Computational Statistics

Saverio Ranciati

Department of Statistical Sciences, University of Bologna.

Via Belle Arti, 41 - 40126 Bologna, Italy.

E-mail: **saverio.ranciati2@unibo.it**

OUTLINE OF THE COURSE

1. Introduction on basic R programming.

2. Random variable generation.

3. Monte Carlo integration.

4. Monte Carlo optimization.

REFERENCES

Notes and slides (available **virtuale.unibo.it)**
(these notes were kindly provided by Prof. Silvia Bianconcini)

C. Robert and G. Casella (2010). *Introducing Monte Carlo Methods with R*. New York: Springer-Verlag.

EXAM

Written exam

1. Basic R programming

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

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.

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.
- 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.
- It is increasingly common to see people who develop new methodology simultaneously producing an R package.

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.

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.
- R operates on those types as a regular function would operate on a scalar.
- Avoid loops in favor of matrix mainpulations.

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.
`> 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`.

MORE ON THE `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.

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 those of the vector `vec` of the dimension np.

- 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.

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`.
`> cbind(x1,x2)`	horizontal merging of `x1` and `x2`.
`> 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.

The `list` and `data.frame` CLASSES

- 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, . . . , 12.

  `> v2=sample(LETTERS[1:10],30,rep=T)` simulate 30 independent uniform $\{A, B, \ldots, J\}$.

  `> 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.

PROBABILITY DISTRIBUTION 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`	
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 STATISTICS: t-TEST

- Test on the mean.

```
> 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
```

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.

2. Random variable generation

- 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.

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 $G(5/2, 9/2)$ 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 $U(2,5)$.

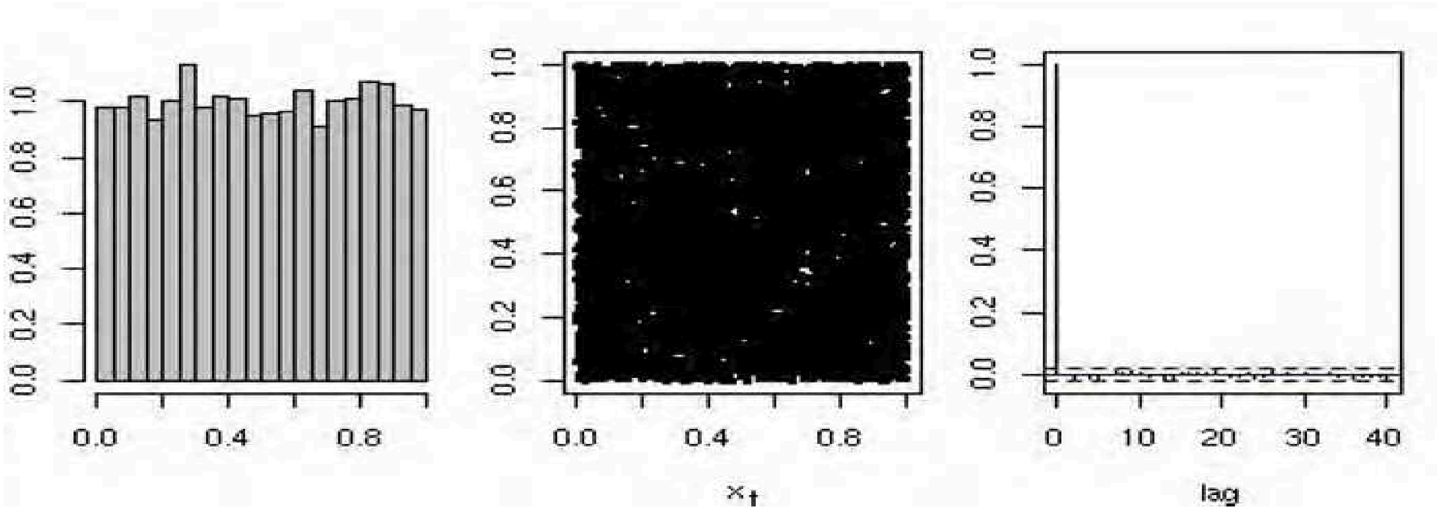
UNIFORM SIMULATION: CHECKING THE GENERATOR

- 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 10000 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

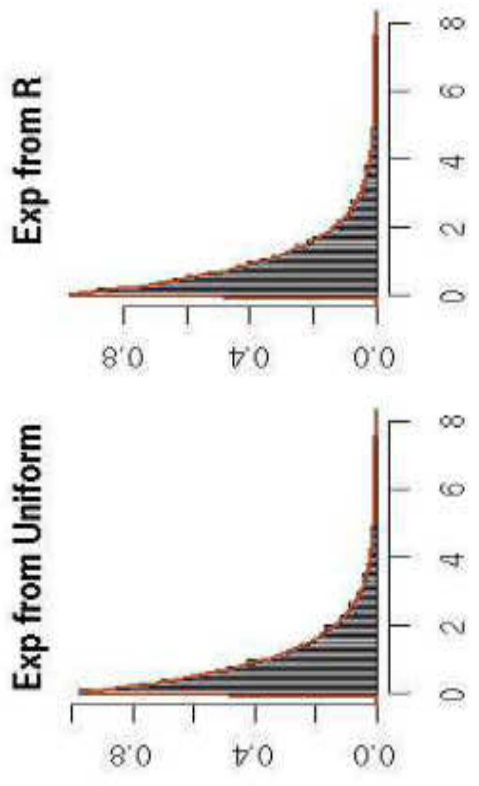
$$F(x) = \int_{-\infty}^{x} f(t)dt,$$

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

- Example 2.1.
  - If $X \sim Exp(1)$, 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*), $Exp(1)$ 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 $F(x) = \frac{1}{2}+\frac{1}{\pi}\arctan\left(\frac{x-\mu}{\sigma}\right)$.

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 i.i.d. $Exp(1)$ random variables,
  - three standard distributions can be derived as

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

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

$$Y = \frac{\sum_{j=1}^{a} X_j}{\sum_{j=1}^{a+b} X_j} \sim Be(a,b), \quad a,b \in \mathbb{N}^*$$

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 $U[0,1]$, the variables X_1 and X_2

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

  are iid $N(0,1)$ 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 $\mathbf{\Sigma} = \mathbf{A}\mathbf{A}'$.
  - $\mathbf{Y} \sim N_p(\mathbf{0}, \mathbf{I}) \Rightarrow \mathbf{AY} \sim N_p(\mathbf{0}, \mathbf{\Sigma})$.
- 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
    ```
- $\mathbf{B}$ 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

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

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

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

DISCRETE DISTRIBUTIONS - BINOMIAL

- Example. To generate $X \sim Bin(10, 0.3)$.
  - The probability values are obtained by `pbinom(k,10,0.3)`.

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

  - First solution: writing your own function.

```
r.bin.dis<-function(n,x,p){

values<-rep(NA,n)

P<-cumsum(p)

for (i in 1:n){u<-runif(1)

j<-1

while (u > P[j]){j<-j+1}

values[i]<-x[j]}

values}
```

  - Second solution: `sample` function.

```
x<-sample(x,n,replace=TRUE,p)
```

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 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.

ACCEPT-REJECTION 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-REJECTION 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 $U \sim U[0, 1]$.
  - 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.

ACCEPT-REJECTION ALGORITHM - R IMPLEMENTATION

- Succinctly, the Accept-Reject Algorithm is

  Accept-Reject Method

```
1. Generate $Y \sim g, U \sim U[0,1]$;
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-REJECTION ALGORITHM - NORMAL 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 $U \leq \exp(-0.5Y^2 + |Y| - 0.5)$.

- Look at R code.

ACCEPT-REJECTION ALGORITHM - THEORY

- Why does this method work?

- A straightforward probability calculation shows

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

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

- The Accept-Reject method is applicable in any dimension.

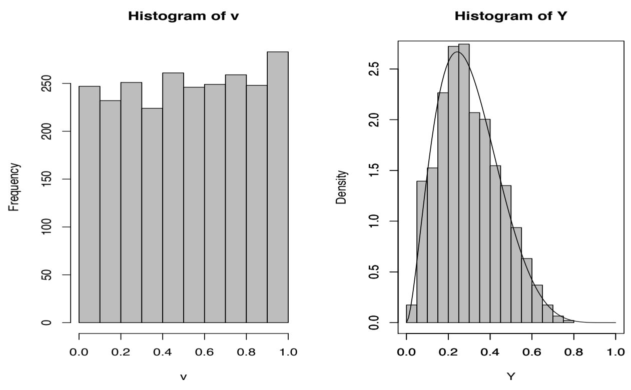
- As long as g is a density over the same space as f.

- Only need to know f/g up to a constant.

- Only need an upper bound on M.

ACCEPT-REJECTION ALGORITHM - BETAS FROM UNIFORMS

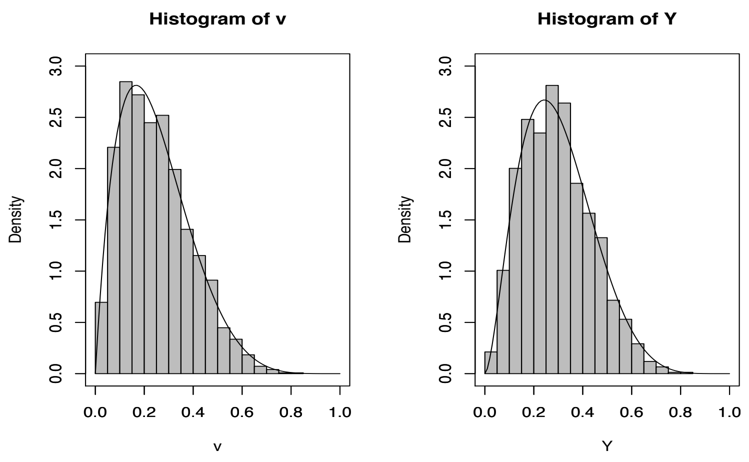
- Generate $X \sim Beta(a, b)$.
- No direct method if a and b are not integers.
- Use a uniform candidate.
- For $a = 2.7$ and $b = 6.3$.



- Acceptance Rate: 37%.

ACCEPT-REJECTION ALGORITHM - BETAS FROM BETAS

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



- Acceptance Rate: 60%.

ACCEPT-REJECTION 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

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

Need $a_1 < a$ and $b_1 < b$.

- Efficiency increases as the candidate gets closer to the target.
- Look at R code.

ACCEPT-REJECTION ALGORITHM - COMMENTS

Some 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.

3. Monte Carlo Integration

- We introduce 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

$$\int_{\chi} h(x) f(x) dx.$$

  - 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.

CLASSICAL MONTE CARLO INTEGRATION - THE MONTE CARLO METHOD

- The generic problem: evaluate

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

  - X takes its values in χ.

- The Monte Carlo Method.

  - Generate a sample $(x_1, \ldots, x_n)$ from the density f.

  - Approximate the integral with

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

CLASSICAL MONTE CARLO INTEGRATION - VALIDATING THE MONTE CARLO METHOD

- The convergence

$$\bar{h}_n = \frac{1}{n}\sum_{j=1}^{n} h(x_j) \to \mathbb{E}_f[h(X)] = \int_{\mathcal{X}} h(x)f(x)dx.$$

  is valid by the Strong Law of Large Numbers.

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

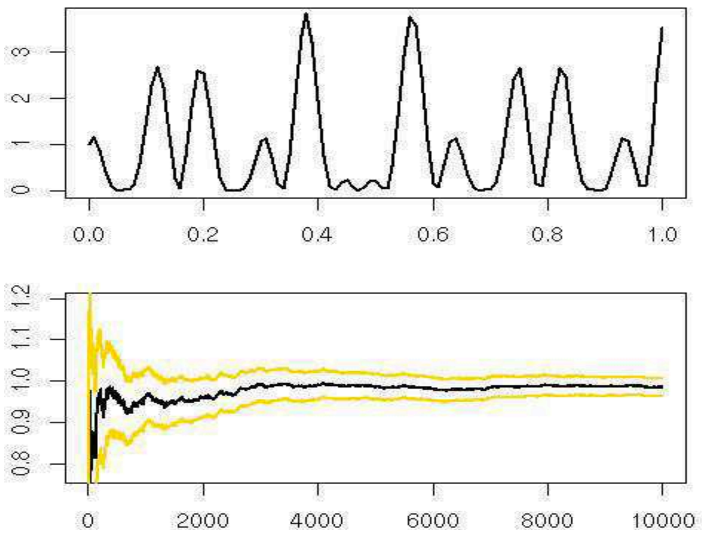
$$\frac{\bar{h}_n - \mathbb{E}_f[h(X)]}{\sqrt{v_n}} \to N(0,1).$$

  – Follows from the Central Limit Theorem.

  – $v_n = \frac{1}{n^2}\sum_{j=1}^{n}[h(x_j) - \bar{h}_n]^2.$

CLASSICAL MONTE CARLO INTEGRATION - A FIRST EXAMPLE

- Look at the function: $h(x) = [\cos(50x) + \sin(20x)]^2$.



- Monitoring convergence.

- R code.

- 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

- The evaluation of the Monte Carlo error is a bonus.

- It assumes that v_n is a proper estimate of the variance of $\bar{h}_n$.

- If v_n does not converge, converges too slowly, the Central Limit Theorem may not apply.

ANOTHER EXAMPLE

- Normal probability

$$\widehat{\Phi}(t) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{x_i \le t} \to \Phi(t) = \int_{-\infty}^{t} \frac{1}{\sqrt{2\pi}} e^{-y^2/2} dy.$$

  - The exact variance $\Phi(t)[1-\Phi(t)]/n$.
  - Conservative: $Var \approx 1/4n$.
  - For a precision of four decimals.
    * Want $2 \times \sqrt{1/4n} \le 10^{-4}$ 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 Strong Law of Large Numbers.

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

  - f is the target density.
  - g is the candidate density.
  - Sound familiar? Just like Accept-Reject.

- So

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

- As long as
  - $Var(h(X)f(X)/g(X)) < \infty$.
  - support of $(h \times f)$ contains the support of g.

REVISITING NORMAL TAIL PROBABILITIES

- $Z \sim N(0,1)$ and we are interested in the probability $P(Z > 4.5)$.

- `pnorm(-4.5,log=T)`

```
[1] -12.59242
```

- Simulating $Z^{(i)} \sim N(0,1)$ only produces a hit once in about 3 million of iterations!
  - Very rare event for the normal.
  - Not-so-rare for a distribution sitting out there!

- Take $g = Exp(1)$ truncated at 4.5:

$$g(y) = \frac{e^{-y}}{\int_{4.5}^{\infty} e^{-x} dx} = e^{-(y-4.5)}.$$

- 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 - SELECTION OF THE IMPORTANCE FUNCTION

Some choices of g are better than others.

While $\frac{1}{n}\sum_{i=1}^{n} h(x_i)\frac{f(x_i)}{g(x_i)} \to \mathbb{E}_f[h(X)]$ almost surely, its variance is finite only when

$$\mathbb{E}_g\left[h^2(X)\frac{f^2(X)}{g^2(X)}\right] = \mathbb{E}_f\left[h^2(X)\frac{f(X)}{g(X)}\right] = \int_{\mathcal{X}} h^2(x)\frac{f(x)}{g(x)}dx < \infty$$

- Instrumental distributions with tails lighter than those of f (those with unbounded ratios f/g) are not appropriate for importance sampling.

- If the ratio f/g is unbounded, the weights $f(x_i)/g(x_i)$ will vary widely, giving too much importance to a few values x_i.

SELECTION OF THE IMPORTANCE FUNCTION - EXAMPLE

- Target: Cauchy density $f(x) = \frac{1}{\pi}\frac{1}{1+x^2}$.

- Importance function: standard Normal density

$$g(x) = \frac{1}{\sqrt{2\pi}} \exp\left[-x^2/2\right].$$

- The ratio $f(x)/g(x) \propto \exp(x^2/2)/(1+x^2)$ is explosive.

- R code

```
x=rnorm(10^6)
wein=decauchy(x)/dnorm(x)
boxplot(wein/sum(wein))
plot(cumsum(wein*(x>2)*(x<6))/cumsum(wein),type="l")
abline(a=pcauchy(6)-pcauchy(2),b=0,col="sienna")
```

IMPORTANCE SAMPLING - OPTIMAL IMPORTANCE FUNCTION

Distributions g with thicker tails than f ensure that the ratio f/g does not cause the divergence of $\mathbb{E}_f\left[h^2(X)\frac{f(X)}{g(X)}\right]$.

Sufficient conditions

(a) $f(x)/g(x) < M$, $\forall x \in \chi$ and $Var_f[h(X)] < \infty$;

(b) χ is compact, $f(x) < F$ and $g(x) > \varepsilon, \forall x \in \chi$.

These conditions are quite restrictive.

Among the distributions g leading to finite variances for the estimator $\frac{1}{n}\sum_{i=1}^{n} h(x_i)\frac{f(x_i)}{g(x_i)}$, the choice of g that minimizes the variance of the estimator is

$$g^*(x) = \frac{|h(x)|f(x)}{\int_\chi |h(z)|f(z)dz}$$

From a practical point of view, this suggests looking for distributions g for which $|h|f/g$ is almost constant with finite variance.

IMPORTANCE SAMPLING - EXAMPLE

Compute the integral $\int_0^\infty e^{-x^3}dx$ through importance sampling from:

- Standard normal density.
- Exponential density function exp(1).

Evaluate the variability of each estimator using a single sequence of length 1000.

```
int1<-function(n){
x=rnorm(n)
fn=rep(0,n)
fn[x>0]=exp(-x[x>0]^3)/dnorm(x[x>0])
fn}

int2<-function(n){
x=rexp(n)
fn=exp(-x^3)/dexp(x)
fn}

Nsim=10^4
i1=int1(Nsim)
i2=int2(Nsim)
mean(i1)
mean(i2)

v1=(mean(i1^2)-mean(i1)^2)/Nsim
v2=(mean(i2^2)-mean(i2)^2)/Nsim
```

4. Monte Carlo Optimization

- Two uses of computer-generated random variables to solve optimization problems.
- The first use is to produce stochastic search technique.
  - 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 $C(\theta, 1)$.

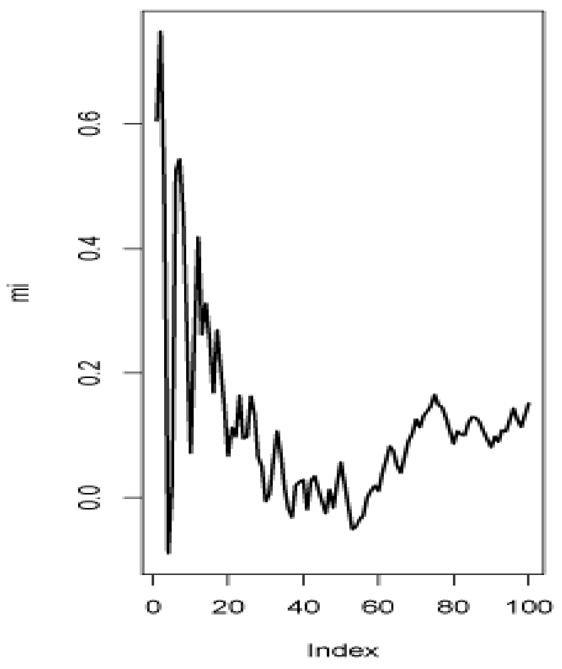
- When maximizing the likelihood of a Cauchy $C(\theta, 1)$ sample,

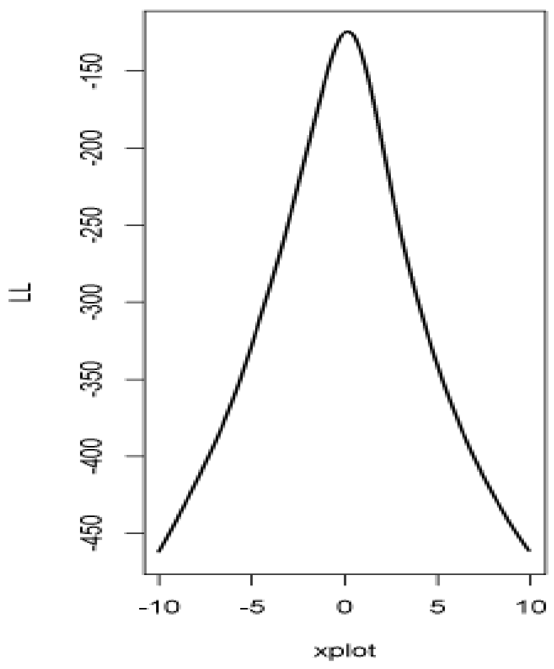
$$\ell(\theta | x_1, \ldots, x_n) = \frac{1}{\pi} \prod_{i=1}^{n} \frac{1}{1 + (x_i - \theta)^2}.$$

- The sequence of maxima (MLEs) $\to \theta^* = 0$ when $n \to \infty$.

- But the journey is not a smooth one . . .

MONTE CARLO OPTIMIZATION - CAUCHY LIKELIHOOD





- MLEs (*left*) at each sample size, $n = 1,500$, and plot of final likelihood (*right*).
  - Why are the MLEs so wiggly?
  - The likelihood is not as well-behaved as it seems.

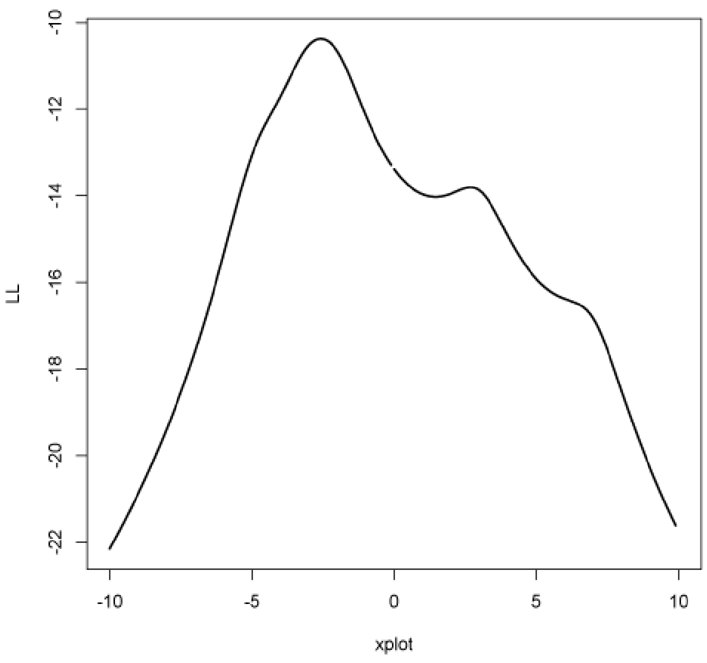
MONTE CARLO OPTIMIZATION - CAUCHY LIKELIHOOD

- The likelihood $\ell(\theta|x_1,\ldots,x_n) = \prod_{i=1}^{n} \frac{1}{1+(x_i-\theta)^2}$ is like a polynomial of degree $2n$.

- The derivative has $2n$ zeros.

- Hard to see if $n = 500$.

- Here is $n = 5$.



- R code.

MONTE CARLO OPTIMIZATION - NEWTON-RAPHSON

- Similarly, `nlm` is a generic R function using the Newton-Raphson method.

- Based on the recurrence relation

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

  where the matrix of the second derivatives is called the *Hessian*

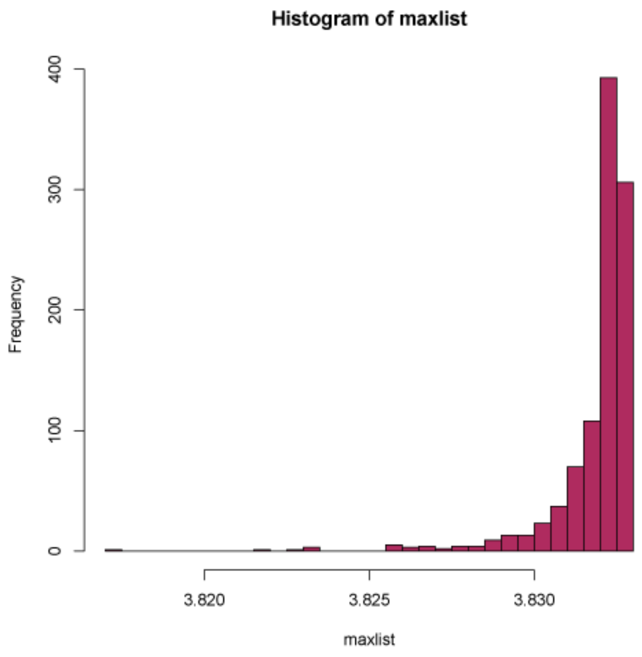
  – This method is perfect when h is quadratic.

  – But may also deteriorate when h is highly nonlinear.

  – It also obviously depends on the starting point θ_0 when h has several minima.

STOCHASTIC SEARCH - A BASIC SOLUTION

- A natural if rudimentary way of using simulation to find $\max_\theta h(\theta)$.
  - Simulate points over Θ according to an arbitrary distribution f positive on Θ.
  - Until a high value of $h(\theta)$ is observed.



  - Recall $h(x) = [\cos(50x) + \sin(20x)]^2$.
  - Max=3.8325.
  - Histogram of 1000 runs.

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.

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

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

STOCHASTIC GRADIENT METHODS

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

- Stochastic Variation.

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

  - β_j is a second decreasing sequence.
  - ς_j is uniform on the unit sphere $||\varsigma|| = 1$.

- We then use

$$\theta_{j+1} = \theta_j + \frac{\alpha_j}{2\beta_j} \nabla h(\theta_j, \beta_j \varsigma_j) \varsigma_j .$$

SIMULATED ANNEALING - INTRODUCTION

- His 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.

METROPOLIS ALGORITHM/ SIMULATED ANNEALING

- Simulation method proposed by Metropolis et al. (1953).
- Update from θ_t to θ_{t+1} is based on Metropolis-Hasting algorithm step.
- ς is generated from a symmetric density g.
- The new value of θ_{t+1} is generated as

$$\theta_{t+1} = \begin{cases} \theta_t + \varsigma & \texttt{with probability} \quad \rho = \exp(\nabla h/T) \wedge 1 \\ \theta_t & \texttt{with probability} \quad 1 - \rho \end{cases}$$

  - $\Delta h = h(\theta_t + \varsigma) - h(\theta_t)$.
  - If $h(\varsigma) \geq h(\theta_t)$, $\theta_t + \varsigma$ is accepted.
  - If $h(\theta_t + \varsigma) < h(\theta_t)$, ς 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 ς from an instrumental distribution with density $g(\varsigma)$.
  2. Accept $\theta_{i+1} = \theta_i + \varsigma$ 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.