Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Estatística

Noções Básicas de *S-PLUS for Windows*

Edna Afonso Reis

Primeira Edição - Setembro de 1997

Prefácio

Esta apostila faz parte do material didático do curso "Introdução ao software S-PLUS", ministrado neste Departamento de Estatística, sendo formada por notas de aula e exercícios práticos. Essas notas foram basedas nos seguintes manuais do S-PLUS (Version 3.2 Release 1 for Sun SPARC, SunOS 4.x):

Noções básicas:

- A Gentle Introduction to S-PLUS
- A Crash Course in S-PLUS

Avançado:

- S-PLUS User's Manual
- S-PLUS Guide to Statistical and Mathematical Analysis
- S-PLUS Programmer's Manual
- S-PLUS Reference Manual I e II

Embora estes sejam manuais da versão S-PLUS para UNIX, são poucas as mudanças quando se trabalha na versão para Windows.

Outro importante guia para elaboração dessas notas foi o livro *Modern Apllied Statistics with S-Plus*, de Willian N. Venables e Brian D. Ripley (1994, Springer-Verlag New York Inc.).

Dentre as diversas tarefas implementadas no S-PLUS, algumas mereceram destaque por terem se mostrado as mais úteis ao longo da minha experiência com o software.

Como a melhor maneira de se aprender um software é usá-lo, recomendo que a leitura desta apostila seja feita interativamente com o uso do S-PLUS. O programa possui um *Help on Line* nos padrões dos softwares para Windows e, portanto, fácil de ser usado.

Edna Afonso Reis
Belo Horizonte, 1997

Índice

1- Aspectos Gerais do S-PLUS

Após iniciar o S-PLUS, aparecerá na janela *Commands* a seguinte mensagem:

```
S-PLUS : Copyright 1988, 1995 MathSoft, Inc.
S : Copyright AT&T.
Version 3.3 Release 1 for MS Windows 3.1 : 1995
Working data will be in _Data
```

O sinal > é o *prompt* da linha de comando, indicando que o S-PLUS está pronto para começar a receber comandos (nos exemplos e exercícios, o sinal > aparecerá antes do comando a ser executado para representar a tela da janela *Commands*. Não digite o *prompt* ao tentar os exercícios).

No S-PLUS, ao contrário de outros softwares estatísticos para Windows, as tarefas requeridas pelo analista não estão todas comodamente dispostas nos *menus* ao alcance do *mouse*. Essas tarefas, como manipular seus dados, fazer determinada análise estatística ou desenhar um gráfico, devem ser requeridas na linha de comando na janela *Commands*. Embora em princípio isso possa parecer uma desvantagem, o uso mais avançado do S-PLUS mostrará que essa orientação é vantajosa, pois permite imensa flexibilidade de uso e expansão das capacidades do software.

O S-PLUS tenta interpretar tudo o que é digitado na linha de comando (seguido de *ENTER*). Veja estes testes:

`> 2`	*ENTER*	Interpretou a entrada 2 como o número 2
`[1] 2`		
`> 2+2`	*ENTER*	Assumiu a tarefa de calculadora
`[1] 4`		
`> 2 < 4`	*ENTER*	Avaliou a expressão e respondeu que é verdadeira (T)
`[1] T`		
`> 2 > 4`	*ENTER*	Avaliou a expressão e respondeu que é falsa (F)
`[1] F`		
`> pi`	*ENTER*	Retornou o valor de π, que está armazenado internamente
`[1] 3.141593`		

No S-PLUS, os dados são armazenados na forma de *objetos*. Na próxima seção veremos os tipos de objetos de dados e como eles podem ser criados. Uma vez criado, um objeto S-PLUS é armazenado no sub-diretório *_Data* do diretório *Home* do S-PLUS (você pode mudar isso). Assim, se a tabela

Sexo	Idade	Altura
1	51	1.78
1	55	1.75
1	40	1.81
2	48	1.75
2	46	1.81
2	42	1.70

já existe no diretório *_Data* com o nome de **tab1**, ao digitar **tab1** na linha de comando, o S-PLUS irá mostrar o conteúdo do *objeto* tab1:

```
> tab1
     Sexo Idade Altura
[1,]    1    51   1.78
[2,]    1    55   1.75
[3,]    1    40   1.81
[4,]    2    48   1.75
[5,]    2    46   1.81
[6,]    2    42   1.70
```

Um objeto S-PLUS é criado como o resultado de um expressão S-PLUS. Para salvar o objeto resultante, simplemente associe um nome ao objeto usando *operadores de atribuição* **<-** , _ ou **->**. Por exemplo, os comandos seguinte criam um objeto **x** que contém o inteiro 10:

```
> x <- 10
```
ou
```
> x _ 10
```
ou
```
10 -> x
```

Para ver o conteúdo de **x**:

```
> x
[1] 10
```

Uma vez criados e associados a nomes (usando o operador de atribuição), os objetos S-PLUS são permanentes, isto é, escritos no disco sem que você precise salvá-los ao sair do programa.

O conteúdo do diretório pode ser visto através do comando **ls()** ou usando o *Object Manager* do *menu Tolls*. Pode-se remover um objeto do diretório através do comando **rm(***nome dos objetos separados por vírgula***)**:

```
> rm(x)
```

O objeto **x** deixa de existir:

```
> x
Error: Object "x" not found
```

No S-PLUS existe distinção entre letras maiúsculas e minúsculas, para os nomes de objetos ou de suas *funções* (comandos). Assim, o objeto **tab1** é diferente do objetos **Tab1**, **TAB1** ou **taB1** e a função **rm** não será reconhecida se você digitar **RM**.

```
> RM(x)
Error: couldn't find function "RM"
```

2- A organização de dados no S-PLUS

Antes de fazer a análise de seus dados no S-PLUS, você deve convertê-los, dentro do programa, para a forma de *objetos de dados S-PLUS*. O *tipo* de dado irá depender da natureza dos seus dados: uma simples seqüência de números é mais facilmente representada como um vetor, enquanto dados de um planilha (colunas representando variáveis e linhas representando indiíduos) são melhor representados como uma matriz ou *data frame*. O S-PLUs possui uma grande variedade de tipos de dados.

Nesta apostila, vamos concentrar nossa atenção em quatro *tipos*:

`vector`	Conjunto de elementos de mesmo *modo* em uma ordem especificada (unidimensional)
`matrix`	Disposição bidimensional de elementos de mesmo *modo*
`data frame`	Disposição bidimensional cujas colunas que podem representar dados de *modos* diferentes
`list`	Conjunto de componentes que podem ser de qualquer um dos outros tipos de objetos

Os elementos de um objeto podem ser dos seguintes *modos*:

`logical`	Modo binário, com valores representados as **T** ou **F** (true ou false)
`numeric`	Números reais
`complex`	Números complexos (parte real e imaginária)
`character`	Caracteres representados como *strings*

A seguir, veremos mais detalhes de cada um dos tipos de dados e como criá-los no S-PLUS.

Vetores

Um *vetor* é um conjunto de elementos em uma ordem específica. A ordem é especificada quando você cria o vetor (usualmente a ordem na qual você digita os dados) e isto é importante porque você pode se referir a um elemento unicamente pela sua posição no vetor.

Todos os elementos de um vetor devem ser de um único *modo*. Como os dados são mais frequentemente números, os vetores númericos são os mais usuais.

Por exemplo, suponha que você tenha medido o volume de gasolina (em litros) que você adicionou a seu carro em 118 paradas para “completar o tanque”. Suponha também que esses dados estão em um arquivo texto chamado *carro.txt* (no diretório *Home* do S-PLUS), com o seguinte conteúdo:

13.3 12.2 11.5 13.5 14.3 25.7 13.3 12.7 8.9 14.2 12 13.2 12.8 13.6 12.3 13.2 13.1 13.6 13.6 13
12.5 11.6 7.5 14.2 11.9 13.1 13.6 14.2 12.1 12.8 13.9 13 12.5 13.6 12.8 14.2 13 13.8 12.3 8.9
10.3 11.6 7.8 14.5 7.1 12 12.3 12.5 14.1 13 12.3 13.2 13.7 13.4 12.5 12.9 13.3 6.2 13.2 12.8
11.2 13.5 13.2 14.1 13.6 13 12.8 13.2 12.9 13 13 12 14.5 21.2 13.2 12.9 12.5 22 13.1 12.3
13.2 13.7 12.5 13.2 13.5 13.9 12.5 12.4 12.3 10.5 11.5 12.8 12 13 5.8 9.5 9 7.7 10.1 11.9
11.4 12.6 12.5 12.7 13.6 11.7 13.5 12.3 13 13.2 13.2 11.6 13.8 13.5 13.9 14.2 13 13

Você pode criar o vetor numérico **carro.gas** com esses dados da seguinte maneira:

```
> carro.gas <- scan(file = "carro.txt")
```

Ou seja, estamos atribuindo (símbolo <-) ao vetor **carro.gas** o resultado da leitura de dados númericos que estão no arquivo *carro.txt*, através da função **scan**. Digitando carro.gas na linha de comando, vemos que o vetor foi criado:

```
> carro.gas
 [1] 13.3 12.2 11.5 13.5 14.3 25.7 13.3 12.7  8.9 14.2 12.0 13.2 12.8 13.6 12.3
[16] 13.2 13.1 13.6 13.6 13.0 12.5 11.6  7.5 14.2 11.9 13.1 13.6 14.2 12.1 12.8
[31] 13.9 13.0 12.5 13.6 12.8 14.2 13.0 13.8 12.3  8.9  10.3 11.6  7.8 14.5  7.1
[46] 12.0 12.3 12.5 14.1 13.0 12.3 13.2 13.7 13.4 12.5 12.9 13.3  6.2 13.2 12.8
[61] 11.2 13.5 13.2 14.1 13.6 13.0 12.8 13.2 12.9 13.0 13.0 12.0 14.5  21.2 13.2
[76] 12.9 12.5 22.0 13.1 12.3 13.2 13.7 12.5 13.2 13.5 13.9 12.5 12.4 12.3 10.5
[91] 11.5 12.8 12.0 13.0  5.8  9.5  9.0  7.7 10.1 11.9 11.4 12.6 12.5 12.7 13.6
[106]11.7 13.5 12.3 13.0 13.2 13.2 11.6 13.8 13.5  13.9 14.2 13.0 13.0
```

Quando mostra um vetor, o S-PLUS começa cada linha com o *índice* do primeiro elemento da linha entre colchetes []. O índice indica a posição do elemento dentro do vetor. Você pode usar o índice do elemento para se referir a ele, como por exemplo, para pedir que somente o sexto elemento seja mostrado:

```
> carro.gas[6]
[1] 25.7
```

Observação: No exemplo acima, o resultado da expressão **carro.gas[6]** começa com o índice **1**, embora nós pedimos pelo elemento com índice **6**. A incinsitência é apenas aparente, entretanto; porque, sendo **carro.gas[6]** uma expressão S-PLUS, o resultado de sua avaliação é um objeto S-PLUS, neste caso um vetor com apenas um elemento. Assim, em **carro.gas**, a observação 25.7 é o sexto elemento, mas em **carro.gas[6]** a observação 25.7 passou a ser o primeiro (e único) elemento.

O número de elementos de um vetor é o seu **length** :

```
> length(carro.gas)
[1] 118
```

Podemos usar a função **scan** para criar um vetor digitando os dados dentro do S-PLUS. Por exemplo, você poderia criar um pequeno vetor com as notas de cinco alunos em uma prova da seguinte maneira:

```
> notas <- scan()
```

Após o *ENTER*, o S-PLUS retorna o sinal de *prompt* **1:** para indicar que ele está pronto para receber o primeiro elemento do vetor (e os próximos também):

```
> notas <- scan()
1:
```

Digite as notas:

```
> notas <- scan()
1: 95 90 98
4: 99 87
```

Quando terminar, pressione *ENTER* em uma linha vazia:

```
> notas <- scan()
1: 95 90 98
4: 99 87
6:
>
```

O *prompt* > aparece para indicar que o S-PLUS terminou a leitura do vetor **notas**. O conteúdo do objeto notas é:

```
> notas
 [1]  95  90  98  99  87
```

Outro modo de criar um vetor dentro do S-PLUS é usar a função **c**:

```
> notas <- c(95,90,98,99,87)
> notas
 [1]  95  90  98  99  87
```

- **Vetores de dados não-numéricos**

Você pode criar um vetor de caracteres (como nomes) usando a função **scan** com o argumento **what=character(***n***)**, onde *n* é o número de nomes que você gostaria de ler:

```
> alunos <- scan(what=character(5))
1: Ariadne Joel Alana Agnes Juan
6:
> alunos
[1] "Ariadne" "Joel"    "Alana"   "Agnes"   "Juan"
```

Podemos dar nomes aos elementos de um vetor através da associação desse vetor a um outro vetor de nomes com o mesmo *length* que ele. Isso é feito usando-se a função **names**:

```
> names(notas) <- alunos
> notas
 Ariadne Joel Alana Agnes Juan
      95   90    98    99   87
```

Um modo mais fácil (de ser lembrado) de criar um vetor de caracteres é usando a função **c**:com os caracteres (nomes) entre aspas duplas:

```
> alunos <- c("Ariadne","Joel","Alana","Agnes","Juan")

> alunos
[1] "Ariadne" "Joel"    "Alana"   "Agnes"   "Juan"
```

Outro tipo de dados não-numéricos são os dados categóricos, que são obervações vindas de um número finito de categorias. Os dados categóricos são representados no S-PLUS por meio de *fatores*. Por exemplo, você pode criar um fator com o sexo dos alunos do exemplo anterior, através da função **factor** (veja outros exemplos interessantes do uso dessa função no *Help*):

```
> sexo <- factor(scan(what=""))
1: Fem Masc Fem Fem Masc
6:
>
```

ou

```
> sexo <- factor(c("Fem","Masc","Fem","Fem","Masc"))
```

A Tabela 1 resume algumas funções úteis para criar vetores. Em particular, a função **:** , que define uma seqüência numérica,

```
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10

> 12:18
[1] 12 13 14 15 16 17 18
```

é útil na extração de subconjuntos de dados. Por exemplo, se desejamos visualizar apenas os dez primeiros ou os cinco últimos elementos do vetor **carro.gas**, fazemos:

```
> carro.gas[1:10]
 [1] 13.3 12.2 11.5 13.5 14.3 25.7 13.3 12.7  8.9 14.2

> carro.gas[114:118]
[1] 13.5 13.9 14.2 13.0 13.0
```

Matrizes

Uma *matriz* é uma disposição bidimensional dos dados, em *linhas* e *colunas*. É a maneira usual de se "enxergar os dados": as colunas são as variáveis e as linhas são os indivíduos.

Os elementos de uma matriz no S-PLUS devem ser de um único *modo*, sendo mais comum termos dados numéricos. Por exemplo, suponha que você tenha o conjunto de notas de cinco alunos em três provas armazenado em um arquivo texto chamado *notas.txt* (no diretório *Home* do S-PLUS), com o seguinte conteúdo:

```
95  94  96
90  89  90
98  95  99
99  97 100
87  86  88
```

onde as linhas representam os alunos e as colunas representam as provas.

Podemos usar a função **matrix** para converter um vetor em uma matriz, especificando o número de linhas (ou colunas) e se o preenchimento da matriz deve ser feito por colunas (*default*) ou por linhas. A função **matrix** tem a seguinte estrutura:

matrix(*dados***, nrow= , ncol= , byrow=F)**

onde o argumento **byrow** tem valor lógico, com **F** (false) para prenchimento por colunas e **T** (true) para preenchimento por linhas.

Para ler matrizes em dados externos como *notas.txt*, a função **matrix** deve ser usada em conjunto com a função **scan**:

```
> notas.mat <- matrix(scan("notas.txt"), ncol=3, byrow=T)
> notas.mat
     [,1] [,2] [,3]
[1,]   95   94   96
[2,]   90   89   90
[3,]   98   95   99
[4,]   99   97  100
[5,]   87   86   88
```

Note que o preenchimento da matriz foi feito por linhas, ou seja, **byrow=T**.

Assim como os vetores, uma matriz pode ser criada digitando-se seus valores diretamente no S-PLUS, através da função **scan** sem argumentos:

```
>  notas.mat <- matrix(scan(), ncol=3, byrow=T)
1: 95 94 96 90 89 90 98 95 99 99 97 100 87 86 88
16:
>
```

ou usando-se a função **c** para combinar os valores:

```
> notas.mat <-matrix(c(95,94,96,90,89,90,98,95,99,99,97,100,87,86,88),
                     ncol=3, byrow=T)
```

Uma matriz também pode ser construída pela combinação de dois ou mais vetores de mesmo **length**. Por exemplo, se temos três vetores com as notas em cada prova,

```
> notas.p1 <- c(95,90,98,99,87)
> notas.p2 <- c(94,89,95,97,86)
> notas.p3 <- c(96,90,99,100,88)
```

usamos a função **cbind** para combiná-los como colunas da matriz **notas.mat**:

```
> notas.mat <- cbind(notas.p1, notas.p2, notas.p3)
```

Se temos cinco vetores com as notas de cada aluno,

```
> notas.a1 <- c(95,94,96)
> notas.a2 <- c(90,89,90)
> notas.a3 <- c(98,95,99)
> notas.a4 <- c(99,97,100)
> notas.a5 <- c(87,86,88)
```

usamos a função **rbind** para combiná-los como linhas da matriz **notas.mat**:

```
> notas.mat <- rbind(notas.a1,notas.a2,notas.a3,notas.a4,notas.a5)
```

Cada elemento de uma matriz é unicamente identificado pela sua posição na matriz, ou seja, indicando-se a linha e a coluna onde o elemento está. Assim, a nota do quarto aluno (linha) na terceira prova (coluna) é identificada por:

```
> notas.mat[4,3]
[1] 100
```

O número de linhas e o número de colunas de uma matriz é a sua dimensão, que pode ser obtida usando-se a função **dim** :

```
> dim(notas.mat)
[1] 5 3
```

A Tabela 2 resume algumas funções úteis para criar e manipular matrizes. Podemos associar nomes às linhas e colunas de uma matriz usando a função **dimnames** , que retorna uma *lista* com dois componentes: os nomes das linhas e os nomes das colunas (veja seção Listas).

Data Frame

Freqüentemente, os dados são uma mistura de variáveis númericas (como idade), variáveis categóricas (como sexo) e caracteres (como nome do indivíduo). Uma tabela deste tipo não pode ser lida como uma matriz, já que mistura dados de diferentes *modos*, mas pode ser lida como um *data frame* (o termo não foi traduzido para clareza do significado).

Um *data frame* também é uma disposição bidimensional dos dados, mas pode ter elementos de diferentes *modos* em diferentes colunas, desde que cada coluna tenha o mesmo tamanho. Assim, um *data frame* é uma estrutura tabular na qual as colunas representam variáveis e as linhas representam os indivíduos.

Um *data frame* pode ser criado de duas maneiras:

read.table	lê os dados de um arquivo externo
data.frame	combina objetos S-PLUS de vários *modos*

Assim, suponha que você tenha em um arquivo texto chamado *notasf.txt* (no diretório *Home* do S-PLUS), que, além do conjunto de notas de cinco alunos em três provas, também tenha duas colunas com o nome e o sexo dos alunos, com o seguinte conteúdo:

Alunos	Sexo	Prova1	Prova2	Prova3
Ariadne	Fem	95	94	96
Joel	Masc	90	89	90
Alana	Fem	98	95	99
Agnes	Fem	99	97	100
Juan	Masc	87	86	88

Para ler essa tabela como *data frame* do S-PLUS, vamos utilizar a função **`read.table`**:

```
> notas.frame <- read.table("notasf.txt", header=T)

> notas.frame
        Sexo Prova1 Prova2 Prova3
Ariadne  Fem     95     94     96
   Joel Masc     90     89     90
  Alana  Fem     98     95     99
  Agnes  Fem     99     97    100
   Juan Masc     87     86     88
```

O argumento **`header=T`** indica que a primeira linha contém os nomes das colunas. Se não for o caso, apenas não forneça argumento **`header`**, pois seu *default* é FALSE.

O número de linhas e o número de colunas de um *data frame* podem ser obtidos usando-se a função **`dim`**:

```
> dim(notas.frame)
[1] 5 4
```

Observação: Note que o S-PLUS tomou a primeira coluna de caracteres da tabela (Alunos) como nomes para as linhas.

Assim como nas matrizes, os elementos de um *data frame* são indexados pelo número da linha e da coluna onde se encontra:

```
> notas.frame[4,3]
[1] 97
```

Um *data frame* também pode ser criado dentro do S-PLUS através da combinação de vários objetos, como vetores de mesmo **`length`** e matrizes com mesmo número de linhas:

```
> notas.frame <- data.frame(sexo,notas.mat)

> notas.frame
        sexo Prova1 Prova2 Prova3
Ariadne  Fem     95     94     96
   Joel Masc     90     89     90
  Alana  Fem     98     95     99
  Agnes  Fem     99     97    100
   Juan Masc     87     86     88
```

Por default, **data.frame** converte todos os vetores de caracteres em *fatores*. Para preservá-los como dados de caracteres, você deve usar a função **I** ao passá-los para **data frame**:

```
> lixo <- data.frame(A=1:4, B=c("a","b","c","d"))
> is.factor(lixo[,2])
[1] T

> lixo <- data.frame(A=1:4, B=I(c("a","b","c","d")))
> is.factor(lixo[,2])
[1] F
```

Observação: a função is.factor(x) testa se o argumento x é do tipo fator. O S-PLUS tem as funções is.numeric, is.vector, is.matrix, etc, para testar se o argumento é do modo numérico, do tipo vetor ou matriz, respectivamente.

Listas

A lista é o mais flexível tipo de objeto no S-PLUS, pois os componentes de uma lista podem ser de qualquer *modo* ou *tipo*, incluindo outras listas. Você pode combinar em uma lista, por exemplo, um vetor numérico com 10 valores e uma matriz 5x5 de valores lógicos.

A lista é a mais freqüente escolha para retornar valores nas análises do S-PLUS. Por exemplo, a saída do ajuste de uma regressão linear retorna em uma lista valores númericos como os coeficientes e os resíduos, que têm comprimentos diferentes, um valor lógico que indica se o intercepto foi ou não ajustado, dentre outros resultados.

Uma lista é criada através da função **list**, com os componentes separados po vírgula:

```
list(componente1, componente2, ...., componenteN)
```

Como foi mencionado na seção Matrizes, o nome das dimensões de um matriz é uma lista com dois componentes: um com os nomes das linhas e o outro com os nomes das colunas. Vamos voltar ao exemplo da matriz **notas.mat**, com as notas de cinco alunos (linhas) em três provas (colunas). Vamos recriar a matriz:

```
> notas.mat <-matrix(c(95,94,96,90,89,90,98,95,99,99,97,100,87,86,88),
                     ncol=3, byrow=T)
> notas.mat
     [,1] [,2] [,3]
[1,]   95   94   96
[2,]   90   89   90
[3,]   98   95   99
[4,]   99   97  100
[5,]   87   86   88
```

Por enquanto a matriz não tem nome nas linhas e colunas:

```
> dimnames(notas.mat)
NULL
```

Vamos criar uma lista com os nomes dos cinco alunos em um componente e os nomes das três colunas (provas) em outro componente:

```
> notas.list <- list(alunos, c("Prova1", "Prova2", "Prova3"))
```

e associar essa lista às dimensões da matriz **notas.mat**:

```
> dimnames(notas.mat) <- notas.list

> notas.mat
        Prova1 Prova2 Prova3
Ariadne     95     94     96
   Joel     90     89     90
  Alana     98     95     99
  Agnes     99     97    100
   Juan     87     86     88

> dimnames(notas.mat)
[[1]]:
[1] "Ariadne" "Joel"    "Alana"   "Agnes"   "Juan"

[[2]]:
[1] "Prova1" "Prova2" "Prova3"
```

O número de componentes de uma lista pode ser obtido através da função **length**:

```
> length(dimnames(notas.mat))
[1] 2
```

Os componentes de uma lista são identificados pela sua posição na lista, entre duplo colchetes **[[#]]**:

```
> dimnames(notas.mat)[[1]]
[1] "Ariadne" "Joel"    "Alana"   "Agnes"   "Juan"

> dimnames(notas.mat)[[2]]
[1] "Prova1" "Prova2" "Prova3"
```

Os nomes dos componentes podem ser dados durante a criação da lista ou através da função **names**:

```
> dimnames(notas.mat)<-list(Alunos=alunos,Provas=c("Prova1","Prova2", "Prova3"))
```

ou

```
> names(dimnames(notas.mat)) <- c("Alunos", "Provas")

> dimnames(notas.mat)
$Alunos:
[1] "Ariadne" "Joel"    "Alana"   "Agnes"   "Juan"

$Provas:
[1] "Prova1" "Prova2" "Prova3"
```

E você pode se referir a cada componente pelo seu nome:

```
> dimnames(notas.mat)$Alunos
[1] "Ariadne" "Joel"    "Alana"   "Agnes"   "Juan"

> dimnames(notas.mat)$Provas
[1] "Prova1" "Prova2" "Prova3"
```

Tabela 1: Funções úteis para criar vetores

Função	Descrição	Exemplos
scan	Lê valores de qualquer *modo*	`scan(),scan(what=character(10))` `scan("dados")`
c	Combina valores de qualquer *modo*	`c(1,2,3), c("yes","no")` `c(vetor1, vetor2, vetor3)`
rep	Repete valores de qualquer *modo*	`rep(NA,5), rep(c(10,20),10)` `rep(c(1,2),c(10,20))`
seq	Cria seqüências no *modo numeric*	`seq(-pi,pi,length=100)` `seq(0,1, by=0.01)`
:	Cria seqüências no *modo numeric*	`0:5, 1:-1, 10*(1:10), 0.5:3.5`
vector	Inicializa um vetor	`vector('complex',3)`
logical	Inicializa um vetor no *modo logical*	`logical(3)`
numeric	Inicializa um vetor no *modo numeric*	`numeric(4)`
complex	Inicializa um vetor no *modo complex*	`complex(5)`
caracter	Inicializa um vetor no *modo character*	`character(6)`

Tabela 2: Funções úteis para criar matrizes

Função	Descrição	Exemplos
matrix	Cria uma matriz	`matrix(1,20,10)` `matrix(1:12, ncol=4, byrow=T)`
cbind	Combina colunas de matrizes	`cbind(21:30,rep(1,10)),` `cbind(matriz1,matriz2,matriz3)`
rbind	Combina linhas de matrizes	`rbind(21:30,rep(1,10)),` `rbind(matriz5,matriz6)`
diag	Retorna a diagonal de uma matriz; Cria uma matriz diagonal	`diag(matrix(1:16,4,4,T))` `diag(mat) <- 1` `mat <- diag(25,10)`
t	Retorna a matriz transposta	`t(mat)`
solve	Retorna a matriz inversa	`solve(mat)`
data.matrix	Converte *data frame* para matriz	`data.matrix(dados.frame)`

Expressando Dados Faltantes no S-PLUS

Um valor faltante (*missing value*) é representado pelo símbolo **NA** (Not Avaliable), com letras maiúsculas. Por exemplo, se no vetor **notas** a nota do terceiro aluno não está disponível, colocamos **NA** na terceira posição do vetor:

```
> notas2 <- c(95,90,NA,99,87)
> notas2
[1] 95 90 NA 99 87
```

Para saber se um objeto tem valores **NA**, fazemos o teste com a função **is.na**, que retorna **T** para os elementos **NA** e **F**, caso contrário:

```
> is.na(notas2)
[1] F F T F F
```

Algumas funções S-PLUS permitem que os dados tenham valores **NA** (veja a função **abs** no *Help*). Dentre as funções que *não* aceitam valores **NA**, algumas têm a opção de automaticamente removê-los (veja a função **mean**), enquanto outras só funcionam se você remover os valores **NA** antes de chamar a função (veja a função **var**).

3- Manipulando dados

Nesta seção apresento algumas formas de extrair e modificar subconjuntos de dados no S-PLUS usando a idéia de "se referir a elemento do objeto de dados pela sua posição nele".

Três tarefas especiais de manipulação de dados são destacadas nessa seção: fazer operações aritméticas com os dados, ordená-los e extrair valores repetidos.

- **Extraindo subconjuntos de dados**

Matrizes e Vetores

Seja a matriz **`notas.mat`** anterior, com as notas de cinco alunos (linhas) em três provas (colunas):

```
> notas.mat
        Prova1 Prova2 Prova3
Ariadne     95     94     96
   Joel     90     89     90
  Alana     98     95     99
  Agnes     99     97    100
   Juan     87     86     88
```

Sabemos que podemos extrair um determinado elemento da matriz indicando sua posição [linha,coluna] na matriz. Por exemplo, a nota da aluna Agnes (linha 4) na segunda prova (coluna 2) pode ser obtida inicando a posição [4,2]:

```
> notas.mat[4,2]
[1] 97
```

ou, como as linhas e colunas têm nomes:

```
> notas.mat["Agnes", "Prova2"]
[1] 97
```

Para extrair uma linha inteira, basta fornecer o número (nome) da linha e deixar em branco o número da coluna. Assim, para obter as notas da aluna Agnes em todas as provas:

```
> notas.mat[4,]
 Prova1 Prova2 Prova3
     99     97    100
```

ou

```
> notas.mat["Agnes",]
 Prova1 Prova2 Prova3
     99     97    100
```

Da mesma forma, para obter as notas de todos os alunos apenas na segunda prova (coluna 2):

```
> notas.mat[,2]
 Ariadne Joel Alana Agnes Juan
      94   89    95    97   86
```

ou

```
> notas.mat[, "Prova2"]
 Ariadne Joel Alana Agnes Juan
      94   89    95    97   86
```

Você pode combinar índices de linhas ou colunas em vetores e usar esses vetores dentro do operador [,]. Por exemplo, para saber apenas as notas dos alunos do sexo masculino nas duas primeiras provas, fazemos:

```
> notas.mat[c(2,5), 1:2]
     Prova1 Prova2
Joel     90     89
Juan     87     86
```

Muitas vezes queremos montar uma matriz como um subconjunto de linhas (ou colunas) de outra matriz, mas o número de linhas que serão mantidas é maior que número de colunas que serão eliminadas. Por exemplo, para montar um matriz formada pelas linhas 1, 3 e 5 da matriz **notas.mat**, podemos manter essas linhas

```
> notas2.mat <- notas.mat[c(1,3,5),]
> notas2.mat
        Prova1 Prova2 Prova3
Ariadne     95     94     96
  Alana     98     95     99
   Juan     88     85     88
```

ou eliminar as linhas 2 e 4, usando o operador **-** antes do vetor que define os índices das linhas a serem elimindas:

```
> notas2.mat <- notas.mat[-c(2,4),]
> notas2.mat
        Prova1 Prova2 Prova3
Ariadne     95     94     96
  Alana     98     95     99
   Juan     88     85     88
```

Você pode também extrair um subconjunto de linhas (ou colunas) que obedecem uma determinada condição. Por exemplo, suponha que queremos retirar da matriz **notas.mat** apenas as notas na segunda prova que sejam maiores ou iguais a 90. Vamos fazer isso passo a passo: primeiro vamos criar um vetor com a notas da segunda prova, ou seja, com a coluna 2 da matriz **notas.mat**:

```
> p2 <- notas.mat[,2]
> p2
 Ariadne Joel Alana Agnes Juan
      94   89    95    97   86
```

Agora vamos saber quais destas notas obedecem à condição:

```
> p2 >= 90
 Ariadne Joel Alana Agnes Juan
       T    F     T     T    F
```

Assim, estamos procurando pelos elementos das posições 1, 3 e 4 do vetor **p2** (equivalentemente, pelos elementos das posições [1,2], [3,2] e [4,2] da matriz **notas.mat**). O vetor lógico que o S-PLUS retorna pode ser armazenado em um objeto:

```
> aux <- p2 >= 90
> aux
 Ariadne Joel Alana Agnes Juan
       T    F     T     T    F
```

Finalmente vamos indicar para o S-PLUS que ele deve extrair do vetor **p2** apenas os elementos com valor **T** no vetor **aux**, pois estes são os elementos que obedecem à condição imposta sobre as notas as segunda prova:

```
> p2.cA <- p2[aux]
> p2.cA
 Ariadne Alana Agnes
      94    95    97
```

Quando você estiver mais familiarizado com expressões encaixadas, você poderá omitir os passos intermediários e obter o resultado desejado com apenas uma expressão:

```
> notas.mat[notas.mat[,2]>=90,2]
 Ariadne Alana Agnes
      94    95    97
```

Esta expressão está "dizendo ao S-PLUS" para selecionar da matriz **notas.mat** os elementos (linhas) da segunda coluna sob a condição de que os elementos da segunda coluna da matriz **notas.mat** sejam maiores ou iguais a 90.

Usando a mesma idéia, podemos, em apenas uma expressão, selecionar as notas da primeira prova dos alunos que tiraram nota menor que 90 na segunda prova:

```
> notas.mat[notas.mat[,2]<90,1]
 Joel Juan
   90   87
```

Listas

Vimos anteriormente que podemos extrair os componentes de uma lista usando o operador duplo colchete [[]] ou, se os componentes forem nomedos, através do operador $:

```
> list1 <- list(codigo=letters[1:5], nota=notas.mat)

> list1
$codigo:
[1] "a" "b" "c" "d" "e"

$nota:
        Prova1 Prova2 Prova3
Ariadne     95     94     96
   Joel     90     89     90
  Alana     98     95     99
  Agnes     99     97    100
   Juan     87     86     88

> list1[[1]]
[1] "a" "b" "c" "d" "e"

> list1$nota
        Prova1 Prova2 Prova3
Ariadne     95     94     96
   Joel     90     89     90
  Alana     98     95     99
  Agnes     99     97    100
   Juan     87     86     88
```

Uma vez que você extraiu um componente da lista, você pode usar a forma apropriada de extrair subconjuntos para aquele tipo de objeto. Por exemplo, para extrair a quarta linha da matriz que é o segundo componente da lista **`list1`**, podemos seguir esses passos:

```
> mat.list1 <- list1[[2]]
> mat.list1[4,]
 Prova1 Prova2 Prova3
     99     97    100
```

ou usar expressões encaixadas

```
> list1[[2]][4,]
 Prova1 Prova2 Prova3
     99     97    100
```

- **Substuindo subconjuntos de dados**

Além de serem úteis na extração de subconjuntos de dados, os operadores de susbscrito ([] e [[]]) são usados também para substituir valores no conjunto de dados por novos valores, apenas informando a posição do elemento que receberá o novo valor.

Por exemplo, a nota do quarto aluno na segunda prova pode ser substituída na matriz **`notas.mat`** pelo novo valor 100 da seguinte forma:

```
> notas.mat[4,2] <- 100
> notas.mat
        Prova1 Prova2 Prova3
Ariadne     95     94     96
   Joel     90     89     90
  Alana     98     95     99
  Agnes     99    100    100
   Juan     87     86     88
```

Você pode substituir mais de um valor ao mesmo tempo. Por exemplo, para substituir as notas do quarto e do quinto aluno na primeira e segunda prova, fazemos:

```
> notas.mat[4:5,1:2] <- c(100,88,99,85)
> notas.mat
        Prova1 Prova2 Prova3
Ariadne     95     94     96
   Joel     90     89     90
  Alana     98     95     99
  Agnes    100     99    100
   Juan     88     85     88
```

Note que a matriz **`notas.mat[4:5,1:2]`** é preenchida por colunas.

- **Operações com vetores e matrizes**

A maioria dos dados que analisamos são numéricos e dados numéricos são freqüentemente manipulados usando funções e operadores matemáticos. A Tabela 3 mostra os operadores matemáticos no S-PLUS (e alguns operadores lógicos e de comparação) e na seção 8 veremos algumas funções matemáticas.

Operações matemáticas no S-PLUS são *vetorizadas*, significando que elas atuam no objeto de dados inteiro de uma só vez, elemento por elemento.

Tabela 3: Operações no S-PLUS

Operadores Aritméticos	
+	adição
-	subtração
*	produto
/	divisão
^	exponenciação
% %	resto da divisão
% / %	inteiro da divisão
% * %	multiplicação matricial
Operadores de Comparação	
==	igual a
!=	não igual a
<	menor que
>	maior que
<=	menor ou igual a
>=	maior ou igual a
Operadores Lógicos	
!	não
|	ou
||	ou *sequencial (para avaliar condições)*
&	e
&&	e *sequencial (para avaliar condições)*

Para ilustrar o uso dos operadores matemáticos, suponha que temos dois vetores **x** e **y**:

```
> x <- c(2,3,5,8,6,4)
> y <- c(5,7,9,1,3,6)
```

Para somar, subtrair, multiplicar e dividir os dois vetores, elemento por elemento, faça:

```
> x + y
[1]  7 10 14  9  9 10

> x - y
[1] -3 -4 -4  7  3 -2

> x*y
[1] 10 21 45  8 18 24

> x/y
[1] 0.4000000 0.4285714 0.5555556 8.0000000 2.0000000 0.6666667
```

Você pode multiplicar todos os elementos do vetor por 10, por exemplo:

```
> 10*x
[1] 20 30 50 80 60 40
```

ou subtrair 2 de todos eles:

```
> x-2
[1] 0 1 3 6 4 2
```

Com as matrizes, os operadores funcionam igualmente, aplicando a operação elemento por elemento da matriz. Sejam as matrizes **M**, **Mt** (transposta de **M**) e **V**:

```
> M <- matrix(1:12,3,4)
> Mt <- t(M)
> V <- matrix(13:24,4,3)

> M
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

> Mt
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12

> V
     [,1] [,2] [,3]
[1,]   13   17   21
[2,]   14   18   22
[3,]   15   19   23
[4,]   16   20   24
```

Como têm a mesma dimensão, as matrizes **Mt** e **V** podem ser somadas ou multiplicadas elemento por elemento:

```
> Mt + V
     [,1] [,2] [,3]
[1,]   14   19   24
[2,]   18   23   28
[3,]   22   27   32
[4,]   26   31   36

> Mt*V
     [,1] [,2] [,3]
[1,]   13   34   63
[2,]   56   90  132
[3,]  105  152  207
[4,]  160  220  288
```

Mas note que a *multiplicação matricial* de **M** por **V** é feita usando-se o operador %*%:

```
> M%*%V
[,1] [,2] [,3]
[1,]  334  422  510
[2,]  392  496  600
[3,]  450  570  690
```

Você pode ter expressões mais complexas no S-PLUS, envolvendo quantos objetos você queira (desde que compatíveis):

```
> (c(2*(1:5), x)/4)^5
[1]  0.0312500  1.0000000  7.5937500 32.0000000 97.6562500  0.0312500
[7]  0.2373047  3.0517578 32.0000000  7.5937500  1.0000000

> 2*x/(x+y)^2
[1] 0.08163265 0.06000000 0.05102041 0.19753086 0.14814815 0.08000000
```

A montagem das expressões no S-PLUS segue a mesma lógica que você usaria se estivesse escrevendo a expressão no papel, inclusive obedecendo a hierarquia das operações matemáticas. Por exemplo, o operador * (multiplicação) tem prioridade sobre o operador + (soma). Se você desejar que a soma seja feita antes da multiplicação, você deve usar a expressão de soma entre parênteses:

```
> (2+4)*2
[1] 12
```

que é diferente de

```
> 2+4*2
[1] 10
```

Os operadores de comparação também operam em cada elemento do vetor ou matriz, retornando um objeto com valores lógicos (T ou F) do mesmo tipo e dimensão do objeto avaliado:

```
> notas.mat == 100
        Prova1 Prova2 Prova3
Ariadne      F      F      F
   Joel      F      F      F
  Alana      F      F      F
  Agnes      T      F      T
   Juan      F      F      F
```

Os operadores lógicos são utilizados para combinar expressões que retornem valores lógicos, como aquelas formadas por operadores de comparação, e a expressão final retorna um valor lógico:

```
> (3>2) & (pi < 4)          # verdadeiro  E   verdadeiro   = verdadeiro
[1] T
> (3>2) | (pi < 4)          # verdadeiro  OU  verdadeiro   = verdadeiro
[1] T

> (1>2) & (4 != 2+2)        # falso       E   falso        = falso
[1] F
> (1>2) | (4 != 2+2)        # falso       OU  falso        = falso
[1] F

> (3>2) & (1>2)             # verdadeiro  E   falso         = falso
[1] F
> (3>2) | (1>2)             # verdadeiro  OU  falso        = verdadeiro
[1] T
```

O operador de negação **!**, quando aplicado a um vetor lógico, converte os valores T em F e vice-versa:

```
> aux <- notas.mat[,2] >= 90
> p2.cA <- p2[aux]
> p2.cB <- p2[!aux]
> p2.cA
 Ariadne Alana Agnes
      94    95    99
> p2.cB
 Joel Juan
   89   85
```

- **Ordenando os dados**

sort	ordena os valores de um vetor
rev	coloca os valores de um vetor na ordem inversa
order	retorna um vetor de inteiros contendo a permutação que irá ordenar o objeto de entrada em ordem crescente
rank	retorna os postos dos valores de um vetor

Exemplos:

```
> rank(c(2,4,5,1,7,6))
     [1] 2 3 4 1 6 5

> rank(c(2,2,5,1,7,6))
   [1] 2.5 2.5 4.0 1.0 6.0 5.0

> sort(c(2,4,5,1,7,6))
[1] 1 2 4 5 6 7

> rev(c(2,4,5,1,7,6))
[1] 6 7 1 5 4 2

> order(c(12,14,15,11,17,16))
[1] 4 1 2 3 6 5
```

Obs: O resultado dessa aplicação da função **order** indica que, para que o vetor dado seja ordenado em ordem crescente, devemos dispor inicialmente o quarto elemento (11), depois o primeiro elemento (12), ..., e finalmente o quinto elemento (17).

■

- **Extraindo valores repetidos**

unique	retorna os valores de um vetor sem qualquer repetição de valores
duplicated	retorna um vetor lógico indicando quais elementos são duplicatas de elementos anteriores

Exemplos:

```
> x <- rep(c(4,2,8,10,6), c(1,2,2,1,3))
> x
[1]  4  2  2  8  8 10  6  6  6

> unique(x)
[1]  4  2  8 10  6

> duplicated(x)
[1] F F T F T F F T T
```

■

Exercícios:

1) Seja a seguinte matriz de dados:

```
> mat <- matrix(c(1,3,2,2,5,4,6,5,6,9,7,8,10,12,11,11,12,14,14,16),ncol=5)
> mat
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    6   10   12
[2,]    3    4    9   12   14
[3,]    2    6    7   11   14
[4,]    2    5    8   11   16
```

Crie um nova matriz, **mat1**, que é a matriz **mat** com linhas ordenadas pela coluna 1:

```
> mat1 <- mat[order(mat[,1]),]
> mat1
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    6   10   12
[2,]    2    6    7   11   14
[3,]    2    5    8   11   16
[4,]    3    4    9   12   14
```

Crie um nova matriz, **mat2**, que é a matriz **mat** com linhas ordenadas pelas coluna 1 e, no caso de empate, pela coluna 2:

```
> mat2 <- mat[order(mat[,1],mat[,2]),]
> mat2
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    6   10   12
[2,]    2    5    8   11   16
[3,]    2    6    7   11   14
[4,]    3    4    9   12   14
```

2) Coloque os elementos do vetor x em ordem crescente e decrescente:

```
> x <- c(2,11,1,15,3,7,13,8,12,6)
> sort(x)
 [1]  1  2  3  6  7  8 11 12 13 15
> rev(sort(x))
 [1] 15 13 12 11  8  7  6  3  2  1
```

3) Seja o vetor x criado a seguir. Crie dois novos vetores: um vetor com os valores ordenados e não repetidos de x e outro vetor com os valores de x que aparecem pelo menos duas vezes em x.

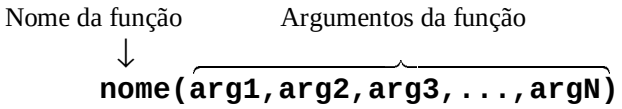
```
> x <- rep(c(4,2,8,10,6), c(1,2,2,1,3))
> x
[1]  4  2  2  8  8 10  6  6  6

> y <- sort(unique(x))
> y
[1]  2  4  6  8 10

> z <- unique(x[duplicated(x)])
> z
[1] 2 8 6
```

■

4- As funções S-PLUS

Todos os "comandos" que vimos anteriormente, como **`ls`**, **`scan`**, **`c`**, **`matrix`**, **`sort`**, são as *funções S-PLUS*. Uma função é uma expressão S-PLUS que retorna um valor, usualmente depois de realizar alguma operação em ou mais argumentos. Por exemplo, a função c retorna um vetor formado pela combinação dos argumentos para c.

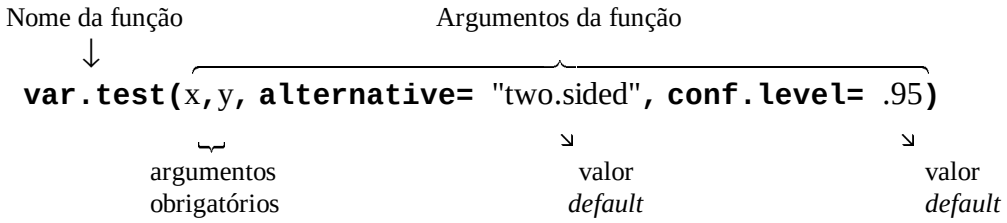
As funções têm a seguinte estrutura:



Assim, você pode "chamar" uma função digitando, na linha de comando, a expressão consistindo do nome da função seguido, entre parênteses, pelos argumentos separados por vírgulas. Alguns argumentos da função possuem um valor *default* e, a não ser que você queira mudar esse valor, não precisam ser especificados quando a função for chamada.

Exemplo:

Veja a função S-PLUS que realiza o teste F de comparação das variâncias de duas amostras independentes vindas de populações normais.



x , **y** vetores que contém as amostras a serem comparadas.
Os vetores não precisam ter os nomes **x** e **y** ao serem chamados na função; estes são apenas os nomes que eles terão durante a execução da função.

- Se você quiser fazer o teste bilateral e o um intervalo com nível de confiança de 95%, então basta fornecer as amostras **`sample1`** e **`sample2`**:

```
var.test(sample1, sample2)
```

- Se você quiser fazer o teste bilateral mas um intervalo com nível de confiança de 90%, então, além de fornecer as amostras **`x`** e **`y`**, você deve modificar o valor do argumento **`conf.level`**:

```
var.test(sample1, sample2, conf.level=.9)
```

- Nesta função, as opções para o argumento **`alternative`** são:
  - "two.sided": as variâncias da populações de **`x`** e **`y`** são diferentes,
  - "greater": a variância da população de **`x`** é maior que a variância da população de **`y`** e
  - "less": a variância da população de **`x`** é maior que a variâncai da população de **`y`**.

Se você quiser fazer um teste unilateral ("maior que") com um intervalo de 90% de confiança, o comando será:

```
var.test(sample1, sample2, alternative="greater", conf.level=.9)
```

ou

```
var.test(sample1, sample2, "greater", .9)
```

O último comando será aceito porque o argumentos **`alternative`** e **`conf.level`** estão em ordem, embora seus nomes não apareçam explicitamente. Note que o comando a seguir resutará em uma mensagem de erro:

```
var.test(sample1, sample2, .9, "greater")
```

Mensagem de erro: Error in var.test(sample1, sample2, 0.9, "greater"): argument 'alternative'
must match one of "greater", "less", "two.sided".
Dumped

Mas a mesma tarefa pode ser realizada através do comando onde, embora fora de ordem, os argumentos **`alternative`** e **`conf.level`** são referidos através de seus nomes:

```
var.test(sample1, sample2, conf.level=.9, alternative="greater")
```

■

O resultado de uma função S-PLUS pode ser númerico, lógico, complexo ou caracter, na forma de um dos tipos de objetos S-PLUS, como vetor, matriz, lista, etc. Assim, você poderá armazenar resultado da função em um objeto.

Exemplo:

Voltando à função **`var.test`**, sejam as seguintes amostras a serem comparadas:

```
> amostra1 <- c(2.7,  6.4, 7.4, 10.9, 16.1, 16.2, 17.7)
> amostra2 <- c(2.3,  4.4, 4.5,  6.9,  9.6, 11.5, 13.8)
```

Aplicando a função **`var.test`**, temos:

```
> var.test(amostra1,amostra2)

    F test for variance equality

data:  amostra1 and amostra2
F = 1.8943, num df = 6, denom df = 6, p-value = 0.4565
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
  0.3254957 11.0243991
sample estimates:
 variance of x variance of y
      33.48952      17.67905
```

Podemos atribuir o resultado da função a um objeto S-PLUS, neste caso uma lista:

```
> vtest1 <- var.test(amostra1,amostra2)
> vtest1

    F test for variance equality

data:  amostra1 and amostra2
F = 1.8943, num df = 6, denom df = 6, p-value = 0.4565
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
  0.3254957 11.0243991
sample estimates:
 variance of x variance of y
      33.48952      17.67905
```

E recuperar cada elemento da lista:

```
> vtest1[[1]]
        F
 1.894306

> vtest1[[2]]
 num df denom df
      6        6

> vtest1[[3]]
[1] 0.4564665

> vtest1[[4]]
[1]  0.3254957 11.0243991
attr(, "conf.level"):
[1] 0.95

> vtest1[[5]]
 variance of x variance of y
      33.48952      17.67905

> vtest1[[6]]
 ratio of variances
                  1
> vtest1[[7]]
[1] "two.sided"

> vtest1[[8]]
[1] "F test for variance equality"

> vtest1[[8]]
[1] "F test for variance equality"

> vtest1[[9]]
[1] "amostra1 and amostra2"
```

■

Uma função pode ser chamada dentro de outra função para combinar tarefas, como no exercício 2 da seção 3, onde usamos a função **`sort`** dentro da função **`rev`**. Podemos também criar nossas próprias funções no S-PLUS, como veremos na seção 8 .

Quando você digita apenas o nome da função na linha de comando, o texto da função é mostrado. Por exemplo, a função **`rev`** tem o seguinte texto:

```
> rev
function(x)
if(length(x)) x[length(x):1] else x
>
```

5- Distribuições de Probabilidade e Números Aleatórios

O S-PLUS tem muitas funções para realizar cálculos de probabilidade, incluindo geração de números aleatórios de determinadas distribuições de probabilidade.

Essas funções têm a seguinte forma geral:

letra***nome***(*argumentos separados por vírgula*)

↓ ↓

letra que indica a operação nome da distribuição de probabilidade

Tabela 4: Funções S-PLUS para geração de números aleatórios e cálculos de probabilidade

Letras	Operação	Argumentos necessários
r	Gera números aleatórios	Tamanho da amostra, parâmetros da distribuição
p	Calcula probabilidades da f.d.a.	Vetor de quantiles, parâmetros da distribuição
q	Calcula quantiles da inversa da f.d.a.	Vetor de probabilidades, parâmetros da distribuição
d	Calcula valores da densidade	Vetor de quantiles, parâmetros da distribuição

Tabela 5: Distribuições de probabilidade em S-PLUS

nome	**Distribuição**	**Argumentos Necessários**	**Argumentos Opcionais**	***Defaults***
beta	Beta	shape1, shape2		
binom	Binomial	size, prob		
cauchy	Cauchy		location, scale	0, 1
chisq	Qui-quadrado	df		
exp	Exponencial			
f	F	df1, df2	rate	1
gamma	Gama	shape		
geom	Geométrica	prob		
hyper	Hipergeométrica	m, n, k		
lnorm	Log-normal		meanlog, sdlog	0, 1
logis	Logística		location, scale	0, 1
nbinom	Binomial Negativa	size, prob		
norm	Normal		mean, sd	0, 1
pois	Poisson	lambda		
stab	***Stable***	index	skewness	--, 0
t	t de Student	df		
unif	Uniforme		min, max	0, 1
weibull	Weibull	shape	scale	--, 1
wilcox	Soma dos postos de Wilcoxon	m, n		

- A função *ppoints(n)* cria um vetor de n valores uniformente espaçados entre 0 e 1 (probabilidades), sendo útil para construir gráficos das distribuições (Veja exemplo da seção Gráficos - Funções *points* e *lines*).

Exercícios:

1) Gere 10 valores de uma Normal(média=25, desvio padrão=5)

```
> rnorm(10,25,5)
[1] 27.64515 20.76271 25.23715 30.93648 30.88803
[6] 28.91334 14.59414 23.49508 16.16100 26.15538
```

2) Seja Z ~ Normal(0,1)

a) Calcule a P(Z<-1.64)

```
> pnorm(-1.64,0,1)
[1] 0.05050258
```

b) Calcule a P(Z>1.96)

```
> 1- pnorm(1.96,0,1)
[1] 0.0249979
```

3) Seja Z ~ Normal(0,1)

a) Encontre k tal que P(Z<k)=0.05

```
> qnorm(0.05,0,1)
[1] -1.644854
```

b) Encontre K tal que P(Z>k)=0.025

```
> qnorm(1-0.025,0,1)
[1] 1.959964
```

4) Calcule P(X=5) onde X ~ Binomial(10,0.5)

```
> dbinom(5,10,0.5)
[1] 0.2460938
```

■

Função *sample*: Geração de amostras aleatórias ou permutação dos dados

A função sample é usada para:

- Tomar uma amostra aleatória (com ou sem reposição) de um vetor população x, permitindo que o usuário determine as probabilidades de selação de cada elemento em x;
- Gerar uma permutação do vetor x

Uso:

```
sample(x, size=<<veja abaixo>>, replace=F, prob=<<veja abaixo>>)
```

x vetor de dados numéricos, complexos ou caracteres a ser amostrado ou permutado (ou seja, a população).
Se x é um inteiro positivo, uma amostra ou permutação será tomada da seqüencia 1: x.

size = tamanho da amostra (*default*: comprimento de x)

replace = FALSE (amostragem sem reposição) *default*
= TRUE (amostragem com reposição)

prob = vetor de probabilidade de seleção de cada elemento de x
(*default*: probabilidades iguais)

Exemplos:

- Se o comando for apenas sample(x), sendo o tamanho de x maior que 1, teremos uma permutação de x.

```
>  potencia2 <- c(2, 4, 8, 16, 32, 64, 128, 256, 312, 624, 1248)
>  sample(potencia2)
 [1]  312  256   64   32   16  128  624 1248    4    2    8
```

Se x é um inteiro positivo, sample(x) resultará em uma permutação da seqüencia 1:x.

```
sample(10)
 [1]  9  7  1  6 10  4  8  2  3  5
```

- Se o comando for sample(x,n,T) ou sample(x,n), sendo n < comprimento de x, teremos, respectivamente, uma amostra aleatória com ou sem reposição de x

```
>  sample(potencia2,5,T)
[1] 312 256 624  64  64
>  sample(potencia2,5)
[1]  32 312 128   8   2
```

ou da seqüencia 1:x, se x é inteiro positivo

```
> sample(10,5,T)
[1] 3 8 4 8 4

> sample(10,5)
[1] 6 7 1 2 4
```

- Você pode definir uma função de probabilidade e gerar números desta distribuição usando a função *sample*:

```
> sample(4, 30, prob=c(0.1,0.2,0.3,0.4), replace=T)
[1] 2 4 1 3 3 3 3 4 4 4 4 2 3 4 4 3 4 3 2 3 3 3 2 1 4 2 4 3 1 1
```

- A função *sample* é útil na construção de rotinas para realizar *testes de permutação ou aleatorização*.

■

Exercícios:

1) Uma urna tem 10 bolas verdes, 8 bolas amarelas, 6 bolas azuis e 4 bolas brancas.
Retire, aleatoriamente e sem reposição, 6 bolas desta urna.

```
> sample(rep(c("verde", "amarela", "azul", "branca"), c(10,8,6,4)), 6, F)
[1] "verde"   "branca"  "amarela" "verde"   "branca"  "branca"
```
ou
```
> sample(c("verde", "amarela", "azul", "branca"), 6, T, c(10,8,6,4))
[1] "azul"    "amarela" "verde"   "amarela" "verde"   "verde"
```

2) Use a função *sample* para gerar 10 números aleatórios de uma distribuição Bernoulli(0.6).

```
> sample(0:1,10, T,c(0.4,0.6))
 [1] 0 1 0 0 1 1 1 0 1 1
```

3) Seja matriz **lx**:

```
> lx <- matrix(c(1:3,10*(1:3)),ncol=2)
> lx
     [,1] [,2]
[1,]    1   10
[2,]    2   20
[3,]    3   30
```

Crie uma nova matriz, **lx2**, que é a matriz **lx** com as linhas permutadas, mantendo a correspondência entre as colunas (Ex: 2 e 20 têm que estar na mesma linha).

```
> lx2 <- lx[sample(3),]
> lx2
     [,1] [,2]
[1,]    3   30
[2,]    1   10
[3,]    2   20
```

■

6- Algumas funções úteis na Análise Exploratória dos dados

Apresento aqui algumas funções para descrever, resumir, fazer transformações numéricas ou operações matemáticas com os dados.

- **Descrevendo e resumindo os dados**

Tabela 6: Funções S-PLUS mais comuns para resumo de dados

Função*	Descrição
min	retorna o menor elemento no objeto numérico
max	retorna o maior elemento do vetor
range	retorna min e max dos elementos do vetor
median	retorna a mediana dos valores
quantile	constrói quantiles para um conjunto de dados
mean	retorna a média
var	retorna a variância se o objeto é um vetor e a matriz de covariância se o objeto é uma matriz, considerando as colunas como variáveis
cor	retorna a a matriz de correlação da matriz, considerando as colunas como variáveis
sum	soma todos os elementos
prod	multiplica todos os elementos
cumsum	somas acumulativas dos elementos de um vetor
cumprod	produtos acumulativos dos elementos de um vetor

*Todas essas funções operam com objetos numéricos

A função **summary(x)**, aplicada a um objeto numérico, retorna os valores médio, mediano, mínimo e máximo, primeiro e terceiro quartis e, se for o caso, o número de obervações faltantes (NA).

```
> x <- rnorm(1000,0,1)
> summary(x)
   Min. 1st Qu.   Median     Mean 3rd Qu.  Max.
 -2.697 -0.6432 0.001229 0.003067  0.6828 3.339
```

A função **table(x1,x2,...,xn)**, retorna uma tabela de contingência com o mesmo número de dimensões quanto o número de argumentos dados. Os argumentos são n objetos de mesmo tamanho, podendo ser numéricos, caracteres ou lógicos.

```
> Pet <- factor(c("Cat","Dog","Cat","Dog","Cat","Cat"))
> Food <- factor(c("Dry","Dry","Dry","Wet","Wet","Wet"))
> table(Pet)
 Cat Dog
   4   2
> table(Food)
 Dry Wet
   3   3
> table(Pet,Food)
    Dry Wet
Cat   2   2
Dog   1   1
```

- **Funções matemáticas - Transformações numéricas nos dados**

Tabela 7: Funções matemáticas S-PLUS

Função*	Descrição
abs	valor absoluto
round	arredonda os valores no objeto numérico
trunc	arredonda os valores no objeto numérico
sqrt	raiz quadrada
exp	exponencial
log	logarítmo natural
gamma	função gamma
sin, cos, tan	seno, cosseno e tangente
asin, acos, atan	arco seno, arco cosseno e arco tangente

*Todas essas funções operam com objetos numéricos

- **Operações repetidas: a função `apply`**

Você pode usar a função **`apply`** para aplicar uma determinada função a todas as colunas ou linhas de uma matriz:

`apply(`*matriz*`, MARGIN, FUN, ...)`**

onde FUN nome da função a ser aplicada
... definição de argumentos opcionais da função FUN
MARGIN 1, para aplicar a função FUN em cada *linha* da matriz;
2, para aplicar a função FUN em cada *coluna* da matriz;

Exemplo:

Seja a matriz **`mat1`** a seguir:

```
> mat1 <- matrix(sample(20), ncol=4, byrow=T)
> mat1
     [,1] [,2] [,3] [,4]
[1,]    3   19    2   12
[2,]   20    1    4   10
[3,]    8   11   17    6
[4,]    9    7   18   16
[5,]   13   14   15    5
```

Vamos criar um vetor que receba a média de cada coluna de **`mat1`**:

```
> mean.mat1 <- apply(mat1, 2, mean)
> mean.mat1
[1] 10.6 10.4 11.2  9.8
```

e um vetor que receba a média truncada a 40% de cada coluna de **`mat1`**:

```
> meantruc.mat1 <- apply(mat1, 2, mean, trim=0.4)
> meantruc.mat1
[1]  9 11 15 10
```

■

Exercícios:

1) Sejam os seguintes conjuntos de dados:

```
> dresumo1a <- sample(10,10,T)
> dresumo1a
 [1] 8 6 6 9 1 6 3 5 8 4

> dresumo1b <- sample(c(T,F), 10, T)
> dresumo1b
 [1] F F F F F T T F T F

> dresumo1c <- sample(c("F","M"),10,T)
> dresumo1c
 [1] "F" "M" "F" "F" "F" "M" "M" "M" "F" "M"

> dresumo1d <- rnorm(10,0,1)
> dresumo1d
 [1] -0.3127440 -0.9258058  1.7417299 -0.2383481 -0.2323846
 [6]  0.6220618  0.3111528  0.4751760 -0.7212297  0.4575782
```

Veja o resultado da aplicação das funções **`summary`** e **`table`** em cada desses conjuntos de dados.

```
> summary(dresumo1a)
 Min. 1st Qu. Median Mean 3rd Qu. Max.
    1    4.25      6  5.6     7.5    9

> summary(dresumo1b)
 Length    Mode
 10     logical

> summary(dresumo1c)
 Length      Mode
 10     character

> summary(dresumo1d)
    Min. 1st Qu.  Median   Mean 3rd Qu.  Max.
 -0.9258 -0.2941 0.03938 0.1177  0.4708 1.742

> table(dresumo1a)
 1 3 4 5 6 8 9
 1 1 1 1 3 2 1

> table(dresumo1b)
 FALSE TRUE
     7    3
> table(dresumo1c)
 F M
 5 5

> table(dresumo1d)
 -0.9258058 -0.7212297 -0.3127440 -0.2383481 -0.2323846
          1          1          1          1          1
  0.3111528  0.4575782  0.4751760  0.6220618  1.7417299
          1          1          1          1          1
```

Veja o resultado do comando **`table(dresumo1b,dresumo1c)`**.

```
> table(dresumo1b,dresumo1c)
      F M
FALSE 4 3
 TRUE 1 2
```

2) Seja a seguinte matriz de dados imaginários onde cada linha representa uma criança. Na primeira coluna temos o código da família da criança e na segunda, a idade da criança em anos.

```
> dresumo2 <- matrix(c(rep(c(1,2,3,4,5,6,7,8),   c(2,3,1,5,3,2,3,1)),
                       c(2,4,4,4,1,3,5,2,3,1,1,4,5,3,4,1,4,3,3,4)),
              ncol=2)

> dimnames(dresumo2) <- list(1:20, c("Familia", "Idade"))
> dresumo2
   Familia Idade
 1       1     2
 2       1     4
 3       2     4
 4       2     4
 5       2     1
 6       3     3
 7       4     5
 8       4     2
 9       4     3
10       4     1
11       4     1
12       5     4
13       5     5
14       5     3
15       6     4
16       6     1
17       7     4
18       7     3
19       7     3
20       8     4
```

Veja o resultado da função **`table(dresumo2[,1]) .`**

```
> table(dresumo2[,1])

 1 2 3 4 5 6 7 8      Códigos de família (8 famílias diferentes)
 2 3 1 5 3 2 3 1      Número de crianças em cada família (soma=20)
```

Interprete o resultado da função **`table(table(dresumo[,1])).`**

```
> table(table(dresumo2[,1]))

 1 2 3 5              Número n de crianças por família
 2 2 3 1              Número de famílias com n crianças
```

⇨A primeira tabela responde perguntas do tipo:

"Quantas crianças tem a família 4" ?
Resposta: "Cinco crianças".

⇨A segunda tabela responde perguntas do tipo:

"Quantas famílias têm 5 crianças" ?
Resposta: "Apenas uma família".

3) Seja a matriz de dados construída a seguir:

```
> dresumo3 <- matrix(runif(20,1,10),ncol=4)
> dimnames(dresumo3) <- list(1:5, c("X1","X2","X3","X4"))
> dresumo3
        X1       X2       X3       X4
1 4.091309 1.922411 4.204570 2.931961
2 6.118130 7.349116 9.274226 2.529495
3 6.482045 5.497112 1.337260 9.730932
4 1.253018 5.434604 7.446593 9.972618
5 7.529399 8.376030 9.178902 7.471203
```

Teste algumas das funções descritivas da Tabela 6, como nos passos dados abaixo:

```
> apply(dresumo3,2,mean)
      X1       X2      X3       X4
 5.09478 5.715854 6.28831 6.527242

> apply(dresumo3,2,var)
       X1      X2       X3       X4
 6.166301 6.06531 11.86693 12.98333

> sqrt(apply(dresumo3,2,var))
       X1       X2       X3       X4
 2.483204 2.462785 3.444841 3.603239

> apply(dresumo3,2,summary)
           X1    X2    X3    X4
   Min. 1.253 1.922 1.337 2.529
1st Qu. 4.091 5.435 4.205 2.932
 Median 6.118 5.497 7.447 7.471
   Mean 5.095 5.716 6.288 6.527
3rd Qu. 6.482 7.349 9.179 9.731
   Max. 7.529 8.376 9.274 9.973

> cor(dresumo3)
            X1        X2          X3         X4
X1  0.99999994 0.5204546  0.02530422 -0.1949437
X2  0.52045465 1.0000001  0.62553924  0.2239809
X3  0.02530422 0.6255392  1.00000000 -0.2736661
X4 -0.19494371 0.2239809 -0.27366608  1.0000000
```

A função **`quantile`** aplicada a cada coluna da matriz, usando as probabilidades *default*:

```
> apply(dresumo3,2,quantile)
           X1       X2       X3       X4
  0% 1.253018 1.922411 1.337260 2.529495
 25% 4.091309 5.434604 4.204570 2.931961
 50% 6.118130 5.497112 7.446593 7.471203
 75% 6.482045 7.349116 9.178902 9.730932
100% 7.529399 8.376030 9.274226 9.972618
```

A função **`quantile`** aplicada a cada coluna da matriz, mas com as probabilidades modificadas no argumento **`probs`**:

```
> apply(dresumo3,2,quantile,probs=seq(0,1,0.20))
       X1       X2       X3       X4
  0% 1.253018 1.922411 1.337260 2.529495
 20% 3.523651 4.732166 3.631108 2.851468
 40% 5.307401 5.472109 6.149784 5.655506
 60% 6.263696 6.237914 8.139517 8.375094
 80% 6.691516 7.554499 9.197967 9.779269
100% 7.529399 8.376030 9.274226 9.972618
```

■

7- Gráficos

Os gráficos (histogramas, box plots, scatter plots, etc...) são mostrados em janelas gráficas que devem ser abertas no menu *Tools/Grafic Device*. Entretanto, os comandos para execução dos gráficos devem ser feitos na linha de coamando, como as funções.

Por *default*, apenas um gráfico será desenhado na janela. Se deseja-se desenhar mais de um gráfico na mesma janela, usa-se, antes de executar os comandos para desenhá-los, a função

`par(mfrow=c(`*nº de linhas*`,`***nº de colunas*`))`**,

imaginando-se a tela gráfica com uma matriz.

Exemplo: para desenhar seis gráficos na tela, dispostos em uma matriz 2x3, usamos o comando

```
> par(mfrow=c(2,3))
```

antes dos comandos que desenham os gráficos.

A tela gráfica continuará particionada desse modo até que se defina outra partição através do mesmo comando par(mfrow=...).

Alguns gráficos úteis na Análise Exploratória dos dados

- **Histograma:** `hist(x,probability=F,...)`

  x vetor numérico de dados para o histograma
  probability = TRUE, a altura das barras do histograma serão as densidades de probabilidade
  = FALSE, a altura das barras do histograma serão as contagens

Exemplo:

```
> hist(rt(100000, 100))
```
(Figura 1)

- **Box-plot:** `boxplot(x1,...,xn, ...)`

  x1,...,xn vetores com os dados. Produz n box-plots no mesmo gráfico

Exemplo:

```
> boxplot(rpois(50,10),rpois(50,20),rpois(50,15))
```
(Figura 2)

- **Série temporal:** `tsplot(x1,...,xn, ...)`

x1,...,xn vetores com os dados. Produz um gráfico com n séries temporais

Exemplo:

```
> ts1 <- sin(seq(-pi,pi,len=100)) + rnorm(100,0,0.1)
> ts2 <- cos(seq(-pi,pi,len=100)) + rnorm(100,0,0.1)
> tsplot(ts1,ts2)
```

(Figura 3)

- **Gráfico de dispersão**: `plot(x,y,...)`

x,y variáveis (vetores) para o gráfico de dispersão

Exemplo:

```
> x <- runif(30,1,50)
> y <- 2 + 0.5*x + rnorm(30,0,1)
> plot(x,y)
```

(Figura 4)

- **Gráfico de dispersão de pares de variáveis:** `pairs(x)`

x uma matriz, por exemplo

Exemplo:

```
> x1 <- runif(30,1,50)
> x2 <- runif(30,0,1)
> y <-  2 + 0.5*x1 + 4*x2 + rnorm(30,0,1)
> pairs(cbind(x1,x2,y))
```

(Figura 5)

- **Desenhando símbolos em um gráfico:**

Podemos fazer um gráfico de dados tridimensionais em duas dimensões codificando a terceira variável de acordo com o tamanho de um símbolo desenhado em cada localização x-y.

Os símbolos podem ser círculos, quadrados, retângulos, box-plots, etc.

`symbols(x,y,circles=,squares=,...)`

x,y **C**oordenadas X e Y dos pontos (vetores)

Exatamente um dos argumentos **`circles`**, **`squares`**, dentre outros, deve ser dado:

circles = vetor contendo o raio dos círculos
squares = vetor contendo o comprimento do lado do quadrado

Exemplo:

Coordenadas da localização certa espécie de árvore em um parque:

```
> x <- runif(20,0,100)
> y <- runif(20,0,100)
```

Diâmetro (em centímetros) do tronco da árvore:

```
> z <- round(abs(rnorm(20,100,50)))
```

Mapa com a localização das árvores e representação de seu diâmetro:

```
> symbols(x,y,circle=z,inches=0.5)
```

(Figura 6)

Figura 1: Exemplo de aplicação da função *hist*

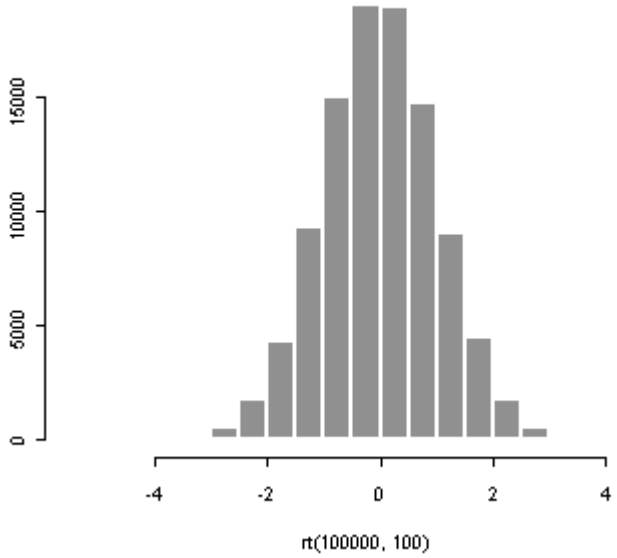


Figura 2: Exemplo de aplicação da função *boxplot*

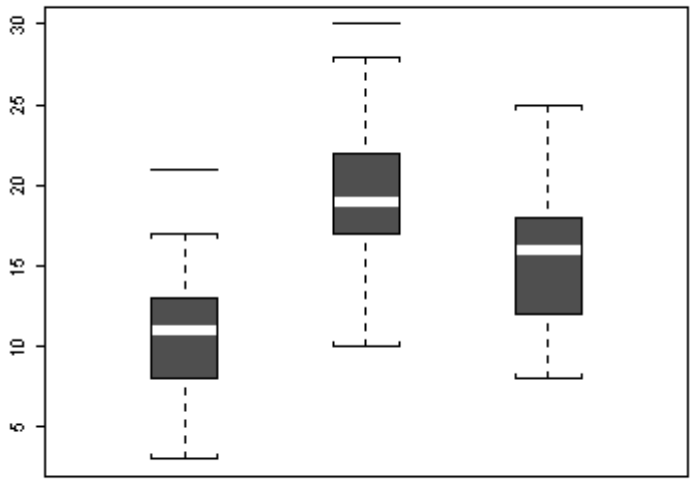


Figura 3: Exemplo de aplicação da função *tsplot*

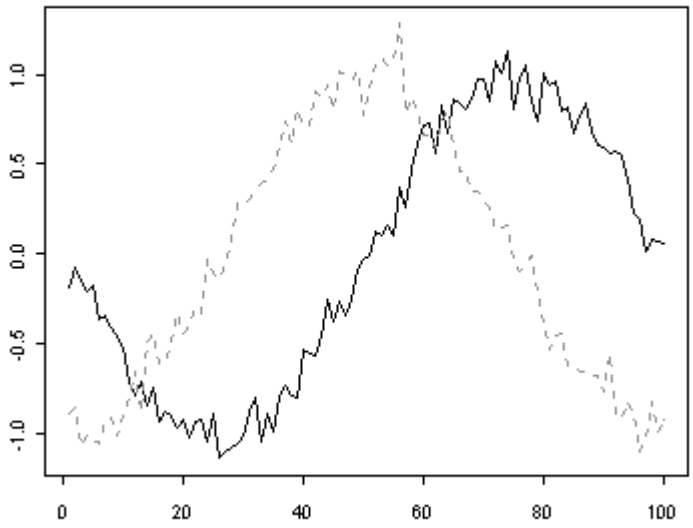


Figura 4: Exemplo de aplicação da função *plot*

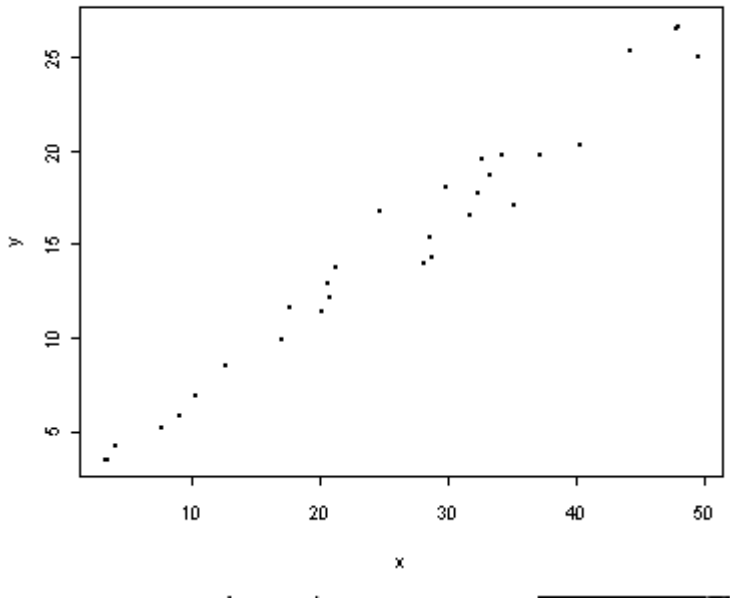


Figura 5: Exemplo de aplicação da função *pairs*

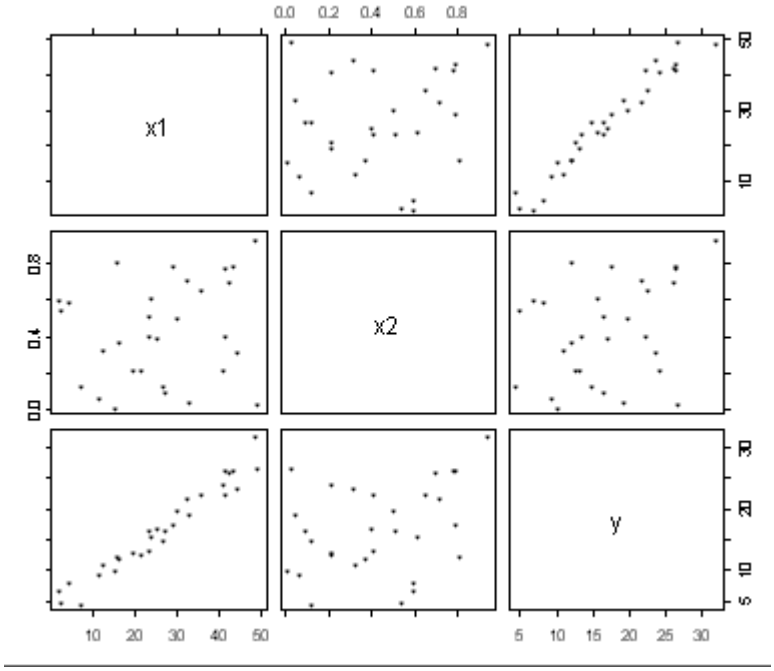
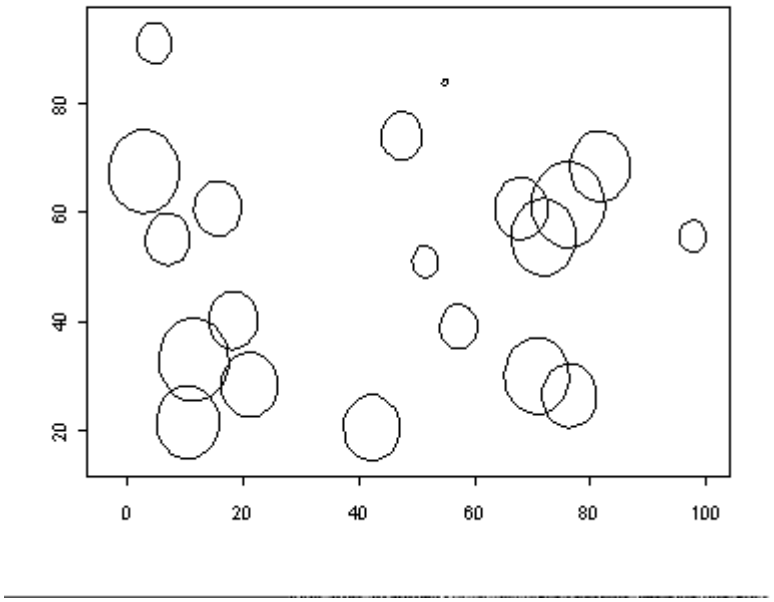


Figura 6: Exemplo de aplicação da função *symbols*



Função *abline*: Adicionando uma linha ao gráfico corrente

Adiciona a linha y=a+b*x ou linhas horizontais e verticais ao gráfico corrente.

Uso:

Após o comando que desenha o gráfico, em uma das seguintes formas:

```
abline(a,b)
abline(coef)
abline(reg)
abline(h=,v=)
```

a,b intercepto e coeficiente de inclinação da linha a ser desenhada no gráfico.

coef vetor contendo intercepto **a** e coeficiente de inclinação **b** da reta y=a+b*x.

reg um objeto de regressão, tal com o retornado pelas funções *lsfit* ou *lm*.

h = vetor de coordinadas y para as linhas horizontais a serem desenhadas no gráfico.
v = vetor de coordinadas x para as linhas verticais a serem desenhadas no gráfico.

Exemplos:

- Gere 100 pontos com coordenadas X ~ Uniforme(1,100) e Y= 5 + 0.9*X + Normal(0,10). Faça o gráfico de dispersão dos pontos e desenhe a linha com intercepto igual a 5 e coeficiente de inclinação igual a 0.9 (Figura 7).

```
> X <- runif(100,1,100)
> Y <- 5 + 0.9*X + rnorm(100,0,10)
> plot(X,Y)
> abline(5,0.9)
```

- Desenhe a reta de regressão ajustada dos Y nos X do exemplo anterior (Figura 8)

```
> plot(X,Y)
> abline(lm(Y~X))
```

Obs: A função *lm* ajusta o modelo de regressão linear.

- Faça o histograma dos Y e desenhe um linha vertical na média dos valores (Figura 9)

```
> hist(Y)
> abline(v=mean(Y))
```

- Desenhe 100 pontos com coordenadas X e Y escolhidas em uma distribuição Uniforme(0,100) e sobreponha ao gráfico uma grade com dez linhas horizontais e dez linhas verticais (Figura 10)

```
> plot(runif(100,0,100),runif(100,0,100))
> abline(v=c(1:10)*10, h=c(1:10)*10)
```

Figura 7

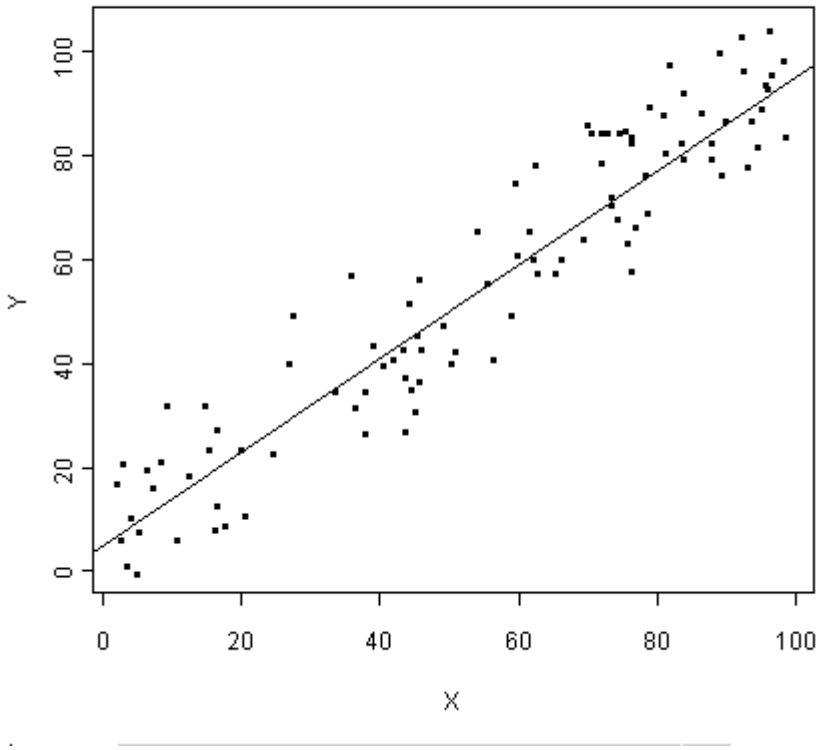


Figura 8

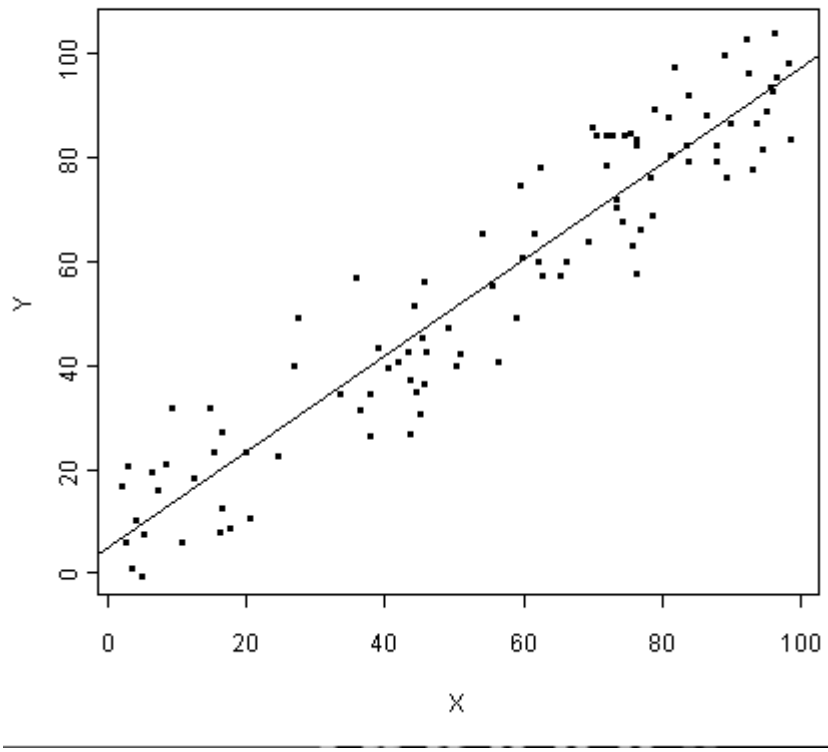


Figura 9

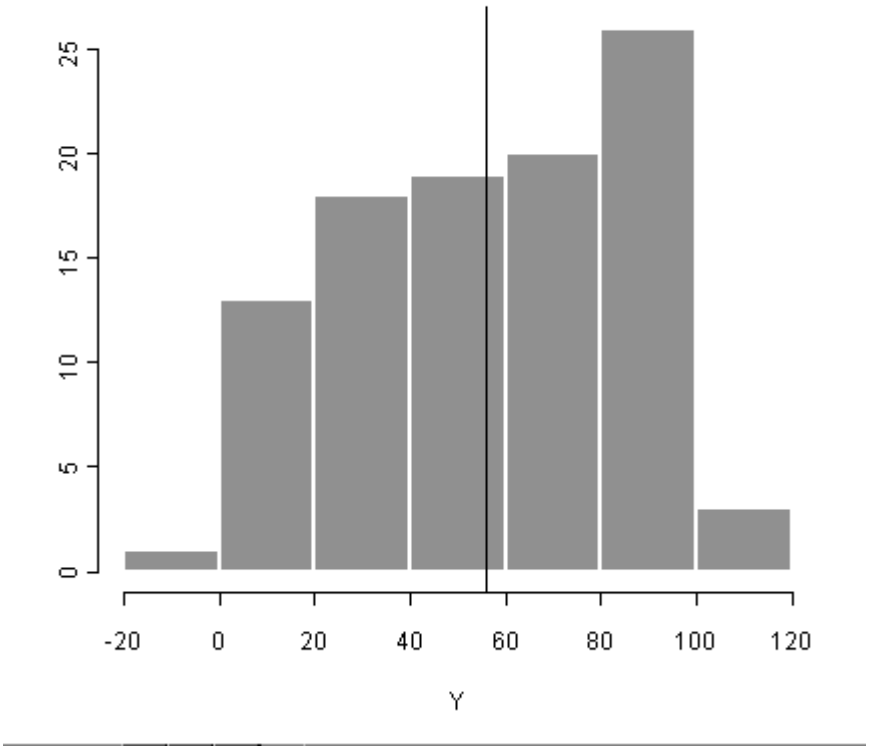
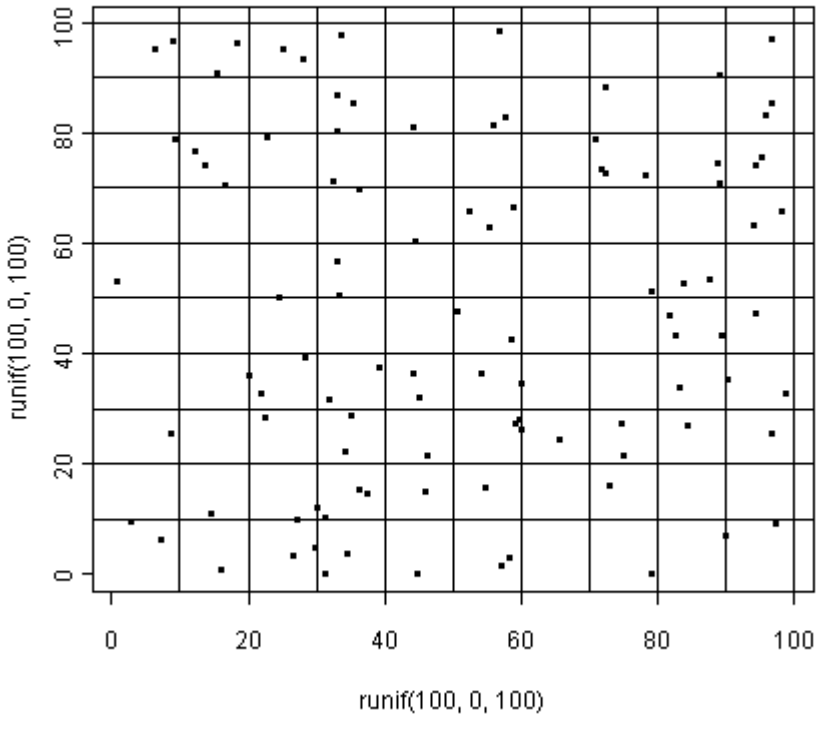


Figura 10



Funções *points* e *lines*: Adicionando pontos ou linhas ao gráfico corrente

`points(x,y)` Adiciona pontos ao gráfico corrente
`lines(x,y)` Adiciona pontos conectados com segmentos de linha ao gráfico corrente

x,y **C**oordenadas X e Y dos pontos.

As coordenadas podem ser dadas, dentro outros, por:

- dois argumentos que são vetores,
- um único argumento que é uma matriz com duas colunas ou
- um único argumento que é uma série temporal univariada.

Exemplo:

Vamos inicialmente construir um histograma (usando **`prob=T`**, ou seja, um histograma da densidade de probabilidade) de uma amostra aleatória de tamanho 100 mil da distribuição Normal(0,1):

```
> amostra <- rnorm(100000,0,1)

> par(mfrow=c(3,1))
> hist(amostra,prob=T)
```

Deseja-se um esboço da curva de densidade de probabilidade da Normal(0,1) desenhado sobre este histograma.

Precisamos, assim, definir quais pontos serão desenhados:

coordenadas (x,y): (quantiles da N(0,1), respectivos valores na densidade N(0,1)).

Vamos inicialmente criar um vetor com os valores de probabilidades para estes pontos:

```
> p <- seq(0.005,0.995,0.010)

> p
 [1] 0.005 0.015 0.025 0.035 0.045 0.055 0.065 0.075 0.085 0.095
[11] 0.105 0.115 0.125 0.135 0.145 0.155 0.165 0.175 0.185 0.195
[21] 0.205 0.215 0.225 0.235 0.245 0.255 0.265 0.275 0.285 0.295
[31] 0.305 0.315 0.325 0.335 0.345 0.355 0.365 0.375 0.385 0.395
[41] 0.405 0.415 0.425 0.435 0.445 0.455 0.465 0.475 0.485 0.495
[51] 0.505 0.515 0.525 0.535 0.545 0.555 0.565 0.575 0.585 0.595
[61] 0.605 0.615 0.625 0.635 0.645 0.655 0.665 0.675 0.685 0.695
[71] 0.705 0.715 0.725 0.735 0.745 0.755 0.765 0.775 0.785 0.795
[81] 0.805 0.815 0.825 0.835 0.845 0.855 0.865 0.875 0.885 0.895
[91] 0.905 0.915 0.925 0.935 0.945 0.955 0.965 0.975 0.985 0.995
```

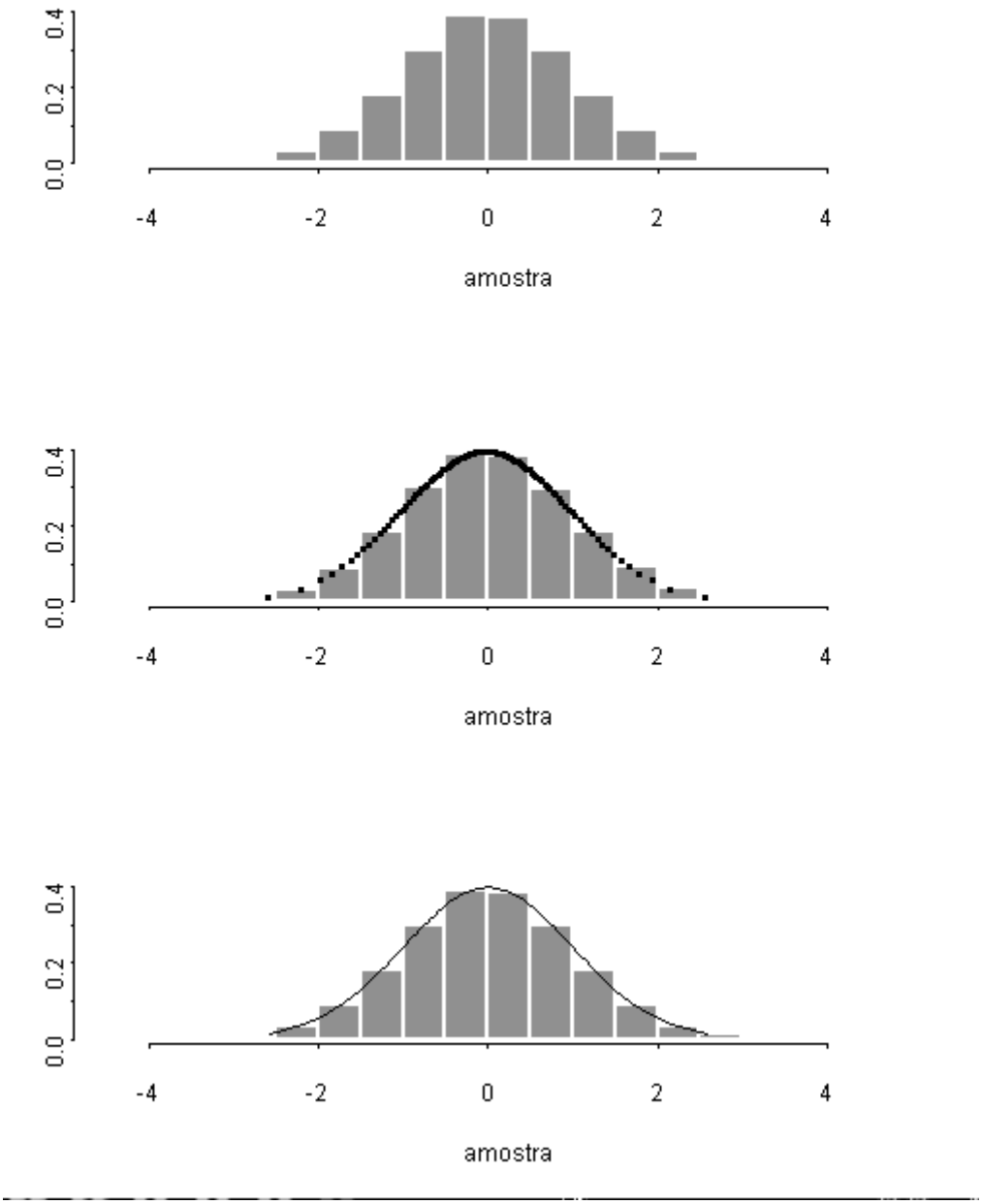
Agora vamos criar as coordenadas dos pontos da Normal(0,1) que serão desenhados:

```
> x <- qnorm(p,0,1)
> y <- dnorm(qnorm(p,0,1),0,1)
```

E agora finalmente esboçar a curva da densidade da Normal(0,1) com pontos ou linhas:

```
> hist(amostra,prob=T)
> points(x,y)

> hist(amostra,prob=T)
> lines(x,y)
```

Figura 11: Exemplos de aplicação das funções *ppoints* e *lines*



Observação: a seqüência em **p** também poderia ser criada através da função **ppoints(100)**. A vantagem de se usar a função **points** está na simplicidade: automaticamente são gerados valores entre 0 e 1 (como requerido para probabilidades), necessitando-se apenas definir o número de pontos desejados, o que está relacionado com a precisão do desenho.

Funções *segments* e *arrows*: Desenhando segmentos de linha deconectados ou flechas no gráfico corrente

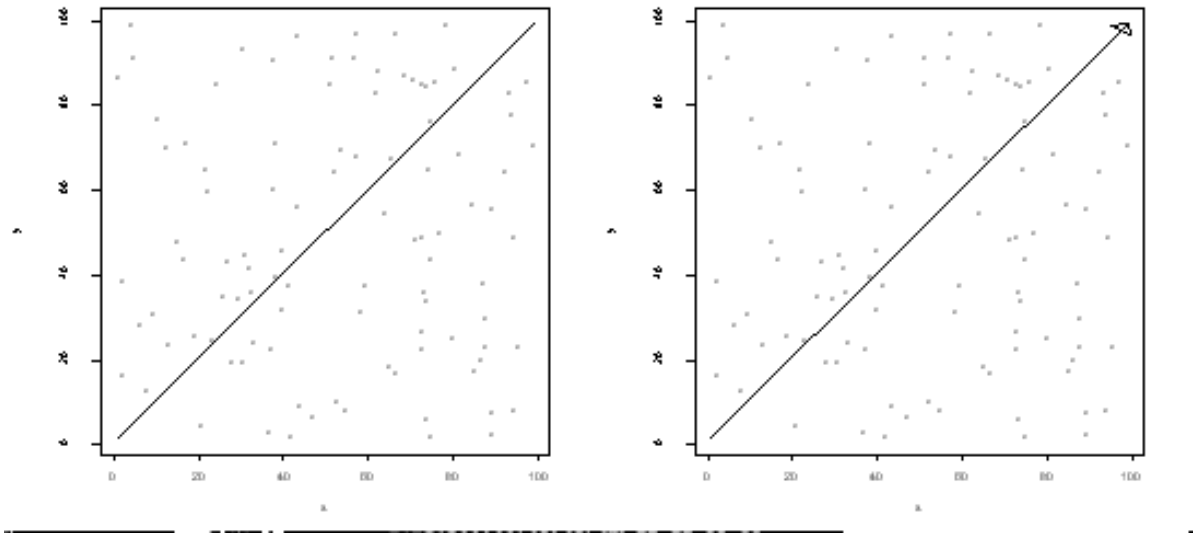
`segments(x1,y1,x2,y2)` Adiciona segmentos de linha ao gráfico corrente
`arrows(x1,y1,x2,y2)` Adiciona flechas ao gráfico corrente

x1,y1,x2,y2 coordenadas X e Y dos pontos extremos dos segmentos ou flechas. Linhas serão desenhadas de (x1[i],y1[i]) a (x2[i],y2[i]).

Exemplos:

```
> x <- runif(100,1,100)
> y <- runif(100,1,100)

> par(mfrow=c(2,2))

> plot(x,y)
> segments(min(x),min(y),max(x),max(y))

> plot(x,y)
> arrows(min(x),min(y),max(x),max(y))
```

Figura 12: Exemplos de aplicação das funções *segments* e *arrows*



8- Escrevendo funções S-PLUS

Uma das grandes vantagens do S-PLUS é que ele nos permite escrever nossas próprias funções na linguagem S-PLUS. Isso o torna uma ferramenta poderosa para testar novas metodologias e realizar simulações.

A nova função S-PLUS que você construir poderá ser completamente nova (um novo estimador que você está propondo, por exemplo) ou apenas uma modificação personalizada de uma função S-PLUS existente. Você pode desejar ainda usar as funções já existentes de modo repetido no seu conjunto de dados - seu trabalho será facilitado incorporando estas tarefas em uma única função.

Vamos ver agora os conceitos básicos da construção de funções S-PLUS e alguns exemplos simples. Para uma discussão completa, veja o *S-PLUS Programmer's Manual*.

• Sintaxe geral

A sintaxe geral para definir uma nova função a partir da linha de comando é:

nome **<- function(** *argumentos* **) {***corpo* **}**

nome: nome que você escolhe para a função
argumentos: os argumentos necessários para a função, separados por vírgula
corpo: conjunto de comandos (tarefas) necessários para que a função retorne o valor desejado

Exemplo:

Vamos construir uma função que, dado um vetor de valores, retorne os valores padronizados.

```
> padroniza <- function(x)
 {
 z <- (x-mean(x))/sqrt(var(x))
 z
 }
```

Testando:

```
> x <- 1:7
> z <- padroniza(x)
> z
[1] -1.3887301 -0.9258201 -0.4629100  0.0000000  0.4629100  0.9258201  1.3887301
```

Obervações sobre função **padroniza**:

x é o único argumento da função.
z Note que **z** não foi criado no diretório de dados.

O último objeto que se faz referência antes de "fechar a função" com } é o objeto que será retorna pela função. Nesse caso o objeto **z**.

Tendo visto esse exemplo bem simples, vamos generalizar algumas observações sobre os elementos da sintaxe geral

nome **<- function(** *argumentos* **) {***corpo* **}.**

1. Você pode colocar valores *default* para alguns ou todos os argumentos colocando na frente do nome do argumento a expressão =*valordefault*. Os argumentos que não tiverem um valor *default* são os argumentos obrigatórios ao chamar a função.

2. Os objetos criados dentro do corpo da função são *locais* àquela função, ou seja, só existem na memória apenas durante a execução da função. Assim, você pode dar nomes a objetos dentro da função iguais a objetos no diretório de dados.

3. Você pode colocar comentários no corpo da função usando o símbolo **#** antes do comentário.

4. A função irá retornar apenas um objeto, que pode ser um vetor, uma matriz ou uma lista, etc. O nome do objeto a ser referenciado deve aparecer (sozinho) na última linha antes do **}** final. Se nenhum nome aparecer no final do corpo da função, será retornado valor da última expressão avaliada no corpo da função.

Exemplo:

Vamos fazer uma função para realizar o Teste t de comparação de duas amostras independentes. Para simplicidade inicial, o teste deverá ser unilateral:

H_1: média da primeira amostra > média da segunda amostra

O teste deve retornar:

- o tamanho das amostras;
- a média e o desvio padrão das duas amostras;
- a diferença entre as duas médias e o desvio padrão combinado ("s_pooled");
- o valor da estatítica de teste t e o valor de comparação na distribuição t nula;
- o valor p do teste e o nível de significância escolhido (com *default*=0.05).

```
ttest <- function(y1,y2,alpha=0.05)
{
#y1 e y2: vetores com a primeira e a segunda amostra, respectivamente
#alpha: nivel de significancia do teste (default=0.05)

n1 <- length(y1)
n2 <- length(y2)
ngl <- n1 + n2 -2                                  # numero de graus de liberdade da distribuicao t sob Ho
sp <- sqrt( ((n1-1)*var(y1) + (n2-1)*var(y2))/ngl)  # "s_pooled"
tstat <- (mean(y1)-mean(y2))/(sp*sqrt(1/n1+1/n2))  # valor da estatística de teste
tc <- qt(0.95,ngl)                                 # valor de comparacaao sob Ho
valorp <- 1-pt(tstat,ngl)                          # valor p

# Vamos agrupar todos os resultados requeridos em uma lista:

resultado <- list( c(n1,n2), c(mean(y1),mean(y2)), c(sqrt(var(y1)),sqrt(var(y2))),
                   c(mean(y1)-mean(y2), sp), c(tstat,tc), c(valorp, alpha))
}
```

Aplicando em duas amostras independentes geradas de distribuições Normais:

```
> result <- ttest(rnorm(30,12,5),rnorm(40,8,5),alpha=0.10)
> result
[[1]]:
[1] 30 40

[[2]]:
[1] 13.903936  6.410418

[[3]]:
[1] 3.230817 5.021282

[[4]]:
[1] 7.493518 4.348808

[[5]]:
[1] 7.134394 1.667572

[[6]]:
[1] 4.06001e-010 1.00000e-001
```

• Expressando condições

Em algumas funções que construímos é necessário avaliar condições para que uma determinada tarefa seja realizada, especialmente dentro de *loops* (veja próxima seção).

Os cálculos condicionais são feitos através da conhecida construção **`if-else`**.

```
      if ( condição1)  { tarefa a ser realizada se condição1 = TRUE }
else if ( condição2)  { tarefa a ser realizada se condição2 = TRUE }
else if ( condição3)  { tarefa a ser realizada se condição3 = TRUE }
                              ...
 else                 { tarefa a ser realizada se todas as condições = FALSE}
```

onde *condição* é uma expressão que resulta em um *único valor lógico* (T ou F).

Assim, as condições 1,2,..., N são mutuamente excludentes, e uma certa condição será realizada se todas as outras forem FALSE. Mas podemos ter estruturas mais simples, como no exemplo a seguir.

Exemplo:

Você pode usar uma expressão condicional **`if`** simples, sem **`else`**:

```
if(mean(x) > median(x))  {estimativa <- y}
```

ou colocar uma alternativa para o caso em que a condição seja falsa:

```
 if(mean(x) > median(x))   {estimativa <- mean(x)}
else                       {estimativa <- median(x)}
```

Podemos testar múltiplas condições usando os operadores & e | representam E e OU, respectivamente:

`if (` *condição1* **`&`** *condição2***`)`** **`{`** *tarefa a ser realizada se as duas condições são TRUE* **`}`**
`if (` *condição1* **`|`** *condição2***`){`** *tarefa a ser realizada se ao menos uma das condições é TRUE* }

Exemplo:

Você pode usar uma expressão condicional **`if`** simples, sem **`else`**:

```
     if(idade < 10)                {idade2 <- 1}
else if(idade >= 10& idade < 20)  {idade2 <- 2}
else if(idade >= 20& idade < 40)  {idade2 <- 3}
else if(idade >= 40& idade < 60)  {idade2 <- 4}
else                              {idade2 <- 5}
```

• Iteração

Voce pode desejar fazer uma série de tarefas repetidas dentro de uma função (ou até mesmo fora). As tarefas iterativas (*loops*) podem ser feitas através dos comandos **`for`**, **`while`** e **`repeat`**.

➡ O comando **`for`** permite que uma tarefa seja repetida à medida que uma variável assume valores em uma seqüência específica:

`for (` *variável* **`in`** *seqüência***`)`** **`{`** *tarefas* **`}`**

Exemplo:

Vamos escrever uma função que gere um vetor com as somas acumuladas dos elementos de um vetor dado (tarefa da função **`cumsum`**, já vista).

```
soma.fun <- function(x)
{
n <- length(x)
soma <- NULL          #vetor onde serão armazenadas as somas
y <- 0
for (i in 1:n)        #Percorrendo cada elemento do vetor x
{
  y <- y + x[i]
 soma <- c(soma, y)
}
soma
}
```

```
> soma.fun(1:5)
[1]  1  3  6 10 15
```

Exemplo:

Seja a seguinte matriz:

```
> blocoV <- matrix(1:25, ncol=5, byrow=T)
> blocoV
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
[4,]   16   17   18   19   20
[5,]   21   22   23   24   25
```

Vamos criar um *loop* que gere uma nova matriz **V** (15x15), bloco diagonal, onde os blocos são todos iguais à matriz **blocoV**:

```
V <- matrix(0, 15, 15)
for (i in seq(1, 15, by=5))
{
V[c(i:(i+4)),c(i:(i+4))] <- blocoV
}
```

```
> V
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15]
 [1,]    1    2    3    4    5    0    0    0    0     0     0     0     0     0     0
 [2,]    6    7    8    9   10    0    0    0    0     0     0     0     0     0     0
 [3,]   11   12   13   14   15    0    0    0    0     0     0     0     0     0     0
 [4,]   16   17   18   19   20    0    0    0    0     0     0     0     0     0     0
 [5,]   21   22   23   24   25    0    0    0    0     0     0     0     0     0     0
 [6,]    0    0    0    0    0    1    2    3    4     5     0     0     0     0     0
 [7,]    0    0    0    0    0    6    7    8    9    10     0     0     0     0     0
 [8,]    0    0    0    0    0   11   12   13   14    15     0     0     0     0     0
 [9,]    0    0    0    0    0   16   17   18   19    20     0     0     0     0     0
[10,]    0    0    0    0    0   21   22   23   24    25     0     0     0     0     0
[11,]    0    0    0    0    0    0    0    0    0     0     1     2     3     4     5
[12,]    0    0    0    0    0    0    0    0    0     0     6     7     8     9    10
[13,]    0    0    0    0    0    0    0    0    0     0    11    12    13    14    15
[14,]    0    0    0    0    0    0    0    0    0     0    16    17    18    19    20
[15,]    0    0    0    0    0    0    0    0    0     0    21    22    23    24    25
```

■

➡ O comando **while** permite que uma tarefa seja repetida enquanto uma *condição* (expressão com resultado lógico) é verdadeira:

while (*condição***) {** *tarefas* **}**

Exemplo: (retirado de Venables and Ripley (1994), página 88)

Considere o problema de encontrar o estimador de máxima verossimilhança do parâmetro λ da distribuição de Poisson truncada em zero:

$$P(Y=y)=\frac{e^{-\lambda}\lambda^{y}}{(1-e^{-\lambda})y!} \qquad y=1,2,...$$

que corresponde a observar apenas valores não nulos da distribuição de Poisson.

A média é

$$E(Y) = \frac{\lambda}{1-e^{-\lambda}},$$

e a estimativa de máxima verossimilhança $\hat{\lambda}$ é encontrada igualando-se a média amostral à sua esperança:

$$\bar{y} = \frac{\hat{\lambda}}{1-e^{-\hat{\lambda}}}, \qquad \text{expressão que pode ser reescrita como} \quad \hat{\lambda} = \bar{y}(1-e^{-\hat{\lambda}}).$$

Assim, o Método de Newton-Raphson, que vamos implementar, leva ao seguinte esquema de iteração:

$$\hat{\lambda}_{m+1} = \hat{\lambda}_m - \frac{\hat{\lambda}_m - \bar{y}(1-e^{-\hat{\lambda}_m})}{1-\bar{y}e^{-\hat{\lambda}_m}},$$

que pode ser reescrito como

$$\hat{\lambda}_{m+1} = \hat{\lambda}_m - \Delta_m.$$

Inicialmente, vamos gerar uma amostra da distribuição de Poisson(λ = 1) e eliminar todos os zeros para obtermos uma amostra da distribuição de Poisson truncada em zero:

```
> y <- rpois(50,1)      # amostra de tamanho 50 da Poisson(1) completa

> table(y)
  0  1  2 3 4
 19 15 11 3 2

> y <- y[y > 0]         # amostra de tamanho 31 da Poisson(1) truncada em zero
```

Vamos ter uma condição de parada baseada na convergência ($\Delta_m < 0.0001$) e no número máximo de iterações que aceitamos (10). O valor inicial será $\hat{\lambda}_0 = \bar{y}$.

```
> ybar <- mean(y)
> ybar
[1] 1.741935

> lambda <- ybar
> delta <- 1
> itera <- 0

> while (abs(delta) > 0.0001 & (itera <- itera  + 1) < 10)
  {
  delta <- (lambda - ybar*(1- exp(-lambda)))/(1- ybar*exp(-lambda))
  lambda <- lambda - delta
  cat(itera, delta, lambda, "\n")            #comando para imprimir
  }

1 0.439167859831138 1.30276762403983
2 0.065047900315307 1.23771972372452
3 0.002068981399284 1.23565074232524
4 0.000002191817381 1.23564855050786
```

■

➡ O comando **`repeat`** permite que uma tarefa seja repetida indefinidadamente, a não ser que a condição no comando **`break`** seja satisfeita:

`repeat {` *tarefas* (*incluindo avaliação para* **`break`**) **`}`**

Exemplo:

No exemplo anterior, o *loop* pode ser feito usando **`repeat`**:

```
> lambda <- ybar
> delta <- 1
> itera <- 0

> repeat
  {
  delta <- (lambda - ybar*(1- exp(-lambda)))/(1- ybar*exp(-lambda))
  lambda <- lambda - delta
  cat(itera, delta, lambda, "\n")
  if(abs(delta) <= 0.0001 | (itera <- itera  + 1) >= 10) break
  }

1 0.439167859831138 1.30276762403983
2 0.065047900315307 1.23771972372452
3 0.002068981399284 1.23565074232524
4 0.000002191817381 1.23564855050786
```

■

9- Considerações Finais

No Apêndice, são apresentados dois exemplos de arquivo de *Help*: funções **`mean`** e **`hist`**. Os arquivos de *Help* do S-PLUS seguem essa estrutura: uma breve descrição da função, seguida pela sintaxe e descrição dos argumentos, os valores retornados, detalhes da função, referências do assunto (se for o caso), indicação de tópicos relacionados e exemplos de uso.

Também no Apêndice, apresento uma cópia do índice do manual *S-PLUS Guide to Statistical and Mathematical Analysis,* que mostra as técnicas estatísticas implementadas naquela versão do S-PLUS. Atualmente, outras técnicas foram implementadas no S-PLUS, seja através de novos módulos (como o módulo para Estatística Espacial) ou das bibliotecas de funções S-PLUS criadas para implementar determinado tipo de análise estatística (como o OSWALD, para Análise de Dados Longitudinais).

10- S-PLUS na Internet

- **Informações e compra:**

`www.mathsoft.com/splus.html/`

- **Cursos e dicas:**

Introdution to S-PLUS (University of Toronto):
`utstat.toronto.edu/splus/contents.html`

Frequently Asked Questions about S (ETH Zurich):
`www.stat.math.ethz.ch/S-FAQ/`

Info on running S-PLUS (University of Wisconsin):
`www.stat.wisc.edu/computing/splus/`

- **Lista eletrônica:**

Tire suas dúvidas sobre S-PLUS com outros usuários ou resolva algumas via *e-mail* através da lista eletrônica *S-News*. Pare se inscrever na lista, mande uma mensagem para:

`s-news-request@utstat.toronto.edu`

As mensagens já enviadas estão em:

`www.stat.cmu.edu/s-news/`

Apêndice

- Exemplos do conteúdo do *Help on Line*
- Conteúdo do manual *Guide to Statistical and Mathematical Analysis*

(cópias xerox a parte)