence:
- Convergence to stationarity
- Convergence of ergodic averages
- We also discuss some convergence diagnostics contained in the coda package

The MCMC algorithms that we have seen

△ Are convergent because the chains they produce are ergodic.

△ Although this is a necessary theoretical validation of the MCMC algorithms

△ It is insufficient from the implementation viewpoint

△ Theoretical guarantees do not tell us

△ When to stop these algorithms and produce our estimates with confidence.

△ In practice, this is nearly impossible

△ Several runs of your program are usually required until

△ You are satisfied with the outcome

△ You run out of time and/or patience

There are three types of convergence for which assessment may be necessary:

1. Convergence to the stationary distribution
2. Convergence of Averages
3. Approximating iid Sampling

1. Convergence to the stationarity distribution

- First requirement for convergence of an MCMC algorithm
- $(x^{(t)}) \sim f$, the stationary distribution
- This sounds like a minimal requirement
- Assessing that $x^{(t)} \sim f$ is difficult with only a single realization
- A slightly less ambitious goal: Assess the independence from the starting point $x^{(0)}$ based on several realizations of the chain using the same transition kernel.

Remarks:

- When running an MCMC algorithm, the important issues are
- The speed of exploration of the support of f
- The degree of correlation between the $x^{(t)}$'s

2. Convergence of Averages

A more important convergence issue is convergence of the empirical average

$$\frac{1}{T}\sum_{t=1}^{T} h(x^{(t)}) \to \mathbb{E}_f[h(X)]$$

- Two features that distinguish stationary MCMC outcomes from iid ones
- The probabilistic dependence in the sample
- The mixing behavior of the transition,

- That is, how fast the chain explores the support of f
- "Stuck in a mode" might appear to be stationarity
- The missing mass problem again
- Also: The CLT might not be available

3. Approximating iid Sampling

- Ideally, the approximation to f provided by MCMC algorithms should
- Extend to the (approximate) production of iid samples from f .
- A practical solution to this issue is to use subsampling (or batch sampling)
- Reduces correlation between the successive points of the Markov chain.
- Subsampling illustrates this general feature but it loses in efficiency
- Compare two estimators
- δ_1: Uses all of the Markov chain
- δ_2: Uses subsamples
- It can be shown that

$$var(\delta_1) \leq var(\delta_2)$$

The **coda** package

- Plummer et al. have written an R package called coda
- Download and install with library(coda)
- Transform an MCMC output made of a vector or a matrix into an MCMC object that can be processed by coda, as in

```
> summary(mcmc(X))
```

or

```
> plot(mcmc(X))
```

"How absurdly simple!", I cried.

"Quite so!", said he, a little nettled. "Every problem becomes very childish when once it is explained to you."

Arthur Conan Doyle

The Adventure of the Dancing Men