Minicurso de Introdução ao R

João Batista M. Pereira
DME - UFRJ

joao@dme.ufrj.br

V Semana de Ciências Atuariais e Estatística - UFRJ

3 a 5 de dezembro de 2014

Sobre o R

- O R é uma linguagem interativa para computação estatística.

Sobre o R

- O R é uma linguagem interativa para computação estatística.
- Possui uma vasta biblioteca de funções estatísticas e matemáticas, além de uma extensa variedade de técnicas gráficas simples e sofisticadas.

Sobre o R

- O R é uma linguagem interativa para computação estatística.
- Possui uma vasta biblioteca de funções estatísticas e matemáticas, além de uma extensa variedade de técnicas gráficas simples e sofisticadas.
- Conta com uma crescente variedade de pacotes para os mais diversos tipos de análises de dados.

Sobre o R

- O R é uma linguagem interativa para computação estatística.
- Possui uma vasta biblioteca de funções estatísticas e matemáticas, além de uma extensa variedade de técnicas gráficas simples e sofisticadas.
- Conta com uma crescente variedade de pacotes para os mais diversos tipos de análises de dados.
- O R começou a ser desenvolvido por Robert Gentleman e Ross Ihaka, do Departamento de Estatística da Universidade de Auckland, Nova Zelândia.

Sobre o R

- O R é uma linguagem interativa para computação estatística.
- Possui uma vasta biblioteca de funções estatísticas e matemáticas, além de uma extensa variedade de técnicas gráficas simples e sofisticadas.
- Conta com uma crescente variedade de pacotes para os mais diversos tipos de análises de dados.
- O R começou a ser desenvolvido por Robert Gentleman e Ross Ihaka, do Departamento de Estatística da Universidade de Auckland, Nova Zelândia.
- Por que  ?

Sobre o R

- O R é uma linguagem interativa para computação estatística.
- Possui uma vasta biblioteca de funções estatísticas e matemáticas, além de uma extensa variedade de técnicas gráficas simples e sofisticadas.
- Conta com uma crescente variedade de pacotes para os mais diversos tipos de análises de dados.
- O R começou a ser desenvolvido por Robert Gentleman e Ross Ihaka, do Departamento de Estatística da Universidade de Auckland, Nova Zelândia.
- Por que  ?
- *Robert Gentleman e Ross Ihaka.*

Sobre o R

- Em 1991, o objetivo inicial de "R & R" era produzir um software para as suas aulas de laboratório baseado na revolucionária linguagem S, utilizada pelo software comercial S-Plus.

Sobre o R

- Em 1991, o objetivo inicial de "R & R" era produzir um software para as suas aulas de laboratório baseado na revolucionária linguagem S, utilizada pelo software comercial S-Plus.
- O S-Plus foi criado por Jonh Chambers, da AT&T, e alguns colegas, que hoje em dia é um dos colaboradores que atuam no desenvolvimento e aperfeiçoamento das análise estatísticas no R.

Sobre o R

- O primeiro relato da distribuição do R foi em 1993, quando algumas cópias foram disponibilizadas no StatLib, um sistema de distribuição de *softwares* estatísticos.

Sobre o R

- O primeiro relato da distribuição do R foi em 1993, quando algumas cópias foram disponibilizadas no StatLib, um sistema de distribuição de *softwares* estatísticos.
- Em 1995, com o incentivo de um dos primeiros usuários deste programa, Martin Mächler, do Instituto Federal de Tecnologia de Zurique, Suíça, "R & R" disponibilizaram o código fonte do R.

Sobre o R

- O primeiro relato da distribuição do R foi em 1993, quando algumas cópias foram disponibilizadas no StatLib, um sistema de distribuição de *softwares* estatísticos.
- Em 1995, com o incentivo de um dos primeiros usuários deste programa, Martin Mächler, do Instituto Federal de Tecnologia de Zurique, Suíça, "R & R" disponibilizaram o código fonte do R.
- Em 1997 foi formado um grupo de profissionais que têm acesso ao código fonte do R, possibilitando assim a atualização mais rápida do *software*.

Sobre o R

- O R é um *software* livre e pode ser obtido em http://www.r-project.org.
- Na opção CRAN, selecione o servidor do qual pretende baixar o *software*
- Escolha a plataforma (Linux, MAC ou Windows) e obedeça as instruções de *download*.

Sobre o R

Vantagens:

- facilidade no manuseio e armazenamento de dados;
- operadores vetoriais e matriciais;
- uma ampla e coerente variedade de ferramentas para análise de dados que atua de forma integrada;
- facilidades gráficas como a confecção imediata de figuras que podem ser salvas em diversos formatos;
- linguagem de programação simples e eficiente;
- integração com outros *softwares*, por exemplo, C++ e WinBUGS;
- calculadora!

Sobre o R

Vantagens:

- facilidade no manuseio e armazenamento de dados;
- operadores vetoriais e matriciais;
- uma ampla e coerente variedade de ferramentas para análise de dados que atua de forma integrada;
- facilidades gráficas como a confecção imediata de figuras que podem ser salvas em diversos formatos;
- linguagem de programação simples e eficiente;
- integração com outros *softwares*, por exemplo, C++ e WinBUGS;
- calculadora!

Desvantagens:

- pode ser lento, por exemplo, quando se trata de simulações intensivas.

Ambiente

- O R trabalha com linhas de comando: você manda $\rightarrow$ ele executa!

Ambiente

- O R trabalha com linhas de comando: você manda $\rightarrow$ ele executa!
- Não há planilhas para digitação de banco de dados ou menu de opções para realização de cálculos.
- Os comandos devem ser digitados na janela de execução (*R Console*).
- As informações ficam armazenadas na mémoria do computador. Elas podem ser acessadas, geradas, salvas, apagadas, manipuladas de diversas formas através de diferentes comandos, o que torna tudo bastante versátil.

Operações Básicas

- +: soma;
- −: subtração;
- ∗: multiplicação;
- /: divisão;
- ∧: potenciação;
- %%: resto da divisão;
- %/%: parte inteira da divisão;
- %∗%: multiplica matrizes;
- ==, !=: argumentos lógicos: igual e diferente, respectivamente;
- <, <=, >, >=: menor, menor igual, maior e maior igual, respectivamente;
- ;: utilizado para separar diferente comandos em uma mesma linha.

Comandos Básicos

- `sqrt()`: raiz quadrada;
- `abs()`: valor absoluto;
- `exp()`: exponencial;
- `log10()` e `log2()`: logaritmo na base 10 e na base 2, respectivamente;
- `log(,base=)`: logaritmo em qualquer base, padrão é o logaritmo neperiano;
- `sin()`, `cos()`, `tan()`: funções trigonométricas (em radianos);
- `asin()`, `acos()`, `atan()`: funções trigonométricas inversas;
- `factorial(n)`: $n!$
- `choose(a,b)`: $\binom{a}{b}$.

Algumas Dicas

- Apertar as setas para cima ou para baixo no teclado permite acesso aos comandos já digitados.
- Apertar `Ctrl + L` limpa a janela de execução dos comandos (*R Console*).
- Inicialmente, *inputs* são na cor vermelha e *outputs* são na cor azul em um fundo branco, mas isto pode ser alterado, assim como tipo e tamanho da fonte, em **Edit** $\rightarrow$ **GUI preferences**.
- É possível trocar o diretório de trabalho (de onde o R lê e onde ele salva arquivos) em **File** $\rightarrow$ **Change dir**.
- Pode-se escrever comentários no R utilizando-se o símbolo #.

Script e Editores

- Um *script* é uma janela em branco, diferente da janela de execução dos comandos (*R Console*), e serve para escrevermos os comandos que eventualmente serão executados.

Script e Editores

- Um *script* é uma janela em branco, diferente da janela de execução dos comandos (*R Console*), e serve para escrevermos os comandos que eventualmente serão executados.
- É possível abri-lo em **File** $\rightarrow$ **New script**.

Script e Editores

- Um *script* é uma janela em branco, diferente da janela de execução dos comandos (*R Console*), e serve para escrevermos os comandos que eventualmente serão executados.
- É possível abri-lo em **File** $\rightarrow$ **New script**.
- Para executar uma linha de comando do *script*, basta colocar o cursor nela e digitar `Ctrl+R`.
- Para executar mais de um comando de uma vez do *script*, basta selecionar o que será executado e digitar `Ctrl+R`.

Script e Editores

- Um *script* é uma janela em branco, diferente da janela de execução dos comandos (*R Console*), e serve para escrevermos os comandos que eventualmente serão executados.
- É possível abri-lo em **File** → **New script**.
- Para executar uma linha de comando do *script*, basta colocar o cursor nela e digitar `Ctrl+R`.
- Para executar mais de um comando de uma vez do *script*, basta selecionar o que será executado e digitar `Ctrl+R`.
- Existem outros editores populares que têm o mesmo papel do *script* do R com mais vantagens:
  - **Tinn-R**: editor gratuito para o R, parecido com o *script* do R, mas tem cores que auxiliam a programação, além de sugestões ao digitar o comando. http://sourceforge.net/projects/tinn-r/.

Script e Editores

- Um *script* é uma janela em branco, diferente da janela de execução dos comandos (*R Console*), e serve para escrevermos os comandos que eventualmente serão executados.
- É possível abri-lo em **File** → **New script**.
- Para executar uma linha de comando do *script*, basta colocar o cursor nela e digitar `Ctrl+R`.
- Para executar mais de um comando de uma vez do *script*, basta selecionar o que será executado e digitar `Ctrl+R`.
- Existem outros editores populares que têm o mesmo papel do *script* do R com mais vantagens:
  - **Tinn-R**: editor gratuito para o R, parecido com o *script* do R, mas tem cores que auxiliam a programação, além de sugestões ao digitar o comando. http://sourceforge.net/projects/tinn-r/.
  - **R-Studio**: é um ambiente de desenvolvimento integrado para o R. Objetos e gráficos são integrados de forma organizada. http://www.rstudio.com.

Objetos

- O R é uma linguagem orientada a objetos armazenados na memória ativa do computador.
- A objetos são atribuídos valores, expressões.

Objetos

- O R é uma linguagem orientada a objetos armazenados na memória ativa do computador.
- A objetos são atribuídos valores, expressões.
- Por exemplo, atribuímos a um objeto de nome `x` o valor `10` fazendo

```
x <- 10      ou      x = 10
```

Objetos

- O R é uma linguagem orientada a objetos armazenados na memória ativa do computador.
- A objetos são atribuídos valores, expressões.
- Por exemplo, atribuímos a um objeto de nome `x` o valor `10` fazendo

```
x <- 10      ou      x = 10
```

- O nome do objeto precisa começar com uma letra, pode conter números, ponto, *underline*... Mas não pode conter símbolos como vírgula, ponto-e-vírgula ou espaço.
- Os valores atribuídos aos objetos (ou variáveis) ficam salvos na memória do computador.

Objetos

- O R é uma linguagem orientada a objetos armazenados na memória ativa do computador.
- A objetos são atribuídos valores, expressões.
- Por exemplo, atribuímos a um objeto de nome `x` o valor `10` fazendo

```
x <- 10      ou      x = 10
```

- O nome do objeto precisa começar com uma letra, pode conter números, ponto, *underline*... Mas não pode conter símbolos como vírgula, ponto-e-vírgula ou espaço.
- Os valores atribuídos aos objetos (ou variáveis) ficam salvos na memória do computador.
- Utilizando o comando `ls()`, podemos ver quais objetos estão salvos na memória do compoutador.

Tipos de Objetos

Alguns dos objetos mais comumente utilizados são

- **Vetores**: conjuntos de elementos de uma mesma natureza.
- **Matrizes**: conjuntos de elementos de uma mesma natureza organizados em linhas e colunas.
- **Arrays**: generalização da ideia de matriz.
- **Data frames**: similar às matrizes, porém diferentes colunas podem possuir elementos de natureza diferentes.
- **Listas**: generalização de vetores, representa uma coleção de objetos.

Tipos de Objetos

Alguns dos objetos mais comumente utilizados são

- **Vetores**: conjuntos de elementos de uma mesma natureza.
- **Matrizes**: conjuntos de elementos de uma mesma natureza organizados em linhas e colunas.
- **Arrays**: generalização da ideia de matriz.
- **Data frames**: similar às matrizes, porém diferentes colunas podem possuir elementos de natureza diferentes.
- **Listas**: generalização de vetores, representa uma coleção de objetos.

Falaremos deles mais tarde!

Importando Dados

- O R pode ler dados de arquivos importando-os para serem analisados.
- Há duas maneiras de fazer isso:

  (i) especificando todo o caminho do arquivo;

  (ii) especificando apenas o nome do arquivo se ele estiver no diretório de trabalho.

Importando Dados (Alguns Comandos)

`read.table("nome.do.arquivo.txt",header=TRUE)`: para ler arquivos *.txt*; se `header=TRUE`, considera a primeira linha como os nomes das colunas (o padrão é `FALSE`).

Importando Dados (Alguns Comandos)

`read.table("nome.do.arquivo.txt",header=TRUE)`: para ler arquivos *.txt*; se `header=TRUE`, considera a primeira linha como os nomes das colunas (o padrão é `FALSE`).

Exemplos:

- Lendo dados de um arquivo no diretório de trabalho:

```
dados <- read.table("dados1.txt",header=TRUE)
```

- Lendo dados de um arquivo em um diretório específico:

```
dados <- read.table("c:\\Users\\joaobmp\\Desktop\\Minicurso
R\\Exemplos\\dados1.txt",header=TRUE)
```

ou

```
dados <- read.table("c:/Users/joaobmp/Desktop/Minicurso
R/Exemplos/dados1.txt",header=TRUE)
```

Importando Dados (Alguns Comandos)

`read.csv("nome.do.arquivo.csv",sep=";",dec=".")`: para ler arquivos *.csv*; geralmente é preciso dizer como os dados são separados: `sep` (o padrão é espaço em branco); e de que forma são os números decimais: `dec` (o padrão é ponto).

Importando Dados (Alguns Comandos)

`read.csv("nome.do.arquivo.csv",sep=";",dec=".")`: para ler arquivos *.csv*; geralmente é preciso dizer como os dados são separados: `sep` (o padrão é espaço em branco); e de que forma são os números decimais: `dec` (o padrão é ponto).

Exemplos:

- Lendo dados de um arquivo no diretório de trabalho:

```
dados <- read.table("dados1.csv")
```

- Lendo dados de um arquivo em um diretório específico:

```
dados <- read.table("c:\\Users\\joaobmp\\Desktop\\Minicurso
R\\Exemplos\\dados1.csv")
```

ou

```
dados <- read.csv("c:/Users/joaobmp/Desktop/Minicurso
R/Exemplos/dados1.csv")
```

Importando Dados (Alguns Comandos)

`scan("nome.do.arquivo")`: para ler arquivos em *.txt* ou *.csv* numéricos; empilha todos os dados em um vetor (empilhamento $\rightarrow$).

Importando Dados (Alguns Comandos)

`scan("nome.do.arquivo")`: para ler arquivos em *.txt* ou *.csv* numéricos; empilha todos os dados em um vetor (empilhamento $\rightarrow$).

Exemplos:

- Lendo dados de um arquivo no diretório de trabalho:

```
dados <- scan("dados1.txt")
```

- Lendo dados de um arquivo em um diretório específico:

```
dados <- scan("c:\\Users\\joaobmp\\Desktop\\Minicurso
R\\Exemplos\\dados1.txt")
```

ou

```
dados <- scan("c:/Users/joaobmp/Desktop/Minicurso
R/Exemplos/dados1.txt")
```

Importando Dados do R

O R conta com uma grande variedade de conjuntos de dados.

Exemplos:

- `beaver1`: série temporal da temperatura de dois castores;
- `beaver2`: série temporal da temperatura de dois castores;
- `cars`: velocidade e distância percorrida de carros;
- `iris`: dados de Edgar Anderson sobre a flor íris.

Estrutura dos Dados

- Conjuntos de dados no R, mais frequentemente, estão no formato de vetores, matrizes ou *data frames*.

Estrutura dos Dados

- Conjuntos de dados no R, mais frequentemente, estão no formato de vetores, matrizes ou *data frames*.
- Para nos referirmos a uma determinada posição `i` de um vetor denominado `vetor`, fazemos

```
vetor[i]
```

- Para nos referirmos a uma determinada posição `(i,j)` de uma matriz denominada `matriz`, fazemos

```
matriz[i,j] # linha i, coluna j
```

Estrutura dos Dados

- Para nos referirmos a uma determinada linha `i` desta matriz, fazemos

```
matriz[i,] # todas as colunas da linha i
```

- Para nos referirmos a uma determinada coluna `j` desta matriz, fazemos

```
matriz[,j] # todas as linhas da coluna j
```

Estrutura dos Dados

- Para nos referirmos a uma determinada linha `i` desta matriz, fazemos

```
matriz[i,] # todas as colunas da linha i
```

- Para nos referirmos a uma determinada coluna `j` desta matriz, fazemos

```
matriz[,j] # todas as linhas da coluna j
```

- Se um conjunto de dados `x` apresenta nomes nas colunas, podemos utilizar o comando `attach(x)` para nos referirmos a estas colunas pelo nome.

Estrutura dos Dados

- Para nos referirmos a uma determinada linha `i` desta matriz, fazemos

```
matriz[i,] # todas as colunas da linha i
```

- Para nos referirmos a uma determinada coluna `j` desta matriz, fazemos

```
matriz[,j] # todas as linhas da coluna j
```

- Se um conjunto de dados `x` apresenta nomes nas colunas, podemos utilizar o comando `attach(x)` para nos referirmos a estas colunas pelo nome.
- Para acessar uma determinada posição de um *data frame*, procede-se da mesma forma que para matrizes.

Estatísticas Descritivas

Algumas das mais importantes estatísticas descritivas são:

- `mean(x)`: média de uma variável `x`;
- `median`: mediana de uma variável `x`;
- `var(x)`: variância de uma variável `x`;
- `sd(x)`: desvio padrão de uma variável `x`;
- `quantile(x,p)`: quantil p de uma variável `x`;
- `summary(x)`: retorna um conjunto de estatísticas da variável `x`;
- `table(x)`: retorna a frequência dos valores da variável `x`;
- `cov(x,y)`: covariância entre as variáveis `x` e `y`;
- `cor(x,y)`: correlação entre as variáveis `x` e `y`.

Pedindo Ajuda! *Help* do R

- O *help* do R é bastante eficiente.
- Quando é chamado, uma página se abre com explicações e exemplos a respeito da função ou comando de interesse.

Pedindo Ajuda! *Help* do R

- O *help* do R é bastante eficiente.
- Quando é chamado, uma página se abre com explicações e exemplos a respeito da função ou comando de interesse.
- Há diversas maneiras de chamar o *help*.
  - `help(nome.da.funcao)`: quando sabe-se o nome da função;
  - `?nome.da.funcao`: faz o mesmo que o comando anterior;
  - `help.search("palavra.chave")`: busca por palavra-chave;
  - `??palavra.chave`: faz o mesmo que o comando anterior;
  - `example(nome.da.funcao)`: mostra exemplos da função.

Análise Gráfica

Histograma: `hist(x)`: faz um histograma dos valores de uma variável `x`.

- `freq=TRUE`: considera o eixo das ordenadas como a frequência absoluta (padrão);
- `prob=TRUE`: considera o eixo das ordenadas como a frequência relativa.

Análise Gráfica

Gráfico de barras: `barplot(x)`: faz um gráfico de barras com dos valores de uma variável `x` (frequência).

- `horiz=FALSE`: gráfico com barras verticias (padrão);
- `horiz=TRUE`: gráfico com barras horizontais;
- `names.arg=vetor.de.nomes`: nomes das categorias.

Análise Gráfica

Gráfico de setores: `pie(x)`: faz um gráfico de setores (pizza) dos valores de uma variável `x` (frequência).

- `clockwise=FALSE`: gráfico no sentido anti-horário (padrão);
- `clockwise=TRUE`: gráfico no sentido horário;
- `labels=vetor.de.nomes`: nomes das categorias.

Análise Gráfica

Box-plot: `boxplot(x)`: faz um gráfico *box-plot* dos valores de uma variável `x`.

- `horizontal=FALSE`: *box-plot* vertical (padrão);
- `horizontal=TRUE`: *box-plot* horizontal.

Análise Gráfica

Gráfico genérico: `plot(x)` ou `plot(x,y)` : faz diversos tipos de gráficos para valores de uma variável `x` ou um par de variáveis `(x,y)`.

- `type="p"`: gráfico de pontos (padrão);
- `type="l"`: gráfico de linhas;
- `type="b"`: gráfico de linhas e pontos;
- `type="c"`: gráfico de linhas sem pontos;
- `type="o"`: gráfico de linhas e pontos sobrespostos;
- `type="h"`: gráfico de linhas verticais;
- `type="s"`: gráfico de degraus;
- `type="S"`: gráfico de degraus invertidos;
- `type="n"`: não plota nada.

Análise Gráfica

- **Linha sobreposta**: `lines(x)` ou `lines(x,y)` : faz um gráfico de linha para valores de uma variável `x` ou um par de variáveis `(x,y)` sobreposto a um gráfico ativo.
- **Pontos sobrepostos**: `points(x)` ou `points(x,y)` : faz um gráfico de pontos para valores de uma variável `x` ou um par de variáveis `(x,y)` sobreposto a um gráfico ativo.

Exercício

Análise exploratória do conjunto de dados `iris`

- Calcule medidas descritivas dos tamanhos das flores comparandos-as entre as espécies.
- Compare o comportamento dos tamanhos das pétalas e sépalas entre as espécies através de histogramas e *box-plots*.
- Veja se existe e como é o tipo de relação entre largura e comprimento das flores para as diferentes espécies através de diagramas de dispersão.

Vetores

Como dito anteriormente, o R trabalha com objetos, que podem ser de diversos tipos.

Vetores

Como dito anteriormente, o R trabalha com objetos, que podem ser de diversos tipos.

- **Vetores**: conjuntos de elementos de uma mesma natureza.
- **Matrizes**: conjuntos de elementos de uma mesma natureza organizados em linhas e colunas.
- **Arrays**: generalização da ideia de matriz.
- **Data frames**: similar às matrizes, porém diferentes colunas podem possuir elementos de natureza diferentes.
- **Listas**: generalização de vetores, representa uma coleção de objetos.

Criando Vetores

Concatenação: c()

- `vetor.numerico <- c(1,4,-0.3,9) # vetor numérico`
- `vetor.numerico[1] # primeiro elemento do vetor`
- `vetor.numerico[-1] # vetor sem o primeiro elemento`
- `vetor.numerico[1] <- NA # vetor recebe um` *missing value*
- `vetor.caracter <- c("casa","bola","hadouken")`
  `# vetor de caracteres`
- `vetor.logico <- c(TRUE, FALSE, TRUE) # vetor lógico`

Criando Vetores

Concatenação: c()

- `vetor.numerico <- c(1,4,-0.3,9) # vetor numérico`
- `vetor.numerico[1] # primeiro elemento do vetor`
- `vetor.numerico[-1] # vetor sem o primeiro elemento`
- `vetor.numerico[1] <- NA # vetor recebe um` *missing value*
- `vetor.caracter <- c("casa","bola","hadouken")`
  `# vetor de caracteres`
- `vetor.logico <- c(TRUE, FALSE, TRUE) # vetor lógico`

Quaisquer operações com vetores (e matrizes) é feita elemento-a-elemento.

Criando Vetores

Sequência: seq()

- -10:10 # {-10,9,...,9,10}
- seq(from=1,to=10,by=2) # {1,3,5,7,9}
- seq(from=5,to=15,length=3) # {5,10,15}

Criando Vetores

Sequência: `seq()`

- `-10:10 # {-10,9,...,9,10}`
- `seq(from=1,to=10,by=2) # {1,3,5,7,9}`
- `seq(from=5,to=15,length=3) # {5,10,15}`

- Sequências permitem criar gráficos de funções conjuntamente com a função `plot(x)`.

Criando Vetores

Sequência: `seq()`

- `-10:10 # {-10,9,...,9,10}`
- `seq(from=1,to=10,by=2) # {1,3,5,7,9}`
- `seq(from=5,to=15,length=3) # {5,10,15}`

- Sequências permitem criar gráficos de funções conjuntamente com a função `plot(x)`.
- Por exemplo,

```
x <- seq(-4,4,length=10)
y <- 1/sqrt(2*pi)*exp(-1/2*x^2)
```

Criando Vetores

Repetição: `rep()`

- `rep(0,times=4)` # {0,0,0,0}
- `rep(c(1,2,3),times=4)` # {1,2,3,...,1,2,3}
- `rep(c(1,3),times=c(8,9))` # {1,...,1,3,...,3}
- `rep(c(1,4),each=5)` # {1,...,1,4,...,4}
- `rep(1:4,each=2,times=3)`
  # {1,1,2,2,3,3,4,4,...,1,1,2,2,3,3,4,4}
- `rep(c("a","b"),times=10)` # {"a","b",...,"a","b"}

Operações com Vetores

- `length(x)`: tamanho do vetor `x`;
- `sort(x)`: ordena `x` em ordem crescente;
- `sort(x,decreasing=TRUE)`: ordena `x` em ordem decrescente;
- `rank(x)`: posições de cada elemento do vetor `x` ordenado;
- `round(x,digits=2)`: arredonda os valores do vetor `x`;
- `min(x)`: elemento de valor mínimo de `x`;
- `which.min(x)`: posição do mínimo;
- `max(x)`: elemento máximo de `x`;
- `which.max(x)`: posição do máximo;
- `sum(x)`: somatório dos elementos de `x`;
- `prod(x)`: produtório dos elementos de `x`;
- `x[x>=10]`: elementos de `x` maiores ou iguais a 10;
- `which(x>=10)`: posição dos elementos de `x` maiores ou iguais a 10;
- `unique(x)`: vetor com elementos não repetidos de `x`;
- `rev(x)`: inverte a ordem do vetor `x`.

Exercício

- Crie um vetor com números entre 0 e 1000 somente com elementos múltiplos de 7; teste usando o comando %%; depois coloque-os em ordem descrecente e informe a posição do número 777.
- Crie um gráfico da função de probabilidade $Bin(50;\, 0,6)$.
- Calcule média e variância do vetor `x <- seq(0,1000,by=30)` utilizando somente os comandos `sum(x)` e `length(x)` e compare com os valores obtidos utilizando os comandos `mean(x)` e `var(x)`.

Matrizes

- **Vetores**: conjuntos de elementos de uma mesma natureza.
- **Matrizes**: conjuntos de elementos de uma mesma natureza organizados em linhas e colunas.
- **Arrays**: generalização da ideia de matriz.
- **Data frames**: similar às matrizes, porém diferentes colunas podem possuir elementos de natureza diferentes.
- **Listas**: generalização de vetores, representa uma coleção de objetos.

Criando Matrizes

Matriz: `matrix()`

- `m = matrix(1:16,nrow=4,ncol=4)`
  `# matriz 4 x 4 (preenchimento ↓)`
- `m = matrix(1:16,nrow=4,ncol=4,byrow=TRUE)`
  `# matriz 4 x 4 (preenchimento →)`
- `m[1,4] # linha 1 e coluna 4`
- `m[1,] # linha 1`
- `m[,1] # coluna 1`
- `m[-1,] # todas as linhas exceto linha 1`
- `m[,-1] # todas as colunas exceto coluna 1`
- `m[c(1,3),] # linhas 1 e 3`
- `m[,c(1,3)] # colunas 1 e 3`
- `m[1:2,1:3] # linhas 1 a 2 e colunas de 1 a 3`

Criando Matrizes

Matriz diagonal: `diag()`

- `diag(4,6) # matriz diagonal de dimensão 6 com valores 4`
- `diag(c(9,5,11))`
  `# matriz diagonal de dimensão 3 com valores {9,5,11}`

Criando Matrizes

Concatenação de matrizes: `cbind()` e `rbind()`

- `M1 <- matrix(1,2,2)`
- `M2 <- cbind(M1,c(2,2))`
  `# acrescenta uma coluna a matriz M1`
- `M3 <- cbind(M1,M2)`
  `# concatena as matrizes M1 e M2 horizontalmente`
- `M2 <- rbind(M1,c(2,2))`
  `# acrescenta uma linha a matriz M1`
- `M3 <- rbind(M1,M2)`
  `# concatena as matrizes M1 e M2 verticalmente`

Operações com Matrizes

- `dim(A)`: dimensão da matriz `A`;
- `A%*%B`: multiplicação matricial;
- `A*B`: multiplicação matricial elemento a elemento;
- `det(A)`: determinante de da matriz `A`;
- `t(A)`: transposta da matriz `A`;
- `solve(A)`: inversa da matriz `A`;
- `eigen(A)`: autovalores e autovetores da matriz `A`;
- `A^(-1)`: valor inverso de cada elemento da matriz `A`.

Algumas Dicas

- Existem objetos no R que são múltiplos, que possuem diferentes atributos.

Algumas Dicas

- Existem objetos no R que são múltiplos, que possuem diferentes atributos.
- Para saber quais são estes atributos, pode-se utilizar o comando `names()`.
- Para acessar algum atributo específico, utiliza-se o nome da variável seguido de `$` e do nome do atributo. Por exemplo,

```
auto <- eigen(A) # autovalores e autovetores da matriz A
names(auto) # retorna os atributos do objeto auto
auto$values # somente os autovalores
auto$vectors # somente os autovetores
```

Algumas Dicas

- Existem objetos no R que são múltiplos, que possuem diferentes atributos.
- Para saber quais são estes atributos, pode-se utilizar o comando `names()`.
- Para acessar algum atributo específico, utiliza-se o nome da variável seguido de `$` e do nome do atributo. Por exemplo,

```
auto <- eigen(A) # autovalores e autovetores da matriz A
names(auto) # retorna os atributos do objeto auto
auto$values # somente os autovalores
auto$vectors # somente os autovetores
```

- É possível editar um vetor ou matriz de forma prática utilizando o comando `fix()` ou `edit()`

Arrays

- **Vetores**: conjuntos de elementos de uma mesma natureza.
- **Matrizes**: conjuntos de elementos de uma mesma natureza organizados em linhas e colunas.
- **Arrays**: generalização da ideia de matriz.
- **Data frames**: similar às matrizes, porém diferentes colunas podem possuir elementos de natureza diferentes.
- **Listas**: generalização de vetores, representa uma coleção de objetos.

Criando *Arrays*

Array: `array()`: *arrays* são generalizações das matrizes e podem ter várias dimensões.

```
A <- array(1:24,dim=c(2,3,4))
# 4 matrizes de dimensão 2 x 3
```

```
A[1,2,4] # linha 1, coluna 2 do bloco 4
```

Data Frames

- **Vetores**: conjuntos de elementos de uma mesma natureza.
- **Matrizes**: conjuntos de elementos de uma mesma natureza organizados em linhas e colunas.
- **Arrays**: generalização da ideia de matriz.
- **Data frames**: similar às matrizes, porém diferentes colunas podem possuir elementos de natureza diferentes.
- **Listas**: generalização de vetores, representa uma coleção de objetos.

Criando *Data Frames*

Data frame: `data.frame()`: similar às matrizes, porém diferentes colunas podem possuir elementos de natureza diferentes.

- `dados <- data.frame(c(27,18,23),c("Ana","Eva","Mel"))`
  `# criando um data frame`
- `names(dados) <- c("Idade","Nome") # nomeando as colunas do data frame`
- `dados <-`
  `data.frame(Idade=c(27,18,23),Nome=c("Ana","Eva","Mel"))`
  `# criando um data frame de outra maneira`
- `attach(dados) # cria variáveis com os nomes das colunas do data frame`

Criando *Data Frames*

- Uma maneira simples de verificar a qual classe um objeto `x` pertence é através dos comandos:
  - `is.numeric`: o objeto é numérico?
  - `is.vector`: o objeto é um vetor?
  - `is.matrix`: o objeto é uma matriz?
  - `is.data.frame`: o objeto é um data frame?
  - `is.list`: o objeto é uma lista?

Criando *Data Frames*

- Uma maneira simples de verificar a qual classe um objeto `x` pertence é através dos comandos:
  - `is.numeric`: o objeto é numérico?
  - `is.vector`: o objeto é um vetor?
  - `is.matrix`: o objeto é uma matriz?
  - `is.data.frame`: o objeto é um data frame?
  - `is.list`: o objeto é uma lista?
- Em alguns casos, é possível transformar um objeto de uma classe em outro substituindo `is` por `as` nos comandos acima.
- Por exemplo,

```
x <- iris[,1:4]
y <- as.matrix(x)
```

Listas

- **Vetores**: conjuntos de elementos de uma mesma natureza.
- **Matrizes**: conjuntos de elementos de uma mesma natureza organizados em linhas e colunas.
- **Arrays**: generalização da ideia de matriz.
- **Data frames**: similar às matrizes, porém diferentes colunas podem possuir elementos de natureza diferentes.
- **Listas**: generalização de vetores, representa uma coleção de objetos.

Criando Listas

Lista: `list()`: generalização de vetores, representa uma coleção de objetos.

- `lista <- list(c(1,2,3),matrix(1,4,3),"compras")`
  `# lista com diferentes objetos`
- `lista[[1]] # acessa o primeiro objeto`
- `lista[1] # acessa o primeiro objeto, mas como uma lista`
- `lista <- list(a=c(1,2,3),b=matrix(1,4,3),c="compras")`
  `# lista com diferentes objetos nomeados`
- `lista$a # primeiro objeto chamado a`

Exportando Dados

- Além de importar dados de arquivos, podemos também exportá-los.
- Alguns comandos para exportação são:
  - `write.table(objeto,"nome.do.arquivo.txt")`: salva os dados em um arquivo *.txt*;
  - `write.csv(objeto,"nome.do.arquivo.csv")`: salva os dados em um arquivo *.csv*.

Exportando Dados

- Além de importar dados de arquivos, podemos também exportá-los.
- Alguns comandos para exportação são:
  - `write.table(objeto,"nome.do.arquivo.txt")`: salva os dados em um arquivo *.txt*;
  - `write.csv(objeto,"nome.do.arquivo.csv")`: salva os dados em um arquivo *.csv*.
- Algumas opções úteis para os comando quando salvamos os dados podem ser `row.names=FALSE` e `col.names=FALSE`, o que faz com que somente os dados sejam gravados no arquivo.

Exercício

- Calcule o valor da funcão de densidade da distribuição normal bivariada

$$f(\mathbf{x}) = (\mathbf{2}\pi)^{-\mathbf{1}} \mid \mathbf{\Sigma} \mid \exp\left\{-\frac{\mathbf{1}}{\mathbf{2}}(\mathbf{x}-\boldsymbol{\mu})^{\mathbf{T}}\mathbf{\Sigma}^{-\mathbf{1}}(\mathbf{x}-\boldsymbol{\mu})\right\},$$

  em que $\boldsymbol{\mu} = (5, 10)$ e $\Sigma = \begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix}$ para diferentes valores de $\mathbf{x}$.
- Crie um data frame com o vetor de médias $\boldsymbol{\mu}$ e a matriz de covariância Σ lado a lado e salve o arquivo.

Comandos para Distribuições de Probabilidade

- O R, como não poderia ser diferente, possui diversos comandos para lidar com distribuições de probabilidade.

Comandos para Distribuições de Probabilidade

- O R, como não poderia ser diferente, possui diversos comandos para lidar com distribuições de probabilidade.
- De forma geral, há quatro tipos de funções:
  - `d`: calcula o valor de uma função de densidade (caso contínuo) ou função de probabilidade (caso discreto) em um determinado ponto;

Comandos para Distribuições de Probabilidade

- O R, como não poderia ser diferente, possui diversos comandos para lidar com distribuições de probabilidade.
- De forma geral, há quatro tipos de funções:
  - `d`: calcula o valor de uma função de densidade (caso contínuo) ou função de probabilidade (caso discreto) em um determinado ponto;
  - `p`: calcula o valor da função de distribuição acumulada em um determinado ponto;

Comandos para Distribuições de Probabilidade

- O R, como não poderia ser diferente, possui diversos comandos para lidar com distribuições de probabilidade.
- De forma geral, há quatro tipos de funções:
  - `d`: calcula o valor de uma função de densidade (caso contínuo) ou função de probabilidade (caso discreto) em um determinado ponto;
  - `p`: calcula o valor da função de distribuição acumulada em um determinado ponto;
  - `q`: calcula o quantil correspondente a uma determinada probabilidade;

Comandos para Distribuições de Probabilidade

- O R, como não poderia ser diferente, possui diversos comandos para lidar com distribuições de probabilidade.
- De forma geral, há quatro tipos de funções:
  - `d`: calcula o valor de uma função de densidade (caso contínuo) ou função de probabilidade (caso discreto) em um determinado ponto;
  - `p`: calcula o valor da função de distribuição acumulada em um determinado ponto;
  - `q`: calcula o quantil correspondente a uma determinada probabilidade;
  - `r`: gera valores aleatórios de uma determinada distribuição.

Distribuições de Probabilidade

Distribuição	Notação no R	Argumentos
Binomial	`binom`	`size,prob`
Geométrica	`geom`	`prob`
Hipergeométrica	`hyper`	`m,n`
Binomial Negativa	`nbinom`	`size, prob`
Poisson	`pois`	`lambda`
Uniforme	`unif`	`min=0, max=1`
Exponencial	`exp`	`rate=1`
Normal	`norm`	`mean=0, sd=1`
t de Student	`t`	`df`
Chi-Quadrado	`chisq`	`df`
F de Snedecor	`f`	`df1, df2`
Weibull	`weibull`	`shape, scale=1`
Gama	`gamma`	`shape, rate=1`
Beta	`beta`	`shape1, shape2`

Distribuições de Probabilidade

Distribuição	Notação no R	Argumentos
Binomial	`binom`	`size,prob`
Geométrica	`geom`	`prob`
Hipergeométrica	`hyper`	`m,n`
Binomial Negativa	`nbinom`	`size, prob`
Poisson	`pois`	`lambda`
Uniforme	`unif`	`min=0, max=1`
Exponencial	`exp`	`rate=1`
Normal	`norm`	`mean=0, sd=1`
t de Student	`t`	`df`
Chi-Quadrado	`chisq`	`df`
F de Snedecor	`f`	`df1, df2`
Weibull	`weibull`	`shape, scale=1`
Gama	`gamma`	`shape, rate=1`
Beta	`beta`	`shape1, shape2`

Por exemplo, o comando `rnorm(100,0,1)` gera 100 valores da distribuição normal padrão.

Exercício

- Gere uma amostra de tamanho 100 da distribuição normal, faça um histograma e compare com a curva teórica.
- Faça um gráfico da função de distribuição acumulada da distribuição chi-quadrado.

Exercício

- Um resultado importante da Teoria das Probabilidades é o seguinte: se $U = F(x) = P(X \leq x)$ é a função de distribuição acumulada de uma variável aleatória X, então U tem distribuição uniforme no intervalo $(0, 1)$.
- Assim, para variáveis contínuas, se é possível encontrar a função $x = F^{-1}(u)$, podemos sortear valores da distribuição de X sorteando valores u da distribuição uniforme (método da inversão).

Exercício

- Um resultado importante da Teoria das Probabilidades é o seguinte: se $U = F(x) = P(X \leq x)$ é a função de distribuição acumulada de uma variável aleatória X, então U tem distribuição uniforme no intervalo $(0, 1)$.
- Assim, para variáveis contínuas, se é possível encontrar a função $x = F^{-1}(u)$, podemos sortear valores da distribuição de X sorteando valores u da distribuição uniforme (método da inversão).
- Gere uma amostra de tamanho 500 da distribuição exponencial com parâmentro $\beta = 5$ utilizando o método da inversão e compare com resultados utilizando o comando `dexp()`.
- Sabe-se que se $X \sim Exp(\beta)$, então $F(x) = 1 - \exp(-\beta x)$.

Intervalos de Confiança e Testes de Hipóteses para a Média

- Através do comando `t.test(x)`, podemos calcular intervalos de confiança para a média μ de uma população normal com variância desconhecida a partir de uma amostra `x`.

Intervalos de Confiança e Testes de Hipóteses para a Média

- Através do comando `t.test(x)`, podemos calcular intervalos de confiança para a média μ de uma população normal com variância desconhecida a partir de uma amostra `x`.
- Além disto, é possível testar hipóteses a respeito deste parâmetro.

Intervalos de Confiança e Testes de Hipóteses para a Média

- Através do comando `t.test(x)`, podemos calcular intervalos de confiança para a média μ de uma população normal com variância desconhecida a partir de uma amostra `x`.
- Além disto, é possível testar hipóteses a respeito deste parâmetro.
- Por exemplo,

```
t.test(x,conf.level=0.95,mu=0,alternative="two.sided")
# teste bilateral (padrão)
```

```
t.test(x,conf.level=0.95,mu=0,alternative="less")
# hipótese alternativa:  mu < 0
```

```
t.test(x,conf.level=0.95,mu=0,alternative="greater")
# hipótese alternativa:  mu > 0
```

Intervalos de Confiança e Testes de Hipóteses para a Média

- O comando `t.test(x)` retorna uma lista.
- Se quisermos acessar objetos específicos desta lista, podemos fazer por exemplo,

```
teste <- t.test(x,confid.level=0.95,mu=0,alternative="two.sided")
interconf <- teste$confid.int # intervalo de confiança
pvalor <- teste$p.value # p-valor
estatteste <- teste$statistic # estatística de teste
```

Intervalos de Confiança e Testes de Hipóteses para a Proporção

- Através do comando `prop.test(x)`, podemos calcular intervalos de confiança para a proporção p de uma população a partir de uma proporção amostral `x/n`.

Intervalos de Confiança e Testes de Hipóteses para a Proporção

- Através do comando `prop.test(x)`, podemos calcular intervalos de confiança para a proporção p de uma população a partir de uma proporção amostral `x/n`.
- Da mesma forma, é possível testar hipóteses a respeito deste parâmetro.

Intervalos de Confiança e Testes de Hipóteses para a Proporção

- Através do comando `prop.test(x)`, podemos calcular intervalos de confiança para a proporção p de uma população a partir de uma proporção amostral `x/n`.
- Da mesma forma, é possível testar hipóteses a respeito deste parâmetro.
- Por exemplo,

```
prop.test(x,n,conf.level=0.95,p=0.5,alternative="two.sided")
# teste bilateral (padrão)
```

```
prop.test(x,n,conf.level=0.95,p=0.5,alternative="less")
# hipótese alternativa:  p < 0.5
```

```
t.test(x,n,conf.level=0.95,p=0.5,alternative="greater")
# hipótese alternativa:  p > 0.5
```

Exercício

Estimação com o conjunto de dados `beaver1`

- Calcule os intervalos de 90% e 95% de confiança para a média da temperatura dos castores.
- Teste $H_0 : \mu = 3,8$ contra a hipótese alternativa $H_1 : \mu > 3,8$.
- Calcule um intervalo de confiança de 99% para a proporção de castores com atividade.
- Teste a hipótese $H_0 : p = 0,6$ contra a hipótse $H_1 : p < 0,6$.

Criando Funções

- O R permite, além das funções disponíveis em suas bibliotecas, que o usuário crie suas próprias funções.

```
nome.da.funcao <- function(){
   comandos
}
```

Criando Funções

- O R permite, além das funções disponíveis em suas bibliotecas, que o usuário crie suas próprias funções.

```
nome.da.funcao <- function(){
   comandos
}
```

- Uma vez definida, a função pode ser chamada como qualquer outra função ou comando interno do R.

Criando Funções

- O R permite, além das funções disponíveis em suas bibliotecas, que o usuário crie suas próprias funções.

```
nome.da.funcao <- function(){
   comandos
}
```

- Uma vez definida, a função pode ser chamada como qualquer outra função ou comando interno do R.
- Uma função pode ou não ter argumentos.

```
nome.da.funcao <- function(argumentos){
   comandos
}
```

Criando Funções

- Para que a função retorne algum valor e não simplesmente execute uma ação, é necessario usar o comando `return()`.

Criando Funções

- Para que a função retorne algum valor e não simplesmente execute uma ação, é necessario usar o comando `return()`.
- Por exemplo, queremos criar uma função que retorne a distância euclidiana entre dois pontos:

```
distancia <- function(x,y){
   d <- sqrt((y[1] - x[1])^2 + (y[2] - x[2])^2)
   return(d)
}
```

Exercício

- Crie uma função que retorne em uma lista a média e desvio padrão do logaritmo de um conjunto de dados.
- Crie uma função que retorne em uma lista as soluções de uma equação de segundo grau, sabendo que a fórmula de Bhaskara é

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Comandos `if` e `else`

- O comando `if` é utilizado para validar uma condição e então executar o código de acordo com o resultado.

```
if(condição verdadeira){
   comandos
}
```

Comandos `if` e `else`

- O comando `if` é utilizado para validar uma condição e então executar o código de acordo com o resultado.

```
if(condição verdadeira){
   comandos
}
```

- Por exemplo,

```
x <- 6
if(x==6){
  print("Aehhh!  Hexa!")
}
```

Comandos `if` e `else`

- O comando `else` é utilizado juntamente com o comando `if`.

```
if(condição verdadeira){
  comandos1
} else{
    comandos2
}
```

Comandos `if` e `else`

- O comando `else` é utilizado juntamente com o comando `if`.

```
if(condição verdadeira){
  comandos1
} else{
    comandos2
}
```

- Por exemplo,

```
x <- 5
if(x==6){
  print("Aehhh!  Hexa!")
} else{
    print("Poxa...")
}
```

Comandos `if` e `else`

- Os operadores lógicos `||` ("ou") e `&&` ("e") são bastante úteis quando precisamos verificar mais de uma condição ao mesmo tempo.
- Por exemplo,

```
x <- 5
if((x>=3)&&(x<7)){
  print("Aluno vai pra final!")
} else{
    print("Aprovado ou reprovado?")
}
```

Comando `for`

- O comando `for` cria um ciclo a partir de um contador e interrompe este ciclo quando uma condiçãoo que depende do contador se torna falsa.

```
for(contador in sequência){
   comandos
}
```

Comando `for`

- O comando `for` cria um ciclo a partir de um contador e interrompe este ciclo quando uma condiçãoo que depende do contador se torna falsa.

```
for(contador in sequência){
   comandos
}
```

- Por exemplo, queremos calcular o traço de uma matriz:

```
d <- dim(A)[1]
traco <- 0
for(i in 1:dim(A)[1]){
   traco <- traco + A[i,i]
}
```

Exercício

- Gere dados artificiais de um modelo $AR(1)$ da forma

$$y_t = 0,5y_{t-1} + \epsilon_t, \quad \epsilon_t \sim N(0,1),$$

para $t = 100$ e faça o gráfico da série temporal.

- Integração de Monte Carlo: seja X uma variável aleatória com distribuição contínua. Então

$$\begin{aligned} P(a < X < b) &= \int_a^b f(x)dx = \int_{\mathbb{R}} I_{(a,b)}(x)f(x) \\ &= E(I_{(a,b)}(X)) \approx \sum_{i=1}^{n} I_{(a,b)}(x_i)f(x_i). \end{aligned}$$

Crie uma função que aproxime a probabilidade de uma distribuição exponencial usando o resultado acima e compare com o resultado exato.

Comando `while`

- O comando `while` cria um ciclo a partir de uma condição e interrompe este ciclo quando a condição se torna falsa.

```
while(condição verdadeira){
   comandos
}
```

Comando `while`

- O comando `while` cria um ciclo a partir de uma condição e interrompe este ciclo quando a condição se torna falsa.

```
while(condição verdadeira){
   comandos
}
```

- Por exemplo, queremos calcular novamente o traço de uma matriz:

```
d <- dim(A)[1]
traco <- 0
i <- 1
while(i <= d){
   traco <- traco + A[i,i]
   i <- i + 1
}
```

Argumentos de Comandos Gráficos

- Já vimos que uma das funções gráficas mais utilizadas é função `plot()`.

Argumentos de Comandos Gráficos

- Já vimos que uma das funções gráficas mais utilizadas é função `plot()`.
- Muitos argumentos podem ser acrescentados a esta função de forma a customizar o gráfico da variável ou variáveis de interesse.
- Além de `type`, que troca o tipo de linha, alguns deles são:
  - `lty`: tipo de linha (cheia, tracejada, pontilhada,...);
  - `pch`: tipo de ponto (círculo, triângulo, quadrado,...);
  - `lwd`: grossura das linhas;
  - `cex`: tamanho dos pontos;
  - `col`: cor dos pontos;
  - `font`: tipo de fonte.

Argumentos de Comandos Gráficos

- Já vimos que uma das funções gráficas mais utilizadas é função `plot()`.
- Muitos argumentos podem ser acrescentados a esta função de forma a customizar o gráfico da variável ou variáveis de interesse.
- Além de `type`, que troca o tipo de linha, alguns deles são:
  - `lty`: tipo de linha (cheia, tracejada, pontilhada,...);
  - `pch`: tipo de ponto (círculo, triângulo, quadrado,...);
  - `lwd`: grossura das linhas;
  - `cex`: tamanho dos pontos;
  - `col`: cor dos pontos;
  - `font`: tipo de fonte.
- Muitos destes comandos funcionam com outras funções gráficas.

Argumentos de Comandos Gráficos

Alguns argumentos importantes para diversos tipos de gráficos são

- `main="título"`: adiciona um título ao gráfico;
- `xlab="eixo x"`: adiciona um nome ao eixo das abcissas;
- `ylab="eixo y"`: adiciona um nome ao eixo das ordenadas;
- `xlim=c(a,b)`: define o intervalo de variação do eixo das abcissas no gráfico;
- `ylim=c(a,b)`: define o intervalo de variação do eixo das ordenadas no gráfico;
- `cex.axis`: controla o tamanho dos eixos;
- `cex.lab`: controla o tamanho dos nomes dos eixos;
- `cex.main`: controla o tamanho do título;
- `axes=FALSE`: retira os eixos e a caixa em volta do gráfico;
- `bty="n"`: retira somente a caixa em volta do gráfico.

Argumentos de Comandos Gráficos: `expression`

- O argumento `expression` permite escrevermos funções matemáticas em gráficos confeccionados no R.

Argumentos de Comandos Gráficos: `expression`

- O argumento `expression` permite escrevermos funções matemáticas em gráficos confeccionados no R.
- Para ver todos os tipos de expressões, basta utilizar o comando `demo(plotmath)`.

Argumentos de Comandos Gráficos: `expression`

- O argumento `expression` permite escrevermos funções matemáticas em gráficos confeccionados no R.
- Para ver todos os tipos de expressões, basta utilizar o comando `demo(plotmath)`.
- Por exemplo,

```
plot(x,xlab=expression(alpha[t]^2),ylab=expression(beta[t]))
```

permite que os nomes dos eixos sejam α_t^2 e β_t.

Exercício

- Amostre 1000 valores da distribuição chi-quadrado, faça um histograma, troque título, nome dos eixos, tamanho da fonte, tipo da fonte, grossura da linha, tipo da linha, cor, título, etc.
- Faça um gráfico de linha (como uma série temporal), troque tipo de linha, cor da linha, grossura da linha, tipo da fonte, limites dos eixos, etc.

Comandos Gráficos: par

- Um comando muito importante para á customização de gráficos e o comando `par()`.

Comandos Gráficos: par

- Um comando muito importante para á customização de gráficos e o comando `par()`.
- Antes de plotar um gráficos, podemos definir diversos parâmetros como tamanho, margem, cores. Por exemplo,

```
par(mfrow=c(2,3),bg=2,cex.axis=3,cex.lab=3,lty=2,+
mar=c(2,2,2,2),pch=4,lty=2,col=4)
```

- `mfrow`: controla quantos gráficos aparecerão em uma mesma figura;
- `bg`: cor do fundo do gráfico;
- `mar`: controla o tamanho das margens do gráfico (fundo, esquerda, cima, direita).

Comandos Gráficos

Outros comandos gráficos úteis são

- `x11()`: permite abrir uma nova janela gráfica;
- `axis(1)`: desenha o eixo das abcissas;
- `axis(2)`: desenha o eixo das ordenadas;
- `box()`: desenha uma caixa em volta do gráfico.

Comandos Gráficos

Outros comandos gráficos úteis são

- `x11()`: permite abrir uma nova janela gráfica;
- `axis(1)`: desenha o eixo das abcissas;
- `axis(2)`: desenha o eixo das ordenadas;
- `box()`: desenha uma caixa em volta do gráfico.

Alguns argumentos do comando `axis` são

- `at`: seleciona as marcações que serão desenhadas;
- `labels`: nomes das marcações que serão desenhadas;
- `las`: permite trocar a orientação dos nomes das marcações.

Comandos Gráficos

Outros tipos de gráficos podem ser confeccionados através de algumas funções específicas:

- `curve(f(x),a,b)`: desenha a função $f(x)$ dentro do intervalo (a, b);
- `abline(h=a)`: desenha uma linha horizontal equivalente a $y = a$;
- `abline(v=a)`: desenha uma linha vertical equivalente a $x = a$;
- `abline(a=c,b=d)`: desenha a reta $y = c + dx$;
- `segments(x0,y0,x1,y1)`: desenha um segmento de reta de um ponto a outro.

Comandos Gráficos

- O comando `polygon()` nos permite criar um polígono sobreposto a um gráfico simplesmente indicado seus vértices.

Comandos Gráficos

- O comando `polygon()` nos permite criar um polígono sobreposto a um gráfico simplesmente indicado seus vértices.
- Por exemplo,

```
plot(0,xlim=c(-2,2),ylim=c(-2,2),type="n")
polygon(c(-1,1,1,-1),c(-1,-1,1,1),col="gray50")
```

Comandos Gráficos

Outros tipos de gráficos podem ser confeccionados através das de algumas funções específicas:

- `contour(x,y,z)`: curvas de nível da função $z = f(x, y)$;
- `persp(x,y,z)`: gráfico de perspectiva da função $z = f(x, y)$;
- `image(x,y,z)`: gráfico de imagem da função $z = f(x, y)$;
- `filled.contour(x,y,z)`: gráfico de imagem suavizado da função $z = f(x, y)$.

Comandos Gráficos

Para criar uma superfície, podemos, por exemplo, usar uma sequência de comandos da forma

```
x <- seq(-10,10,length=40)
y <- seq(-10,10,length=40)
z <- matrix(NA,40,40)
for(i in 1:40){
  for(j in 1:40){
    z[i,j] <- x[i]^2 + y[j]^2
  }
}
```

Comandos Gráficos

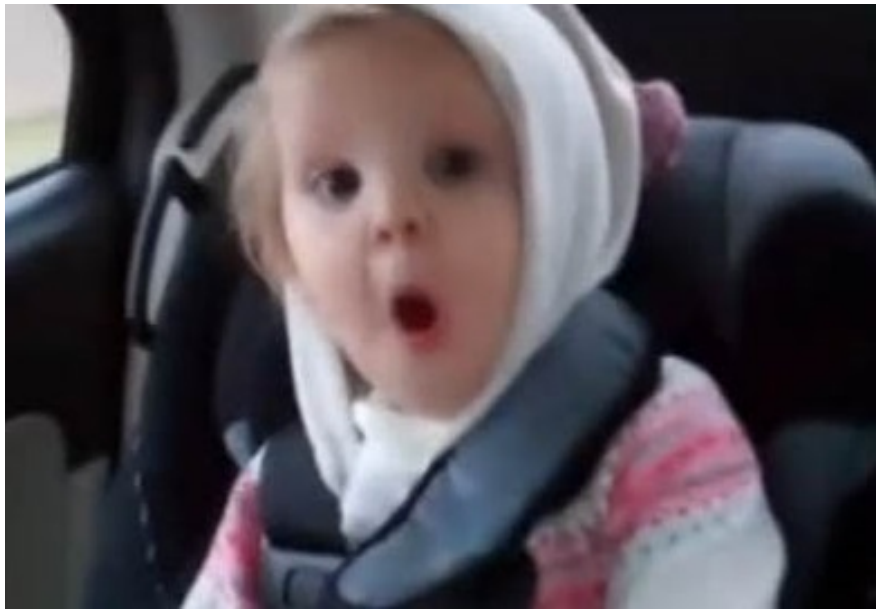
Algumas demonstrações interessantes são

- `demo(persp)`: vários exemplos de gráficos de superfície;
- `demo(image)`: vários exemplos de gráficos de imagem.

Comandos Gráficos

Algumas demonstrações interessantes são

- `demo(persp)`: vários exemplos de gráficos de superfície;
- `demo(image)`: vários exemplos de gráficos de imagem.



Palhetas de Cores

- Algumas vezes, ao invés de escolhermos uma única cor, podemos escolher uma palheta de cores para um gráfico.
- Algumas opções são `heat.colors()`, `terrain.colors()`, `topo.colors()` e `cm.colors()`.
- Por exemplo,

```
persp(x,y,z,col=heat.colors(10,0.9)) # 10 cores da palheta
com transparência de 90%
```

Exercício

- Crie superfícies de funções de sua escolha e faça gráficos de perspectiva, de imagem, de contorno, etc.
- Troque cores, sombras, refinamento, etc.

Gráficos

- Gráficos podem ser salvos de maneira simples em diversos formatos em **File** $\rightarrow$ **Save as**.
- Podem também ser salvos através do comando `savePlot()`.

Gráficos

- Gráficos podem ser salvos de maneira simples em diversos formatos em **File** $\rightarrow$ **Save as**.
- Podem também ser salvos através do comando `savePlot()`.
- Por exemplo,

```
savePlot("meugrafico.pdf",type="pdf")
```

- As extensões possíveis são “png”, “jpg”, “jpeg”, “bmp”, “ps”, “eps” e “pdf”.

Como Instalar Pacotes

- Além das funções básicas (que já são muitas) o R dispõe de muitas bibliotecas/pacotes com funções específicas.
- Eles podem ser instalados através da internet ou de arquivos no computador.

Como Instalar Pacotes

- Além das funções básicas (que já são muitas) o R dispõe de muitas bibliotecas/pacotes com funções específicas.
- Eles podem ser instalados através da internet ou de arquivos no computador.
- Pela internet: vá em **Packages** $\rightarrow$ **Install package(s)**. Em geral, é preciso primeiramente escolher o CRAN.
- Através de um arquivo no computador: vá em **Packages** $\rightarrow$ **Install package(s) from local zip files**.

Como Instalar Pacotes

- Além das funções básicas (que já são muitas) o R dispõe de muitas bibliotecas/pacotes com funções específicas.
- Eles podem ser instalados através da internet ou de arquivos no computador.
- Pela internet: vá em **Packages** → **Install package(s)**. Em geral, é preciso primeiramente escolher o CRAN.
- Através de um arquivo no computador: vá em **Packages** → **Install package(s) from local zip files**.
- Para carregar um pacote vá em **Packages** → **Load packages** ou simplesmente utilize o comando `library(nome.do.pacote)`.

Referências

- *An Introduction to R - Notes on R: A Programming Environment for Data Analysis and Graphics, Version 3.1.1* (2014-07-10).
- *Curso Básico de R*, Kelly C. M. Gonçalves
- *Histórico do programa R* obtido em http://www.estatisticanor.xpg.com.br/4.html (2013).
- *Minicurso de Introdução ao R*, Felipe R. R. Melo (2010).
- *O que eu Lembro de R*, João B. M. Pereira (2014).

OBRIGADO!