

CONTENTS

Chapter 1

Montecarlo Simulations

1. Simulation ([Sampling]) 2. Integration ([Expectations]) 3. Optimisation

Called "Montecarlo" (casino) because of its use of randomly generated numbers. These are used to solve something that isn't random.

Example 1.0.1

Given a field with a fixed area (eg. 20km^2) and a lake inside of it with an irregular shape with its own area.



How do we estimate (compute/measure is impossible - Coastline Paradox) the area of the lake? Note: the area of the lake isn't random, but injecting randomness makes it much easier to solve.

We can shoot cannon balls into the field **at random**, recording if it hit water or not. By repeating this process, we get a long string of binary values which were **randomly generated** (in a weird way).

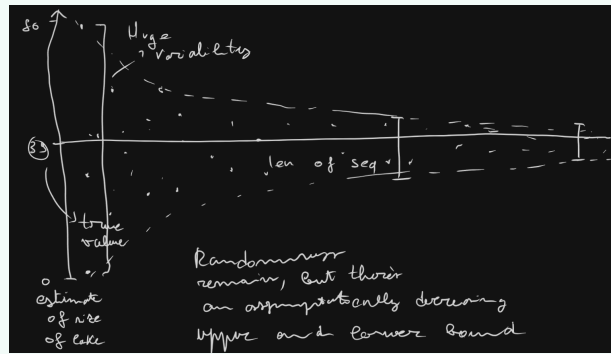
We can now define a random variable X (in this case a Bernoulli) with n recorded outcomes X_1, \dots, X_n

Seen as the area of the lake and field don't change, the probability of hitting water remains the same. So for each random variable $P(X_1 = 1) = \dots = P(X_n = 1)$, they are **identically distributed**.

Even if we keep hitting grass in one direction, we can't decide to move the cannon based on this information, because this wouldn't be random and it would cause dependence between the n random variables, which need to be iid (independent identically distributed).

Given $A = \text{"hit water"}$, $P(A) \propto \frac{\text{Area Lake}}{\text{Area Field}}$. So if we want to estimate the area of the lake, we can multiply the field area by the probability of hitting water. The measurement has become probabilistic.

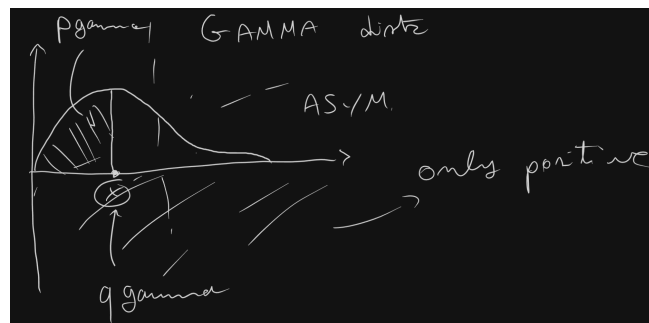
To find such probability, we can find the avg. value of X (sample proportion/mean). The more samples we have, the more precise the measurement (thanks to CLT).



1.1 R

Suite of pre-implemented functions, use them!

Gamma distribution is a continuous distribution of positive random variables.



Use 'rexp', 'rgamma', ect. for preloaded distributions (you can search for the name in the help, selecting from a set of 4 different functions, d for density, p for probability, q to get value from probability, r for random selections). How to sample from *any* distribution? For example, in Bayesian statistics we don't know the shape of the posterior.

To view data, we can use the 'hist' function for a histogram. If we want densities, we can use 'freq = FALSE' (this is what we'll use). This is actually an estimator of the distribution of the random variable. We can also use 'curve', which is also an estimator, but a smooth one.

Be careful about the size of the plots (don't be tricked). Height of the histogram estimates the density, but we can't tell if they're independent. The second plot is used for this case, plotting points in a 2d space with slightly shifted values. The acf (auto correlation function) plot measures correlation of a time series, by taking a sequence and shifting it many times, recording the correlation of a vector with itself. The x axis indicates the "lag", so at 0 it's 1 (correlation between two identical vectors). If our values are independent, we should see no correlation (close to 0).

By setting the seed, we can fix the pseudo-random sequence.

Example 1.1.1 (Field example in R)

Matrix with ones (lake) and zeros (field). We can plot this now. We then randomly select a matrix element and record if it's a 0 or 1.

Note that we're using discrete values. We can sample using the 'sample' function, with which we can define the set of tuples to sample from, the amount, with or without replacement (in our case the former with 'replace=TRUE').

The accuracy depends on the amount of random variables whose outcome we can observe, at a cost to computational time.

To define a function, the correct syntax is: Name <- function(parameters)

Chapter 2

Sampling

2.1 Direct Continuous

2.1.1 PIT

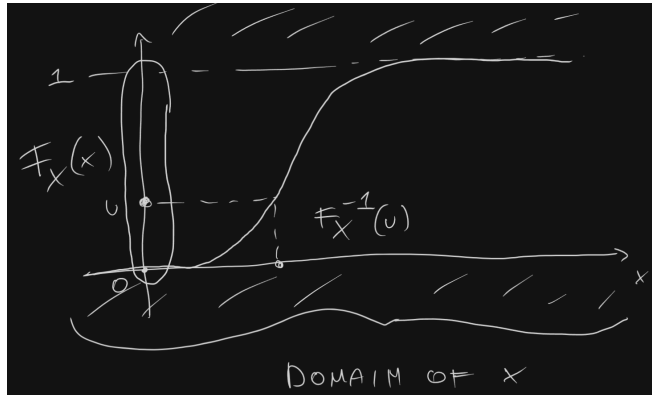
Remember to search for other packages that might implement the function

We have a starting function of which we take the inverse. We are using the cumulative integral function, thus the "Integral" part.

The PIT can produce any random variable starting from a uniform. Lets say we want a random variable with density f and cdf F :

$$F_X(x) = \int_{-\infty}^x f_X(t) dt$$

We set $U = F_X(X)$ and solve for X . Note that F_X has to be invertible, although this is almost always the case.



The cdf always (with some exceptions) exists, but the density might not. So the first thing we find is the cumulative, and if it exists we can calc the integral and get the density.

We use the definition of the generalised inverse:

$$F^{-1}(u) = \inf\{x; F(x) \geq u\}$$

If $U \sim Unif(0, 1)$, then $F_X^{-1} = X$. This only works for the uniform distribution.

We have that $F_X(x) = P(X \leq x) = \int_{-\infty}^x f_X(t) dt$. In general, it's a **monotonic, non-decreasing** function. Outside of weird cases (greater than numerable amount of discontinuous points in f_X i think), it's continuous.

We can't sample directly from X because we don't know its distribution. But we can select a random value from $F_X(x)$ then find its inverse, but does this truly give the correct distribution?

The uniform distribution has some cool props: - $U \sim Unif(a, b) \iff \forall u \in [a, b]. P(U \leq u) = F_U(u) = u$

Thus:

$$P(F_X^{-1}(U) \leq x) = P(F_X(F_X^{-1}(U)) \leq F_X(x)) = P(U \leq F_X(x)) = F_X(x)$$

So we can say $X = F_X^{-1}(U)$

2.1.2 Known Distributions

Used when a distribution is easy to construct using other distributions which are easy to simulate. Given $X_i \sim \text{Exp}(1)$ i.i.d, three standard distributions can be derived as:

- $Y = 2 \sum_{j=1}^v X_j \sim \chi_{2v}^2$ (Chi-squared distribution)
- $Y = \beta \sum_{j=1}^a X_j \sim G(a, \beta)$ (Gamma distribution)
- $Y = \frac{\sum_{j=1}^a X_j}{\sum_{j=1}^{a+b} X_j} \sim \text{Be}(a, b)$ (Beta distribution)

Where $v, a, b \in \mathbb{N}^* = \{1, 2, 3, \dots\}$.

Gaussian

$F_X(x) = P(X \leq x) = P(X \leftarrow x)$ $X \sim N(\mu, \sigma^2)$, has no closed form solution (no analytical solution), so we can't use PIT (we can't invert it).

Box-Muller algorithm is a direct transformation method which gives two normals from two uniforms. We don't have to remember the algorithm. It's **exact**, but not the most efficient.

Why is there π in a Gaussian distr? Looking at the algorithm, we use trigonometric functions, which is weird. R uses the PIT by using a very close approximation of the cdf which is invertible.

Multivariate Gaussian

Multivariate Gaussian instead of having a single variable $X \sim N(\mu, \sigma^2)$, we have a vector:

$$() \sim N_p(\mu, \Sigma)$$

Where μ is a vector and Σ is a Matrix that indicates the variances (on the diag) and covariances between each pair of RV. It's called a variance and covariance matrix, and it's symmetric and positive definite (all variances are different to 0).

Each individual RV of a multivariate gaussian is a gaussian (gaussian family is closed under). Note, the opposite is not true.

If Σ is not diagonal, it means at least two RV are dependent.

This pops up for PCA.

Inverse standardisation for multivariate: How do we square a matrix? Cholesky! We take the Cholesky decomposition $\Sigma = AA^T$ (more or less the square root of Σ).

$$Y \sim N_p(0, 1) = AY \sim N_p(0, \Sigma)$$

There are R packages that replicate these steps, but it's faster and better to sample from a gaussian and then stretch and squeeze the vector to rebuild the covariance structure. 2

2.1.3 General Case (Using Derivative)

Given $X \sim f_X(x)$ and $Z = g(X) : X = g^{-1}(Z)$ (g allows an inverse), then:

$$f_Z(z) = f_X(g^{-1}(z)) \cdot \left| \frac{\delta g^{-1}(z)}{\delta z} \right|$$

Note:

We don't always have a density.

Example 2.1.1 (Uniform R.V.)

$$U \sim \text{Unif}(0, 1) \rightarrow f_U(u) = \begin{cases} 0 & u \notin [0, 1] \\ 1 & u \in [0, 1] \end{cases}$$

$X = (b - a) \cdot U + a$ (linear transformation of a uniform). What is the distribution of X ?

First we calc the inverse:

$$X - a = (b - a) \cdot U$$

$$U = \frac{X - a}{b - a} = g^{-1}(\cdot)$$

Now the derivative:

$$\frac{\partial g^{-1}(X)}{\partial x} = \frac{\partial [\frac{X-a}{b-a}]}{\partial x}$$

$$\frac{\partial [\frac{X-a}{b-a} - \frac{a}{b-a}]}{\partial x}$$

$$\frac{\partial [\frac{X}{b-a}]}{\partial x} = \frac{1}{b-a}$$

So, applying the general transform:

$$f_X(x) = f_U\left(\frac{x-a}{b-a}\right) \cdot \frac{1}{b-a} = \begin{cases} 0 & f_U = 0 \iff x \notin [a, b] \\ \frac{1}{b-a} & \end{cases}$$

Thus $X \sim \text{Unif}(a, b)$

Example 2.1.2 (Gaussian)

$X \sim N(\mu, \sigma^2)$ and $Z = \frac{X-\mu}{\sigma} = g(X)$ (never seen before transformation)

$X = Z \cdot \sigma + \mu = g^{-1}(Z)$

So the derivative is simply

$$\frac{\partial g^{-1}(Z)}{\partial z} = \sigma$$

$$f_Z(z) = f_X(g^{-1}(z)) \cdot \sigma$$

$$\frac{1}{\sqrt{2\pi}} \cdot \frac{1}{\sigma} \cdot e^{-\frac{(z\sigma + \mu - \mu)^2}{2\sigma^2}} \cdot \sigma$$

$$\frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{z^2}{2}}$$

This is the density of the standard gaussian distribution $N(0, 1)$, and this is why we standardize in this way.

We now look at direct distributions for discrete random variables. After, we'll look at indirect meethods with the accept-reject algorithm.

2.2 Direct Discrete

Non-continuous cdf.

2.2.1 General Algorithm

Based on inverse transform, but not the same as PIT. This is not used much, because most discrete distributions are known.

We have to store the values of the cdf (we have to be able to compute it). Then sample from a uniform and check where the value falls:

$$X = k \iff p_{k-1} < U < p_k$$

Note:

If we have a very large Poisson, we'll have to store many probability values for computation.

$$P_X(x) = P(X = x) = \binom{10}{x} p^x (1-p)^{10-x}$$

The problem is calculation (binomial can get very large), especially for unlimited distributions like poisson. Given two RV X, Y , then

$$X = Y \iff F_X(x) = F_Y(y)$$

The binomial distribution isn't very used, in contrast to Poisson which was very used in sports analysis seen as it **counts** "stuff" in a specific time window (highly connected to exponential). It's also known as the "rare distribution", seen as the only parameter λ tells you on average the number of events you expect to happen, but this is also the value of the variance (which doesn't happen in real life)

$$X \sim \text{Poisson}(\lambda) \rightarrow \begin{cases} E(X) = \lambda \\ \text{Var}(X) = \lambda \end{cases}$$

This is called the **overdispersion problem** ($\text{Var}(X) \gg E(X)$), and it's the main reason for moving on to more complex models. So the computation is quite difficult, also because of factorial and exponential terms.

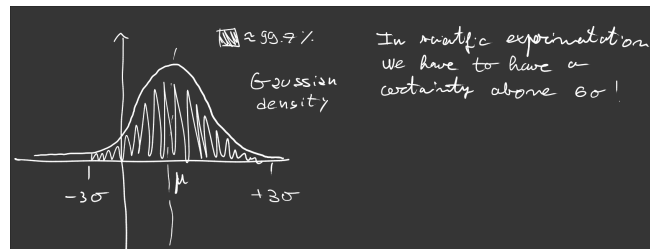
$$\text{Poisson}(\lambda) \rightarrow \lambda \rightarrow +\infty \rightarrow \sim N(\lambda, \lambda)$$

So we can use a Gaussian to approximate a Poisson distribution.

When $\lambda > 50$ we can say that most of the probability of the RV is

$$\lambda \pm 3\sqrt{\lambda}$$

So this is the range of values for X that we should consider to calculate the distribution. This is because of its similarity to the gaussian.



So some of the values in the domain of X are useless to check, seen as they're highly unlikely

Example 2.2.1

$X \sim \text{Poisson}(100)$. If we consider the Gaussian approx. most of the values will be in $(70, 130)$. So if

2.3 Indirect

2.3.1 Bayes' Theorem

Giuro che questo non lo rifaccio

Continuous

Given two cont RV X, Y

$$f_{X|Y=y}(x)$$

is the distribution of X given some info (Y)

$(X|Y = y)$ random variable

we're just focusing on a specific cross section of the population (e.g. income at a specific age).

We want to find its density, we can use Bayes

$$f_{X|Y=y}(x) = \frac{f_{X,Y}}{f_Y(y)} = \frac{f_{Y|X=x}(y)f_X(x)}{f_Y(y)}$$

When you want to move a RV behind the conditioning line, you divide by the marginal.

We can use the marginal distribution for the denominator

$$\frac{f_{X,Y}(x, y)}{\int_Y f_{X,Y}(x, t) dt}$$

by marginalizing out Y (same idea as the discrete version).

2.3.2 Expectation

Given X cont. RV $X \sim f_X(x)$:

$$E_X(X) = \int_{D_X} x f_X(x) dx$$

Note:

Always indicate the distribution used as a subscript. This will become important.

$$E_Y(f_{X|Y}(X|Y)) = \int_{D_X} f_Y(y) f_{X|Y}(X|Y) dy$$

This is a product of a conditional times a marginal (moving Y from the back to the front). So we have the joint

$$= \int_{D_Y} f_{X,Y}(X, Y) dy = f_X(x)$$

So we can use expectation to marginalize out variables. Why is this important?

If we have values sampled from a conditional distribution, taking the expected values (average) we're removing the influence of Y . This is the building block for the Montecarlo-Markov Chain (most widely used Montecarlo method).

2.3.3 Accept-Reject Method

We need to know:

- When it works
- Main ingredients
- No proof!
- The actual algorithm

It's the first indirect method for sampling. It works with two distributions:

- A **candidate** dist.
- A **target** dist.

Remember the lake? Let's say the area is $1 \times 1 = 1 \text{ km}^2$. We are shooting cannon balls at random, so

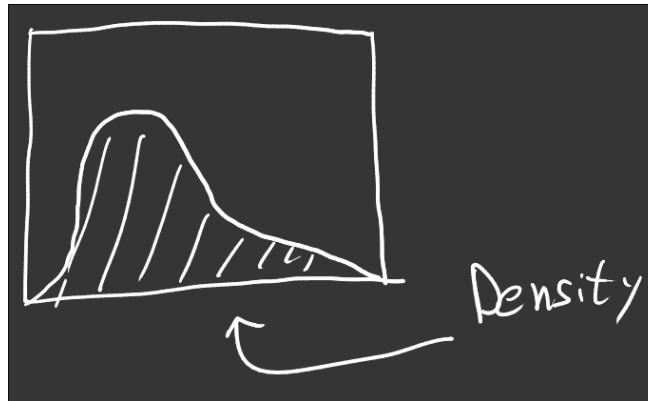
$$X \sim \text{Unif}(0, 1)$$

For every two random values we get a result

$$(u_1, u_2) = \text{HIT!}$$

So we generate a uniform for each side.

By changing the shape of the lake, we can get a more interesting result



We're now randomly testing a **density function** with two uniform RV.

Ingredients

- **Density** f of interest (only functional form) and be able to compute it.
- **Candidate** distribution g .

The candidate choice is up to us. Some are smarter than others, but as long as we don't pick functions that don't satisfy a certain condition (we'll see later), it's ok. In the exam we'll be given the candidate.

The candidate generates a value. We need to test

$$u \leq \frac{f_X(y)}{M g_Y(y)}$$

If it's satisfied we accept (it comes from f_X). Otherwise, we reject it and try again (similar to the discrete case).

Proposal density constraints:

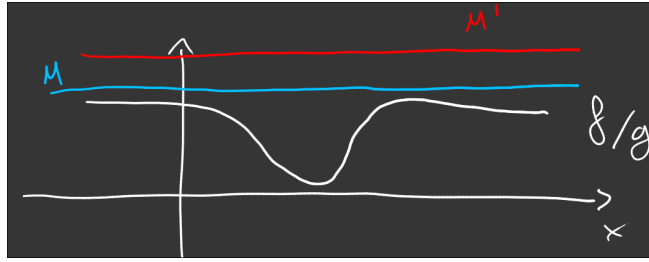
- Target and candidate have to have compatible supports ($S_X \subseteq S_Y$).
- There is a constant M with $f(x)/g(x) \leq M \forall x$.

The last requirement is less trivial. It means that the ratio between the distributions has to be bounded.

If these are satisfied, then

1. Generate $Y \sim g$ and **independently** generate from $U \sim \text{Unif}(0, 1)$.
2. Check if $U \leq \frac{1}{M} \frac{f(Y)}{g(Y)}$ then set $X = Y$ (accept)
3. If it's not satisfied, discard Y and U and start again.

Note that $M = \sup_x \frac{f(x)}{g(x)}$. The crucial requirement that M exists means that M can have many different values.



The best value for M is the one that reduces the area where the points fall and don't belong to the density, reducing efficiency

$$P(\text{Accept}) = \frac{1}{M}$$

So in the case of the graph you should choose the blue line.

Proof

Does it work? We need to think from a statistical pov: Are the accepted values coming from $f_X(x)$? What is the distr. of the accepted values? What is the CDF?

Let's define the event

$$A = \text{"Accept a value"}$$

Where A is basically

$$Y|U \leq \frac{1}{M} \frac{f(Y)}{g(Y)}$$

so a distribution conditional to a test where $U \sim \text{Unif}(0, 1)$ and $Y \sim g(y)$. Lets look at the CDF

$$\begin{aligned} P(Y \leq x | U \leq \frac{1}{M} \frac{f(Y)}{g(Y)}) &= \frac{P(C, D)}{P(D)} \\ &= \frac{P(Y \leq x, U \leq \frac{1}{M} \frac{f(Y)}{g(Y)})}{P(U \leq \frac{1}{M} \frac{f(Y)}{g(Y)})} \\ &= \frac{\int_{-\infty}^x \left(\int_{-\infty}^{\frac{1}{M} \frac{f(y)}{g(y)}} 1 \, du \right) \cdot g_Y(y) \, dy}{\int_{-\infty}^{+\infty} \left(\int_{-\infty}^{\frac{1}{M} \frac{f(y)}{g(y)}} 1 \, du \right) \cdot g_Y(y) \, dy} \\ &= \frac{\int_{-\infty}^x \frac{1}{M} \frac{f(y)}{g(y)} g_Y(y) \, dy}{\int_{-\infty}^{+\infty} \frac{1}{M} \frac{f(y)}{g(y)} g_Y(y) \, dy} \\ &= \frac{\int_{-\infty}^x f(y) \, dy}{\int_{-\infty}^{+\infty} f(y) \, dy} \\ &= \frac{P(X \leq x)}{1} = F_X(x) \end{aligned}$$

So the accepted values have the same cdf as X !

Note:

g and M are arbitrary and don't matter for the correctness of the algorithm.

2.4 Accept - Reject algorithm

TODO

Example 2.4.1 (AC- algh)

We aim to simulate a random variable X from the Standard Normal distribution using the Accept-Reject method with a Laplace (Double Exponential) candidate distribution.

- Target density (f): The Standard Normal distribution ($\mathcal{N}(0, 1)$):

$$f(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) \quad x \in \mathbb{R}$$

- Candidate Density (g): The Laplace distribution ($Laplace(1)$):

$$g(y) = \frac{1}{2} \exp(-|y|), \quad y \in \mathbb{R}$$

We need to find a constant M such that $f(x) \leq Mg(x)$ for all x . This is equivalent to finding the maximum of the ratio $f(x)/g(x)$:

$$\frac{f(x)}{g(x)} = \frac{\frac{1}{\sqrt{2\pi}} e^{-x^2/2}}{\frac{1}{2} e^{-|x|}} = \sqrt{\frac{2}{\pi}} \exp\left(-\frac{x^2}{2} + |x|\right)$$

To maximize this ratio, we maximize the expression $h(x) = \sqrt{\frac{2}{\pi}} \exp\left(-\frac{x^2}{2} + |x|\right)$, for doing that we have to set the derivate to 0:

$$h'(x) = \sqrt{\frac{2}{\pi}} \exp\left(-\frac{x^2}{2} + |x|\right) \left(-x + \frac{x}{|x|}\right) = 0$$

Solving this, we find that the maximum occurs at $x = 1$ and $x = -1$. Evaluating $h(x)$ at these points gives:

$$M = h(1) = h(-1) = \sqrt{\frac{2}{\pi}} e^{-1/2+1} = \sqrt{\frac{2e}{\pi}}$$

So we have

$$U \leq \frac{1}{M} \frac{f(Y)}{g(Y)} = \frac{1}{\sqrt{\frac{2e}{\pi}}} \cdot \sqrt{\frac{2}{\pi}} \exp\left(-\frac{Y^2}{2} + |Y|\right) = e^{-1/2} \exp\left(-\frac{Y^2}{2} + |Y|\right)$$

```
dlaplace <- function(x) 0.5 * exp(-abs(x)) # this is the Laplace density
curve(dlaplace(x), 6,6) # BE CAREFUL, if you dont specify the range curve will assume [0,1]
plaplace <- function(x) ifelse(x < 0, 0.5 * exp(x), 1 - 0.5 * exp(-x)) # Laplace CDF
qlaplace <- function(u) ifelse(u < 0.5, log(2 * u), -log(2 * (1 - u))) # Laplace Quantile function -> IM
rlaplace <- function(n) qlaplace(runif(n)) # n random variables from Laplace distribution, using PIT

# Accept-Reject algorithm to sample from N(0,1) using Laplace(1) as candidate
rnorm_ar <- function(n, g, rg, M, report = true){ # g is the density of candidate, rg is the random generator
  x<- rep(0,n) # vector to store accepted values
  ntry <- 0 # number of tries
  for (i in 1:n){
    done <- false
    while (!done){
      # FOUR STEPS OF AR ALGORITHM
      ntry <- ntry + 1
      y <- rg(1) # sample from the Proposal
      u<- M * runif(1) # sample from Uniform
      if (u < (dnorm(y)/g(y))) done <- true # accept
    }
  }
}
```

```

        # PAY ATTENTION: dnorm is the density of the target distribution SO IT MUST BE CHANGED IF THE TARGET DISTRIBUTION CHANGES
    }
    x[i] <- y # store accepted value
}
if (report){
    cat("Acceptance Rate:", n / ntry, "\n")
}
return (x)
}

Nsim <- 10^4
M <- sqrt(2*exp(1)/pi)
x<- rnorm_ar (Nsim, dlaplace, rlaplace, M)

```

An other algh in R:

```

# Generating beta from uniform
a = 2.7
b = 6.3
curve (dbeta(x,a,b), col="red")

# f is already our ratio cause g(y)=1 (uniform density)
f<-function(x) dbeta(x,a,b)
M <- optimize (f, interval = c(0,1), maximum=T)$objective #

rbeta <- function (n,a,b,M,report = TRUE){
    x<- rep(0,n)
    ntry <- 0
    for(i in 1:n){
        done = FALSE
        while(!done){
            ntry<- ntry +1
            y=runif(1)
            u=M*runif(1)
            if (u<dbeta(y,a,b)/dunif(y)) done = TRUE
        }
        x[i]<- y
    }
    if (report) cat("I needed ", ntry, "trials to get", n, return(x))
}
Nsim <- 10^4
set.seed(10)
x = rbeta2(Nsim, a, b, M)

par(mfrow = c(1,2))
hist(x, freq= FALSE)
curve(dbeta(x,a,b), col = "red", add= TRUE)

```

Alternative for ar IF YOU HAVE REDUCED COMPUTATIONAL RESOURCE

```

### Generating betas from betas
# -----
# OBIETTIVO: Simulare da una Beta(2.7, 6.3) usando una Beta(2, 6)
# come densità strumentale (Proposal g).
# -----

# 1. Definizione dei parametri della distribuzione Target f(x)
a = 2.7

```

```

b = 6.3

# 2. Definizione della funzione rapporto  $r(x) = f(x) / g(x)$ 
#  $f(x) = \text{dbeta}(x, a, b)$  [Target]
#  $g(x) = \text{dbeta}(x, 2, 6)$  [Proposal]
f <- function(x) dbeta(x, a, b) / dbeta(x, 2, 6)

# 3. Calcolo della costante M
# M deve essere tale che  $f(x) \leq M * g(x)$  per ogni x.
# Troviamo il massimo del rapporto  $f(x)/g(x)$  usando 'optimize'.
# $objective estrae il valore numerico del massimo.
M <- optimize(f, maximum=T, interval=c(0,1))$objective

# 4. Definizione della funzione di generazione (Algoritmo AR)
rbeta3 <- function(n, a, b, M, report=TRUE){

  # Inizializzazione del vettore che conterrà i campioni accettati
  x <- rep(0, n)

  # Contatore per monitorare l'efficienza (totale tentativi fatti)
  ntry <- 0

  # Ciclo per generare n valori accettati
  for (i in 1:n){
    done <- FALSE

    # Ciclo while: continua a provare finché non accetta un candidato
    while (!done){
      ntry <- ntry + 1

      # A. Generazione del Candidato Y dalla Proposal  $g \sim \text{Beta}(2, 6)$ 
      y <- rbeta(1, 2, 6)

      # B. Generazione della variabile ausiliaria U
      # Qui si usa u scala M:  $u = M * \text{Uniform}(0,1)$ 
      u <- M * runif(1)

      # C. Test di Accettazione
      # La condizione teorica è:  $U_{\text{raw}} \leq f(y) / (M * g(y))$ 
      # Moltiplicando per M, diventa:  $u \leq f(y) / g(y)$ 
      if (u < dbeta(y, a, b) / dbeta(y, 2, 6)) done <- TRUE
    }
    # Se la condizione è vera, salviamo il candidato y nel vettore x
    x[i] <- y
  }

  # Report sull'efficienza dell'algoritmo (tasso di accettazione)
  if (report) cat("I needed ", ntry, " trials to get ", n, " samples.\n")

  return(x)
}

# 5. Esecuzione della simulazione
Nsim = 10^4
# Esempio di chiamata (nota: la riga è tagliata nell'immagine)
# campioni <- rbeta3(Nsim, a, b, M)

```

Example 2.4.2 (Penso esercizio)

So we have $f(x) \propto \exp\left(-\frac{x^2}{2}\right) [\sin(6x)^2 + 3 \cos(x)^2 \cdot \sin(4x)^2 + 1]$

1. Plot $f(x)$ to show $M \cdot g(x)$ bounds it where g_Y is the density of a Std. GAUSSIAN $g_Y(y) = \frac{1}{\sqrt{2\pi}} \cdot \exp\{-\frac{y^2}{2}\}$ and find M .
2. Generate 2500 numbers from f_X .
3. Deduce from the acceptance rate of the AR algorithm, an approximation of the normalizing constant.

```
UNNORMALIZED (you have proportional to in the text of the exer) fx <- function(x)
exp(-x^2/2) * (sin(6 * x)^2 + 3 * (cos(x)^2) * (sin(4 * x)^2) + 1)
fx.over.g <- function(x) fx(x)/dnorm(x)  if we have a differente propoasal dnorm is
gonna change
m <- optimize(fx.over.g, interval=c(-6,6), maximum = T) objectivecurve(fx, -9, 9, ylim =
c(-1, 7)) this is the density curve(m * dnorm(x), -5, 5, col = "red", add = T) curve(15 *
dnorm(x), -6, 6, col = "blue", add = T)
B AR.sampling <- function(n, f, g, rg, m, report=TRUE) x <- rep(0, n) tries <- 0 for(i
in 1:n) done <- FALSE while(!done)
y <- rg(1) u <- runif(1) * m if (u <= f(y) / (g(y)))
done <- TRUE tries <- tries + 1 x[i] <- y if(report) cat("Total tries:", tries, "")
return(list(x=x, tries = tries))
```