ESTATÍSTICA BÁSICA COM USO DO SOFTWARE R

©Adilson dos Anjos
Departamento de Estatístitica
- UFPR -

Curitiba, 22 de junho de 2010.

Módulo 1: Introdução ao Software R

Objetivo do Módulo

Ao final desse módulo o aluno deverá ser capaz de utilizar os principais recursos do R. Reconhecer e manipular objetos, utilizar funções básicas, conseguir ler dados de arquivos externos, criar e salvar um gráfico e conseguir buscar ajuda para utilizar recursos mais avançados.

1.1 Introdução

O **R** é um *software livre*. Isto significa que ele pode ser utilizado, copiado, distribuído, alterado e melhorado de forma livre. Tudo isso de forma legal. O **R** não é vendido. Você pode "baixá-lo" da Internet, de forma gratuita (www.r-project.org).

O **R** pode ser utilizado tanto no ambiente Windows, quanto no ambiente Linux/Unix, entre outros. Por isso, preste atenção na hora de selecionar o arquivo de instalação no site do **R**.

O **R** divide-se entre a instalação básica e pacotes (packages). Na instalação básica, você encontrará os principais comandos necessários para efetuar suas análises estatísticas e matemáticas. Ainda, na instalação básica, são disponibilizados alguns pacotes adicionais que normalmente são mais utilizados.

Além da instalação básica, no site do **R** existem vários outros aplicativos que podem ser utilizados em uma grande variedade de análises (principalmente estatísticas). Atualmente existem mais de mil pacotes adicionais que podem ser utilizados. Alguns destes pacotes serão utilizados no curso.


Para conhecer um pouco mais, acesse o endereço http://www.r-project.org

Alguns links são importantes para conhecer em uma primeira visita no site do **R**:

CRAN : Existem vários *mirrors* espalhados pelo mundo. Mirrors ou espelhos, são computadores servidores que armazenam os arquivos de instalação. No Brasil,

você pode encontrar os arquivos de instalação na UFPR, UFV e ESALQ/USP e FIOCRUZ. Para fazer o download, basta selecionar um espelho, escolher o sistema operacional: Windows, Linux ou MAC; e o arquivo para instalação.


Para conhecer, entre no endereço http://cran-r.c3sl.ufpr.br/web/packages/index.html para ver os pacotes disponíveis no **R**.

Search : Existe uma lista de discussão no site do **R**, onde os emails são armazenados. Você pode consultá-los através de uma busca no site. Isso é altamente recomendável se estiver pensando em mandar alguma pergunta para a lista. No Brasil há uma lista de discussão sobre o **R** em português no site do Yahoo Grupos.


No Yahoo Groups há uma lista de discussão do **R**. As discussões são em vários níveis de conhecimento e em português.

Documentation: Existem vários arquivos disponíveis que tratam de diferentes assuntos. A maioria está em inglês, mas existem em outros idiomas, inclusive em português. O Wiki do **R** ainda está em construção e possui muitas lacunas. Recentemente, foi lançado uma revista (Journal) especializada em assuntos do **R**. Apesar de serem em um nível avançado, vale a pena conhecer.

Misc: Alguns assuntos e alguns pacotes tiveram um destaque no site. São assuntos e temas específicos que, de tanto serem explorados no **R**, acabaram gerando um conjunto de pacotes e funções que estão organizados em uma área especial. Apesar de tratar de temas bastante específicos é uma boa fonte de informação sobre a aplicação do **R** e, claro, de estatística.

O site ainda contém outras áreas mais específicas sobre o **R**, que poderão ser exploradas.

Através de um site de busca, também é possível encontrar muito material disponível na Internet. Muitos cursos de estatística, no mundo e no Brasil, utilizam o **R** para o ensino de estatística e outras matérias.

No **R**, o arquivo de *Help* pode ser acessado como uma página *html*. Digite `help.start()` e um navegador (Browser) será aberto. Navegue nesta página e descubra um pouco mais dos recursos do **R**. No link *Packages* serão mostradas apenas os pacotes instalados no seu computador.

1.2 Iniciando e finalizando uma sessão no R

Existem muitas maneiras de se trabalhar com o **R**. Isso vai depender do nível do usuário em termos de conhecimento das funcionalidades do software e de computação de uma forma geral, além de suas preferências individuais.

No Windows, quando iniciar uma sessão do **R**, mude o diretório de trabalho no menu *File* e em seguida *Change Dir* e escolha um diretório de sua preferência.

Em uma próxima sessão, quando voltar a trabalhar com o **R** basta selecionar novamente o mesmo diretório para recuperar as informações que foram salvas. Se preferir, antes de abrir o **R**, entre no diretório onde você trabalhou pela útlima vez. Lá, você encontrará dois arquivos: .Rhistory e .Rdata. Clique no arquivo que possui o logo do **R**e o programa será aberto já configurado para o diretório de trabalho.

Uma sugestão, no Linux, é criar um diretório com o comando `mkdir` , entrar no diretório e iniciar o **R** na janela de comandos da seguinte maneira:

```
$ mkdir cursoR  # cria o diretório cursoR
$ cd cursoR     # entra no diretório cursoR
$ R # abri o R
```

Isso será feito apenas na primeira vez. Nas próximas vezes, basta entrar no diretório e abrir o **R**

Tanto no ambiente Windows quanto no Linux, você também pode definir o diretório de trabalho com o comando `setwd()`. Para ver em qual diretório você está trabalhando utilize o comando `getwd()`.

```
> getwd()

[1] "/home/adilson/Documentos/extensao/cursoDistancia/apostila"
```

Na linha de comando, você pode utilizar as seguintes opções:

```
> setwd("/home/adilson/documentos/Aulas")
```

para definir o diretório de trabalho e, para descobrir em qual diretório você está no momento digite:

```
> getwd()
[1] "/home/adilson/documentos/Aulas"
```

No Windows, basta abrir o **R**, ir em Arquivo e depois Mudar dir ... . Escolha um diretório de sua preferência.

Toda vez que uma sessão é aberta você pode salvar seu trabalho em um diretório. Basicamente você pode salvar o histórico de comandos da sessão e os objetos gerados durante a sessão. Automaticamente o **R** gera o arquivo `.Rhistory` que armazena o histórico de comandos e o arquivo `.RData` que armazena os objetos gerados durante uma sessão de trabalho. O **R** pergunta se você quer salvar esses objetos quando uma sessão é finalizada.

Para finalizar uma sessão digite *q()* e depois *ENTER*. O **R** irá perguntar se você quer salvar a sessão responda *y* para sim, cancelar responda *c* ou não salvar e retornar ao **R** responda*n* .

Para recuperar uma sessão, abra o **R** no mesmo diretório onde a sessão foi iniciada ou salva, se você estiver usando Linux. Se estiver no Windows, vá até o diretório onde a sessão foi salva e clique sobre o ícone do **R**.

Dica: Para cada módulo do curso, você pode ter um diretório com a sessão do **R** gravada. No dia-a-dia, utilize um diretório para cada tarefa ou trabalho.

Uma outra possibilidade é utilizar `scripts`. Tanto no ambiente Windows quanto em Linux.

No Windows, inicie o **R**, vá em `Arquivo` e depois `Novo script`. Para começar a trabalhar, digite um comando e as teclas <CTRL> + R simultaneamente. O comando será processado na janela do **R**. O script pode ser salvo com

De uma maneira mais simples, utilize qualquer editor de textos para escrever os comandos do **R**. Escreva um comando e depois copie e cole na janela de comandos (editor) do **R**.

1.3 Início do trabalho

O símbolo > indica que o programa está pronto para receber um comando.

O **R** trabalha com programação orientada à objetos . Utilize o símbolo <- para indicar uma atribuição. Por exemplo,

```
> a<-2          #pressione <ENTER>
```

O comando anterior significa que a recebe o número 2. Pressione <ENTER> após digitar o comando. O símbolo ♯ é utilizado como comentário . Tudo o que vier após esse símbolo na linha será desconsiderado como comando pelo **R**.

Agora, a é um objeto.

Agora, digite a e pressione <ENTER> e veja o resultado.

```
>  a

[1] 2
```

O número [1] indica a posição do algarismo.

O **R** é sensível a letras maiúsculas e minúsculas.

```
>  A<-3
>  A

[1] 3

>  a

[1] 2
```

Atenção! Os objetos podem ser sobrepostos sem que o **R** pergunte se você realmente quer fazer tal substituição.

Por exemplo:

```
> A<-4
> A
```

```
[1] 4
```

Os objetos podem receber nomes maiores. Os nomes não podem começar com números ou com '.' (ponto).

```
> Adilson<-1
> Adilson.Anjos<-2
```

Evite utilizar nomes de funções do **R**. Isso pode gerar algum conflito. Aos poucos você descobrirá os nomes das principais funções.

Comandos podem ser separados em uma mesma linha por ponto e vírgula (;).

```
> x<-2; y<-3; x;y
```

```
[1] 2
```

```
[1] 3
```

Se um comando não for completado, o sinal de + aparecerá ao invés do *prompt* >.

```
> dados<-c(2,3
+
```

Se isso ocorrer, digite o que estiver faltando ou cancele o comando com *ESC*, no Windows ou <CTRL C> se estiver no Linux.

```
> dados<-c(2,3
+ )  #digite `)' e pressione <ENTER>
>
```

O **R** utiliza do método recursivo. Isso significa que ele pode executar expressões dentro de expressões . Veja um exemplo:

```
> a<-6
> c<-a/(b<-2);c
```

```
[1] 3
```

1.4 Linha do Editor

Para deletar caracteres use a tecla *DEL* ou *Backspace* e as setas para movimentar-se na linha do editor de comandos.

As teclas <*CTRL*> + *a* e <*CTRL*> + *e* movimentam o cursor para o início e fim da linha de comando respectivamente. Você também pode utilizar as teclas <Home> e <End>. Use as teclas <SHIFT>+ <Page Up> e <SHIFT> +<Page Down> para subir e descer uma tela, respectivamente.

Todos os comandos executados continuam disponíveis no editor. Basta pressionar a seta direcional para cima ou para baixo. Quando encontrar o comando que deseja repetir pressione <ENTER>. Se desejar, ainda pode editar o comando antes de reexecutá-lo.

1.5 O que tenho armazenado

Para ver os objetos armazenados use `ls()`:

```
> ls()
```

```
[1] "a"             "A"             "Adilson"
[4] "Adilson.Anjos" "b"             "badfonts"
[7] "c"             "x"             "y"
```

Para remover um objeto da sua área de trabalho (.GlobalEnv) no **R**, use

```
> rm(x,Adilson)
```

Lembre-se que todos os objetos são armazenados no arquivo *.RData* para uso em uma sessão futura.

Os comandos são armazenados no arquivo *.Rhistory*.

1.6 Decimais

Por ser um software de língua inglesa, os algarismos decimais são separados por ponto. Quando for entrar com dados diretamente na linha do editor, não esqueça disso.

Quando os dados são obtidos de arquivos externos, onde é utilizado vírgula como separador decimal, existem mecanismos para que a leitura seja efetuada de forma correta. Isso será abordado no decorrer do curso!

1.7 O R como calculadora

Nas linhas de comando você pode executar operações como:

```
> 2+2

[1] 4

> 2*3

[1] 6

> 4/2

[1] 2

> 2^2

[1] 4
```

Outras operações podem ser executadas. Por exemplo, o **R** possui algumas funções prontas para uso:

```
> sqrt(4)  # raiz quadrada

[1] 2

> sin(45)  # seno
```

```
[1] 0.8509035
```

Otras funções serão abordadas no decorrer do curso.

Algumas aplicações com vetores (será discutido com mais detalhes na seção 1.14):

```
> a<-c(2 ,4 ,6)
> b<-c(1, 2, 3)
> a+b

[1] 3 6 9

> a*b

[1]  2  8 18

> a/b

[1] 2 2 2

> x<-2
> y<-c(4,8,10)
> y*x

[1]  8 16 20

> y/x

[1] 2 4 5
```

Outros comandos com uso de vetores:

```
> sum(y) # soma os valores de y

[1] 22
```

```
> length(y) # número de elementos em y
```

```
[1] 3
```

```
> media<-sum(y)/length(y)
> media
```

```
[1] 7.333333
```

ou, simplesmente,

```
> mean(y) # média de y
```

```
[1] 7.333333
```

No Módulo 2, sobre estatística descritiva, serão apresentadas outras funções estatísticas!

1.8 Como obter ajuda no R

Como o software é baseado em comandos, você dificilmente conseguirá gravar tudo. Será necessário consultar os arquivos de ajuda para sanar suas dúvidas. Essas informações podem ser obtidas no próprio programa por meio dos arquivos de ajuda.

Existem várias formas de obter ajuda no **R**. Na linha de comando você pode digitar, por exemplo

```
>  help(anova)
```

ou

```
> ?anova
```

O arquivo *help* possui um formato padrão na maioria das funções:

- No cabeçalho ele informa qual o nome da função e onde ela está armazenada.
- No item *description* ele informa o que a função faz;
- No item *usage* ele informa como usar a função. Os $\cdots$ indicam que outros argumentos podem ser utilizados.
- No item *arguments* você específica o que a função deve fazer.
- No item *value* ele informa qual o resultado que a função retorna.
- No item *warning* o **R** informa quais cuidados devem ser tomados quando utilizar a função.
- No item *references* ele apresenta as referências bibliográficas que foram baseadas as análises efetuadas pela função.
- No item *See also* são apresentados outras funções que podem estar relacionados com a função e que podem ser úteis para complementar a análise.
- Normalmente, no arquivo *help*, também aparecem alguns exemplos no final.

Ainda, é possível fazer uma busca usando

```
> help.search("anova")
```

ou

```
> help.search('anova')
```

Observe que o objeto de procura está **entre aspas**. Experimente também

```
> example(t.test)
```


Dica:Nem todas as funções possuem arquivos de exemplo.

Cada função no **R** possui argumentos necessários para sua execução. Para saber quais são os argumentos de uma função digite `args(nome.da.função)`.

A função `help.start()` pode ser usada para visualizar o arquivo de ajuda no formato *html*. Um navegador (browser) será aberto para pesquisa.

Por último, no site do **R** ou com auxílio de ferramentas de busca da internet você pode encontrar muitas outras fontes de informação sobre o **R**

1.9 Tipos de Objetos

Os tipos de objetos mais importantes e que serão trabalhados nesse curso são:

Vetor: é um objeto que possui pelo menos um elemento (número ou caracter); cada elemento de um vetor possui um número que indica sua posição e pode ser acessado individualmente;

Fator: Um fator, geralmente representado por um vetor, é um tipo de objeto onde os elementos são categóricos (podem ser números ou letras). Um fator possui níveis definidos;

Listas: São um conjunto de objetos, não necessariamente da mesma classe ou tamanho;

Data frame: é uma tabela, com colunas de mesmo tamanho, onde as colunas podem ser fatores ou vetores diferentes;

Funções: são objetos que realizam alguma tarefa. Por exemplo, `sum(x)` é uma função que faz a soma dos elementos de x.

A classe do objeto pode ser investigada das seguintes maneiras, Aqui, x é o objeto avaliado:

`is.vector(x)`; `is.matrix(x)`; `is.factor(x)` ; `is.data.frame(x)`; `is.list(x)`; `is.function(x)`.

ou simplesmente `class(x)`

```
> x<-c(1,2,3,4,5);x

[1] 1 2 3 4 5

> is.numeric(x)

[1] TRUE

> class(x)

[1] "numeric"
```

Os atributos de um objeto também podem ser vistos ou adicionados usando-se a função `attributes(x)`.

Alguns objetos também podem ser convertidos para outros tipos de objetos com o comando correspondente, por exemplo,

```
> x<-as.factor(x)
> class(x)

[1] "factor"

> is.numeric(x)

[1] FALSE
```

Dica: Os objetos podem ser acessados somente por letras iniciais. Em geral as três primeiras letras são suficientes para acessar um objeto. Por exemplo, na linha do editor do **R** digite *mea* e aperte a tecla <TAB>. Veja que aparecerá a palavra *mean* que é uma função do **R** que obtém a média de um objeto.

1.10 Gerando sequências

Sequências de números são úteis na geração e manipulação de arquivos de dados. Existem várias maneiras de gerar sequências no **R**. Por exemplo, gerar a sequência de 1 até 20,

```
> a<-1:20
> a

 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

ou ainda,

```
> b<-20:1
> b
```

```
 [1] 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
```

Uma sequência mais elaborada pode ser construída usando a função `seq` . Os argumentos básicos são `seq(to=0,from=10,by=2)`

```
> seq(0,10, by = 2)

[1]  0  2  4  6  8 10
```

Ainda, pode-se especificar o tamanho do vetor e o início,

```
> seq(length=12,from=0,by=3)

 [1]  0  3  6  9 12 15 18 21 24 27 30 33

> seq(0,10,length=11)

 [1]  0  1  2  3  4  5  6  7  8  9 10

> seq(0,20,length=11)

 [1]  0  2  4  6  8 10 12 14 16 18 20
```

Usando a função `rep`. Procure entender os argumentos da função observando o resultado!

```
> rep(c(1,2,3),2)

[1] 1 2 3 1 2 3

> rep(c(1,2,3),each=2)

[1] 1 1 2 2 3 3

> rep(1:3,each=3,times=2)
```

```
 [1] 1 1 1 2 2 2 3 3 3 1 1 1 2 2 2 3 3 3
```

```
> rep("r",5)
```

```
[1] "r" "r" "r" "r" "r"
```

A função `rep` gera um vetor numérico ou alfanumérico.


Dica: Sempre que entrar com letras ou palavras, por exemplo, não esqueça de utilizar aspas.

1.11 Fornecendo nomes

Para cada elemento de um vetor, é possível atribuir nomes, por exemplo,

```
> notas<-c(7,5,8.5,9)
> names(notas)<-c("João","Pedro","Paulo","José")
> notas
```

```
 João Pedro Paulo  José
  7.0   5.0   8.5   9.0
```

```
> aprovados<-notas>=7
> aprovados
```

```
 João Pedro Paulo  José
 TRUE FALSE  TRUE  TRUE
```

1.12 Gráficos

O **R** possui um grande número de possibilidades gráficas. Por exemplo, a figura de abertura da página do projeto www.r-project.org é um conjunto de vários gráficos

construídos no próprio **R**. Clique na figura inicial, na página do **R** e veja como foi programada!

Ao longo do curso serão vistos alguns tipos de gráficos. Nesse momento, será visto como gerar gráficos e como definir alguns parâmetros.

Quando um gráfico é gerado, o **R** abre uma nova janela. Nessa janela são inseridos os gráficos criados. A janela gráfica irá sobrepor todos os gráficos criados. Assim, cada vez que você pedir para criar um novo gráfico o anterior será totalmente apagado e o novo gráfico será adicionado na janela gráfica.

Existem algumas funções que apenas acrescentam elementos novos ao gráfico já existente, como por exemplo, uma legenda.

Também existem opções para controlar a janela gráfica. Essas opçõs não serão abordadas nesso curso.

Para mais informaçoes sobre gráficos, consulte o livro: Paul Murrell. R Graphics. Chapman & Hall/CRC, Boca Raton, FL, 2005. ISBN 1-584-88486-X. Há um exemplar disponível na biblioteca do Setor de Tecnologia no Cenro Politécnico.

1.13 Como gerar um gráfico

Um gráfico simples pode ser gerado da seguinte maneira (figura 1):

Toda vez que um gráfico é gerado, uma *janela gráfica é aberta*, exceto quando os comandos salvar do tipo `postscript()` e `pdf()`, por exemplo, são inseridos.

Os gráficos podem ser salvos em diferentes formatos. Com o gráfico aberto, basta clicar com o botão direito do mouse (Windows) em cima do gráfico e escolher salvar como: metafile (.emf), ou postscript (.ps ou .eps).

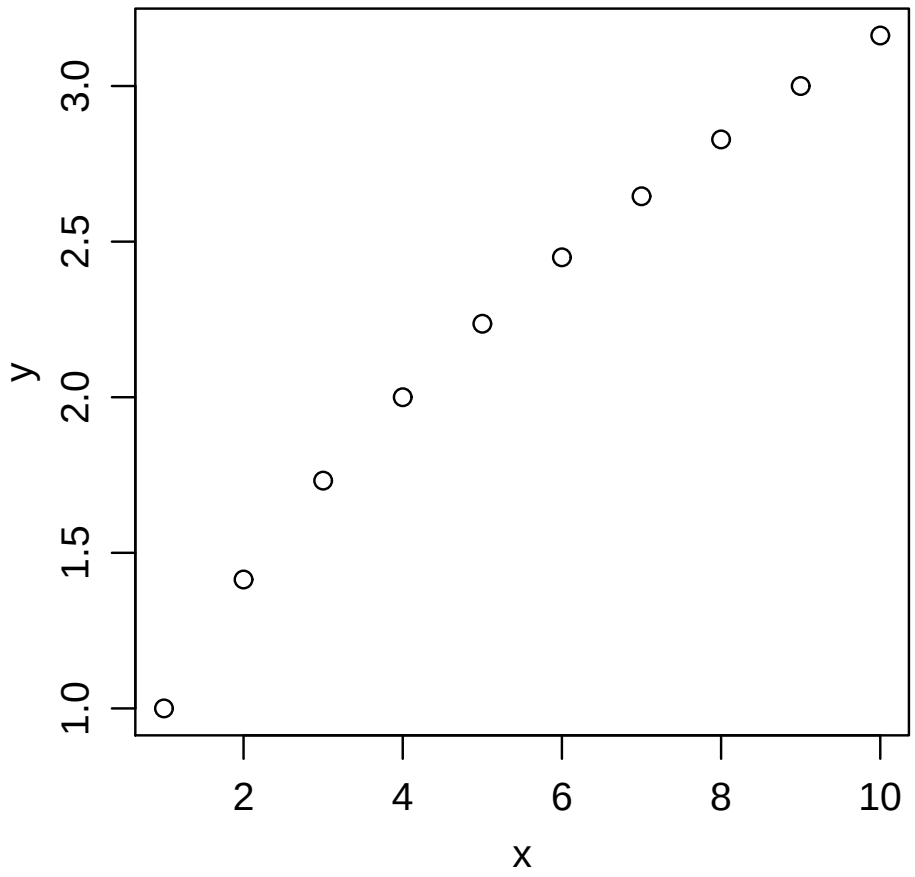
Esse gráfico pode ser modificado acrescentando-se textos e outros elementos.

Experimente utilizar os seguintes comandos. Execute cada um deles e veja o resultado. Não se esqueça, tente entender o que cada argumento realiza no gráfico!

```
> plot(x,y,main="Título")
> plot(x,y,main="Título \n inclusive com acento")
> plot(x,y,main="Título",sub="sub título")
> plot(x,y,main="Título",xlab="Eixo x",ylab="Eixo y")
```

Figura 1: Um simples gráfico de dispersão.

```
> x<-1:10
> y<-sqrt(x) # sqrt=raiz quadrada
> plot(x,y)
```



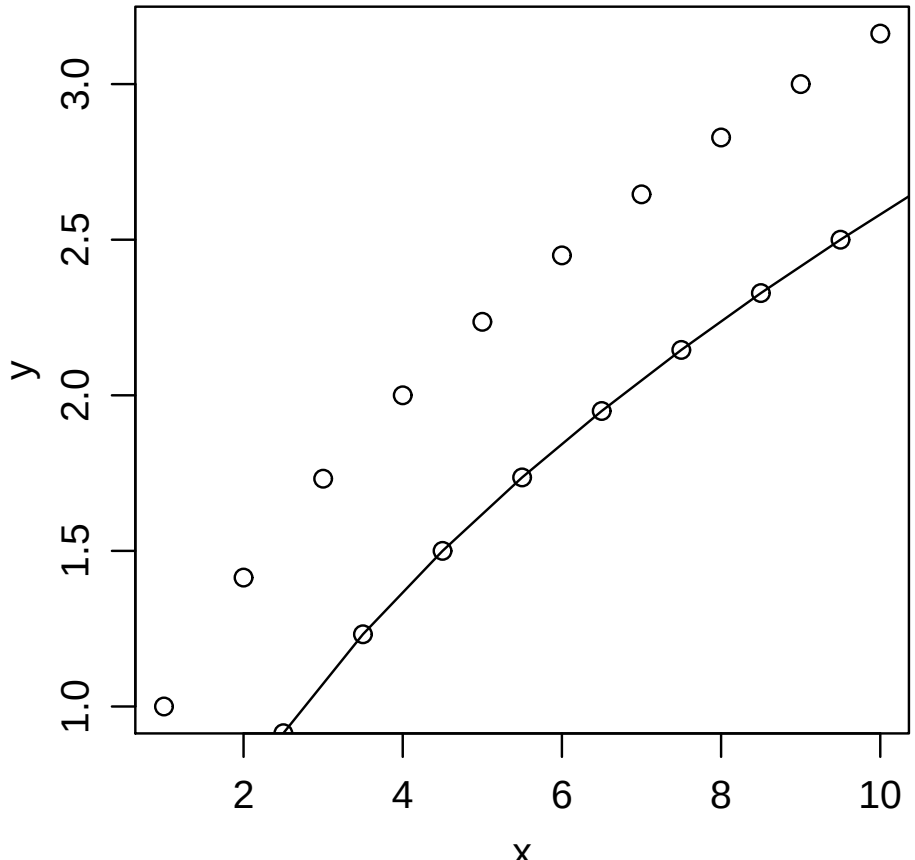
```
> plot(x,y,main="Título",xlab="Eixo x",ylab="Eixo y",type="l")
> plot(x,y,main="Título",xlab="Eixo x",ylab="Eixo y",type="l",col=2)
> plot(x,y,main="Título",xlab="Eixo x",ylab="Eixo y",type="l",col=2,axes=F)
> plot(x,y,main="Título",xlab="Eixo x",ylab="Eixo y",type="b",col=2)
```

Quando um novo gráfico é gerado, ele sobrepõe o anterior. Se você quer abrir uma nova janela gráfica, digite `windows()`,`X11()`, `x11()`.

Você pode acrescentar pontos e linhas em um gráfico (figura 2):

Figura 2: Acrescentar pontos e linhas

```
> plot(x,y)
> points(x+.5,y-.5)
> lines(x+.5,y-.5)
```



Linhas também podem ser adicionadas com a função `abline` (Figura 3),

O argumento `pch` define o símbolo que será utilizado no gráfico. Ele é definido por um número ou então pelo símbolo. Veja o exemplo a seguir (figura 4):

O argumento `cex=`, aumenta ou diminui o tamanho dos caracteres gráficos. O padrão (default) é 1 (figura 5).

Figura 3: Função abline.

```
> plot(x,y)
> abline(h=2)
> abline(v=4)
```

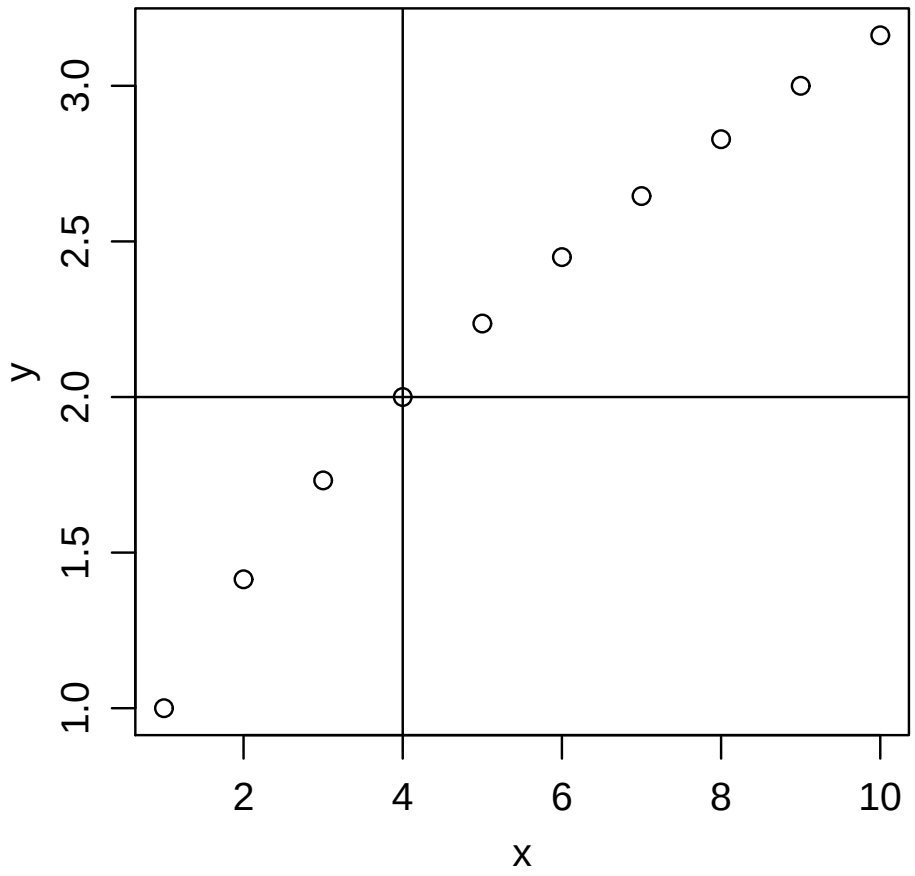


Figura 4: Uso do `pch`.

```
> x<-1:20
> y<-1:20
> plot(x,y,pch=1:20)
```

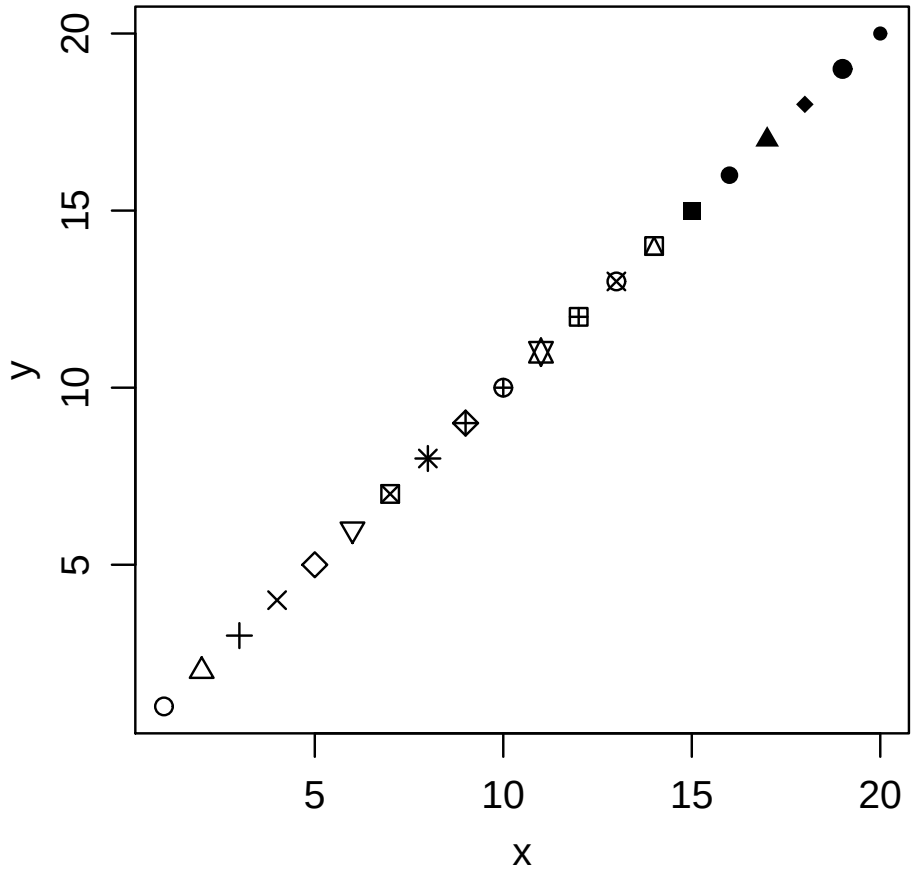
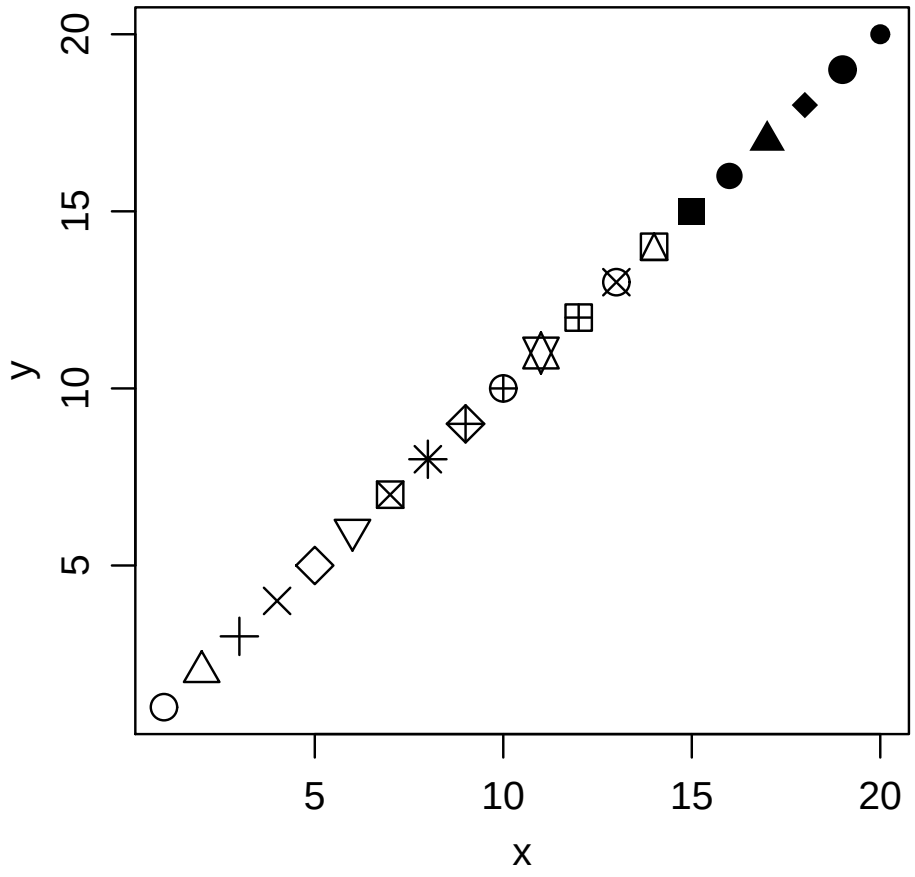


Figura 5: Modificando o amanho de caracteres.

```
> plot(x,y,pch=1:20,cex=1.5)
> plot(x,y,pch=1:20,cex=0.5)
```



A função `locator()` serve para obter as coordenadas de pontos no gráfico. Com a janela gráfica aberta, execute o comando `locator(n=1)`. Ao executar esse comando, observe que o cursor fica piscando e nada acontece. Nesse momento, vá para a janela gráfica e clique em algun ponto dentro do gráfico.

```
locator(n=1) #
```

O resultado, voltando para a janela do editor é a posição do ponto clicado com o mouse na forma de coordenadas x e y.

Se você selecionar n=2, deverá clicar duas vezes dentro do gráfico para obter as coordenadas de dois pontos. **Experimente!**

O comando `identify()` serve para identificar os pontos dentro do gráfico. Execute o comando a seguir,

```
identify(x,y) # identificando vários pontos
```

Clique em um ponto do gráfico!

Cuidado! Se você clicar muito longe de um ponto aparecerá na janela do editor uma mensagem de aviso (warning), indicando que não havia como identificar um ponto.

Para terminar, clique com o botão direito do mouse no Linux e no Windows, clique com o botão direito e selecione 'stop'.

Para inserir uma legenda em um gráfico, utilize a função `legend()`. Forneça as coordenados x e y, o título da legenda, o caracter para representação entre outras opções (figura 6).

Veja esse outro exemplo (figura 7):

Para inserir um texto no meio do gráfico utilize a opção `text()`. Aqui, utilizamos uma forma interativa com a função `locator()`. Na maneira como mostrada a seguir, execute o comando e vá para a janela gráfica e clique em um ponto no gráfico.

```
> text(locator(1),"outlier")
```


Dica: Se você precisa inserir o texto de maneira repetida, aumente o valor da função `locator()`.

Figura 6: Inserir legenda.

```
> plot(x,y)
> legend(x=15,y=5,legend="pontos",pch=1,cex=.5)
```

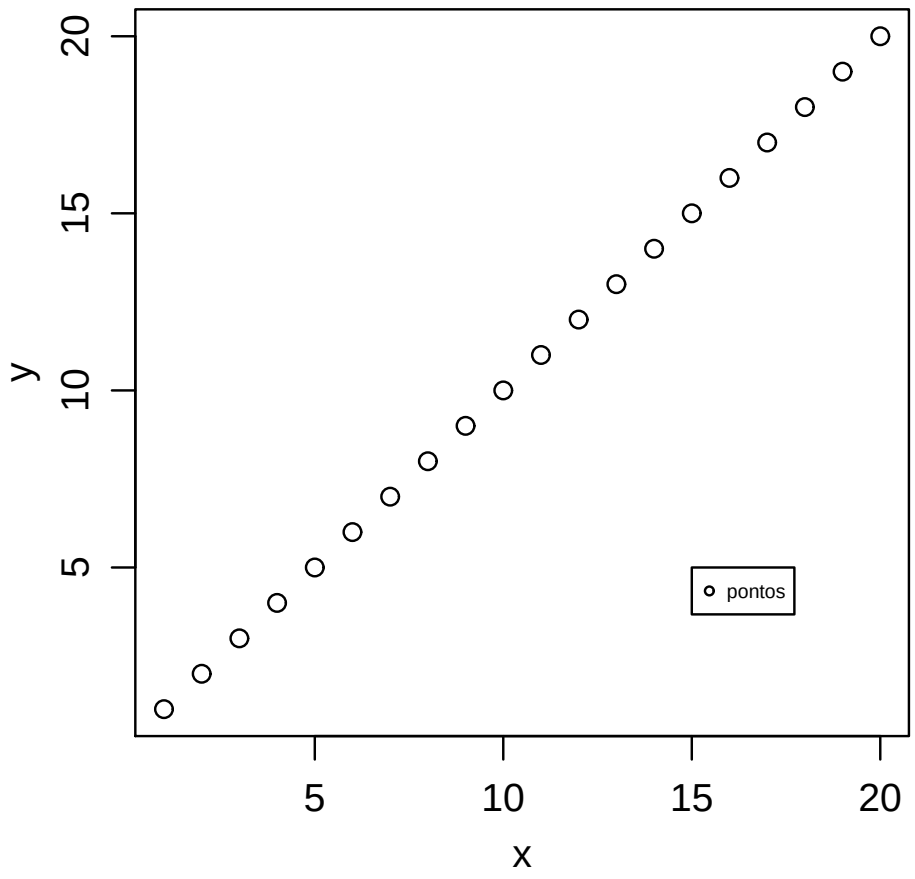
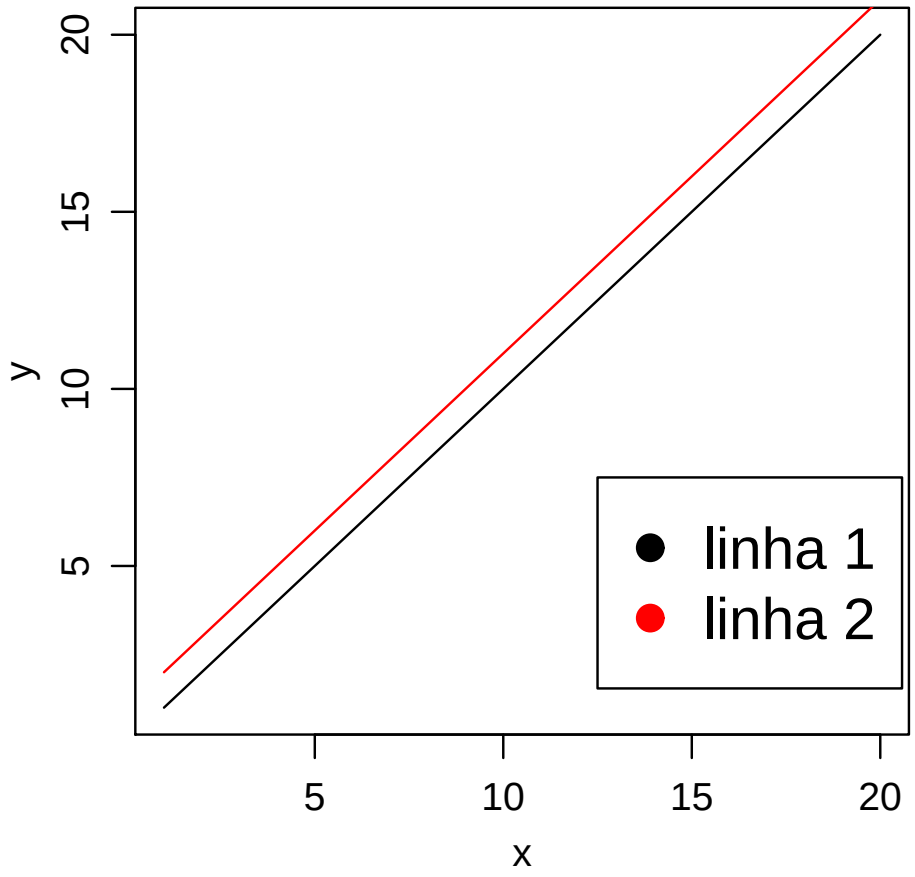


Figura 7: Outro exemplo de legenda.

```
> plot(x,y,type="l")
> lines(x,y+1,col=2)
> legend(12.5,7.5,c("linha 1","linha 2"),pch=19 ,col=1:2,cex=1.5)
```



Como inserir vários gráficos em uma janela? Com o comando `par()`.

O comando `par()` altera muitos parâmetros da janela gráfica. Veja

```
> par()
```

para saber qual a configuração de sua janela gráfica.

Por exemplo, se você quer que dois gráficos sejam postos lado a lado, use

```
> par(mfrow=c(1,2)) #uma linha e duas colunas
```

Com essa opção, a janela gráfica será formatada de modo que você poderá gerar dois gráficos e eles serão colocados lado a lado.

Para que a janela gráfica volte ao normal, é preciso reconfigurá-la.

```
> par(mfrow=c(1,1)) # voltando ao padrão
```

Como salvar um gráfico :

No Windows, utilize a função `savePlot()`

```
> x<-1:10
> y<-sqrt(x)
> plot(x,y)
> savePlot("grafico",type="wmf")
```

No Windows basta apertar o botão direito do mouse sobre o gráfico e escolher salvar como. Selecione o tipo arquivo e o diretório.

No Linux e também no Windows,

```
> pdf("grafico") ou postsript("grafico",hor=F)
> plot(x,y) # o gráfico não é mostrado
> dev.off() # fecha a janela gráfica
```

Nessa opção o gráfico não é mostrado. Por isso, gere o gráfico definitivo e somente após salve o gráfico.

1.14 Trabalhando com números e vetores

O objeto mais simples no **R** é um vetor, que pode consistir de um único número ou letra ou ainda um conjunto deles.

```
>  x.n<-c(89,65,76.5);x.n
```

```
[1] 89.0 65.0 76.5
```

```
>  x.l<-c('a','b','c');x.l
```

```
[1] "a" "b" "c"
```

```
> x.l<-letters[1:3];x.l  # a função letters gera letras minúsculas

[1] "a" "b" "c"

> x.L<-LETTERS[4:5];x.L

[1] "D" "E"
```

Para digitar um vetor de forma mais ágil, pode-se utilizar a função `scan()` . Digite `scan()` e pressione *enter*. Após, aparecerá o algarismo `[1]` indicando a posição 1. Digite um número e pressione *enter*. Continue até terminar a digitação. Quando chegar no último algarismo, pressione *enter* duas vezes.

```
> dados<-scan() #enter
1: 13 # enter
2: 14 # enter
3:    # enter duas vezes
Read 2 items
> dados
[1] 13 14
```

Se os dados estiverem disponíveis em uma planilha, por exemplo, os dados de uma coluna podem ser copiados para a memória utilizando <CTRL> + <C> e depois podem ser copiados para o **R** utilizando

Abra uma planilha eletrônica (por exemplo Excel) e digite alguns dados em uma coluna na planilha. Copie os dados da coluna e cole os dados no **R** após executar o seguinte comando:

```
> dados<-scan() #enter
1:
```

Nesse ponto use <CTRL> + <V> e <ENTER> ao final.

1.15 Selecionando um subconjunto de um vetor

Cada elemento de um objeto pode ser acessado de várias formas:

```
>   x<-1:10;x

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

>   y<-x[5:10] # retira os seis útlimos números do vetor x
>   y

[1]  5  6  7  8  9 10

>   z<-y[-3];z #retira a terceira observação de y

[1]  5  6  8  9 10

>   z[1]<-77   # insere o valor 77 na posição 1
>   z

[1] 77  6  8  9 10

>   a<-1:5
>   c<-a[a>2] # retira os números de a que são maiores do que 2
>   c

[1] 3 4 5
```

1.16 Listas e Data frames

Uma lista é uma coleção de objetos no **R**. Por exemplo,

```
>  dados<-list(nome="Adilson",civil="casado", cachorros=2);dados

$nome
[1] "Adilson"

$civil
```

```
[1] "casado"

$cachorros
[1] 2
```

Observe que os componentes de uma lista não precisam ser da mesma classe e tamanho. Veja também que os objetos *Adilson* e *casado* aparecem entre *aspas* porque não são numéricos.

Os componentes podem ser referenciados de várias maneiras, por exemplo,

```
> dados[3]

$cachorros
[1] 2

> dados[[3]]

[1] 2

> dados$cachorro

[1] 2
```

A função `length(dados)` aplicada a uma lista, fornece o número de objetos na lista ou o tamanho da lista.

```
> length(dados)

[1] 3
```

Pode-se digitar apenas o mínimo de letras para identificar o objeto. Mas, se houver mais de um elemento com a mesma inicial deve-se digitar mais letras para diferenciar os elementos.

Os elementos de uma lista podem ser acessados pelo nome do objeto seguido pelo símbolo $ e o nome do elemento da lista.

```
> dados$n

[1] "Adilson"

> dados$c

NULL

> dados$ca

[1] 2
```

1.17 Como alterar valores da lista

Para alterar algum valor da lista, ou acrescentar mais elementos você pode utilizar

```
> dados[[3]]<-1 # ou
> dados$cachorros<-1
> dados$profissao<-"professor"
> dados

$nome
[1] "Adilson"

$civil
[1] "casado"

$cachorros
[1] 1

$profissao
[1] "professor"
```

Observe nesse caso que a palavra *professor* está entre aspas. Isso é necessário porque uma palavra não é algo conhecido dentro do **R**.

1.18 Data frame

Um `data.frame` nada mais é do que uma lista organizada, onde todos os elementos são vetores e tem o mesmo tamanho. Um `data.frame` pode ser criado com a função `data.frame()`.

Mas, quando usamos a função `read.table()` , o resultado é um `data.frame` atribuído a algum objeto no **R**. Esse objeto basicamente será uma "tabela" com vetores que podem ser lógicos, caracteres ou números.

1.19 Lendo dados de arquivos externos

Existem vários formatos de arquivos que podem ser lidos diretamente no **R**.

A função `read.table()` é utilizada para a leitura externa de dados no formato txt, por exemplo. Em geral, usa-se quando temos uma tabela para ser inserida.

Aqui, uma tabela é entendida como um objeto contendo linhas e colunas. Todas as colunas possuem o mesmo tamanho. Podem ter um cabeçalho ou não.

A sintaxe do comando é

```
> dados<-read.table("c:\\arquivo.txt",header=T,na.strings="-")
```

Para ver o arquivo basta digitar `dados`.

O argumento `header`= T indica que o arquivo de dados possui um cabeçalho , ou seja, as colunas possuem um título.

O argumento *na.strings*= *"-"* indica que as observações faltantes estão representadas pelo sinal -. Se não houver algum caracter indicador, basta colocar as `aspas vazias` ou simplesmente não colocar essa opção.

No lugar de *arquivo.txt* pode ser usado um endereço da Web . Tente, por exemplo, *http://www.ufpr.br/∼aanjos/ead/dados/planilha.txt*.

O arquivo .txt, pode ser gerado, por exemplo, através de uma planilha eletrônica ou algum editor de textos (Pico, Nano, Vi, Notepad, Word). A planilha deve ser salva como texto separado por espaços.

Há outras funções de importação de arquivos do tipo texto, como por exemplo: `read.csv()`, `read.delim()`.

Existem muitos outros formatos de leitura de dados no **R**. Veja o pacote `foreign` para saber como importar dados de outros aplicativos como o SAS e SPSS por exemplo.

Para ler um arquivo no formato .xls é necessário utilizar o pacote `gdata`. Dentro do pacote há uma função chamada `read.xls()`. A sintaxe é a mesma utilizada na função `read.table()`.

1.20 Usando search(), attach() e detach()

A função `search()` mostra o caminho de procura do **R**. Quando executa-se essa função o **R** mostra o .GlobalEnv, pacotes, autoloads ou data.frames e listas adicionadas com o comando `attach()`.

Para ver qual ou quais objetos estão "atachados", digite

```
> search()
```

```
 [1] ".GlobalEnv"        "package:MASS"      "package:stats"
 [4] "package:graphics"  "package:grDevices" "package:utils"
 [7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
```

Com isso, aparecerá o nome do objeto ou dos objetos que estão "atachados".

Quando necesssitamos fazer muitas referências para um objeto, podemos minimizar a digitação, colocando o `data.frame` ou `lista` no caminho de procura, através do comando `attach()`.

Observe que o objeto *dados* não está atachado. Se você digitar

```
> nome
```

Aparecerá uma mensagem de erro, alertando que o objeto não foi encontrado. Nesse caso, o correto seria utilizar o seguinte comando:

```
> dados$nome
```

```
[1] "Adilson"
```

Para evitar ter que digitar `dados$nome` utiliza-se a função `attach()` da seguinte maneira:

```
> attach(dados)
> search()
```

```
 [1] ".GlobalEnv"        "dados"             "package:MASS"
 [4] "package:stats"     "package:graphics"  "package:grDevices"
 [7] "package:utils"     "package:datasets"  "package:methods"
[10] "Autoloads"         "package:base"
```

Veja que o objeto ”dados”aparece agora no caminho de procura do **R**!

Isso permite referenciar os componentes de uma lista , por exemplo, simplesmente digitando

```
> nome
```

```
[1] "Adilson"
```

Cuidado!

Se você possui objetos com o mesmo nome em diferentes listas, isso pode causar alguma confusão. Antes de começar outra análise com outros dados, não esqueça de usar a função `detach()` para retirar o objeto do caminho de procura:

```
> detach(dados)
```

1.21 Como instalar um pacote

No site do **R** www.r-project.org existem centenas de pacotes disponíveis. Pacotes ou packages em inglês, são, em geral, formados por um conjunto de funções que realizam análises estatísticas específicas. Podem, contudo, ser apenas um repositório de arquivos de dados. Por exemplo, alguns autores de livros que utilizam o **R** disponibilizam os arquivos de dados em pacotes.

Todo pacote possui obrigatoriamente um responsável por fazer atualizações e corrigir as possíveis falhas.


Para saber mais: A função `package.skeleton()` é utilizada para criação de pacotes no **R**.

Na instalação do **R** somente alguns pacotes importantes são instalados junto com o programa. Se você necessitar de algum pacote em especial, deverá fazer o download diretamente da página do **R**. A obtenção dos pacotes pode ser realizada de várias maneiras.

No Windows, se você estiver conectado a internet, vá no menu Pacotes e em seguida selecione a opção ”Escolher espelho CRAN“. Selecione um repositório qualquer. Em geral, escolhe-se o mais próximo geograficamente. Por exemplo Brazil(1) PR. Em seguida, vá em Instalar pacotes e escolha o pacote desejado. Será iniciado o download do pacote. O tempo de download varia para cada pacote escolhido e claro da qualidade de conexão da internet. Finalizado o download com sucesso, você terá instalado no seu computador o pacote. Para utilizá-lo no **R**, você deverá chamar o pacote toda vez que desejar alguma função, utilizando a função `library(nomedopacote)` ou `require()`, por exemplo.

Se você não tiver acesso a Internet em seu computador, os pacotes podem ser instalados de arquivos `.zip` previamente copiados. Basta apenas indicar o arquivo.

No Laboratório de Estatística, todos os pacotes já estão instalados. Basta apenas saber qual pacote deve ser carregado e utilizar a função `library(nomedopacote)`.

1.22 Obtendo dados de outros “pacotes” (packages)

Vários packages do **R** possuem arquivos de dados bastante interessantes. Eles podem ser usados para ilustrar exemplos de uso das funções indicadas.

Para utilizar os dados, utilize o comando `data()`. Digitando somente `data()`, aparecerá uma lista de todos os conjuntos de dados disponíveis.

Se o package não estiver carregado, utilize, por exemplo,

```
> data(cats,package="MASS") # ou
> library(MASS) # no Linux ou Windows
```

```
> require(MASS) # no Linux ou Windows
> data(cats)
```

Para ter informações sobre o arquivo de dados use

```
> ?cats
```

1.23 Exercícios:

Não é necessário entregar esse exercício. Ele serve apenas para você praticar o que aprendeu nesse módulo.

1. Gere a seguinte sequência utilizando a função `seq` do **R**:

```
> x

[1]  1  4  7 10 13 16 19
```

2. Obtenha: $\sum(x), \sum(x^2), \sum(x)^2$

3. Utilize a função `rep()` e gere as seguintes sequências:

```
> A

[1] 1 1 1 2 2 2

> B

[1] 1 2 3 1 2 3
```

4. Considere as informações contidas na Tabela 1. A partir dessa tabela, crie um no **R** um *data frame* com o nome `dados`. As colunas A e B devem ser do tipo `factor` e a coluna resposta deve ser do tipo `numeric`.

Tabela 1: Dados para análise.

A	B	Resposta
1	1	5
1	2	6
1	3	7
2	1	9
2	2	10
2	3	11

   Após criar o objeto dados, utilize a função `summary()`. Se o objeto estiver correto, deverá aparecer o número de níveis de cada fator (A e B) e as estatísticas descritivas da variável Resposta.

5. Crie um arquivo de dados em uma planilha eletrônica. Salve esse arquivo com as extensões `.xls` `.txt` e carregue esses arquivos para o **R**.

6. Escolha um pacote disponível na página do **R**. Baixe e carregue esse pacote no seu computador. Teste o pacote com algum exemplo. **Atenção**: Lembre-se que existem pacotes que exigem que outros pacotes também sejam instalados para poderem funcionar corretamente.

7. Construa o seguinte gráfico no **R**. Utilize as funções `abline` e `points`.