ANEXO 1:

Códigos feitos no Python usando os comandos do R

```
###### Método da aceitação-rejeição ######
# Objetivo: gerar valores de uma distribuição Beta(alfa=2, beta=2)
# Distribuição proposta:  Uniform(0,1) --> 1 + 0*x

%%R
curve(6*x*(1-x), 0, 1, col = 4, ylim = c(0, 2))
curve(1 + 0 * x, add = TRUE, lty = 2)
curve(1.5 * 1 + 0 * x, add = TRUE, lty = 2, lwd = 2)
legend("topright", legend = c("f(x)", "g(x)", "M g(x)"),
       lty = c(1, 2, 2), col = c(4, 1, 1), lwd = c(1, 1, 2), bty = "n")

%%R
## Criar os elementos necessários.
f <- function(x) 6*x*(1-x)
g <- function(x) 1 + 0 * x
M <- 1.5
x <- NULL

# Gerar y cuja densidade é g()
set.seed(1)
(y <- runif(n = 1, 0, 1))

# Gerar u de uma uniforme padrão
(u <- runif(n = 1))

# Calcular a razão entre as densidades
(r <- f(y)/(M * g(y)))

# Comparar e decidir
if (u < r) {
  x <- y
  print("u < r então valor aceito.")
} else {
  print("u >= r então valor descartado.")
}
%%R
# Verificar os pontos no gráfico:

curve(6*x*(1-x), 0, 1, col = 4)
curve(M*1 + 0 * x, add = TRUE, lty = 2, lwd = 2)
legend("bottomright", legend = c("f(x)", "g(x)", "M*g(x)"),
       lty = c(1, 2, 2), col = c(4, 1, 1), lwd = c(1, 1, 2), bty = "n")

points(y, f(y), pch = 19, col = 4)
text(y, f(y), "f(y)", pos = 4, col = 4)

points(y, M*g(y), pch = 19, col = 1)
text(y, M*g(y), "M g(y)", pos = 1, col = 1)

points(y, u*M*g(y), pch = 19, col = 2)
```

```
text(y, u*M* g(y), "u", pos = 4, col = 2)
%%R
# Definir um algoritmo que repete o processo acima para um número fixo de amostras
da distribuição proposta.
# Simula de uma única vez, com um valor fixo de simulações
Nsim <- 10000
set.seed(1)
# Amostra da proposta
y <- runif(Nsim, -1, 1)
# Amostra da U(0,1)
u <- runif(Nsim)
# Calcula a razão
r <- f(y)/(M * g(y))
# x será um vetor com os valores de y onde u < r
x <- y[u < r]
# Valores de u aceitos (apenas para o grafico)
ua <- u[u < r]
# Valores de u rejeitados (apenas para o grafico)
ur <- u[u >= r]
# Curvas
%%R
curve(6*x*(1-x), 0, 1, col = 4)
curve(M*1 + 0*x, add = TRUE, lty = 2, lwd = 2)
points(x, ua*M*g(x), col = 3)
points(y[u >= r], ur * M * g(y[u >= r]), col = 2)
%%R
# No R, podemos passar a função que desejamos maximizar para a função optimize()
(max <- optimize(f = function(x) {6*x*(1-x)},
                      interval = c(0, 1), maximum = TRUE))

# determinar o valor de M
(M <- max$objective/1)
%%R
# Proporção de pontos aceitos
length(x)/length(y)
%%R
# Taxa (teorica) de aceitacao
1/M
%%R
# Proporção de pontos rejeitados
length(ur)/length(u)
%%R
## Simula 1000 valores de f
N <- 1000L
x <- numeric(0)
while(length(x) < N) {
  y <- runif(1, 0, 1)
  u <- runif(1)
  r <- f(y)/(M * g(y))
  if(u < r) {
    ## Não é a forma mais eficiente!
    x <- c(x, y)
  }
}
```

```
length(x)
%%R
hist(x, freq = FALSE); lines(density(x), col = 2)
```

ANEXO 2:

Códigos feitos no Python usando os comandos do R

```
%load_ext rpy2.ipython

# Utilizando a dcauchy como distribuição proposta para obter a Beta(2,4)
%%R
alfa = 2
beta = 4
d.beta = function(x){(gamma(alfa+beta)*x^(alfa-1)*((1-x)^(beta-
1)))/(gamma(alfa)*gamma(beta))}

BetaCauchy  <- function(n){
  X = numeric(n)
  M = 7.1
  for(i in 1:n){
    Y = tan(pi*(runif(1)-1/2))
    while( runif(1) > d.beta(Y)/(M*dcauchy(Y))){
      Y = tan(pi*(runif(1)-1/2))
    }
    X[i] = Y
  }
  X
}
hist(BetaCauchy(1000000), 30, prob = T)
curve(d.beta(x), add = T, col = "red")
```

ANEXO 3:

Códigos feitos no Python usando os comandos do R

```
%load_ext rpy2.ipython

# Simular beta(2,4) usando a Cauchy
%%R
alfa = 2
beta = 4
d.beta = function(x){(gamma(alfa+beta)*x^(alfa-1)*((1-x)^(beta-
1)))/(gamma(alfa)*gamma(beta))}

%%R
curve(dcauchy(x),-1,1, lty = 3, ylim = c(0, 2.5))
curve(dcauchy(x),0,1, add = TRUE, lty = 5, col = "red")
curve(d.beta(x), add = TRUE, col = 4)

%%R
# Beta e Cauchy.
par(mfrow = c(1, 2))
curve(d.beta(x), 0, 1)
```

```
curve(dcauchy(x), add = TRUE, lty = 2)

# Razão entre Beta e Cauchy
curve(d.beta(x)/dcauchy(x), 0, 1)
abline(v = c(0, 1), lty = 2)
layout(1)
d.beta(1)/dcauchy(1)

%%R
M <- 7.1      ## Valor obtido no "olho" com base no gráfico da Razão entre Beta e C
auchy

## Taxa(teorica)de aceitacao:
1/M

# Visualização das curvas
%%R
curve(M * dcauchy(x), 0, 1,
      lty = 2,
      ylim = c(0, M * dcauchy(0)),
      ylab = "Densidade")
curve(d.beta(x), add = TRUE)
legend("topright",
       legend = c("f(x)", "M g(x)"),
       lty = c(1, 2),
       bty = "n")
```

ANEXO 4:

Códigos feitos no Python usando os comandos do R

```
%load_ext rpy2.ipython
%%R
install.packages("extRemes")
library(extRemes)
%%R
## Considere que deseja-se gerar valores de uma distribuição Beta(2,10)
# Distribuição proposta evd(x, 0.1,0.1)) ; evd = Extreme Value Distribution


alfa = 2
beta = 10
d.beta = function(x){(gamma(alfa+beta)*x^(alfa-1)*((1-x)^(beta-
1)))/(gamma(alfa)*gamma(beta))}

curve(d.beta(x), from = 0, to = 1, col = 4, ylim = c(0, 5))
curve(devd(x, 0.1, 0.1), add = TRUE, lty = 2)
legend("topright", legend = c("f(x) - Beta(2,10)", "g(x) - evd(0.1, 0.1))"),
       lty = c(1, 2), col = c(4, 1), bty = "n")
%%R
(M <- optimize(f = function(x) {d.beta(x)/devd(x, 0.1,0.1)},
               interval = c(0, 1), maximum = TRUE)$objective)
```

```
%%R
## Define funções
f <- function(x){(gamma(alfa+beta)*x^(alfa-1)*((1-x)^(beta-
1)))/(gamma(alfa)*gamma(beta))}
g <- function(x) devd(x, 0.1,0.1)

## Simula
Nsim <- 1000000
## Amostra da proposta
y <- revd(Nsim, 0.1,0.1)
## Amostra da U(0,1)
u <- runif(Nsim)
r <- f(y)/(M * g(y))
x <- y[u < r]
ua <- u[u < r]
ur <- u[u >= r]
%%R
curve(d.beta(x), from = 0, to = 1, col = 4, ylim = c(0, 5))
curve(M * devd(x, 0.1,0.1), from = 0, to = 1, add = TRUE, lty = 2, lwd = 2)
points(x, ua * M * g(x), col = 3)
points(y[u >= r], ur * M * g(y[u >= r]), col = 2)
%%R
## Quantos foram aceitados
length(x)/length(y)
%%R
## Taxa (teorica) de aceitacao é
1/M
%%R
plot(ecdf(x))
legend("right", legend = c("Empírica", "Teórica"),
       lty = 1, col = 1:5, bty = "n")
par(mfrow = c(1, 2))
```

ANEXO 5:

Códigos feitos no Python usando os comandos do R

```
%load_ext rpy2.ipython
%%R
# Considere que deseja-se gerar valores de uma distribuição Beta(2,2)
#  Queremos amostrar de uma Distribuição Minimax(1.5,1.5)

alfa = 2
beta = 2
d.beta = function(x){(gamma(alfa+beta)*x^(alfa-1)*((1-x)^(beta-
1)))/(gamma(alfa)*gamma(beta))}


teta = 1.5
gama = 1.5
d.minmax = function(x){teta*gama*x^{teta-1}*(1-x^teta)^{gama-1}}
r.minmax = function(x){   (1-(1-runif(x))^(1/gama))^(1/teta)}
```

```
# curvas
curve(d.beta(x), from = 0, to = 1, col = 4, ylim = c(0, 1.5))
curve(d.minmax(x), from = 0, to = 1, add = TRUE, lty = 2)
legend("topright", legend = c("f(x) - Beta", "g(x) - minmax"),
       lty = c(1, 2), col = c(4, 1), bty = "n")
%%R
(M <- optimize(f = function(x) {d.beta(x)/d.minmax(x)},
               interval = c(0, 1), maximum = TRUE)$objective)
M
%%R
curve(d.beta(x), from = 0, to = 1, col = 4, ylim = c(0, 1.6))
curve(d.minmax(x), from = 0, to = 1, add = TRUE, lty = 2)
curve(M*d.minmax(x), add = TRUE, lty = 2, lwd = 2)
legend("topright", legend = c("f(x)", "g(x)", "M*g(x)"),
       lty = c(1, 2, 2), col = c(4, 1, 1), lwd = c(1, 1, 2), bty = "n")
%%R
## Define funções
f <- function(x) d.beta(x)
g <- function(x) d.minmax(x)

## Simula
Nsim <- 1000000
## Amostra da proposta
y <- r.minmax(Nsim)
## Amostra da U(0,1)
u <- runif(Nsim)
r <- f(y)/(M * g(y))
x <- y[u < r]
ua <- u[u < r]
ur <- u[u >= r]
%%R
# pontos amostrados que foram aceitados (verde) e aqueles rejeitados (vermelho)
curve(d.beta(x), from = 0, to = 1, col = 4)
curve(M*d.minmax(x), from = 0, to = 1, add = TRUE, lty = 2, lwd = 2)
points(x, ua * M * g(x), col = 3)
points(y[u >= r], ur * M * g(y[u >= r]), col = 2)
%%R
## Quantos foram aceitados
length(x)/length(y)
%%R
## Taxa (teorica) de aceitacao é
1/M
# Gráficos
%%R
hist(x, freq = FALSE); lines(density(x), col = 2)
curve(d.beta(x), add = TRUE, from = 0, to = 1, col = 4)
```

ANEXO 6:

Códigos feitos no Python usando os comandos do R.

O código abaixo serve para gerar uma amostra por importância da distribuição Beta (alfa, beta).

A função de importância considerada aqui é de distribuição Minimax. Lembre-se que nesse caso, não é necessário que a função proposta seja maior do que a função alvo.

Fonte do conhecimento: http://cursos.leg.ufpr.br/ce089/07_MC_aprox. Ver também: http://cursos.leg.ufpr.br/ce089/06_MC_intro.html e http://cursos.leg.ufpr.br/ce089/07_MC_aprox.htmlhtml#22_Re-amostragem_por_import%C3%A2ncia

```
%load_ext rpy2.ipython
%%R
# Considere que deseja-se gerar valores de uma distribuição Beta(2,2)
# Queremos amostrar de uma Distribuição Minimax(1.5,1.5).

# Obs: para fazer análises adicionais de Minimax(1,1), Minimax(2,2), Minimax(3,1.5)
e Minimax(1.5,5), basta alterar os valores de teta e gama abaixo e, na sequência,
rodar o código.

alfa = 2
beta = 2
d.beta = function(x){(gamma(alfa+beta)*x^(alfa-1)*((1-x)^(beta-
1)))/(gamma(alfa)*gamma(beta))}

teta = 1.5
gama = 1.5
d.minmax = function(x){teta*gama*x^{teta-1}*(1-x^teta)^{gama-1}}
r.minmax = function(x){(1-(1-runif(x))^(1/gama))^(1/teta)}

%%R
curve(d.beta(x), from = 0, to = 1, ylim = c(0, 2), lty = 1, col = 1)
curve(d.minmax(x), from = 0, to = 1, add = TRUE, lty =2, col = 2)


legend("topright", legend = c("Beta", "Minmax"),
       lty = 1:2, col = 1:2, bty = "n")
# Valor teórico : Beta(beta, alfa)
%%R

E = alfa/(alfa+beta)
E

%%R
#### Integral de uma funcao de uma variavel
f = function(x){x*(gamma(alfa+beta)*x^(alfa-1)*((1-x)^(beta-
1)))/(gamma(alfa)*gamma(beta))} #obs: d.beta = function(x){(gamma(alfa+beta)*x^(alf
a-1)*((1-x)^(beta-1)))/(gamma(alfa)*gamma(beta))}
integrate(f, 0, 1)

%%R
curve(d.beta(x), from = 0, to = 1, ylim = c(0, 3), lty = 1, col = 1)
curve({1*1*x^{1-1}*(1-x^1)^{1-1}}, from = 0, to = 1, add = TRUE, lty =2, col = 2)
curve({1.5*1.5*x^{1.5-1}*(1-x^1.5)^{1.5-
1}}, from = 0, to = 1, add = TRUE, lty = 3, col = 3)
curve({2*2*x^{2-1}*(1-x^2)^{2-1}}, from = 0, to = 1, add = TRUE, lty = 4, col = 4)
```

```
curve({1.5*3*x^{1.5-1}*(1-x^1.5)^{3-
1}}, from = 0, to = 1, add = TRUE, lty = 5, col = 5)
curve({3*1.5*x^{3-1}*(1-x^3)^{1.5-
1}}, from = 0, to = 1, add = TRUE, lty = 6, col = 6)


legend("topright", legend = c("Beta(2,2)", "Minmax(1,1)", "Minmax(1.5,1.5)", "Minma
x(2,2)", "Minmax(1.5,3)", "Minmax(3,1.5)"),
       lty = 1:6, col = 1:6, bty = "n")

%%R
## Define funções
g <- function(x){(gamma(alfa+beta)*x^(alfa-1)*((1-x)^(beta-
1)))/(gamma(alfa)*gamma(beta))}  # d.beta(x)
f <- function(x){teta*gama*x^{teta-1}*(1-x^teta)^{gama-
1}}                                  # d.minmax(x)
## 1. Amostragem
m <- 1e6
x <- r.minmax(m)
## 2. Calcula pesos
w <- g(x)/f(x)
w.norm <- w/sum(w)
## 3. Re-amostragem por importância
k <- 1e5
## Amostra considerando os pesos normalizados para cada X
g.sample <- sample(x, size = k, replace = TRUE, prob = w.norm)
## Tamanho da amostra
length(g.sample)

%%R
# É possível ver porque o método funciona quando vemos um gráfico dos valoreas amos
trados de X com seus respectivos pesos:
curve(d.beta(x), from = 0, to = 1, col = 4,  ylim = c(0, 2))
curve(d.minmax(x), from = 0, to = 1, add = TRUE, lty = 2)
legend("right", legend = c("f(x)", "g(x)"),
       lty = c(1, 2), col = c(4, 1), bty = "n")
points(x, w)

%%R
# A obtenção de uma amostra da distribuição completa da VA de interesse possui dive
rsas finalidades. Por exemplo, no caso da beta, podemos obter um resumo

summary(g.sample)

%%R
# intervalo onde se encontram 95% dos dados:

quantile(g.sample, probs = c(.025, .975))

%%R
(I.chapeu = mean(g.sample))
I.chapeu

%%R
```

```
(erro.padrao.I = sd(g.sample)/sqrt(k))
erro.padrao.I

%%R
(IC = I.chapeu + qt(c(.025, .975), k-1)*erro.padrao.I)
IC
```