

Introdução à programação com a linguagem R

Felipe Rafael Ribeiro Melo
Departamento de Métodos Quantitativos - UNIRIO
(felipe.ribeiro@uniriotec.br)

II Seminário Internacional de Estatística com R
23 e 24 de maio de 2017
Niterói/RJ - Brasil

I - *Apresentação do Programa e Principais Comandos*

- O R começou a ser desenvolvido por Robert Gentleman e Ross Ihaka (Departamento de Estatística da Universidade de Auckland, na Nova Zelândia), mais conhecidos por “R & R”, apelido do qual originou-se o nome do programa.
- O objetivo inicial de “R & R”, em 1991, era produzir um *software* para as suas aulas de laboratório baseado na já revolucionária linguagem S, utilizada pelo software comercial S-Plus (criado por John M. Chambers, da AT&T).

- O primeiro relato da distribuição do R foi em 1993, quando algumas cópias foram disponibilizadas no StatLib, um sistema de distribuição de softwares estatísticos.
- Com o incentivo de um dos primeiros usuários deste programa, Martin Mächler (do Instituto Federal de Tecnologia de Zurique, na Suíça), “R & R”, em 1995, lançaram o código fonte do R, disponível por *ftp*.
- Em 1997, foi formado um grupo de profissionais que têm acesso ao código fonte do R, possibilitando assim a atualização mais rápida do *software*. Desde então, o R vem ganhando cada vez mais adeptos em todo o mundo.

Vantagens do R


- Além de gratuito, é um programa potente, estável e pode ser copiado e distribuído sem nenhum problema.
- É um programa que tem uma longa história, com mais de 20 anos de desenvolvimento.
- É apoiado por uma grande equipe de desenvolvedores em todo o mundo.
- Pode ser usado nos sistemas operacionais Windows, Linux e Mac OS.
- Amplamente utilizado no meio acadêmico.

Download e Instalação

- O *software* R é gratuito e pode ser obtido em <http://www.r-project.org>.
- Nesta página, selecione CRAN (lado esquerdo da tela do seu navegador), e opte por um dos *mirrors* disponíveis.
- Logo após, escolha o seu sistema operacional no campo *Download and Install R*, o link *base* e, finalmente, selecione o link em destaque para baixar o programa.
- Uma vez realizado o *download*, o processo de instalação do programa é simples, bastando apenas dar um duplo-clique no arquivo executável e seguir os passos do assistente de instalação.

Interface do Programa

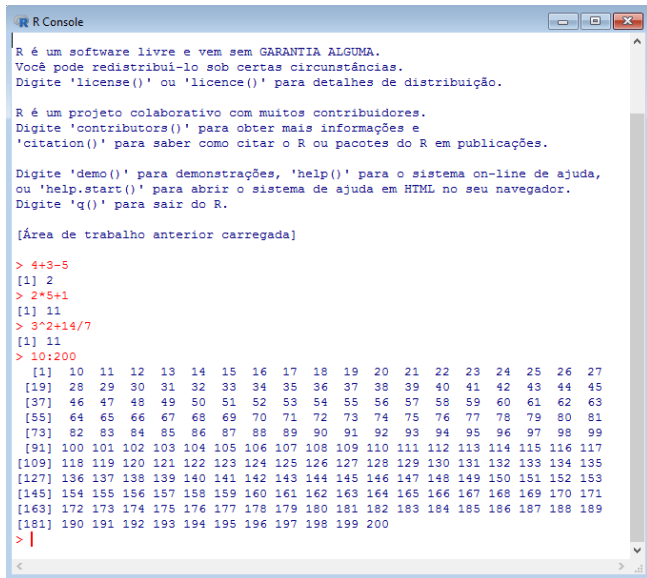
- Na janela do R (*R Console*), é exibida uma série de créditos na cor azul e, em seguida, o sinal `>` na cor vermelha.
- Este símbolo é chamado ***prompt de comando***, e significa que o R está apto a receber um ***comando*** nesta linha, ou seja, uma linha de comando.
- Note que não nos deparamos com planilhas e tampouco com menu de opções para realização de fórmulas e gráficos. Aqui, tudo é programável. Algumas vantagens disto são geração e armazenamento de diversas informações, uma vastidão de comandos possíveis e maior flexibilidade nestes comandos.

- Como mencionado anteriormente, a linha indicada pelo sinal `>` está pronta para receber qualquer tipo de comando (como, por exemplo, uma expressão com operações aritméticas). Para executar um comando (isto é, obter o resultado proveniente do dado de entrada digitado na linha do *prompt*), basta digitá-lo e apertar a tecla . O resultado do comando é o que chamamos de *output* ou dado de saída, exibido na cor azul.

Exemplo 1.1

Execute no R as quatro linhas de comando a seguir:

```
4+3-5 ; 2*5+1 ; 3^2+14/7 ; 10:200.
```

```
R Console

R é um software livre e vem sem GARANTIA ALGUMA.
Você pode redistribuí-lo sob certas circunstâncias.
Digite 'license()' ou 'licence()' para detalhes de distribuição.

R é um projeto colaborativo com muitos contribuidores.
Digite 'contributors()' para obter mais informações e
'citation()' para saber como citar o R ou pacotes do R em publicações.



Digite 'demo()' para demonstrações, 'help()' para o sistema on-line de ajuda,
ou 'help.start()' para abrir o sistema de ajuda em HTML no seu navegador.
Digite 'q()' para sair do R.

[Área de trabalho anterior carregada]



> 4+3-5
[1] 2
> 2*5+1
[1] 11
> 3^2+14/7
[1] 11
> 10:200
 [1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
[19] 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
[37] 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
[55] 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
[73] 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
[91] 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
[109] 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135
[127] 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153
[145] 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171
[163] 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189
[181] 190 191 192 193 194 195 196 197 198 199 200
> |
```

Figura 1: Executando linhas de comando do Exemplo 1.1

- A primeira expressão apresenta operações de soma (+) e subtração (-). Na expressão seguinte, temos multiplicação (*) e soma, com a prioridade dada para a multiplicação. Analogamente, na terceira expressão, são realizadas potenciação (^), divisão (/) e soma, exatamente nesta ordem.
- Para expressões nas quais desejarmos mudar as prioridades, devemos usar parênteses – e *nunca* colchetes ou chaves – como delimitadores (a rigor, as chaves podem ser utilizadas, mas não é recomendável).
- Por exemplo, se desejarmos somar 5 com 1 e depois multiplicar o resultado por dois, devemos fazer $2 * (5 + 1)$, e não $2 * [5 + 1]$.

- Na quarta expressão, temos a sequência de números inteiros de 10 a 200. Repare que o número entre colchetes indica a posição do primeiro elemento da linha do *output*. Como o dado de saída desta operação possui vários elementos, por vezes é útil saber a posição dos elementos obtidos no resultado (por exemplo: 82 é o 73^o termo da sequência, 136 é 127^o termo da sequência,...).
- Notas:
 - (i) Na atual linha de comando, é possível acessar comandos já executados utilizando as setas  e  do teclado.
 - (ii) Para executar um comentário no R (ou seja, uma sequência de caracteres que deve ser desprezada como dado de entrada), basta digitar o símbolo # e, em seguida, o comentário.

Novo *Script*

- Um *Script* é uma janela diferente da janela *R Console* que serve para “rascunharmos” (e registrarmos) as linhas de comando de interesse.
- Para abrir um novo *script*, vá em *Arquivo* (na barra de ferramentas do R) e selecione *Novo Script*.
- Qualquer linha ou sequência de linhas de comando no *script* pode ser executada diretamente do *script*: basta selecioná-las e pressionar  +  (no caso de apenas 1 linha, basta deixar o cursor sobre ela, não havendo necessidade de selecioná-la por completo).

Exemplo 1.2

Na janela do *script*, digite

```
# Exemplo de comentário:  
2 + log(1) + exp(0) # usando logaritmo e exponencial  
sqrt(2) # usando raiz quadrada (square root)  
2+5+3  
2+5#+3
```

e execute estas linhas de comando conforme explicado acima.

Observação: Para dividir elegantemente a janela *R Console* e a janela do *script*, clique em *Janelas* (na barra de ferramentas) e selecione alguma das quatro opções possíveis. Sugestão particular: *Dividir Lado a lado (Tile Vertically)*.

Objetos

- São os “rótulos” que daremos para as nossas expressões.
- Os nomes dos objetos devem começar com uma letra (pode inclusive ser apenas uma letra), e pode conter letras, números, pontos, *underline* (não são permitidos vírgula, ponto e vírgula, dois pontos e espaço).
- Para “rotular” uma expressão em um objeto, devemos digitar na seguinte forma:

objeto <- expressão

Atualmente, o R também aceita o sinal de igual no lugar do **operador de atribuição** <- , produzindo o mesmo resultado.

Exemplo 1.3

```
y <- 4 # lê-se "y recebe 4"
y
x.A <- 2/3
x.B = 0.5 # mesmo resultado de x.B <- 0.5
x.A + x.B
y == 3*x.A + 4*x.B # avalia se expressões são iguais
y != 3*x.A + 4*x.B # avalia se expressões são diferentes
alpha = 0.35
a = cos(alpha)
b = sin(alpha)
(a^2) + (b^2) # famosa relação trigonométrica
```

- É recomendável atribuir nomes que tenham um significado lógico. Isto facilita lidar com um grande número de objetos.
- Letras maiúsculas e minúsculas são consideradas diferentes para o R.
- O comando `ls()` lista todos os objetos da área de trabalho.
- O comando `rm()` remove os objetos, que devem ser especificados dentro dos parênteses (separados por vírgulas).

Exemplo 1.4

```
ls()  
rm(x.A,x.B)  
ls() # não lista mais os objetos x.A e x.B  
x.A # não reconhecerá mais x.A  
a # permanece intacto
```

- Um objeto pode conter não apenas um valor escalar, mas também um vetor, uma matriz, um *data frame*...
- Ainda, objetos podem incorporar resultados provenientes de comandos que geram resultados aleatórios (ou, genericamente, de alguma distribuição de probabilidades).

Exemplo 1.5

Execute as seguintes linhas

```
# Sorteio de 1 número do conjunto {1,2,...,10}
amostra = sample(1:10,1)
amostra
amostra
amostra
```

e observe que os resultados obtidos são iguais. Agora, execute três vezes a sequência de linhas

```
# Sorteio de 1 número do conjunto {1,2,...,10}
amostra = sample(1:10,1)
amostra
```

e observe os – com alta probabilidade – diferentes resultados obtidos.

Criando Vetores

- O comando geral utilizado para criação de vetores é

c(entradas do vetor separadas por vírgulas)

Exemplo 1.6

```
c(3,7,2,12)
c("Luiz","Maria","Rafael") # vetor de nomes
10:30 # também é um vetor
30:10 # ordem decrescente
v = c(7,9)
c(v,11,3)
v[1] # 1ª entrada do vetor (7,9)
c(v,11,3)[3] # 3ª entrada do vetor (7,9,11,3)
c(v,11,3)[-2] # vetor (7,9,11,3) sem a 2ª entrada
```

Comandos que geram sequências e repetições

Gerando Sequências: Comando *seq*(`_` , `_` , `_`)

- O comando *seq* possui três argumentos, e gera uma sequência do primeiro argumento até o segundo argumento, com salto entre valores consecutivos de acordo com o terceiro argumento.
- Para sequências decrescentes, o terceiro argumento obrigatoriamente deverá vir acompanhado do sinal negativo.
- O término da sequência será o segundo argumento ou algum número anterior a este na sequência, a depender do valor do terceiro argumento.

- O comando `seq` pode ser escrito apenas com os dois primeiros argumentos. Neste caso, o terceiro argumento será, por padrão, igual a 1 (para sequências crescentes) ou -1 (para sequências decrescentes).

Exemplo 1.7

```
seq(10,22,3)
seq(10,22,5)
seq(22,10,-3)
seq(22,10,-5)
seq(3,8) # 3<8, logo 3º argumento implícito = 1
3:8 # gera o mesmo resultado acima
seq(8,3) # 8>3, logo 3º argumento implícito = -1
8:3 # gera o mesmo resultado acima
```

Gerando Repetições: Comando *rep*(,)

- Primeiro argumento: expressão a ser repetida.
- Segundo argumento: número de repetições.

Exemplo 1.8

```
rep(0,6) # repetição de um escalar  
rep(c(1,7),4) # repetição de um vetor  
rep(seq(-2,1,0.5),2) # repetição de uma sequência  
rep(c(1,3,5),c(3,2,1)) # gera vetor (1,1,1,3,3,5)
```

Outros comandos interessantes aplicados à vetores

```
A = c(1,3,5,7,9)
```

```
B = A-1
```

```
C = 0:9
```

```
union(A,B) # união de A e B
```

```
intersect(A,B) # intersecção de A e B
```

```
intersect(B,C)
```

```
setdiff(C,A) # operação  $C \setminus A$ 
```

```
setequal(A,B) # avalia igualdade
```

```
setequal(A,setdiff(C,B))
```

```
w = c(5,3,14,8,3)

rev(w) # reverte a ordem dos componentes de w

# Componentes de w em ordem crescente:
sort(w)

# Componentes de w em ordem decrescente:
sort(w, decreasing=TRUE)

min(w) # menor valor em w
max(w) # maior valor em w

sum(w) # somatório dos componentes de w
prod(w) # produtório dos componentes de w
length(w) # tamanho do vetor w (número de entradas)

unique(w) # elementos distintos de w
```


Criando Matrizes

- O comando geral utilizado para criação de matrizes é

```
matrix(entradas, nrow, ncol, byrow=)
```

Exemplo 1.9

```
x = 1:16
A1 = matrix(x,4,4) # ou matrix(x,4,4,byrow=F)
A2 = matrix(x,4,4,byrow=T)
A1
A2
matrix(2,5,3) # matriz 5x3 com entradas 2
matrix(x,ncol=8) # definindo apenas n° de colunas
diag(4,6) # matriz diagonal 6x6 com entradas 4
diag(c(9,5,11)) # matriz (3x3) diagonal
diag(c(9,5,11),8) # matriz diagonal 8x8
```

Operações com matrizes

`A1 + A2`

`A1 - A2`

`A1 * A2` # multiplicação `a1_ij * a2_ij`

`A1 %*% A2` # multiplicação matricial

`A1 / A2` # divisão `a1_ij / a2_ij`

`det(A1)` # determinante de A1

`t(A1)` # matriz transposta de A1

`y = c(1:4,rep(0,3),rep(5,2))`

`B = matrix(y,3,3)`

`solve(B)` # matriz inversa de B

`P = matrix(rep(c(0.5,0.2,0.3),3),3,3,byrow=T)`

`eigen(P)`

`eigen(P)$values` # autovalores de P

`eigen(P)$vectors` # autovetores de P

- **Concatenação de matrizes:** Comandos *cbind* (concatenação horizontal) e *rbind* (concatenação vertical).
- O comando *cbind* concatena apenas matrizes com o mesmo número de linhas; e o comando *rbind* concatena apenas matrizes com o mesmo número de colunas.

Exemplo 1.10

```
P
B
cbind(P,B)
rbind(P,B)

Q = matrix(1:12,3,4)
cbind(P,Q) # ambas possuem 3 linhas
rbind(P,t(Q)) # ambas possuem 3 colunas
```

Extraindo entradas/linhas/colunas/submatrizes de uma matriz

```
A = matrix(1:16,4,4,byrow=T) # A = A2 criada antes
```

```
A[2,4] # entrada na 2ª linha e 4ª coluna de A
```

```
A[3,] # 3ª linha de A
```

```
A[,3] # 3ª coluna de A
```

```
A[-3,] # matriz A sem a 3ª linha
```

```
A[,-3] # matriz A sem a 3ª coluna
```

```
A[1:3,] # submatriz de A
```

```
A[,2:4] # submatriz de A
```

```
A[1:3,2:4] # submatriz de A
```

```
A[c(1,3,4),c(2,4)] # submatriz de A
```


- Um *data frame* possui estrutura ideal para comportar conjuntos de dados. Cada coluna possui um nome, que podem ser vistos como os nomes das variáveis.

Exemplo 1.11

```
data.frame(v1=c(3,-1,5),v2=c(0,10,30))

Turma = data.frame(Alunos = c("Bia","Thaís","Luca","Leo"),
  Notas=c(7.0,8.0,9.5,6.5), Faltas = c(2,0,0,1))

Turma
attach(Turma) # para R reconhecer os nomes das colunas
mean(Notas) # média das notas
max(Faltas) # maior número de faltas
detach(Turma) # cancela o attach(Turma)
```

- Para ajuda sobre comandos do R, basta digitar o nome da comando precedido de uma interrogação na janela R Console e apertar . Para fins gerais, usam-se duas interrogações em vez de uma, ou ainda o comando

`help.search("nome para a busca")`

Exemplo 1.12

```
# Qual comando retorna o desvio padrão?  
?deviation # nada encontrado  
??deviation  
help.search("deviation")  
?sd
```

Pacotes (*Packages*)

- Alguns comandos e funcionalidades mais sofisticados no R requerem o carregamento de **pacotes**.
- Caso o usuário já possua determinado pacote instalado e queira utilizar comandos deste pacote, há duas formas de carregá-lo:
 - 1 na barra de ferramentas do R, selecionar *Pacotes* > *Carregar Pacote* e, na janela aberta em seguida (com todos os pacotes instalados), selecionar o pacote desejado; ou
 - 2 executar o comando `library(nome do pacote)` ou o comando `require(nome do pacote)`.

- Entretanto, caso o pacote de interesse não esteja **instalado**, não será possível carregá-lo. Logo precisamos **instalá-lo** para, em seguida, carregá-lo conforme mencionado anteriormente.
- Tal instalação é frequentemente feita diretamente da internet, por meio do menu *Pacotes > Instalar pacote(s)* da barra de ferramentas do R. Após acessar este menu, deve-se selecionar um dos *mirrors* disponíveis (sugestão: *0-Cloud [https]*) para então selecionar o(s) pacote(s) desejado(s).
- Alternativamente, é possível instalar pacotes por meio da linha de comando

```
install.packages("nome do pacote")
```


II - *Importação/Exportação de planilhas*

Importando planilha de arquivo *.xls/.xlsx*

- Suponha que desejemos fazer com que o R entenda uma planilha construída no *Excel*, conforme a figura abaixo. Ainda suponha que tal planilha é a primeira deste arquivo (*Plan1*) e este foi salvo com o nome *teste.xls*.

	A	B	C	D	E
1	Nome	Peso	Altura		
2	Pedro	90	1.92		
3	Marcos	70	1.76		
4	Bruno	65	1.67		
5	Diego	80	1.84		
6	Tiago	85	1.80		
7					
8					
9					

Figura 2: Pequeno conjunto de dados na *Plan1* do arquivo *teste.xls*

- Para a importação de planilhas em qualquer formato, primeiramente vamos “adiantar” ao R o caminho completo da pasta está o arquivo que desejamos importar. Isto é feito por meio da linha de comando

```
setwd("caminho da pasta onde está o arquivo")
```

- Em particular, para planilhas em arquivos nos formatos *.xls* e *.xlsx*, usaremos o comando **read_excel**, disponível no pacote **readxl**. Tal pacote pode ser instalado e carregado executando as seguintes linhas de comando:

```
install.packages("readxl")  
library(readxl)
```

Exemplo 2.1

Importando a planilha exibida no Slide 34 (primeira planilha do arquivo *teste.xls*), salva na Área de Trabalho (*Desktop*) do computador cujo nome de usuário é “Felipe Rafael”:

```
setwd("C:/Users/Felipe Rafael/Desktop")
perfil = read_excel("teste.xls",1,col_names=T)
perfil
```

- Na primeira linha de comando acima, note que as barras invertidas foram substituídas por barras (também é comum substituir cada barra invertida por `\\`).
- O número 1 no segundo argumento do comando `read_excel` indica em qual planilha do arquivo encontra-se o que desejamos importar (neste caso, a 1ª planilha).

- O terceiro argumento em `read_excel` refere-se ao fato de que a primeira linha da planilha contém os nomes das variáveis (caso contrário, devemos usar `col_names=F`).

Exemplo 2.2

```
# Explorando o conjunto de dados perfil:
perfil$Peso # entradas da coluna "Peso"
Peso # R não reconhece
attach(perfil) # R reconhecerá nomes das colunas
Peso

sum(Peso) # somatório dos pesos
mean(Altura) # média das alturas
sd(Altura) # desvio padrão das alturas

IMC = Peso/(Altura^2)
cbind(perfil,IMC)
```

Importando dados em outros formatos

- Cada formato de arquivo possui um comando específico para importação. Listaremos aqui alguns dos mais utilizados. Para aplicação correta de todos os comandos de importação nos slides a seguir, cabe lembrar que é necessário informar anteriormente ao R o caminho completo da pasta na qual está o arquivo que será importado por meio do comando:

```
setwd("caminho da pasta onde está o arquivo")
```

(é necessário substituir cada barra invertida no caminho completo da pasta por uma barra ou por um par de barras invertidas).

- **Arquivos no formato *.txt*** (supondo que a primeira linha do arquivo contém o nome das colunas, é usado ponto e vírgula como separador de campos e ponto como separador de decimal):

```
read.table("nome do arquivo.txt", header=T,  
           sep=";", dec=".")
```

- **Arquivos no formato *.csv*** (supondo que é usado ponto e vírgula como separador de campos e ponto como separador de decimal):

```
read.csv("nome do arquivo.csv", sep=";", dec=".")
```

- **Arquivos no formato *.dat*** (necessário carregar o pacote `foreign` antes da importação):

```
read.dta("nome do arquivo.dta")
```

Salvando e carregando arquivos de dados

- Para salvar o objeto *perfil* (criado no Slide 36) com o nome *teste.RData* (*.RData* é a extensão de arquivos de dados do R) na mesma pasta configurada pelo comando `setwd`, devemos executar a linha de comando

```
save("perfil",file="teste.RData")
```

- Para carregar o conjunto de dados *teste.RData* (e ter acesso ao objeto *perfil*), é necessário informar o caminho completo da pasta onde está *teste.RData* com o comando `setwd` conforme visto anteriormente, e logo após executar a linha de comando

```
load("teste.RData")
```


Exportando dados para o Bloco de Notas

- Uma forma de exportar o conjunto de dados no objeto *perfil* (criado no Slide 36) para o formato *.txt* (com o nome, digamos, *teste.txt*) é usar o comando `write.table`:

```
write.table(perfil,"teste.txt",sep=" ", quote=F,  
            row.names=F)
```

(o arquivo *teste.txt* será salvo na mesma pasta configurada pelo comando `setwd`).

- `sep=" "` configura um simples espaço como separador de campos numa mesma linha, `quote=F` inibe a geração de aspas em torno dos nomes e `row.names=F` inibe a numeração de cada linha.

III - *Probabilidade e Estatística no R*

Distribuições de Probabilidades

- Temos quatro funções aplicadas às distribuições de probabilidades:

d: calcula a **função densidade de probabilidade** (caso contínuo) ou **função de probabilidade** (caso discreto) aplicada em um ou mais valores.

p: calcula a **função de distribuição acumulada** aplicada em um ou mais valores.

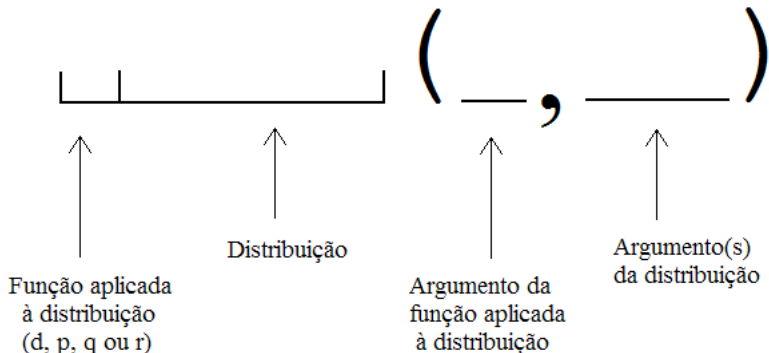
q: calcula **quantil** correspondente a uma ou mais probabilidades.

r: extrai **amostra** da distribuição.

Tabela 1: Algumas distribuições de probabilidades no R

Distribuição	Notação	Argumentos com <i>default</i>
Binomial	binom	n= p=
Geométrica	geom	prob=
Binomial negativa	nbinom	size= prob=
Hipergeométrica	hyper	m= n= k=
Poisson	pois	lambda=
Uniforme	unif	min=0 max=1
Exponencial	exp	rate=1
Gama	gamma	shape= rate=1
Normal	norm	mean=0 sd=1
t de Student	t	df=
Qui-Quadrado	chisq	df=
F de Snedecor	f	df1= df2=
Beta	beta	shape1= shape2=

- Mas como usar as funções **d**, **p**, **q** e **r** nestas distribuições?



Exemplo 3.1

```
dexp(0,3) # densidade da Exp(3) no ponto 0
dexp(0) # densidade da Exp(1) no ponto 0
pnorm(2,5,3) # acumulada da N(5;9) no ponto 2
pnorm(1.96) # acumulada da N(0;1) no ponto 1.96
qnorm(0.975) # quantil 97,5% da N(0;1)
rnorm(15,5) # amostra de tamanho 15 da N(5;1)
rnorm(15,sd=5) # amostra de tamanho 15 da N(0;25)

ppois(c(10,20,30),12) # acumuladas da Poisson(12)
dnbinom(c(5,12),10,0.6) #fçs de prob da BN(10;0.6)
qgamma(c(0.25,0.75),5,2) # quantis da Gama(5;2)

rnorm(7,c(0,10,100),c(2,2,2))
dexp(0,c(2,3,5))
pbeta(0.5,c(2,4),c(2,6))
```

Gráficos de distribuições de probabilidades

- As linhas a seguir mostram uma forma de construir gráficos de distribuições no **caso contínuo** com o comando `curve` (seja o gráfico da função densidade de probabilidade, da função de distribuição acumulada ou da função quantil). Como as janelas de gráficos no R se sobrepõem, execute uma linha de cada vez.

```
# Densidade da  $N(100;64)$ , de  $x=60$  até  $x=140$ :  
curve(dnorm(x,100,8),60,140)
```

```
# F.d.a. da  $N(100;64)$ , de  $x=60$  até  $x=140$ :  
curve(pnorm(x,100,8),60,140)
```

```
# Função quantil da  $N(100;64)$ , de  $x=0$  até  $x=1$ :  
curve(qnorm(x,100,8),0,1)
```

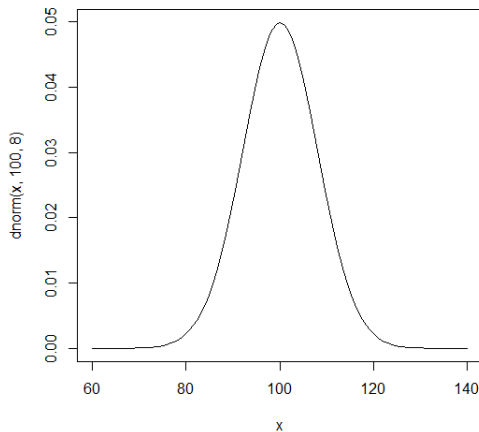


Figura 3: Densidade de uma variável aleatória $N(100; 64)$

Incluindo formatação

Exemplo 3.2

```
# Densidade da  $N(0;1)$  com título:  
curve(dnorm,-3.5,3.5, main=  
      "Distribuição Normal Padrão\n $X \sim N(0,1)$ ")
```

Exemplo 3.3

```
curve(dnorm(x,100,8),60,140,  
      xlab="valores de x", ylab="densidade de x")  
curve(dnorm(x,90,8),60,140,add=T,col=2)  
curve(dnorm(x,100,15),60,140,add=T,col=3)  
legend("topright",c("N(100;64)", "N(90;64)",  
                    "N(100;225)"), fill=c(1,2,3))  
title("Várias curvas da distribuição Normal")
```

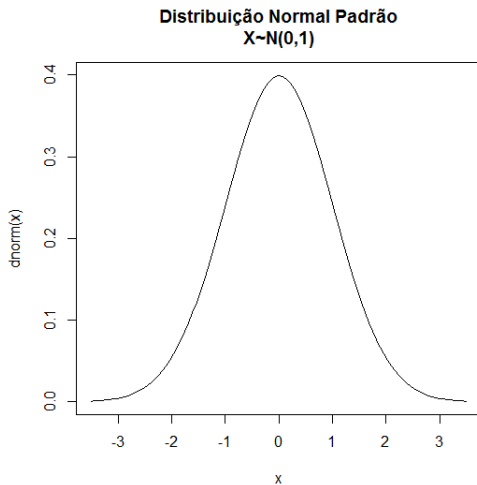


Figura 4: Densidade de uma variável aleatória Normal Padrão

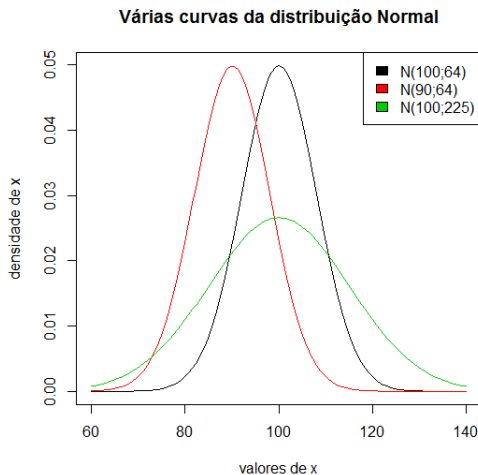


Figura 5: Várias densidades na família Normal, conforme legenda

- Para o **caso discreto**, ilustramos como obter gráficos de função de probabilidade e função de distribuição acumulada de uma variável aleatória com distribuição binomial de parâmetros $n=10$ e $p=0,4$ por meio do comando `plot`:

```
x = 0:10
```

```
px = dbinom(x,10,0.4)
```

```
plot(x,px,type="h") # gráfico estilo haste
```

```
Fx = pbinom(x,10,0.4)
```

```
plot(x,Fx,type="s") # gráfico estilo escada
```

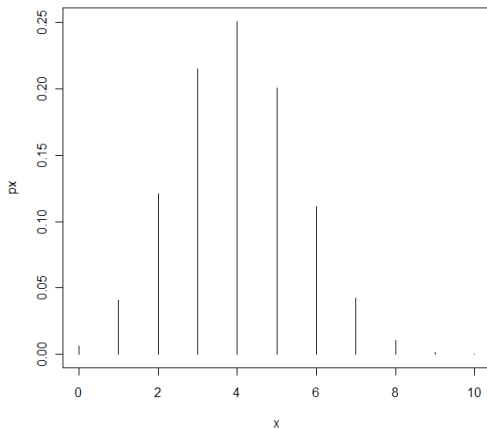


Figura 6: Função de probabilidade de uma v.a. $Bin(10; 0,4)$

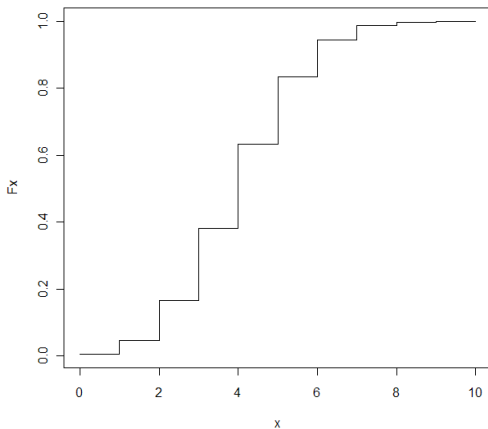


Figura 7: Função de distribuição acumulada de uma v.a. $Bin(10; 0,4)$

Comando `sample`

- O comando `sample` gera amostra (com ou sem reposição) proveniente de um vetor de dados (numérico ou não).
- Possui três argumentos:
 - 1º argumento: população (da qual será extraída a amostra);
 - 2º argumento: número de extrações (i.e., o tamanho da amostra);
 - 3º argumento: reposição (T, para amostras com reposição; e F, para amostras sem reposição).

Exemplo 3.4

```
sample(0:10,4,F)

# Extraíndo bolas de uma urna:

urna = rep(c("branca","preta","vermelha"),c(5,7,13))
urna
sample(urna,6,F)
sample(urna,6,T)

# Sorteio da Mega-sena:

dezenas = 1:60
sorteio = sample(dezenas,6,F)
sorteio
```


- A ausência da distribuição Uniforme discreta no quadro de distribuições visto anteriormente não é uma perda potencial, pois a geração de amostras da distribuição uniforme discreta é facilmente executada pelo comando `sample`.
- Por exemplo, para simular 10 lançamentos (independentes) de um dado equilibrado, devemos executar o comando: `sample(1:6, 10, T)`.
- Além disto, o comando `sample` pode ser utilizado com uma distribuição personalizada ao adicionar um quarto argumento, que denota a probabilidade de cada elemento na população ser sorteado. Para que isto faça sentido, devemos utilizar amostras *com* reposição.

Exemplo 3.5

#20 arremessos independentes de uma moeda viciada:

```
sample(c("cara","coroa"),20,T,c(0.75,0.25))
```

9 ligações indep's para uma central telefônica:

```
sample(c("tel. livre","tel. ocupado"),9,T,  
       c(0.8,0.2))
```

Outros comandos usuais em Estatística

```
am1 = sample(1:6,40,T)
am2 = rbinom(40,5,0.5) + 1
am1
am2

# Distribuição de frequências (absolutas) de am1:
table(am1) # distrib de frequências (absolutas) de am1

# Distribuição de frequências (relativas) de am1:
table(am1)/sum(table(am1))
prop.table(table(am1)) # alternativa ao comando acima

# Tabela de contingência entre am1 e am2 (pareado):
table(am1,am2)
```

```
mean(am2) # média de am2
median(am2) # mediana de am2
quantile(am2,0.8) # quantil 80% de am2
summary(am2) # medidas de posição de am2

sd(am2) # desvio-padrão de am2
var(am2) # variância de am2
IQR(am2) # distância interquartil de am2

cov(am1, am2) # covariância entre am1 e am2 (pareado)

# Coeficiente de correlação entre am1 e am2 (pareado):
cor(am1,am2) # padrão é coef. de correl. de Pearson
cor(am1,am2, method="spearman") # Spearman
```

IV - *Gráficos no R*

Comando plot

- Seus únicos argumentos obrigatórios são dois vetores de mesmo tamanho (o primeiro associado ao eixo horizontal e o segundo associado ao eixo vertical), gerando assim pares ordenados.

Exemplo 4.1

```
x = 1:20
y = x^3
plot(x,y) # default é type="p" (pontos)
plot(x,y,type="b") # linha entre os pontos
plot(x,y,type="c") # segmentos que aumentam
plot(x,y,type="h") # haste
plot(x,y,type="l") # linha
plot(x,y,type="o") # linha por cima dos pontos
plot(x,y,type="s") # escada
```

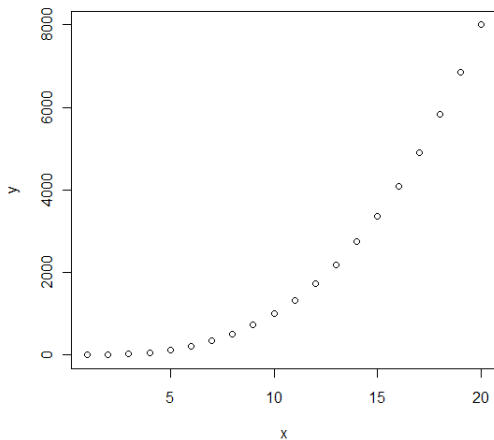


Figura 8: Gráfico gerado pelo comando `plot(x,y)` do Exemplo 4.1

Cores nos gráficos

- Para gráficos com cores específicas, usamos no comando `plot` o argumento `col="cor de sua preferência"`. Ainda, há uma codificação numérica para poucas cores no R que pode ser usada no lugar do nome da cor (sem aspas).
- Código numérico de cores do R: 1-preto; 2-vermelho; 3-verde; 4-azul; 5-ciano; 6-magenta; 7-amarelo; e 8-cinza. A sequência de cores é a mesma de 9 a 16, de 17 a 24, e assim por diante. Além disto: 0-branco.

Exemplo 4.2

```
plot(x,y,col="blue") # contornos em azul  
plot(rev(x),y,col=7) # contornos em amarelo  
plot(x,y,type="l",col=10) # linha em vermelho
```


- Felizmente, o R possui (muito) mais que 8 cores para gráficos. Uma forma de listar todas as cores disponíveis no R é executar o comando `colors()`.

Exemplo 4.3

```
plot(x,y,col="orange") # contornos na cor laranja  
plot(rev(x),y,col="plum") # contornos em ameixa  
plot(x,y,type="l",col="green") # linha verde
```

- **Observação:** O argumento `col` também pode ser utilizado em outros comandos gráficos além de `plot`.

Sobrepondo gráficos

- É possível sobrepor gráficos (em um gráfico gerado pelo comando `plot`) com o uso de comandos como `points`, `lines` e `abline`.

Exemplo 4.4

```
plot(x,y,col="orange")
```

```
points(rev(x),y,col="plum")
```

```
lines(x,y,col="green")
```

```
abline(5,200) # equação da reta  $f(u) = 5u + 200$ 
```

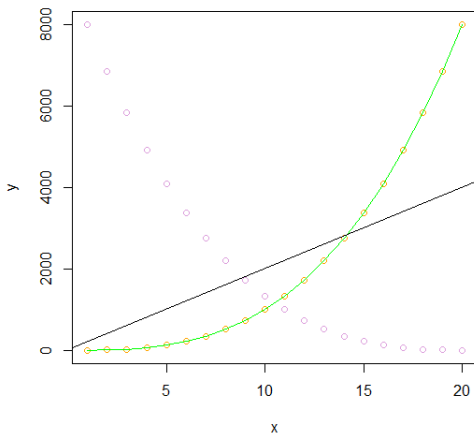


Figura 9: Gráficos obtidos com os comandos do Exemplo 4.4

Formatando pontos

Exemplo 4.5

```
plot(x,y,pch=3) # estilo dos pontos  
points(rev(x),y,pch=6)  
points(x,8000-y,pch="6") # atente às aspas!
```

Exemplo 4.6

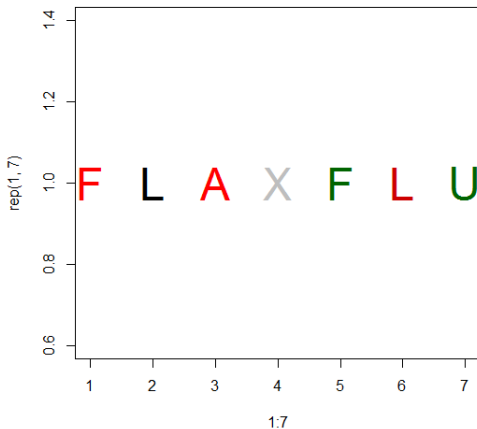
```
plot(x,y,pch=c("2","7","4"))  
points(rev(x),y,pch=c(2,7,4))
```

Exemplo 4.7

```
plot(x,y,cex=3) # tamanho dos pontos  
# Colorindo "interior do ponto"(com pch=21):  
points(rev(x),y,pch=21,bg="red")
```

Exemplo 4.8

```
plot(1:7,rep(1,7),cex=3,pch=c("F","L","A","X","F","L",  
"U"),col=c(2,1,2,8,"darkgreen","red3","darkgreen"))
```



Formatando linhas

- Argumentos `lwd=` para a espessura da linha (vale também para pontos); e `lty=` para o estilo da linha.

Exemplo 4.9

```
plot(x,y,lwd=3,col="red") # pontos "espessos"  
lines(x,8000-y,lwd=4) # linha ainda mais espessa  
lines(rev(x),y,lty=2) # linha interrompida
```

Exemplo 4.10

```
plot(x,y,type="l", lty=4,lwd=3)  
points(rev(x),y,lwd=2,col="khaki")
```

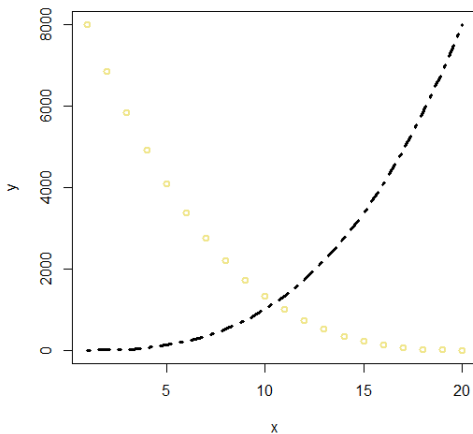


Figura 10: Gráficos obtidos por meio do Exemplo 4.10

Editando eixos

- Os argumentos `xlab` e `ylab` editam os rótulos dos eixos horizontal e vertical, respectivamente, e os argumentos `xlim` e `ylim` definem onde começam e terminam os eixos.

Exemplo 4.11

```
# Editando rótulos associados aos eixos:
plot(x,y,xlab="valores de x",ylab="valores de x ao cubo")

# Editando tamanho do eixo horizontal:
plot(x,y,xlim=c(-5,25)) # eixo-x de -5 até 25

# Editando tamanho do eixo vertical:
plot(x,y,ylim=c(0,10000)) # eixo-y de 0 até 10000

# Editando tamanho dos dois eixos:
plot(x,y,xlim=c(-5,25),ylim=c(0,10000))
```


- Para editar os valores explicitamente mostrados em cada eixo, devemos usar o comando `axis` após a execução do gráfico desejado.

Exemplo 4.12

```
# Editando eixo horizontal:
plot(x,y,xaxt="n") # eixo horizontal sem valores
axis(1,seq(1,20,3)) # 1º argumento de axis = 1

# Editando eixo vertical:
plot(x,y,yaxt="n") # eixo vertical sem valores
axis(2,seq(0,8000,1600)) #1º argumento de axis = 2

# Adicionado expressões genéricas:
plot(x,y)
axis(1,c(8,14),c("a","b"),col.axis=2,col.ticks=2)
axis(2,c(8^3,14^3),c("f(a)","f(b)"),col.axis=2,las=2)
```

Recorte em branco do plano cartesiano

- O comando `plot` também possui uma opção no argumento `type` para abrir uma janela com um recorte do plano cartesiano em branco: `type="n"`.

Exemplo 4.13

```
# Recorte em branco na caixa [0;20]x[-8000;8000]:  
plot(c(0,20),c(-8000,8000),type="n")  
  
lines(x,y, col="navyblue")  
points(x,-y, col="gold")  
lines(c(5,10,7,5),c(5000,5000,0,5000),col=8)  
segments(5,-5000,10,0,col="brown")
```

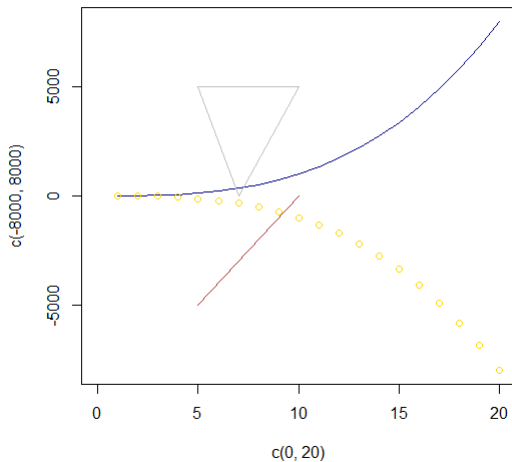


Figura 11: Gráficos obtidos com o Exemplo 4.13

Adicionando texto e título

- Os comandos `title` e `text` adicionam ao gráfico respectivamente título e texto (este último na janela do gráfico). Os dois primeiros argumentos do comando `text` indicam o par ordenado em torno do qual o texto será escrito.

Exemplo 4.14

```
plot(x,y,type="l",lwd=2,col=6)
title("Aula de hoje\nGráficos no R")
text(18,3000, "y=x ao cubo")

# Ilustrando o comando expression:
plot(x,y,type="l",lwd=2,col=6)
title("Aula de hoje\nGráficos no R",cex.main=1.8)
text(18,3000, expression(y==x^3))
```

Exemplo 4.15

```
# Ilustrando os comandos expression e paste:  
plot(x,y,type="l",lwd=2,col=6)  
title(expression(paste("Função ",y==x^3)))  
text(18,3000, expression(y==x^3),cex=2)
```

- **Observação:** Os comandos `expression` e `paste` também podem ser usados fora de `title` ou `text` (como, por exemplo, em `xlab` e `ylab`). Para maiores informações sobre tipografia de notações matemáticas no R, execute o comando

`?plotmath`

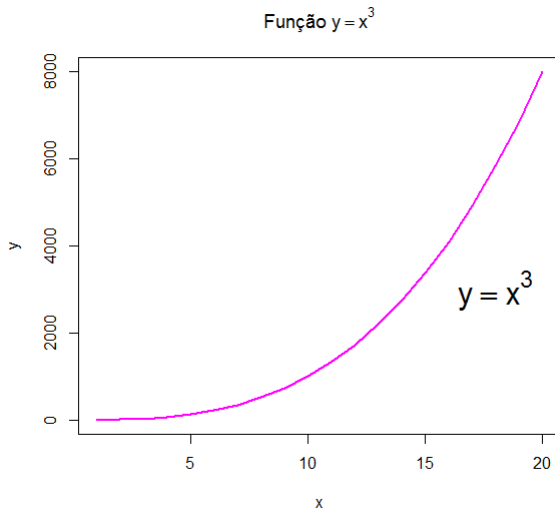


Figura 12: Gráfico obtido por meio do Exemplo 4.15

Comando `curve`

- O comando `curve` é ideal para exibir graficamente funções contínuas em um intervalo pré-definido. Seu primeiro argumento utiliza o caracter `x` para representar a função a ser exibida, sendo que este `x` não representa um objeto. Ainda, o argumento `add=T` no comando `curve` permite sobrepor gráficos, e vários argumentos válidos em `plot` também são válidos em `curve`.

Exemplo 4.16

```
curve(2*x,0,1) # gráfico de  $f(x)=2x$  de  $x=0$  até  $x=1$   
curve(1/(x+1),col=2,add=T) #  $g(x)=1/(x+1)$   
curve(dexp(x,5),col=4,add=T) # densidade da  $\text{Exp}(5)$   
legend("right",c("f(x)=2x",expression(g(x))==frac(1,x+1)),  
      ,expression(h(x)==5*e^{5*x})),fill=c(1,2,4))
```

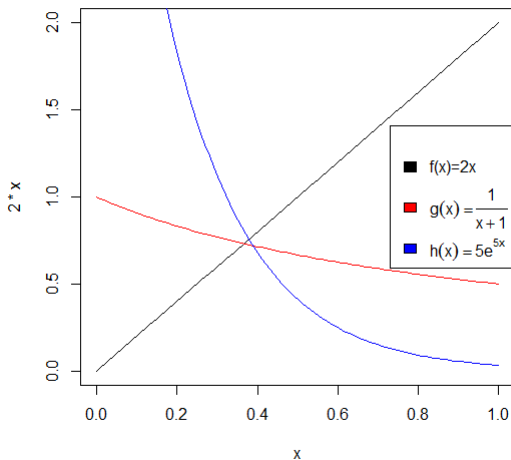


Figura 13: Gráficos e legenda obtidos por meio do Exemplo 4.16

Janela com vários gráficos (não sobrepostos)

Exemplo 4.17

```
par(mfrow=c(2,3)) # 2 linhas, 3 colunas

plot(x,y)
curve(x^3,col="red")
plot(x,2*y)
curve(2*x^3,col="green3",add=T)
plot(x,log(y),col="blue",type="l")
plot(x^2,y^2,type="h")
lines(x^2,y^2,col=5,lwd=2)
plot(-x,-y, type="s")
```

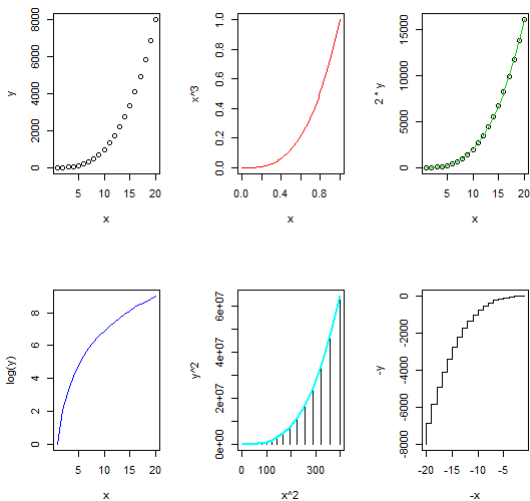


Figura 14: Gráficos obtidos no Exemplo 4.17

- Ao executar um novo comando gráfico (como `plot` ou `curve`) que gere um novo gráfico não sobreposto, os $2 \times 3 = 6$ gráficos plotados sumirão, e o gráfico fruto desta última linha de comando tomará a 1ª linha e 1ª coluna da janela de gráficos.
- Para retornar ao padrão com apenas uma caixa tomando toda a janela, execute o comando

```
par(mfrow=c(1,1))
```

ou simplesmente feche a janela *R Graphics*, na qual os gráficos estão dispostos.

Abrindo novas janelas *R Graphics*

- No software R, quando geramos um gráfico, ele automaticamente **sobrescreve** o gráfico anterior, fazendo com que o percamos. Uma forma de evitar que isto aconteça é abrir uma nova janela *R Graphics* com o comando `x11()`, para então gerar um novo gráfico.

Exemplo 4.18

```
plot(x,y,xlab="valores de x",  
      ylab=expression(paste("valores de ",x^3)))  
x11()  
plot(x^2,y^2,type="h")  
x11()  
curve(dbeta(x,8,2),0,1,main="X~Beta(8;2)")
```

- Histograma (comando hist):

```
z = rgamma(180,4,2)
z
par(mfrow=c(1,2))

# Frequências absolutas no eixo-y:
hist(z,nclass=8,freq=T,main="Histograma (f.a.)")

# Densidades de frequências no eixo-y:
hist(z,nclass=8,freq=F,main="Histograma (dens.)")

# Adicionando curva da densidade Gama(4;2):
curve(dgamma(x,4,2),col=4,lwd=2,add=T)

#Único argumento obrigatório de hist é o primeiro!
```

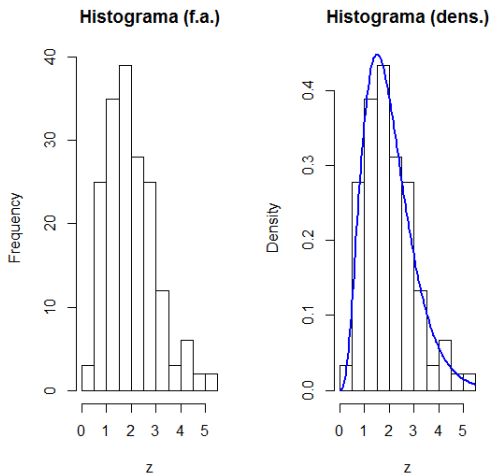


Figura 15: Histogramas e curva de densidade obtidos por meio das linhas de comando do *slide* 85

- Gráfico de barras (comando barplot):

```
conceitos = rep(c("A","B","C"),c(11,25,4))
```

```
par(mfrow=c(1,2))
```

```
# Gráfico de barras na vertical:
```

```
barplot(table(conceitos),main="Conceitos")
```

```
# Gráfico de barras na horizontal:
```

```
barplot(table(conceitos),main="Conceitos",horiz=T)
```

```
# Gráfico de barras colorido:
```

```
par(mfrow=c(1,1))
```

```
barplot(table(conceitos),main="Conceitos",  
        col=c("green","yellow","red"))
```

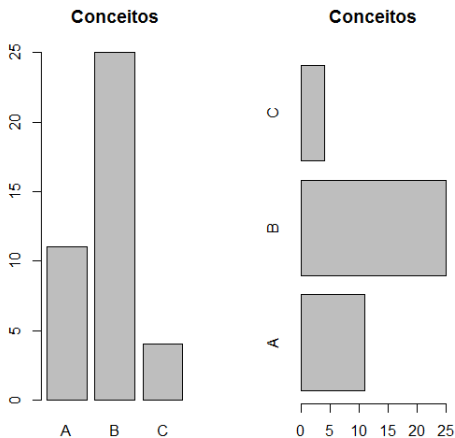


Figura 16: Gráficos de barras na vertical e na horizontal obtidos com as linhas de comando do *slide* 87

- Gráfico de setores (comando pie):

```
par(mfrow=c(1,1))

futrj= rep(c("BOT","FLA","FLU","VAS"),c(20,33,31,24))
pie(table(futrj), main="Títulos estaduais - RJ",
      col=c(1,2,"darkgreen",0))

# Cores definidas conforme ordem alfabética dos
# nomes!  BOT-1, FLA-2, FLU-"darkgreen" e VAS-0.
```

Títulos estaduais - RJ

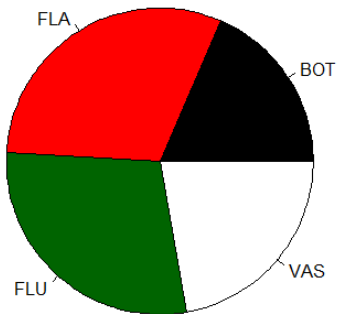


Figura 17: Gráfico obtido com as linhas de comando do *slide* 89

- **Boxplot (comando boxplot):**

```
x1 = rnorm(200,10,2)
```

```
x2 = rgamma(200,5,0.5)
```

```
x3 = rexp(200,0.1)
```

```
boxplot(x1,x2,x3,names=c("N(10;4)", "Gama(5;1/2)",  
      "Exp(1/10)"),main="Exemplos de boxplots")
```

Exemplos de boxplots

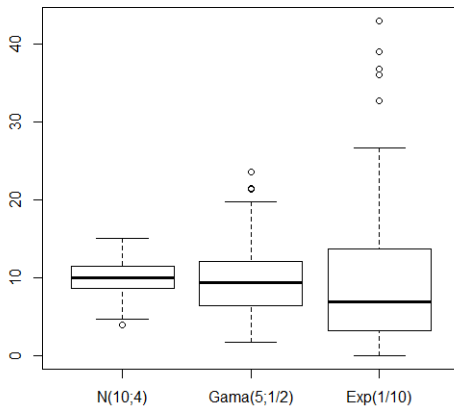


Figura 18: Boxplots obtidos por meio das linhas de comando do *slide* 91

- Para obter mais detalhes sobre estes (e outros) gráficos de interesse em Estatística, consulte a ajuda do R.
- Cabe ressaltar que vários dos argumentos que servem para o comando `plot` (tais como `col`, `pch`, `bg`, `cex`, `xlab`, `ylab`, `xlim`, `ylim`) também funcionam para gráficos de interesse em Estatística.

Salvando gráficos no R

- Gráficos no R podem ser salvos como arquivos em diferentes formatos (*.png*, *.jpeg*, *.pdf*, entre outros).
- Para tal, primeiramente clique em qualquer lugar na janela *R Graphics*. A barra de ferramentas padrão do R (associada à janela *R Console*, na qual são executados os comandos) se torna barra de ferramentas da janela *R Graphics*.
- Feito isto, acesse o menu *Arquivo* e selecione *Salvar como*. Uma aba com várias opções de formato de arquivo será aberta para salvar o conteúdo exibido na janela *R Graphics*.

V - Gerando funções e rotinas

Operadores comuns em funções/rotinas

Tabela 2: Operadores comuns em funções e rotinas

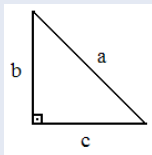
==	igual a
!=	diferente de
<	menor
<=	menor ou igual
>	maior
>=	maior ou igual
	ou (avalia condições)
&&	e (avalia condições)
%%	retorna o resto (ex.: $7\%3 = 1$)
%/%	divisão inteira (ex.: $7\%/3 = 2$)

- Uma função é feita de argumentos e uma sequência de comandos.
- Os argumentos são as entradas (*input*) para que a função realize a tarefa em questão. Uma sequência de comandos forma o corpo da função.
- Em linhas gerais, a estrutura de uma função criada no R é a seguinte:

```
nome_da_função = function(argumentos)  
{  
  Sequência de comandos  
  return(resposta)  
}
```

Exemplo 5.1

Função que retorna o valor da hipotenusa de um triângulo (retângulo).



```
hipotenusa = function(b,c){  
  a = sqrt(b^2 + c^2)  
  return(a)  
}
```

```
hipotenusa(1,1)
```

```
hipotenusa(9,12)
```

Editando uma função construída

- Caso terminemos de construir uma função tendo cometido algum erro na sua construção (por exemplo, poderíamos ter equivocadamente denotado a hipotenusa por $a = b^2 + c^2$), podemos editá-la por meio do comando `fix()`, com o nome da função dentro dos parênteses.
- Uma janela com o corpo da função será aberta imediatamente após a execução deste comando.
- Após a edição no corpo da função, será necessário salvar esta janela antes de fechá-la para que a modificação seja considerada.

Exemplo 5.2

Função que retorne um resumo descritivo.

```
resumo = function(x){ # x um vetor numérico
x1 = mean(x)
x2 = sd(x)
x3 = IQR(x)
x4 = min(x)
x5 = quantile(x,0.25)
x6 = median(x)
x7 = quantile(x,0.75)
x8 = max(x)
resposta = c(x1,x2,x3,x4,x5,x6,x7,x8)
names(resposta) = c("Média","Desvio Padrão","DIQ",
  "Mínimo","1ºQ","Mediana","3ºQ","Máximo")
return(resposta)
}
```

```
resumo(rep(c(7,12,10,2),c(4,7,8,6)))
```

```
resumo(rpois(100,3))
```

```
resumo(sample(1:20,1000,T))
```

```
resumo(rnorm(200,10,1))
```

Exemplo 5.3

Função que retorna o módulo de um número real.

```
modulo = function(x){  
  if(x<0){x=-x}  
  return(x)  
}
```

```
modulo(-3.5)
```

```
modulo(0.72)
```

```
y = log(runif(1)) # número não-positivo  
modulo(y)
```

Exemplo 5.4

Função que verifica se o número é par ou ímpar.

```
verifica = function(x){  
  if(x%%2==0){"O número é par"}  
  else{"O número é ímpar"}  
}
```

```
verifica(8)
```

```
verifica(27)
```

```
verifica(139)
```

```
verifica(4650)
```


Rotinas - comando `for`

- Rotinas permitem realização de uma mesma tarefa de forma cíclica, e também podem ser úteis fora do corpo de uma função.
- Para a implementação de rotinas, em geral utiliza-se o comando `for`, que permite que uma tarefa seja repetida a medida que um índice assume valores em uma sequência específica, na seguinte forma:

```
for(índice in sequência)  
{  
  Sequência de comandos  
}
```

Exemplo 5.5

Função que retorna o traço de uma matriz (quadrada), ou seja, a soma das entradas na diagonal principal.

```
traco = function(A){  
  tr=0  
  for(i in 1:nrow(A)){  
    tr = tr + A[i,i]}  
  return(tr)  
}  
  
traco(matrix(1:9,3,3))  
traco(matrix(6,10,10))  
traco(diag(c(9,3,6)))
```

Rotinas com comando `while`

Exemplo 5.6

Função que retorna o traço, mas usando `while` em vez de `for`:

```
rm(traco) # removendo função criada anteriormente
traco = function(A){
  tr=0
  i=1
  while(i <= nrow(A)){
    tr = tr + A[i,i]
    i=i+1}
  return(tr)}

traco(matrix(1:9,3,3))
traco(matrix(6,10,10))
traco(diag(c(9,3,6)))
```

Exemplo 5.7

```
# Outro exemplo de uso do comando while:
x = rnorm(2000)
y = sort(x) # ordem crescente

k=0
i=1
while(y[i]<1.645){
  k=k+1
  i=i+1}

k/length(y) # espera-se algo em torno de 0,95
```

Exemplo 5.8

Ilustrando graficamente a convergência da média amostral para a média populacional de uma amostra aleatória proveniente da distribuição $N(10; 64)$ conforme o tamanho da amostra aumenta.

```
n = c(2, 3, seq(5,5000,10))
xbarra = rep(0,length(n)) # vetor de n zeros

for (i in 1:length(n)){
  amostra = rnorm(n[i],10,4)
  xbarra[i] = mean(amostra)
}

plot(n, xbarra,col="violet",pch=19,
      main="Gráfico de convergência\nMédia amostral")
abline(10,0,lwd=3)
```

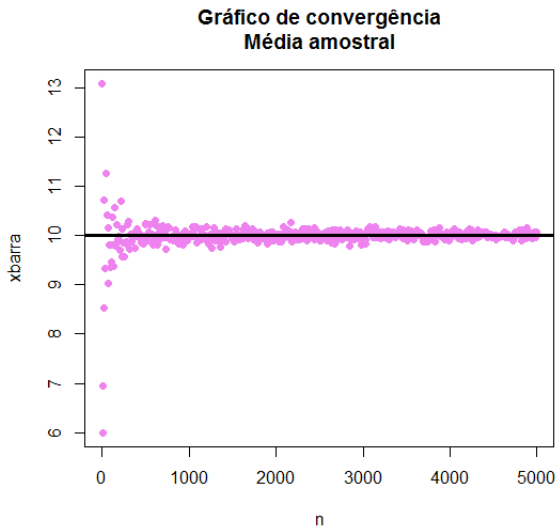


Figura 19: Gráfico obtido pelo Exemplo 5.8

- ❶ Alcoforado, L.F. & Cavalcante, C.V. *Introdução ao R utilizando a Estatística Básica*. Editora da UFF, Niterói, 2014.
- ❷ Carmo, C.N. *Noções Básicas de R*. Rio de Janeiro, 2003.
- ❸ História do programa R - disponível em <http://www.estatisticador.xpg.com.br/4.html>
- ❹ Justiniano, P. (2011). *Introdução ao Ambiente Estatístico R*. Disponível em: <http://www.leg.ufpr.br/~paulojus/embrapa/Rembrapa/Rembrapa.pdf>
- ❺ Terrón, A., Cabellero, P. & Alcaraz, L. (2011). *Estadística Básica con R-Commander*. Disponível em: <http://www.bubok.es/libros/203887/Estadistica-basica-con-RCommander>
- ❻ Venables, W.N., Smith, D.M. & the R Development Core Team (2017). *An Introduction to R- Version 3.3.3*. Disponível em: <https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf>