ANÁLISE GRÁFICA
COM USO DO R

©Adilson dos Anjos
Departamento de Estatística
- UFPR -

Curitiba, 27 de outubro de 2006.

Apresentação

O objetivo dessa disciplina é fomentar o uso da análise gráfica com o uso do software **R**. Não será dado enfoque em técnicas quantitativas de análise de dados, mas, em alguns momentos essa associação é de certa forma indispensável.

Além das técnicas gráficas usuais, pretende-se incentivar o uso de alternativas de análise e interpretação de dados estatíticos. O **R**, pelas suas características de programação, permite que sejam criados vários tipos de análise gráfica, seja uma nova metodologia ou uma combinação de metodologias já existentes.

Por fim, pretende-se que o aluno desenvolva a capacidade de utilizar metodologias gráficas para análise de dados estatísticos utilizando o software **R**.

Sumário

Lista de Figuras

Lista de Tabelas

Parte I
Manipulação de gráficos no R

Capítulo 1
Sessão inicial

Sumário

1.1 Gerenciamento das atividades

Para gerenciar os materiais produzidos durante o curso, sugere-se que cada aluno crie um diretório dentro da sua área de trabalho:

```
$ mkdir ce231
$ cd ce231
$ R
```

Assim, todas as tarefas realizadas com o **R** ficará armazenadas nesse diretório. Toda vez que iniciar um sessão de trabalho, entre nesse diretório, e abra o **R**.

Não se esqueça de, ao final, salvar sua sessão de trabalho.

```
> q()
Save workspace image? [y/n/c]: y
```

1.1.1 Emacs

Se você possui algum conhecimento sobre o *Emacs* é recomendável utilizá-lo durante a utilização do conteúdo desse material. Além de agilizar alguns comandos é possíel fazer comentários e modificar funções gráficas mais facilmente.

No caso de utilizar o sistema operacional Windows, você pode utilizar um *script* de forma semelhante ao *Emacs*.

1.2 Gráficos no R

No **R** existem pacotes gráficos (`packages`) que geram os gráficos propriamente ditos e, além disso, existem diversos outros pacotes que possuem funções gráficas específicas. Existem ainda, os sistemas gráficos que implementam outras funcionalidades aos gráficos gerados pelos diferentes pacotes. Ainda, esses sistemas permitem integrar os pacotes gráficos, através de funções no pacote `grDevices`, possibilitando uma maior versatilidade de trabalho com cores e tipos de fontes, por exemplo.

No **R** existem três tipos de funções gráficas:

- funções de alto nível (*high-level*), que produzem gráficos completos;
- funções de baixo nível (*low-level*), que adicionam informações a um gráfico existente e,
- funções para trabalhar interativamente com gráficos.

Existem muitos pacotes que trabalham com funcionalidades gráficas no **R**. Por isso, antes de tentar criar algo novo, faça uma busca detalhada no site do **R**, principalemnte nas listas de emails do site do **R** e na internet.

Funções de "alto nível"

As funções de alto nível podem ser aplicadas através do comando `par()` ou como argumentos em funções gráficas como `plot()`, por exemplo. Experimente ver quais são as opções da função `par()`. Para isso, digite o seguinte

comando no **R**:

Aparecerá no arquivo de ajuda, uma lista das opções que podem ser utilizadas, os possíveis valores de cada uma e, no final, alguns exemplos de aplicação. Na tabela 1.1, são apresentadas algumas dessas funções. Experimente aplicá-las em um gráfico!

Antes de iniciar com um exemplo, é recomendável 'salvar' a configuração original da função `par()`.

```
> op <- par()
```

Para restaurar a função original, após ter experimentado os comandos da tabela 1.1, basta utilizar

```
> par(op)
```

Tabela 1.1: Funções gráficas de alto nível.

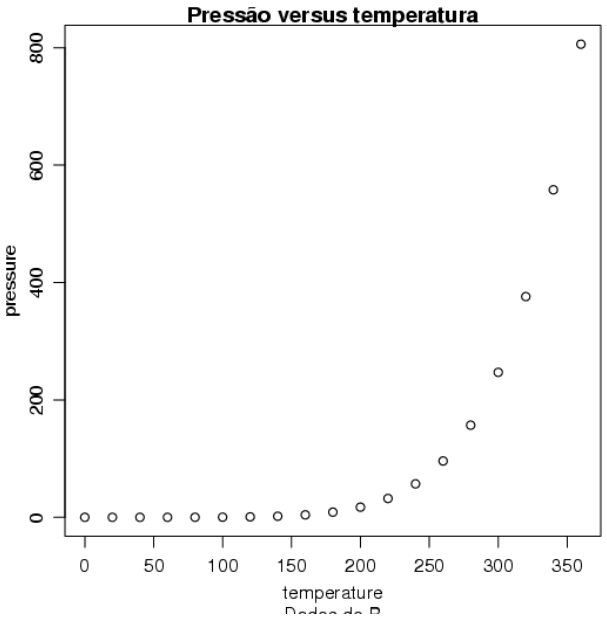
Função	descrição
`adj`	justificação de texto
`ann`	inserção de títulos
`bg`	cor de fundo
`bty`	tipo de caixa do gráfico
`cex`	tamanho do texto
`cex.axis`	tamanho do 'tick' dos eixos
`cex.lab`	tamanho dos nomes dos eixos
`cex.main`	tamanho do título do gráfico
`cex.sub`	tamanho do sub-título do gráfico
`col`	cor de linhas e símbolos
`col.axis`	cor do 'tick' dos eixos
`col.lab`	cor dos nomes dos eixos
`col.main`	cor do título do gráfico
`col.sub`	cor do sub-título
`fg`	cor das linhas externas
`font`	fonte para texto
`font.axis`	fonte para 'tick' dos eixos
`font.lab`	fonte para nome dos eixos
`font.main`	fonte para o título
`font.sub`	fonte para o subtítulo
`gamma`	correção para cores
`lab`	número de 'ticks' nos eixos
`las`	rotação de textos nas margens dos eixos
`lty`	tipo de linha
`lwd`	largura da linha
`mgp`	localização de 'ticks' e nomes de 'ticks'
`pch`	tipo de símbolo
`srt`	rotação de de texto na região do plot
`tck`	comprimento dos 'ticks' (em relação ao gráfico)
`tcl`	comprimento dos 'ticks' (em relação ao texto)
`tmag`	tamanho do título do gráfico (em relação a outros nomes)
`type`	

	tipo de gráfico (pontos, linhas, ambos)
`xaxp`	número de 'ticks' sobre eiso 'x'
`xaxs`	cálculo da amplitude de escal do eixo x
`xaxt`	estilo do eixo x (padrão ou nenhum)
`yaxp`	número de 'ticks' sobre eiso 'y'
`yaxs`	cálculo da amplitude de escal do eixo y
`yaxt`	estilo do eixo y (padrão ou nenhum)

Como sugestão, tente aplicar as opções da função `par()` no gráfico:

Figura 1.1: Um gráfico para treinar.

```
> plot(pressure, main = "Pressão versus temperatura", sub = "Dados do R")
```



Para ver quais são os parâmetros atuais, definidos pela função `par()`, digite o seguinte:

```
> par(c("bty", "ann"))

$bty
[1] "o"

$ann
[1] TRUE
```

As mudanças efetuadas através da função `par()` permanecem inalteradas até que outra mudança seja efetuada nessa função. Ou seja, para qualquer gráfico gerado a partir de uma nova configuração da função `par()`, todos terão as mesmas características. Se a mudança for efetuada através da função `plot()`, por exemplo, essa mudança será temporária e não afetará novos gráficos.

Funções de "baixo nível"

As funções de “baixo nível”, ao contrário da anterior, podem ser modificadas apenas através da função `par()`. Estas funções são listadas na tabela 1.2.

Tabela 1.2: Funções gráficas de baixo nível.

Função	descrição
`ask`	perguntar antes de criar um novo gráfico
`family`	família da fonte de texto
`mfcol`	número de figuras na página
`mfrow`	número de figuras na página
`mfg`	define a figura atual
`new`	novo gráfico em um região da figura

Ainda existem outras funções de ’baixo nível’ que tratam de margens e localização de figuras dentro da região do gráfico que serão tratadas mais adiante.

Experimente utilizar alguns dos exemplos a seguir, para treinar o uso de algumas funções da tabela 1.2:

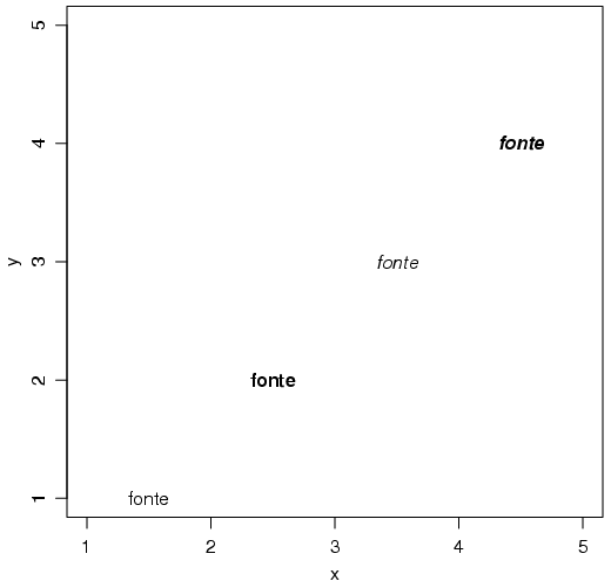
```
> plot(pressure)

> par(mfrow = c(1, 2))
> plot(pressure)
> plot(pressure)

> par(mfrow = c(2, 1))
> plot(pressure)
> plot(pressure)

> plot(pressure, cex = 1.5)
> par(new = TRUE)
> plot(pressure)
```

```
> par(mfrow = c(1, 2))
> plot(pressure)
> plot(pressure)
> text(100, 400, "lá")
> par(mfg = c(1, 1))
> text(100, 300, "aqui")
```

A função `text()` é utilizada aqui para adicionar um texto no gráfico.

```
> x <- 1:5
> y <- 1:5
```

```
> par(family = "", mfrow = c(1, 1))
> plot(x, y, type = "n")
> text(1.5, 1, "fonte", font = 1)
> text(2.5, 2, "fonte", font = 2)
> text(3.5, 3, "fonte", font = 3)
> text(4.5, 4, "fonte", font = 4)
```



Consulte outras opções da função `par()` e experimente alterá-las no gráfico anterior!

1.2.1 Tipos de saídas gráficas

Os gráficos gerados no **R** podem ser salvos em diferentes formatos de arquivo: `postscript`, `pdf`, `pictex` (LaTeX), `xfig`, `bitmap`, `png` e `jpeg` e exclusivamente no Windows, `win.metafile` e `bmp`.

No **R**, uma 'saída' gráfica é direcionada para um dispositivo em particular, que gerencia o formato do arquivo que será criado. Assim, quando deseja-se que uma saída gráfica seja salva, deve-se abrir um dispositivo gráfico, para receber essa saida, e depois esse dispositivo deve ser fechado.

Veja o seguinte exemplo para gerar um simples gráfico no formato `postcript`:

```
> postscript("meugrafico.ps")
> plot(uspop)
> dev.off()
```

1.2.2 Linhas, pontos e texto

As funções `lines()`, `points()` e `text()` permitem inserir, em um gráfico, linhas, pontos e texto, respectivamente. Para saber quais opções podem ser utilizadas, utilize a opção de ajuda de cada uma das funções.

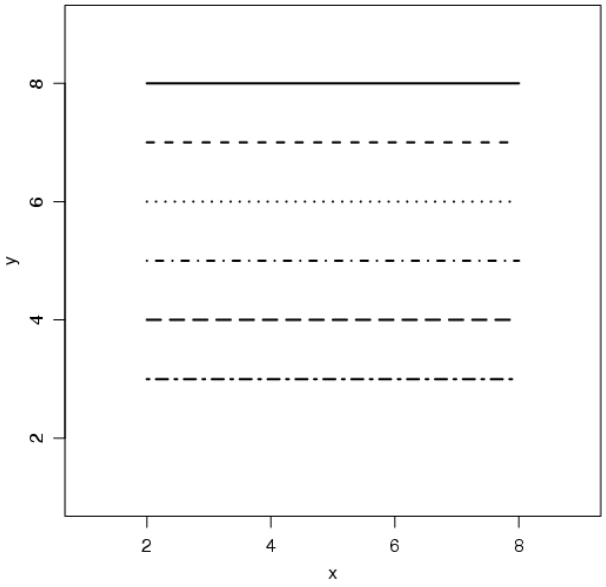
Linhas

Os dois principais comandos relacionados com linhas são `lty` e `lwd`. O primeiro controla o tipo de linha: sólida, pontilhada. O segundo, controla a largura da linha.

A largura da linha é especificada por um número, por exemplo `lwd=2`. Esse valor varia com o tipo de dispositivo empregado para visualização. Na tela de um monitor, o número 1, por exemplo, representa um *pixel*. Veja alguns

exemplos a seguir.

```
> x <- 1:9
> y <- 1:9
> plot(x, y, type = "n")
> lines(c(2, 8), c(8, 8), lwd = 2)
> lines(c(2, 8), c(7, 7), lty = 2, lwd = 2)
> lines(c(2, 8), c(6, 6), lty = 3, lwd = 2)
> lines(c(2, 8), c(5, 5), lty = 4, lwd = 2)
> lines(c(2, 8), c(4, 4), lty = 5, lwd = 2)
> lines(c(2, 8), c(3, 3), lty = 6, lwd = 2)
```

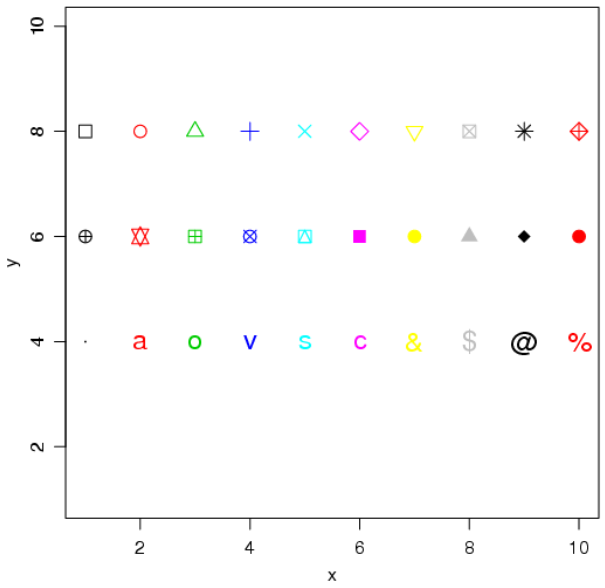


No **R** é possível, além de utilizar os formatos pré-definidos de linhas em `lty`, utilizar versões 'pessoais'. Esse formatos devem ser fornecidos como *strings* (entre aspas). Um número ímpar especifica o comprimento da linha e um número par especifica o tamanho do espaço vazio (*gap*).

Pontos

No **R** pode-se adicionar diferentes tipos de pontos em gráfico. Basta definir qual símbolo deve ser utilizado entre os 26 disponíveis. Utiliza-se a função `points()`. Ainda, é possível utilizar caracteres como síbolos. Simplesmente, defina o símbolo entre aspas.

```
> x <- 1:10
> y <- 1:10
> plot(x, y, type = "n")
> points(1:10, rep(8, 10), pch = 0:9, col = 1:10, cex = 1.5)
> points(1:10, rep(6, 10), pch = 10:19, col = 1:10, cex = 1.5)
> points(1:10, rep(4, 10), pch = c(".", "a", "o", "v", "s",
+      "c", "&", "$", "@", "%"), col = 1:10, cex = 1.5)
```



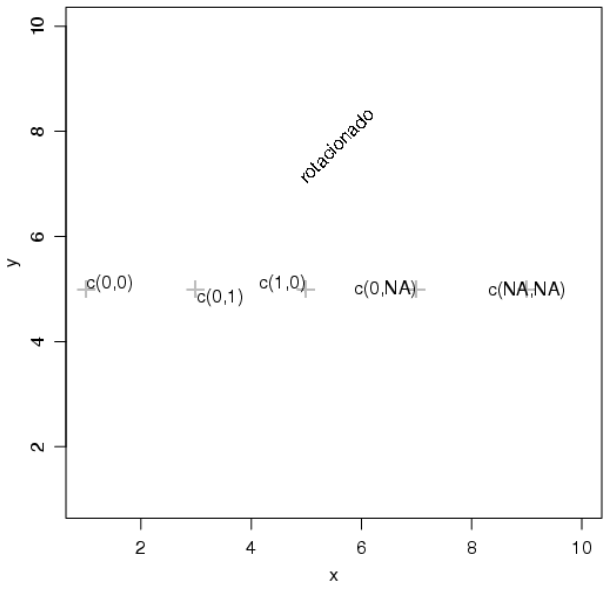
Experimente utilizar a função `show.pch()` do pacote `Hmisc`,

```
> require(Hmisc)
> show.pch()
```

Texto

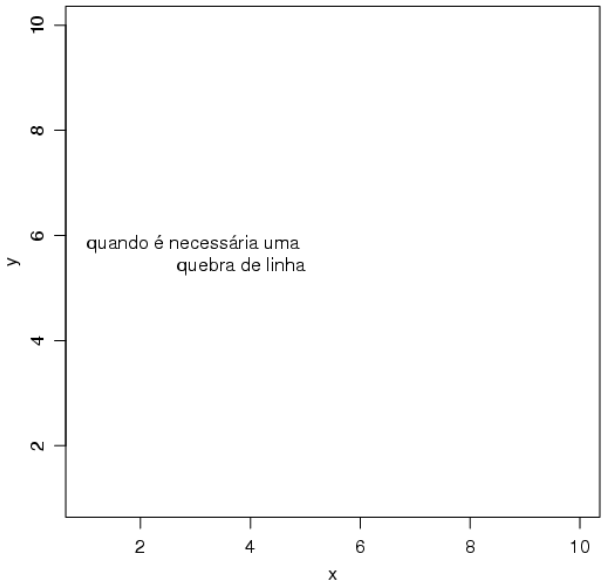
Com a função `text()`, pode-se inserir textos no gráfico. Nessa função deve-se informar a posição do texto, fornecendo as coordenadas em relação à região do gráfico. Uma opção útil nessa função é `adj`, que indica a justificação do texto. Nessa opção, 0 significa um alinhamento à esquerda, 0.5 centrado e 1 justificado à direita. Ainda, na opçãp `adj`, pode-se fornecer a informação da forma *c(hjust,vjust)*, onde *hjust* indica o alinhamento horizontal e *vjust* um alinhamento vertical. Veja alguns exemplos:

```
> x <- 1:10
> y <- 1:10
> plot(x, y, type = "n")
> points(seq(1, 9, by = 2), rep(5, 5), pch = "+", col = "gray",
+       cex = 2)
> text(1, 5, "c(0,0)", adj = c(0, 0))
> text(3, 5, "c(0,1)", adj = c(0, 1))
> text(5, 5, "c(1,0)", adj = c(1, 0))
> text(7, 5, "c(0,NA)", adj = c(1, NA))
> text(9, 5, "c(NA,NA)", adj = c(0.5, 0.5))
> text(5, 7, "rotacionado", adj = c(0, 0), srt = 45)
```



Quando um texto é muito longo ou deseja-se fazer a quebra de uma linha, deve-se utilizar a opção '\ n' no ponto de quebra de uma linha.

```
> x <- 1:10
> y <- 1:10
> plot(x, y, type = "n")
> text(5, 6, "quando é necessária uma \ n quebra de linha",
+      adj = c(1, 1))
```



Comandos locator() e identify()

Estes comandos são utilizados depois que os gráficos são criados.

- `locator()`: serve para encontrar as coordenadas de um ponto no gráfico;
- `identify()`: serve para mostrar os nomes (labels) de pontos em um gráfico.

locator()

Experimente usar o exemplo seguinte. Gere o gráfico e depois, na janela gráfica, selecione um ponto com o 'mouse' (dê um 'clique' dentro do gráfico). Nesse momento, aparecerá na janela do **R** as coordenadas do ponto.

```
> x <- 1:10
> y <- 1:10
> plot(x, y, type = "n")
> locator(n = 1)
```

Agora, utilize o mesmo gráfico, definindo a opção com *n* = 2 e *type='l'*.

```
> locator(2, type = "l")
```

Com o 'mouse', selecione dois pontos dentro do gráfico.

Experimente, também, utilizar outras opções para *type* (veja *help(locator)*).

A função `locator()`, também, pode ser utilizada em associação com a função `points()`.

```
> points(locator(3), type = "l")
```

e com a função `text()`,

```
> text(locator(1), "meu texto")
```

Da maneira a seguir, você deve clicar duas vezes no gráfico.

```
> text(locator(2), c("meu texto", "outro texto"))
```

Para cancelar o processo antes de localizar todos os pontos, aperte o botão direito do 'mouse'. No Windows, apertando o botão direito é mostrada a opção *'stop'*.

Cuidado! Se a janela gráfica for alterada antes que a total identificação dos pontos seja feita, os pontos desaparecem.

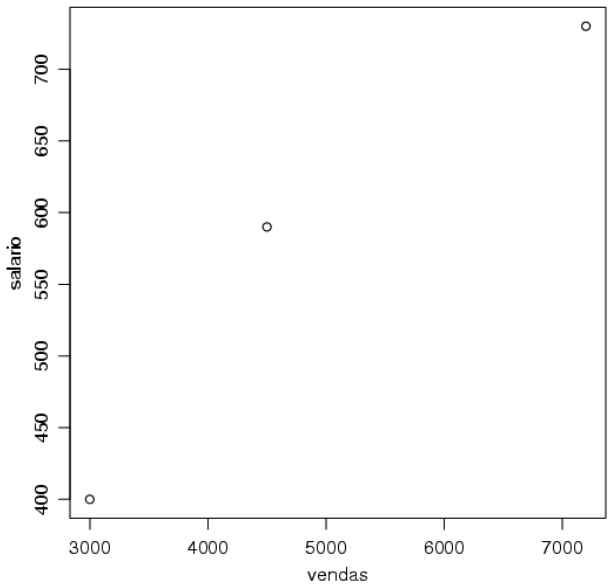
identify()

Utiliza-se a função `identify()` para dar nomes aos pontos de um gráfico. Para utilizar essa função são necessárias as coordenadas (**x**,**y**) e um vetor com nomes dos pontos (string):

```
> identify(x, y, labels = nomes)
```

Como exemplo, considere o seguinte conjunto de dados e o respectivo gráfico:

```
> nomes <- c("João", "Pedro", "Paulo")
> salario <- c(400, 590, 730)
> vendas <- c(3000, 4500, 7200)
> plot(vendas, salario)
```



Para identificar cada ponto no gráfico, utilize a função `identify()` da seguinte maneira:

```
> identify(vendas, salario, labels = nomes)
```

Agora, clique próximo aos pontos para poder identificar cada um deles. Caso não deseje identificar todos, aperte o botão direito do 'mouse'.

Se o ponteiro do 'mouse' ficar muito distante do ponto, aparecerá uma mensagem de aviso no **R**. Continue 'clicando' até conseguir se aproximar do ponto desejado.

1.2.3 Cores

Existem, basicamente, três maneiras de definir cores nos gráficos. Pode-se utilizar os comandos `fg`, `bg` e `col`. A função `col` possui variações, podendo ser aplicada em várias situações, como mudança de cor de eixos, títulos etc, como visto na tabela 1.1.

Dependendo do local onde a função `col` é utilizada, ela pode funcionar de diferentes maneiras. Por exemplo, dentro da função `plot()` ela pode ser utilizada para colorir símbolos, texto, linhas. Já, dentro da função `barplot()` ela é utilizada para colorir as barras.

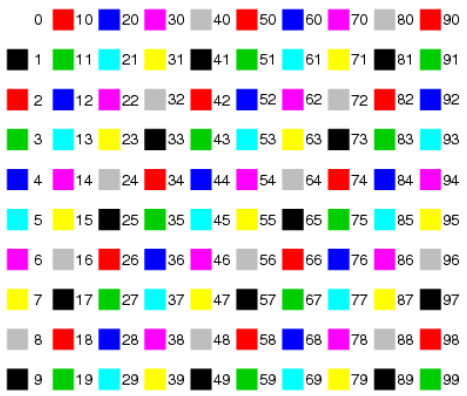
A função `fg` é utilizada para colorir eixos e bordas. Observe que, essa opção pode causar alguma sobreposição de cores se utilizada junto com comandos específicos de cores para eixos e texto do tipo `col.axis`, `col.main`.

A função `bg` é utilizada para controlar a cor de fundo de um gráfico.

Para especificar uma cor, pode-se utilizar números ou *strings* (nome da cor). Por exemplo, *'blue'* indica que a cor azul foi definida. No **R** existem 657 nomes de cores para serem utilizados. Experimente a opção `colors()` para ver os nomes de cores disponíveis. A função `palette()` também fornece alguns nomes de cores.

Experimente utilizar a seguinte função do pacote `Hmisc`:

```
> show.col()
```



Outras formas de especificar cores

A função `rgb()` permite que sejam especificadas cores como uma combinação de VERMELHO-VERDE-AZUL (**R**ed,**G**reen,**B**lue). por exemplo, a cor vermelha é especificada como `rgb(1,0,0)`.

Se você quer saber qual são os números que devem ser fornecidos para uma determinada cor, utilize a função `col2rgb()`. Veja o exemplo a seguir:

```
> col2rgb("green")
```

```
      [,1]
red      0
green  255
blue     0
```

```
> col2rgb("gray")
```

```
      [,1]
red    190
green  190
blue   190
```

Os valores da função `rgb()` também podem ser fornecidos na forma de um *string* `#RRGGBB`, em que cada par corresponde a um numero de 0 a 255(FF).

A função `rgb()` possui uma opção maxColorValue que define a amplitude de variação dos números.

Veja o exemplo, a seguir, para ver como utilizar cores com `rgb` (aproveite para conhecer a função `rect()`):

```
> plot(c(100, 200), c(300, 450), type = "n", xlab = "", ylab = "")
> rect(100, 300, 125, 350)
> rect(100, 400, 125, 450, col = "green", border = "blue")
> rect(115, 375, 150, 425, col = par("bg"), border = "transparent")
> rect(150, 300, 175, 350, density = 10, border = "red")
> rect(150, 400, 175, 450, density = 30, col = "blue", angle = -30,
```

```
+       border = "transparent")
> minha.cor <- rgb(5, 55, 10, max = 100)
> rect(100, 300, 125, 350, col = minha.cor)
```

Conjuntos de cores

Existem ainda alguns conjuntos de cores já definidos no **R**. Esses conjuntos formam alguns padrões que podem ser utilizados em diferentes tipos de gráficos. Esses padrões, em geral, são utilizados em gráficos que necessitam de mais de uma cor para ser aplicada.

Tabela 1.3: Conjuntos de cores.

Conjunto	descrição
`rainbow()`	cores do arco-íris
`heat.colors()`	varia do branco até laranja/ vermelho
`terrain.colors()`	varia do branco até marrom/ verde
`topo.colors()`	varia do branco até marron/ verde/ azul
`cm.colors()`	varia do azul claro até branco/ magenta
`grey()` ou `gray()`	tipos de cinza

Dentro dos parênteses, são inseridas opções, em geral, o número de cores. Experimente o seguinte exemplo no **R**:

```
> example(rainbow)
```

Observe que, a cor final pode depender da resolução do vídeo, do tipo de impressora, tipo de papel além de outras condições. Todas as cores geradas, são armazenadass pelo **R** como cores do tipo `rgb()`.

Veja alguns exemplos de cores na função de demonstração `graphics`:

```
> demo(graphics)
```

Experimente ainda, utilizar essa função para entender um pouco mais sobre cores

```
> cores.ex <- function() {
+       plot(1, 1, xlim = c(0, 14), ylim = c(0, 3), type = "n",
+            axes = F, xlab = "", ylab = "")
+       text(1:6, rep(2.5, 6), paste(1:6), col = palette()[1:6],
+            cex = 2.5)
+       text(10, 2.5, "palette (default)", adj = 0)
+       rainchars <- c("R", "O", "Y", "G", "B", "I", "V")
+       text(1:7, rep(1.5, 7), rainchars, col = rainbow(7), cex = 2.5)
+       text(10, 1.5, "rainbow(7)", adj = 0)
+       cmtexto <- substring("cm.colors", 1:9, 1:9)
+       text(1:9, rep(0.5, 9), cmtexto, col = cm.colors(9), cex = 3)
+       text(10, 0.5, "cm.colors(9)", adj = 0)
+ }
> cores.ex()
```

1.2.4 Segmentos, flechas, abline

Nessa seção, veremos como construir mais linhas, com os comandos `segments()` e `abline`, além de flechas (arrows).

segments()

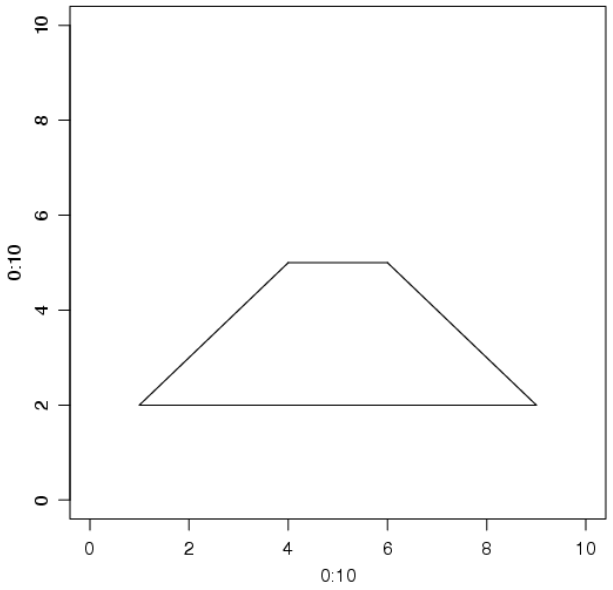
A sintaxe da função `segments()` é a seguinte:

```
> segments(x0, y0, x1, y1, ...)
```

x e *y* representam as coordenadas dos pontos que devem ser unidos. No exemplo, a seguir, pode-se colocar em um gráfico várias linhasbaseadas na função `segments()`.

Figura 1.2: Uso da função segments().

```
> plot(0:10, 0:10, type = "n")
> segments(1, 2, 4, 5)
> segments(4, 5, 6, 5)
> segments(6, 5, 9, 2)
> segments(1, 2, 9, 2)
```



Experimente alterar opções da função `segments()` e gere outras figuras!

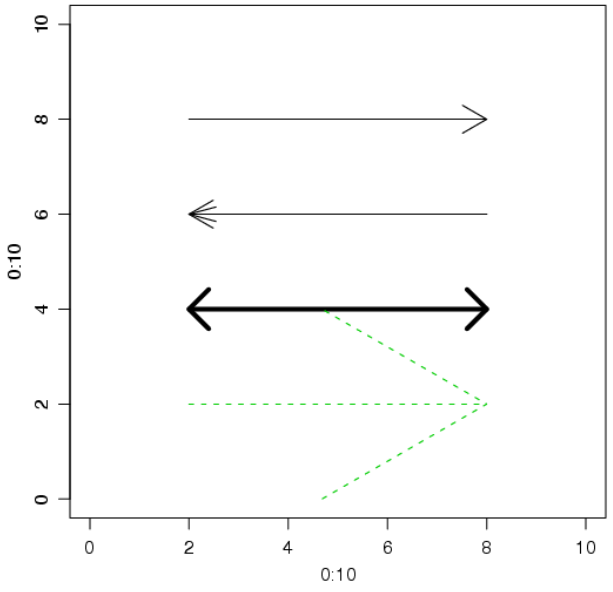
arrows() -flechas

De forma muito semelhante a função `segments()`, a função `arrows()` permite que sejam feitas setas dentro de um gráfico. Entre os argumentos, `code` (0 a 3) indica em que lada a ponta da flecha vai estar, `angle` indica o ângulo da flecha e `length` indica o tamanho da flecha. Outras opções como tipo e cor da linha também são aceitas.

A função `arrows()` pode ser utilizada da seguinte maneira (firura 1.3):

Figura 1.3: Uso da função arrows().

```
> plot(0:10, 0:10, type = "n")
> arrows(2, 8, 8, 8)
> arrows(2, 6, 8, 6, code = 1, angle = 30)
> arrows(2, 6, 8, 6, code = 1, angle = 15)
> arrows(2, 4, 8, 4, code = 3, angle = 45, lwd = 4)
> arrows(2, 2, 8, 2, lty = 2, col = 3, length = 1.7)
```



---

abline()

A função `abline()` é utilizada, em geral, para representar coeficientes de um modelo, mas, pode ser utilizada para gerar uma linha em um gráfico. A idéia é utilizar os coeficientes de um modelo, em geral de uma regressão linear simples para gerar uma reta no gráfico. Mas, pode-se utilizar um simples argumento para gerar uma linha vertical ou horizontal. Veja os exemplos a seguir:

Primeiro, geramos um gráfico de dispersão entre *x* e *y*.

Figura 1.4: Uso da função abline().

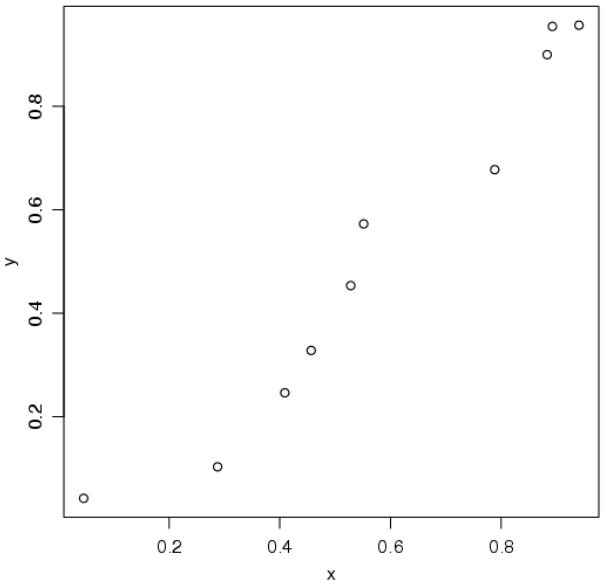
```
> set.seed(123)
> x <- sort(runif(10))
> x
```

```
 [1] 0.0455565 0.2875775 0.4089769 0.4566147 0.5281055 0.5514350 0.7883051
 [8] 0.8830174 0.8924190 0.9404673
```

```
> y <- sort(runif(10))
> y
```

```
 [1] 0.04205953 0.10292468 0.24608773 0.32792072 0.45333416 0.57263340
 [7] 0.67757064 0.89982497 0.95450365 0.95683335
```

```
> plot(x, y)
```



Em seguida, pode-se construir um modelo para representar a relação entre *x* e *y*. Nesse caso, um modelo de regressão linear simples (uma reta):

```
> reg.lm <- lm(y ~ x)
> reg.lm$coef
```

```
(Intercept)           x
 -0.1356696   1.1397177
```

Existem pelo menos três maneiras de inserir uma reta no gráfico. No caso de uma regressão linear simples, pode-se inserir o objeto do tipo `lm`, 'chamar' os coeficientes do modelo ou simplesmente digitando os números correspondentes aos coeficientes.

```
> abline(reg = reg.lm)
> abline(coef = c(reg.lm$coef[1], reg.lm$coef[2]))
> abline(-0.1356696, 1.1397177)
```

Ainda, pode-se inserir, com o comando `abline()` uma reta horizontal e ou uma reta vertical.

```
> abline(h = 0.6)
> abline(v = 0.4)
```

1.2.5 Retângulos, polígonos e círculos

Algumas figuras geométricas podem ser geradas através de algumas funções no **R**. Nesta seção trataremos de algumas destas funções que podem auxiliar na criação de gráficos.

rect()

A primeira delas é a função `rect()` que pode ser utilizada para gerar retângulos. A sintaxe da função é a seguinte:

```
> args(rect)
```

```
function (xleft, ybottom, xright, ytop, density = NULL, angle = 45,
    col = NA, border = NULL, lty = par("lty"), lwd = par("lwd"),
    ...)
NULL
```

onde `xleft`, `ybottom`, `xright`, `ytop` são as coordenadas do retângulo dentro da região do gráfico, `density` e `angle` são argumentos que definem a existência ou não de linhas dentro do retângulo e o ângulo de inclinação das linhas, respectivamente.

Outras opções fazem parte da configuração de cor, tipo de linha etc.

Veja o exemplo da figura 1.2.3 para saber como trabalhar com a função `rect()`. Faça alterações nos argumentos da função para entender como ela funciona.

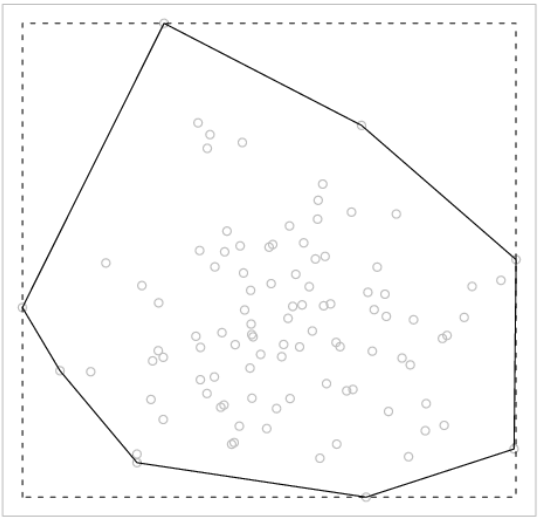
Experimente também, utilizar o exemplo disponível no documento de ajuda sobre a função no **R**.

polygon()

Em muitas situações, podemos estar interessados em construir um polígono, ou seja, unir diferentes pontos em um gráfico. A função `polygon` permite esse tipo de tarefa.

Figura 1.5: Uso da função polygon().

```
> set.seed(123)
> x <- rnorm(100)
> y <- rnorm(100)
> plot(x, y, ann = FALSE, axes = FALSE, col = "grey")
> box(col = "grey")
> rect(min(x), min(y), max(x), max(y), lty = "dashed")
> perif <- chull(x, y)
> polygon(x[perif], y[perif])
```



Duas novas funções aparecem nos comandos da figura 1.5.

A primeira é a função `box()`. Essa função cria uma 'caixa' ao redor da região do gráfico (questões sobre dimensão e margens serão estudadas mais adiante).

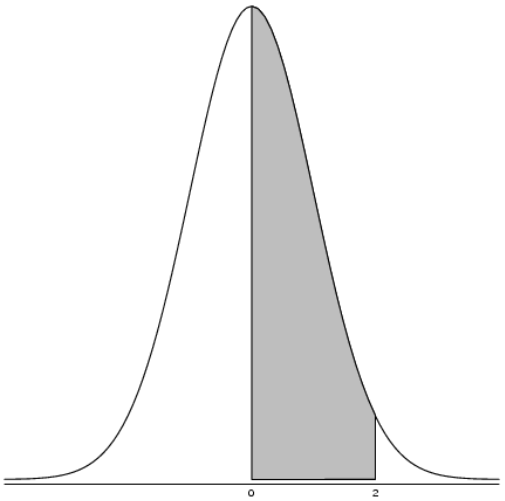
A segunda é a função `chull()`, que retira do conjunto de coordenadas os pontos extremos de forma a contornar toda a extensão periférica dos pontos. Observe que, o objeto `perif` contém os pontos que formam essa região e `x[perif]` e `y[perif]` contém as coordenadas desses pontos.

Nesse caso, a função `polygon()` simplesmente faz a união desses pontos através de uma linha.

Um aplicação pode ser com o uso da distribuição Normal.

Figura 1.6: Uso da função polygon() com a distribuição Normal.

```
> a <- 0
> b <- 2
> a.n <- "0"
> b.n <- "2"
> plot(dnorm, -4, 4, xlab = "", ylab = "", axes = F, bty = "n")
> polygon(x = c(a, seq(a, b, l = 20), b), y = c(0, dnorm(seq(a,
+       b, l = 20)), 0), col = "gray")
> text(c(a, b), c(-0.011, -0.011), c(a.n, b.n), cex = 0.6)
> segments(-4, -0.004, 4, -0.004)
```



Lembre-se que a função `dnorm()` gera os valoes de uma distribuição Normal padrão. Nesse caso, a função `polygon()` utiliza a informação sobre a distribuição para poder acompanhar a linha da curva da distribuição Normal.

Como sugestão, tente implementar outras opções para símbolos e ainda, abra o arquivo de ajuda dessa função para ver mais exemplos.

symbols()

Esta função pode ser utilizada para construir círculos, quadrados, entre outros desenhos. Nesse momento, vamos trabalhar apenas com círculos. Se tiver mais interesse, consulte a documentação de ajuda dessa função.

Observe nos argumentos da função, que existem muitas opções:

```
> args(symbols)
```

```
function (x, y = NULL, circles, squares, rectangles, stars, thermometers,
    boxplots, inches = TRUE, add = FALSE, fg = par("col"), bg = NA,
    xlab = NULL, ylab = NULL, main = NULL, xlim = NULL, ylim = NULL,
    ...)
NULL
```

x e *y* fornecem as coordenadas do centro do símbolo e `circles` define o símbolo do tipo círculo. Observe a diferença entre as figuras 1.7 e areffig:simbolo2. Na figura 1.7 a opção `inches` foi definida como `TRUE`. Isso faz com que o tamanho dos círculos seja relacionado a escala em polegadas. Já, quando a opção é `FALSE`, a escala está relacionada com o eixo *x*. Veja os exemplos nas figuras 1.7 e 1.8:

Figura 1.7: Uso da função symbol() para círculos.

```
> plot(1:5, 1:5, type = "n")
> symbols(2, 2, circles = 0.1, inches = TRUE, bg = 2, add = TRUE)
> symbols(4, 4, circles = 0.3, inches = TRUE, fg = 3, add = TRUE)
```

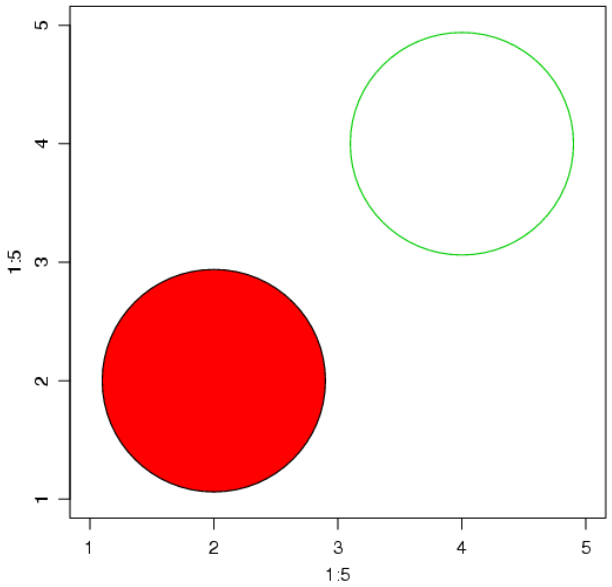
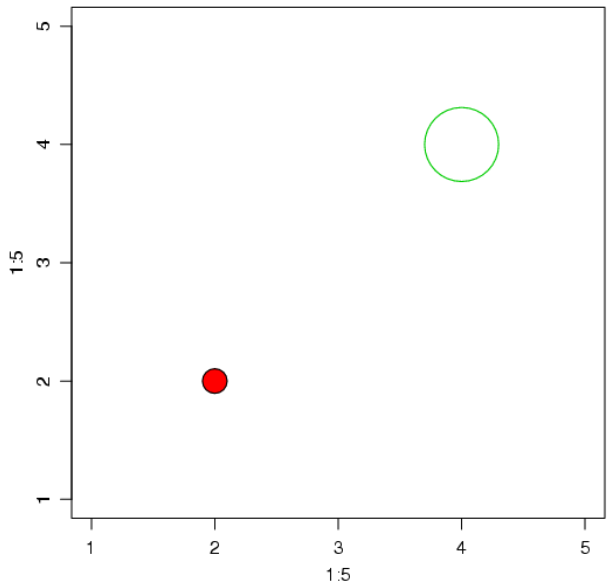


Figura 1.8: Uso da função symbol() para círculos.

```
> plot(1:5, 1:5, type = "n")
> symbols(2, 2, circles = 0.1, inches = FALSE, bg = 2, add = TRUE)
> symbols(4, 4, circles = 0.3, inches = FALSE, fg = 3, add = TRUE)
```

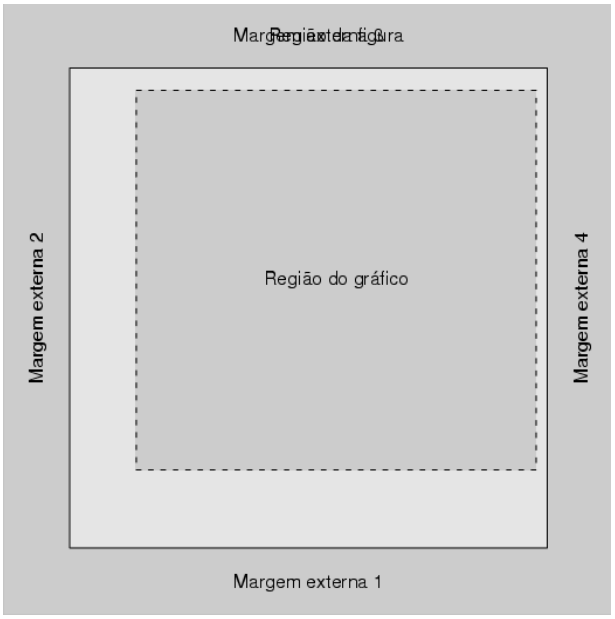


Observe nas linhas de comando que a opção `add` foi definida como `TRUE`. Esta opção indica que os novos símbolos devem ser *adicionados* ao gráfico. Experimente utilizar a opção `FALSE` para ver o resultado.

1.2.6 Regiões do gráfico

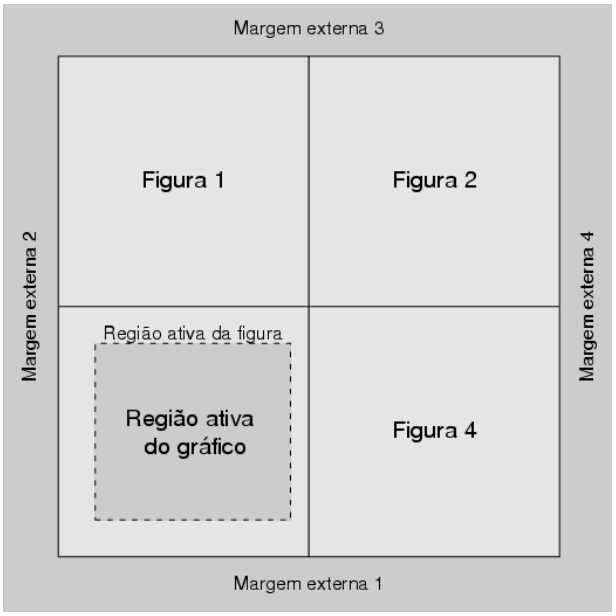
Toda vez que um dispositivo gráfico é aberto no **R**, ele é dividido em *três regiões*: margem externa, a região da figura ativa e a região do gráfico ativo (figura 1.9).

Figura 1.9: . Regiões de um dispositivo gráfico no R.



No caso de existirem múltilos gráficos, as regiões ficam caracterizadas conforme a figura 1.10

Figura 1.10: . Múltiplas regiões de um dispositivo gráfico no R.



O tamanho dessas regiões é controlado pela função `par()`.

Margem externa

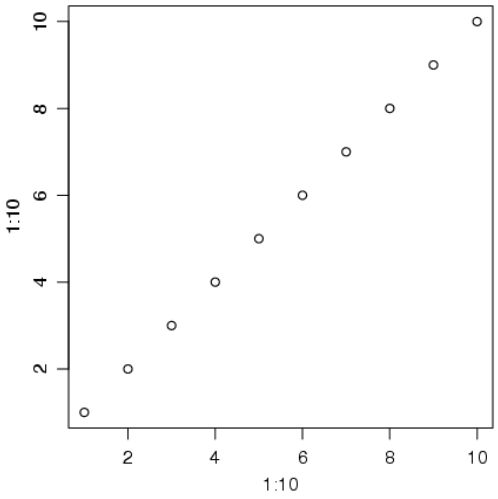
Por 'default' não há margens externas quando da geração de um gráfico. As margens externas podem ser inseridas de três maneiras:

- Usando o comando `oma`: estabelece os espaços considerando a referência de linhas de texto. Um valor 1 significa espaço para uma linha de texto. Como existem quatro margens, os valores são especificados na ordem `bottom, left, top, right`;
- Usando o comando `omi`: as margens tmabém podem ser especificadas em polegadas (inches);
- Usando o comando `omd`: pode-se utilizar também uma proporção da região do dispositivo gráfico. Nesse caso a ordem de entrada é `left, right, bottom, top`.

Experimente alterar as opções gráficas da função `par()`, como no exemplo da figura 1.11.

Figura 1.11: . Exemplos de `oma, omi e omd`.

```
> op <- par()
> par(oma = c(1, 2, 3, 3))
> plot(1:10, 1:10)
> par(op)
```



Nesse caso, você deverá perceber mudanças no dispositivo gráfico. Experimente alterar as opções do parâmetro `oma`.

Região da figura

O **R** determina a região da figura considerando as dimensões das margens externas e do número de figuras no dispositivo gráfico. A região da figura pode ser determinada pelo comando `fig` especificando `left, right, bottom, top`. Nesse caso, cada valor é uma proporção da figura desconsiderando os valores das margens externas. O comando `fin` especifica a largura e altura, `width, height` da região da figura em polegadas e faz a centralização da região da figura no gráfico.

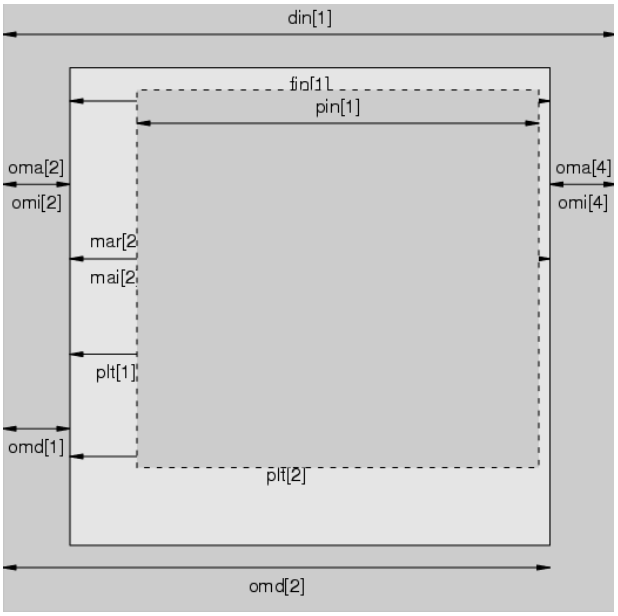
Margens da figura

As margens da região da figura podem ser controladas utilizando o comando `mar`. Esse comando também utiliza a ordem `bottom, left, top, right`. Cada valor representa o número de linhas de texto. Os números default são c(5,4,4,2) +0.1. Da mesma forma como em outros comandos, pode-se determinar o tamanho das margens em termos de polegadas utilizando o comando `mai`.

Região do gráfico

A região do gráfico é determinda pela diferença entre a região da figura e a região das margens da figura. As dimensões podem ser controladas pelo comando `plt`, fornecendo as especificações da forma `left, right, bottom, top`. Pode-se ainda definir as dimensões, `width, height`, em termos de polegadas (inches). Ainda, existe o comando `pty` que controla o quanto do espaço disponível que a regiã oda figura deve ocupar. O valor padrão é 'm', que indica que todo o espaço disponível deve ser ocupado. Pode-se utilizar a opção 's' que indica que a região do gráfico deve ocupar o espaço disponível (região da figura menos a região das margens), mas, esse espaço deve ser um quadrado.

Figura 1.12: . Dimensões de um gráfico.



Fora da região do gráfico - Clipping()

Quando um gráfico é gerado, o **R** irá colocar o resultado (output) na 'região do gráfico'. Qualquer resultado que seja gerado fora das coordenadas da região do gráfico não aparecerá. Mas, existem alguns mecanismos que permitem que se insiram informações fora da região do gráfico (clipping). Por exemplo, pode ser necessário inserir uma legenda em uma posição fora das delimitadas pelas dimensões da região do gráfico.

A opção `xpd` controla a 'permissão' para inserir informações além da região do gráfico. Para `xpd=NA`, é permitido inserir informações em toda a região do dispositivo gráfico. Já, para uma opção `xpd=TRUE`, é permitido utilizar a região da figura e para a região do gráfico, somente, utiliza-se a opção `xpd=FALSE`, que é, por sua vez, o valor padrão.

```
> par(op)
> plot(1:5, 1:5)
> points(0.3, 0.3, pch = "x", col = 2)
> points(0.3, 0.3, pch = "x", col = 2, xpd = TRUE)
```

Para saber quais as dimensões da janela gráfica, basta digitar

```
> par("din", "fin", "pin", "oma", "plt", "mar")
```

```
$din
[1] 6.992249 6.995686

$fin
[1] 6.992249 6.995686

$pin
[1] 6.246351 6.036675

$oma
[1] 0 0 0 0

$plt
```

```
[1] 0.09143565 0.98476072 0.10662252 0.96953642


$mar
[1] 3.5 3.0 1.0 0.5
```

Para alterar, pode-se utilizar, por exemplo:

```
> par(op)
> par(pin = c(5.5, 5.5))
> plot(1:5, 1:5)
> points(0.3, 0.3, pch = "x", col = 2)
> par(op)
```

1.2.7 Arranjo de gráficos

O comando `mfrow` ou `mfcol` permitiam inserir vários gráficos em um mesmo dispositivo. Mas, todos ocupavam as mesmas dimensões.

Nessa sessão, veremos como utilizar a função `layout()`. Basicamente, essa função permite inserir vários gráficos com dimensões diferentes.

Para utilizar essa função é necessário fornecer uma **matriz** especificando o número de posições que serão utilizadas. No exemplo, a seguir, deverão ser gerados quatro gráficos sobre a janela gráfica.

```
> layout(matrix(c(1, 2, 3, 4), byrow = TRUE, ncol = 2))
> plot(1:5, 1:5, ann = FALSE)
```

Observe que nesse caso, todos os espaços terão o mesmo tamanho. Mas, é possível definir tamanhos diferentes para cda gráfico. A opção `heights` ou `widths` podem ser utilizadas para definir esses tamanhos.

```
> layout(matrix(c(1, 2)), heights = c(1.5, 1))
> plot(1:2, 1:2, ann = FALSE, cex.lab = 0.5, cex.axis = 0.5)
> plot(1:2, 1:2, ann = FALSE, cex.lab = 0.5, cex.axis = 0.5)
```

Nesse caso, o gráfico superior ocupa dois terços do espaço disponível (2/(2 + 1)), enquanto que o gráfico inferior ocupa um terço do espaço disponível (1/(2 + 1)).

Ainda, a altura das linhas, nesse exemplo, funciona de forma independente das colunas. Para forçar que as colunas também tenham o mesmo formato das linhas, utiliza-se a opção `respect=TRUE`. Veja o que acontece com o seguinte exemplo:

```
> layout(matrix(c(1, 2)), heights = c(1.5, 1), respect = TRUE)
> plot(1:3, 1:3, ann = FALSE, cex.lab = 0.5)
> plot(1:3, 1:3, ann = FALSE, cex.lab = 0.5)
```

Pode-se, ainda, definir dimensões em centímetros, utilizando a função `lcm`. O exemplo a seguir, mostra os gráficos separados 0,5 cm.

```
> layout(matrix(c(1, 0, 2)), heights = c(2, lcm(0.5), 1), respect = TRUE)
> plot(1:3, 1:3, ann = FALSE, cex.lab = 0.5)
> plot(1:3, 1:3, ann = FALSE, cex.lab = 0.5)
```

Assim como `matrix()`, pode-se utilizar as funções `rbind()` ou `cbind()`.

```
> layout(rbind(c(1, 3), c(0, 0), c(2, 2)), heights = c(2, lcm(0.5),
+     1), respect = TRUE)
> plot(1:3, 1:3, ann = FALSE, cex.lab = 0.5)
```

Experimente gerar outras configurações de gráficos utilizando a função `layout()`.

Ainda é possível utilizar a função `layout.show()` para ver como está dividida a janela gráfica.

```
> layout.show()
```

1.2.8 Texto nas margens - mtext()

A função `text()` permite inserir texto dentro da região do gráfico. Para inserir um texto na região da figura ou nas margens, utiliza-se a função `mtext()`. O argumento `outer` permite definir onde o texto será inserido. O argumento `side` informa em qual margem: 1-abaixo; 2-esquerda; 3-acima; 4-direita.

O texto é inserido em número de linhas, na região da figura 1.13 ou nas margens externas 1.14, podem ser definidas as coordenadas pelo usuário.

Figura 1.13: . Margens da região da figura no R.

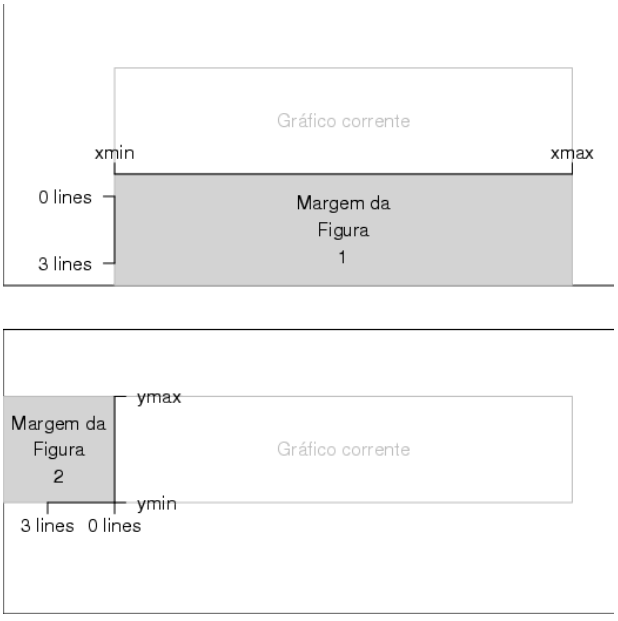
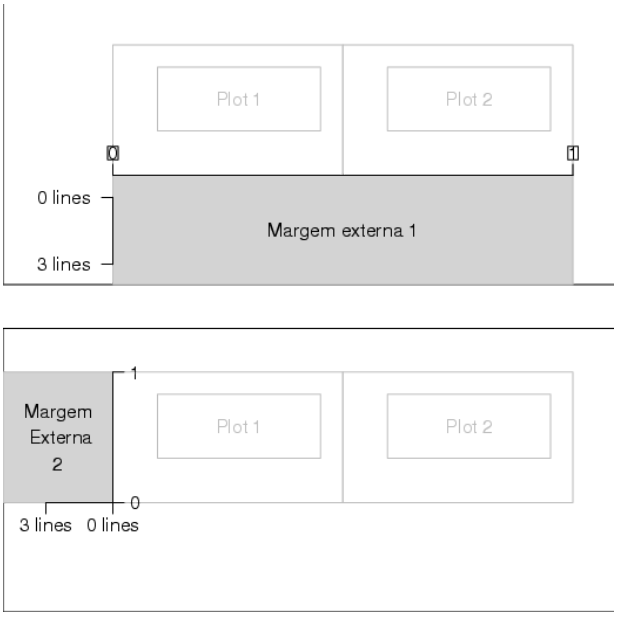


Figura 1.14: . Margens externas da figura no R.

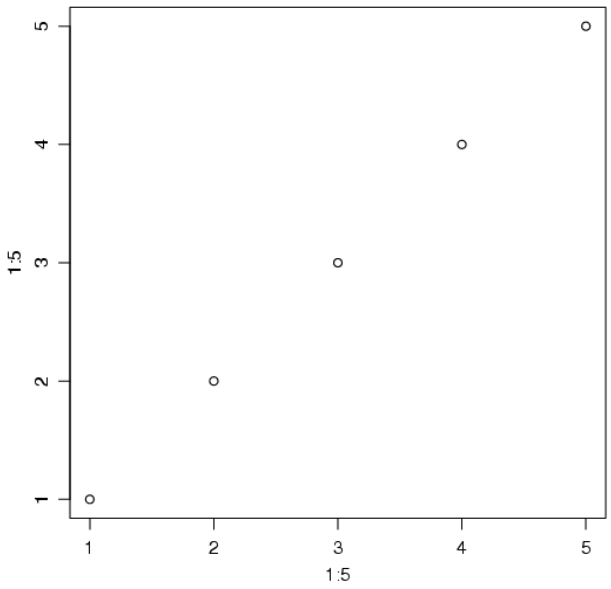


Um exemplo

Veja o exemplo 1.15 para ver como utilizar a função `mtext()`. Nesse caso, quando você tenta inserir um texto com a função `text()` aparece uma BF mensagem de erro, pois a posição requerida fica na região da figura e não do gráfico. Por isso, a função `mtext()` deve ser utilizada nessa situação. Experimente ver os argumentos da função `mtext()` e tentar novas opções.

Figura 1.15: . Inserindo texto nas margens.

```
> op <- par()
> plot(1:5, 1:5)
> text(3, 6, "região da figura")
> mtext("região da figura", side = 3, line = 1)
> par(op)
```



1.2.9 Eixos e o comando mgp

Já vimos alguns comandos sobre eixos quando estudamos funções de alto nível. Nessa seção veremos como trabalhar com o comando `mgp`. Esse comando controla a distância de alguns componentes dos eixos em relação a borda da região do gráfico.

Como valor padrão, o comando `mgp` tem o valor `c(3,1,0)`. Os comandos `xaxs` e `yaxs` controlam o estilo dos eixos de um gráfico. As opções correntemente implementadas são `i` e `r`. Experimente mudar algumas opções no exemplo a seguir.

```
> par(mgp = c(3, 0, 1))
> plot(1:5, 1:5, xaxs = "i")
> box(col = "gray")
```

Ainda, pode-se alterar as marcações de escala dos eixos com o comando `tcl`. `tcl` especifica o tamanho da marca como uma fração do tamanho da linha de texto. Alterando o sinal, muda-se o sentido da marcação.

```
> par(mgp = c(3, 0, 1))
> plot(1:5, 1:5, xaxs = "i", tcl = 0.5)
> box(col = "gray")
```

1.2.10 Controle de janelas gráficas

Em algumas situações, podemos estar interessados na geração de vários gráficos em diferentes janelas gráficas. No **R**, existem algumas funções que auxiliam no controle dessas janelas gráficas.

Para abrir uma nova janela gráfica, pode-se utilizar, no Windows, `x11()` ou `window()`. No Linux, utilize `X11()`. Experimente abrir várias janelas gráficas ao mesmo tempo. Observe que em cada janela, na parte superior, aparece o número da janela e se ela está ativa ou não.

Para movimentar-se entre as janelas, você pode utilizar `dev.set()`. Entre parênteses, insira o número da janela gráfica que você quer trabalhar. Ainda, você pode utilizar `dev.next()` para saber qual a próxima ou `dev.prev()` para saber qual o número da janela anterior (considerando a ordem de abertura das janelas).

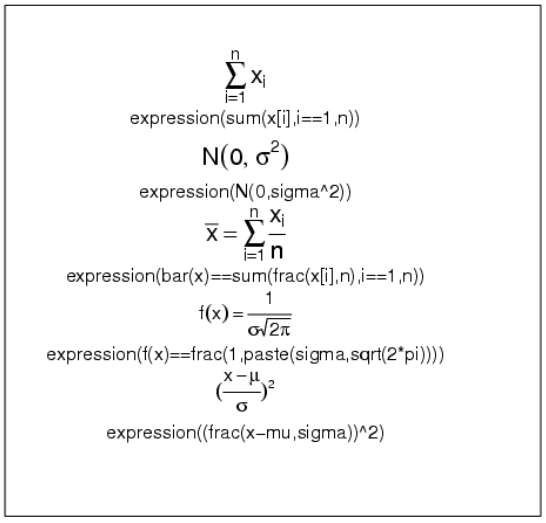
O comando `dev.off()` fecha a janela gráfica.

1.2.11 Fórmulas matemáticas

Em alguns gráficos, pode-se ter a necessidade de se inserir *fórmulas matemáticas*. No **R**, pode-se utilizar tanto o formato de texto quanto expressões que gerem um resultado de interesse. A função `expression()`, pode 'desenhar' símbolos que poderão ser utilizados em gráficos.

Utilize a `demo(plotmath)` ou `help(plotmath)` para saber quais expressões podem ser utilizadas em um gráfico no **R**. Após, tente reproduzir as expressões contidas na figura 1.16:

Figura 1.16: . Inserindo fórmulas no gráfico.



1.2.12 Explorando o plot

Já utilizamos a função `plot()` para construir alguns gráficos simples. Nessa seção, vamos explorar um pouco mais os argumentos dessa função. Acompanhe na seqüência, a seguir, alguns dos principais argumentos da função `plot()`.

Como inserir títulos:

```
> x <- 1:5
> y <- 1:5
> plot(x, y, main = "Explorando plot()", sub = "um sub-título",
+      xlab = "eixo x", ylab = "eixo y")
```

Como mudar o tamanho dos títulos:

```
> plot(x, y, main = "Explorando plot()", sub = "um sub-título",
+      xlab = "eixo x", ylab = "eixo y", cex.axis = 1.5, cex.lab = 2,
+      cex.main = 2.2, cex.sub = 1.5)
```

Como alterar cores:

```
> plot(x, y, main = "Explorando plot()", sub = "um sub-título",
+      xlab = "eixo x", ylab = "eixo y", col = 3, col.axis = 4,
+      col.lab = 5, col.main = 6, col.sub = 7, fg = 8)
```

Como trabalhar com as "marcas de eixos":

```
> plot(x, y, main = "Explorando plot()", lab = c(8, 15, 15),
+      mgp = c(5, 2, 3), tck = 1)
```

Como trabalhar com eixos:

```
> plot(x, y, main = "Explorando plot()", xaxp = c(0, 7, 10))
```

Como controlar a exibição de eixos:

```
> plot(x, y, main = "Explorando plot()", axes = F)
```

```
> plot(x, y, main = "Explorando plot()", xaxt = "n")
```

Como controlar os limites de eixos:

```
> plot(x, y, main = "Explorando plot()", xlim = c(-2, 6), ylim = c(-1,
+     8))
```

Ainda, com relação ao tipo de gráfico, podem ser definidas as seguintes opções:

```
> par(mfrow = c(3, 3))
> plot(x, y, main = "Explorando plot()", sub = "p", type = "p")
> plot(x, y, main = "Explorando plot()", sub = "l", type = "l")
> plot(x, y, main = "Explorando plot()", sub = "o", type = "o")
> plot(x, y, main = "Explorando plot()", sub = "b", type = "b")
> plot(x, y, main = "Explorando plot()", sub = "c", type = "c")
> plot(x, y, main = "Explorando plot()", sub = "s", type = "s")
> plot(x, y, main = "Explorando plot()", sub = "S", type = "S")
> plot(x, y, main = "Explorando plot()", sub = "h", type = "h")
> plot(x, y, main = "Explorando plot()", sub = "n", type = "n")
```

1.2.13 Legendas

Outro tópico bastante usual em gráficos é a inserção de legendas em um gráfico.

Considere o seguinte conjunto de dados para iniciar o trabalho:

```
> dados <- data.frame(x = 1:5, y = 1:5, nome = letters[1:5])
```

Um simples legenda pode ser inserida com o comando `legend()`. Deve-se inserir as coordenadas, o texto da legenda e o tipo de caracter:

```
> plot(x, y)
> legend(x = 4, y = 2, legend = "pontos", pch = 1, cex = 0.5)
```

Pode-se utilizar os nomes dos objetos:

```
> plot(dados$x, dados$y, col = rainbow(5), pch = 1:5)
> legend(x = 2, y = 4.5, dados$nome, pch = 1:5, col = rainbow(5))
```

O comando `fill` pode ser utilizado para preencher os símbolos:

```
> plot(dados$x, dados$y, col = rainbow(5), pch = 15)
> legend(x = 2, y = 4.5, dados$nome, fill = rainbow(5), bg = "lightgray")
```

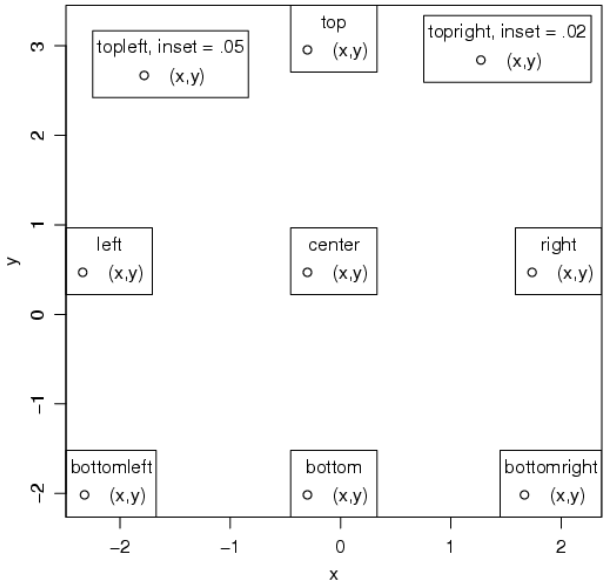
Se você tiver linhas, pode-se representá-las da seguinte forma (uma opção):

```
> plot(x, y, type = "l")
> lines(x, y + 1, col = 2)
> legend(3.5, 2, c("linha 1", "linha 2"), pch = "-", col = 1:2,
+       cex = 1.5)
```

A posição da legenda também pode ser determinada por comandos, ao invés de coordenadas (figura 1.17):

Figura 1.17: . Inserindo legendas no gráfico.

```
> plot(x, y, type = "n")
> legend("bottomright", "(x,y)", pch = 1, title = "bottomright")
> legend("bottom", "(x,y)", pch = 1, title = "bottom")
> legend("bottomleft", "(x,y)", pch = 1, title = "bottomleft")
> legend("left", "(x,y)", pch = 1, title = "left")
> legend("topleft", "(x,y)", pch = 1, title = "topleft, inset = .05",
+       inset = 0.05)
> legend("top", "(x,y)", pch = 1, title = "top")
> legend("topright", "(x,y)", pch = 1, title = "topright, inset = .02",
+       inset = 0.02)
> legend("right", "(x,y)", pch = 1, title = "right")
> legend("center", "(x,y)", pch = 1, title = "center")
```



Veja outros exemplos e opções no `help` da função `legend()`.

1.2.14 Exercícios

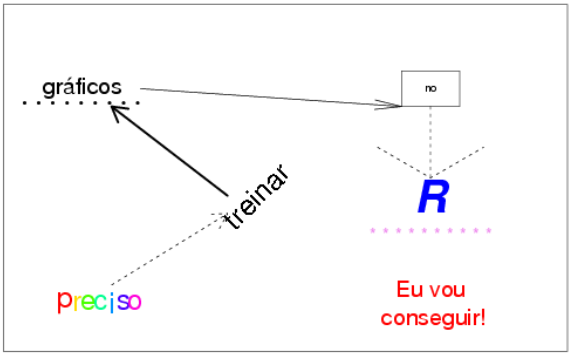
Tente reproduzir as figuras a seguir, utilizando comandos básicos do **R**. Em cada figura são listadas as principais funções utilizadas em cada uma delas.

Obviamente, a reprodução não precisa ser perfeita (considerando, principalmente, dimensões e cores) mas, você deve tentar aproximar ao máximo cada componente do gráfico.

1. As principais funções utilizadas nesse gráfico foram:
`plot()`, `text()`, `arrows()`, `segments()`, `points` e `rect()`.

Figura 1.18: Exercício sobre texto.

```
> plot(1:10, 1:10, type = "n", ann = FALSE, axes = TRUE, bg = "green",
+      xaxt = "n", yaxt = "n")
> texto <- substring("preciso", 1:7, 1:7)
> text(seq(1.7, 2.9, by = 0.2), rep(2, 7), texto, col = rainbow(7),
+      cex = 2.5)
> text(5, 5, "treinar", srt = 45, cex = 2.5)
> segments(1, 7.6, 3, 7.6, lty = 3, lwd = 5)
> text(2, 8, "gráficos", cex = 2)
> text(8, 8, "no")
> rect(7.5, 7.5, 8.5, 8.5)
> text(8, 5, "R", font = 4, cex = 4, col = "blue")
> arrows(2.5, 2.5, 4.5, 4.5, srt = 45, code = 2, lty = 2)
> arrows(4.5, 5, 2.5, 7.5, srt = 90, code = 2, lty = 1, lwd = 3)
> arrows(3, 8, 7.5, 7.5, srt = 0, code = 2, lty = 1, angle = 10,
+      len = 0.4)
> arrows(8, 7.5, 8, 5.5, srt = -90, code = 2, lty = 2, len = 0.9,
+      angle = 60)
> text(8, 2, "Eu vou\ n conseguir!", cex = 2, col = "red")
> points(seq(7, 9, len = 10), rep(4, 10), pch = "*", col = "violet",
+      cex = 1.5)
```



---

2. As principais funções utilizadas nesse gráfico foram:
`plot()`, `text()`, `symbols()`, `polygon()`, `points` e `rect()`.

Figura 1.19: Exercício sobre cores e polígonos.

```
> plot(1:10, 1:10, type = "n", ann = FALSE, axes = FALSE, main = "Bandeira do BRASIL")
> rect(1, 2, 10, 9, col = "darkgreen")
> polygon(c(5.5, 1.2, 5.5, 9.8), c(2.2, 5.5, 8.8, 5.5), col = "yellow")
> symbols(5.5, 5.5, 2, bg = rgb(0, 0, 1), add = TRUE)
> rect(3.54, 5.3, 7.46, 5.8, col = "white")
```

```
> text(5.5, 5.5, "ORDEM E PROGRESSO", col = "darkgreen")
> points(5.92, 5.98, , pch = 8, col = "white")
> points(c(6.06, 5.91, 6.07, 6.13, 6.24, 5.68, 5.29, 5.19,
+     5.03, 4.87, 5.13, 5.18, 5.38), c(5.14, 4.82, 4.67, 4.99,
+     5.1, 4.9, 4.55, 4.86, 5.06, 5.06, 4.69, 4.47, 4.31),
+     pch = 8, col = "white")
> points(c(5.73, 5.88, 5.51, 5.79, 5.96, 6.08, 5.34, 5.5, 5.23,
+     5.12, 5.55, 5.87, 6.04), c(5.01, 4.58, 4.66, 4.84, 5.1,
+     4.71, 4.88, 4.56, 4.55, 4.95, 4.42, 4.49, 4.64), pch = 8,
+     col = "white")
```

---

3. As principais funções utilizadas nesse gráfico foram:
`plot()`, `symbols()`, `points` e `segments()`.

Figura 1.20: Exercício sobre cores e polígonos.

```
> plot(1:10, 1:10, type = "n", ann = FALSE, axes = FALSE)
> symbols(5, 5, 2, add = TRUE, inc = FALSE)
> symbols(c(4, 6), c(6, 6), c(0.3, 0.3), add = TRUE, inc = FALSE)
> points(c(4.041275, 6.086176), c(6.006826, 6.006826), pch = c(44,
+     44), cex = 4, col = "blue")
> points(c(7.022771, 2.997524), c(4.974744, 5.011604), pch = c(41,
+     40), cex = 7)
> segments(4.467296, 3.426621, 5.617553, 3.297611, col = "red")
> segments(4.22, 7.1, 5.04, 9, 1)
> segments(5.04, 9.01, 5.96, 7.02)
> symbols(5.16, 8.97, 0.2, add = TRUE, inc = FALSE, bg = "yellow")
> points(4.989172, 4.827304, pch = 94, cex = 3)
> points(c(3.49, 3.54, 3.84, 4.07), c(4.71, 4.31, 4.21, 4.6),
+     pch = 42, col = 5:8)
> points(c(5.81, 6.05, 6.41, 6.13), c(4.53, 4.01, 4.18, 4.66),
+     pch = 42, col = 5:8)
```

Parte II
Gráficos estatísticos

Capítulo 2
Gráficos estatísticos

Sumário

2.1 Explorando Pacotes

2.1.1 Introdução

No **R** existem, além dos pacotes básicos, vários outros pacotes (packages) que geram muitos tipos diferentes de gráficos. Nesta seção, iremos estudar como gerar gráficos estatísticos utilizando estes diferentes pacotes.

Entre na página do R e explore alguns pacotes, claro, dando enfoque para a parte gráfica.

Para 'baixar' um pacote, no **R** digite

```
> require("nome.do.pacote")
```

Também daremos algum enfoque na forma de apresentação de gráficos, procurando seguir as normas de publicação e, claro, dentro dos aspectos estatísticos.

Segunda avaliação: Para a segunda avaliação, cada aluno deverá escolher um pacote do **R**, obviamente, que trabalhe com funções gráficas. Um das questões será a entrega de um trabalho (um gráfico) baseado em uma ou mais funções gráficas de um pacote. Você deve utilizar somente as funções do pacote e/ou funções básicas do **R**, ou seja, você deve usar a função `require()` apenas uma vez. Procure utilizar dados originais ou então, simule-os no próprio **R** (indique a semente `set.seed()`).

Após a escolha do pacote, envie uma email para o professor com o *subject* **pacote**, informando qual o nome do pacote. A preferência será dada pela ordem de chegada dos emails.

O trabalho deverá ser entregue no dia da segunda avaliação.

Utilize a aula de hoje para pesquisar esses pacotes.

2.2 Apresentação de Gráficos

2.2.1 Introdução

Nesta seção veremos alguns pontos que tratam da apresentação de gráficos, como por exemplo, títulos, formato etc. As informações contidas desta seção são baseadas nas Normas para apresentação de trabalhos científicos publicada pela UFPR. Estas normas, por sua vez, baseiam-se nas normas da ABNT (Associação Brasileira de Normas Técnicas). Mesmo assim, muitas revistas científicas, por exemplo, podem possuir normas ou recomendações diferentes das apresentadas nesse e em outros textos.

Procure, daqui para frente, observar os detalhes de apresentação de gráficos, não só em relação as normas de apresentação, mas também com relação à qualidade de apresentação das informações.

2.2.2 Percepção gráfica

Existem algumas características fundamentais em gráficos que podem gerar diferentes percepções em diferentes indivíduos:

1. Posição ao longo de uma escala comum;
2. Posição ao longo de uma escala não alinhada;
3. Longitude;
4. Angulação;
5. Área;
6. Volume, densidade e saturação de cores;
7. Escala de cores.

Cada indivíduo possui uma percepção diferente em determinados aspectos.

2.2.3 Classificação de gráficos

Dependendo da finalidade, um gráfico pode ser classificado em:

- Gráficos de Propaganda;
- Gráficos Analíticos: informam o que está acontecendo. Em geral, os gráficos estatísticos;
- Gráficos que substituem tabelas: necessariamente deve-se ler os números para poder interpretá-lo;
- Gráficos decorativos: apenas são apresentados porque são bonitos mas não oferecem muita informação.

2.2.4 Princípios básicos de um gráfico

Em um gráfico deve-se procurar observar alguns princípios. Basicamente, um gráfico deve ser:

- Compreensível: permite visualizar as relações entre variáveis;
- Claro: todos os componentes de um gráfico devem ser perfeitamente perceptíveis, principalmente os componentes mais imprtantes devem estar visivelmente destacados;
- Consistente: os elementos gráficos devem fornecer informações coerentes com gráficos anteriores; novos elementos podem requerer uma descrição adicional para fazerem sentido
- Eficiência: os elementos gráficos devem representar exatamente o que os dados querem dizer; pode ser que existam elementos que representem mais de uma informação
- Necessidade: um gráfico deve ser útil para representar dados; deve haver uma necessidade para inserir elementos em um gráfico
- Confiabilidade: os dados devem estar corretamente representados, principalmente no que diz respeito à escala

2.2.5 Elementos de um gráfico

- Título Principal
- Título Secundário o Subtítulo

- Descrição do gráfico
- Região de dados e símbolos
- Eixo horizonta e escala
- Eixo vertical e escala
- Indicadores: flechas, números
- Descrição de sinais e marcas

2.2.6 Recomendações gerais para elaborar um gráfico

1. Os dados devem se sobressair, aparecer. Evite informações desnecessárias.
2. Utilize elementos destacados para mostrar os dados.
3. Não exagere no número de símbolos.
4. Utilize uma linha de referência quando há algum valor importante que deva ser visto em todo o gráfico, por exemplo, uma linha média. Cuide para que isso não interfira na apresentação do gráfico.
5. Não deixar que símbolos ou outros elementos interfiram em dados quantitativos, por exemplo, se amontoando.
6. Evite colocar notas, marcas ou sinais em um lado da região do gráfico. Coloque notas no texto de como explicação.
7. Gráficos sobrepostos devem ser visualmente diferenciáveis.
8. Datos sobrepostos devem ser visualmente diferenciáveis.
9. Um gráfico deve mostras os dados.
10. Um gráfico deve induzir o aobservador a pensar sobre o conteúdo e não no desenho do gráfico, na tecnologia ou outros atributos.
11. Evitar distorcer a mensagem que os dados devem passar.
12. Evitar muitos números em um espaço pequeno.
13. Fazer com que grandes conjuntos de dados tenham coerência.
14. Induzir que os olhos do observador comparem diferentes partes dos dados.
15. Revelar diferentes detalhes dos dados, desde a perspectiva global até detalhes particulares.
16. Ter um objetivo bastante claro: descrição, exploração, tabulação e decoração.
17. Estar bastante integrado as descrições estatísticas e verbais do conjunto de dados.

2.2.7 Erros mais comuns em gráficos

1. Em geral, excessso de decoração é um problema;
2. Ausência de um título, marcas e indicadores;
3. Ausência ou erro na escala de eixos;
4. Excesso de informação;
5. Falta de dados;
6. Má qualidade de impressão.

2.2.8 Objetivo de um gráfico

Inicialmente, um gráfico tem a finalidade de apresentar uma informação de maneira clara, rápida e objetiva, além de resumir, organizar e apresentar dados de qualquer natureza.

Obviamente, existem naturezas diversas, bem como objetivos variados na apresentação de gráficos. Pode-se, por exemplo, citar gráficos com fins promocionais. Muitas vezes, eles não seguem quaisquer normas rígidas de apresentação. Mesmo nesses casos, deve-se ter sempre em mente que a informação não pode ser modificada. Aspectos relacionados com a maneira de evidenciar ou obscurecer informações em gráficos não serão tratados nesse texto.

Elementos básicos:

2.2.9 Númeração

Assim como tabelas, os gráficos devem ser referenciados no texto. Se eles ocorrem em um trabalho, é porque possuem alguma importância, fornecem alguma informação. Portanto, eles precisam ter alguma explicação ou comentário a respeito deles no corpo do trabalho.

Por esse motivo, os gráficos devem ser numerados seqüencialmente ao longo do texto, independente do tipo. Insere-se, no texto, a palavra Gráfico e em seguida o número. Pode-se abreviar a palavra Gráfico. Em alguns textos ou revistas, podeser exigida a palavra figura.

Exemplo:

GRÁFICO 1 - Número médio mensal de acidentes de trânsito com ciclistas em Curitiba no ano de 2005.

No caso de capítulos, deve-se utilizar a numeração correspondente ao capítulo.

Exemplo: No capítulo 1 use GRÁFICO 1.1
GRÁFICO 1.2

No capítulo 2 use

GRÁFICO 2.1
GRÁFICO 2.2

2.2.10 Título

De maneira simples, o título de um gráfico deve ser suficiente para o leitor entender seu conteúdo e sua inserção no contexto do material publicado.

Basicamente, o título traz uma breve descrição do conteúdo do gráfico e uma data de referência, quando for o caso.

2.2.11 Fonte

Em gráficos estatísticos, é recomendável que a fonte (responsável) dos dados seja mencionada. Normalmente a fonte é apresentada abaixo do gráfico, com uma tamanho de letra menor do que a do título, iniciada com a expressão **Fonte**.

2.2.12 Escala

Em um gráfico, as informações que possuem uma escala devem ser apresentadas com clareza. Os números da escala devem estar na posição horizontal e na região externa dos eixos. A unidade de medida deve aparecer no final de cada linha do gráfico.

Caso não seja possível apresentar a escala por completo, pode-se fazer um corte no eixo correspondente para informar que a escala não está completa.

2.2.13 Diferença de linhas

Cores: dependem dos recursos e da finalidade. Em geral, para publicação em revistas científicas, em se tratando de gráficos estatísticos, são utilizados apenas gráficos em preto e branco. Uma alternativa é utilizar tonalidades de preto, passando pelo cinza, até o branco.

Além de cores, podem-se utilizar hachuras ou linhas pontilhadas

legenda: Tanto cores, como linhas pontilhadas ou hachuras devem ser referenciadas em uma legenda.

2.2.14 Apresentação

Em geral, pode-se dizer que quando um gráfico não consegue expressar a devida informação sem a necessidade de números, é porque uma tabela deve ser uma opção mais adequada.

2.3 Gráficos univariados

Nesta seção, estudaremos alguns gráficos univariados. O principal objetivo destes gráficos é fazer uma análise descritiva de dados. Entende-se por análise descritiva a obtenção de informações que auxiliem na interpretação e na avaliação da qualidade dos dados em determinadas situações. Basicamente a freqüência e a forma de distribuição dos dados são os aspectos que podem ser considerados mais relevantes em uma análise gráfica univariada. Obviamente, em casos particulares, pode-se ter interesses diferentes dos aqui apresentados.

No **R**, iremos utilizar alguns conjuntos de dados provenientes de pacotes e em algumas situações, utilizaremos funções específicas de alguns pacotes.

2.3.1 Ramo-e-folhas

O objetivo de um gráfico ramo-e-folhas é resumir os dados de tal forma que se possa ter uma noção da forma da distribuição dos dados (simétrica, assimétrica), a freqüência de observações, vazios entre os dados e possíveis dados discrepantes (*outliers*).

As observações devem ser divididas em duas partes. A primeira, chamada de ramo, é colocada à direita de uma linha vertical, e a segunda, chamada de folhas, é colocada á esquerda. Por exemplo, se a variável possui valores do tipo 2, 34; 2, 98, o número 2 será o ramo e 34 e 98 serão as folhas.

A maneira como são distribuídos os ramos e folhas podem variar em função da quantidade de dados. Em alguns casos, pode-se ter uma visualização melhor com uma separação dos dados por ramo.

Esse gráfico pode não ser muito útil quando existem muitas observações.

Exemplo

Vamos considerar um conjuto de dados simulados no **R**:

```
> set.seed(2209)
> dados <- sample(1:100, 50)
```

Para construir o gráfico, utiliza-se a função `stem()`.

```
> stem(dados)
```

```
  The decimal point is 1 digit(s) to the right of the |

   0 | 267
   1 | 01257
   2 | 03579
   3 | 0123
   4 | 345678
   5 | 1369
   6 | 346789
   7 | 23468
   8 | 3456789
```

```
  9 | 2679
 10 | 0
```

Veja os argumentos e o help da da função `stem()`:

```
> args(stem)
```

```
function (x, scale = 1, width = 80, atom = 1e-08)
NULL
```

Experimente alterar alguns dos parâmetros da função. Tente, também, aumentar o tamanho e o tipo de dado gerado. Por exemplo, gere dados de uma distribuição Normal, utilizando `rnorm()`.

2.3.2 Box-Plot

O box-plot é um gráfico que mostra a posição central, dispersão e simetria dos dados de uma amostra , comprimento de caudas e dados discrepantes. É utilizado para resumir as informações de um conjunto de dados.

Esse gráfico é baseado no “resumo dos 5 números"(`fivenum()`).

Considere o seguinte conjunto de dados: 17, 22, 23, 27, 29, 32, 38, 42, 46, 52, 60, 92

Para construir o box-plot dessas observações, devemos seguir os seguintes passos:

- Calcular a mediana , o primeiro e terceiro quartis, e a amplitude interquartílica.

```
> box <- c(17, 22, 23, 27, 29, 32, 38, 42, 46, 52, 60, 92)
> quantile(box, type = 2)
```

```
  0%  25%  50%  75% 100%
  17   25   35   49   92
```

```
> AIQ <- quantile(box, 0.75, type = 2) - quantile(box, 0.25,
+       type = 2)
```

- Calcular 1,5 × AIQ =1,5 × 24=36 e,

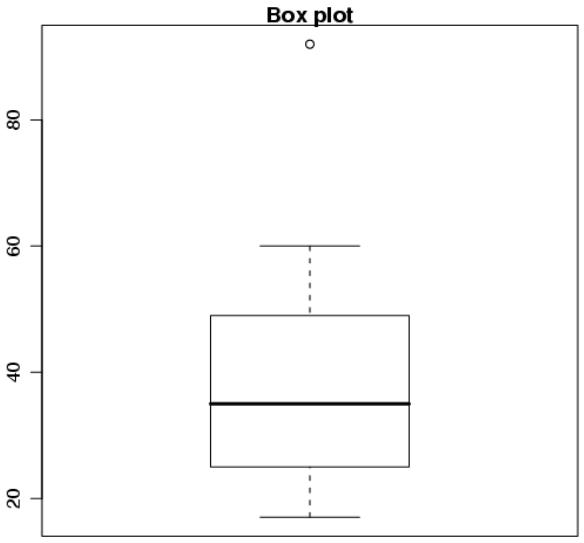
  Limite inferior = Q1-36=25-36=-11
  Limite superior = Q3+36=49+36=85

- Encontrar o menor valor e o maior valor dentro dos limites inferiores e superiores, respectivamente: menor=17 e maior=60;
- Construir o gráfico (figura 2.1)

  No **R**, basta utilizar a função `boxplot()` informando a variável que será utilizada.

Figura 2.1: Gráfico Box-Plot.

```
> boxplot(box, main = "Box plot")
```



Além do gráfico, a função `boxplot()` retorna outras estatística que podem ser de interesse.

```
> boxplot.stats(box)
```

```
$stats
[1] 17 25 35 49 60

$n
[1] 12

$conf
[1] 24.05344 45.94656

$out
[1] 92
```

Veja outras opções da função `boxplot()`.

Se o objeto que se deseja utilizar a função `boxplot()` é apenas uma variável numérica, será construído apenas um box-plot. No caso de existir mais de uma variável, por exemplo, um `data.frame`, a função `boxplot()` irá construir um box-plot para cada coluna.

Se existir uma variável do tipo `factor`, pode-se utilizar uma fórmula para construir um box-plot para cada categoria.

Por exemplo, considere os dados sobre gatos (cats) do pacote `MASS`. Existem três colunas: Sex (sexo), Bwt (peso do corpo), Hwt (peso do coração).

```
> library(MASS)
> data(cats)
```

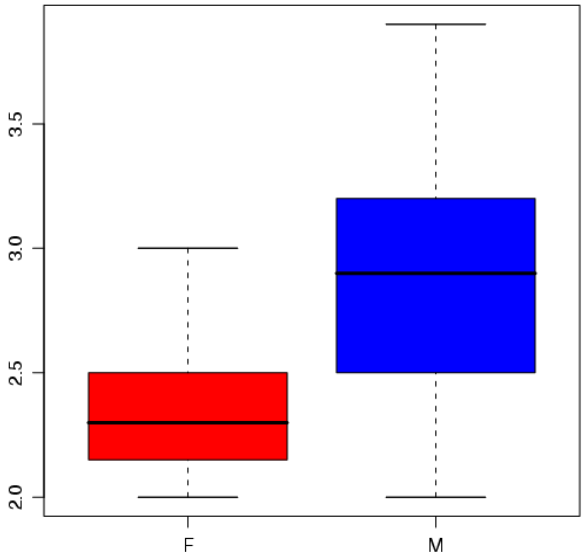
Para construir um box-plot para cada um dos sexos, utiliza-se o seguinte:

```
> boxplot(Bwt ~ Sex, data = cats)
```

Experimente alterar algumas opções da função `boxplot()`. Veja, por exemplo, como mudar cores.

Figura 2.2: .Gráfico Box-Plot com cores.

```
> boxplot(Bwt ~ Sex, col = c("red", "blue"), data = cats)
```



A opção `notch=TRUE` insere um corte (entalhe) em cada box-plot. Quando esses cortes se sobrepõem, isso indica que não deve haver diferenças entre as medianas dos grupos.

2.3.3 Gráficos de setores ou circulares

O gráfico de setores ou circular, também conhecido como 'pizza' pode ser utilizado para representar a freqüência de observações de diferentes categorias. O tamanho de cada setor é proporcional ao número de observações em cada categoria. O tamanho pode ser definido em números percentuais ou absolutos.

Em geral, um gráfico de setores não é um bom modo de representar dados porque o olho humano tem dificuldades para comparar áreas relativas. em comparação com medidas lineares.

No **R**, a função utilizada para esse tipo de gráfico é `pie()`.

Vamos utilizar os dados do arquivo *cats* do pacote MASS:

```
> require(MASS)
> data(cats)
> attach(cats)
```

Experimente obter algumas informações,

```
> summary(cats)
```

```
Sex         Bwt             Hwt
F:47   Min.   :2.000   Min.   : 6.30
```

```
 M:97   1st Qu.:2.300   1st Qu.: 8.95
        Median :2.700   Median :10.10
        Mean   :2.724   Mean   :10.63
        3rd Qu.:3.025   3rd Qu.:12.12
        Max.   :3.900   Max.   :20.50
```

A função `tapply()` (*t* de table) pode ser utilizada para obter estatísticas por grupos:

```
> tapply(Bwt, Sex, mean)
```

```
       F        M
2.359574 2.900000
```

Nesse exemplo, utiliza-se a variável resposta *Bwt*, agrupa-se por *Sex* e estima-se a média de cada grupo.

ou fazendo tabelas,

```
> table(Sex)
```

```
Sex
 F  M
47 97
```

Suponha uma nova variável, onde, se o peso do coração for maior ou igual a 9,5, o gato está apto e, caso contrário, estará inapto.

```
> aptidao <- ifelse(Hwt >= 9.5, "apto", "inapto")
> table(Sex, aptidao)
```

```
    aptidao
Sex apto inapto
  F   23     24
  M   72     25
```

Obtenção de uma soma marginal:

```
> sex.t <- table(Sex, aptidao)
> margin.table(sex.t, 1)
```

```
Sex
 F  M
47 97
```

```
> margin.table(sex.t, 2)
```

```
aptidao
  apto inapto
    95     49
```

Para obter freqüências relativas, utilize

```
> prop.table(sex.t, 1)
```

```
   aptidao
Sex      apto    inapto
  F 0.4893617 0.5106383
  M 0.7422680 0.2577320
```

```
> prop.table(sex.t, 2)
```

```
   aptidao
Sex      apto    inapto
  F 0.2421053 0.4897959
  M 0.7578947 0.5102041
```

Se for de interesse obter valores em percentual, multiplique por 100.

Para obter as proporções em função do total geral, basta utilizar

```
> sex.t/ sum(sex.t)
```

```
   aptidao
Sex      apto    inapto
  F 0.1597222 0.1666667
  M 0.5000000 0.1736111
```

```
> round(sex.t/ sum(sex.t), 2)
```

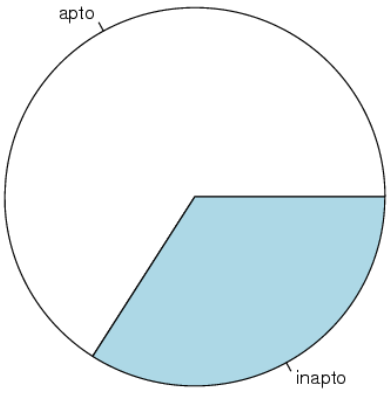
```
   aptidao
Sex apto inapto
  F 0.16   0.17
  M 0.50   0.17
```

(com duas casas decimais)

O gráfico de setores pode ser obtido da seguinte maneira:

Figura 2.3: Gráfico de Setores.

```
> pie(margin.table(sex.t, 2))
```



Outros exemplos

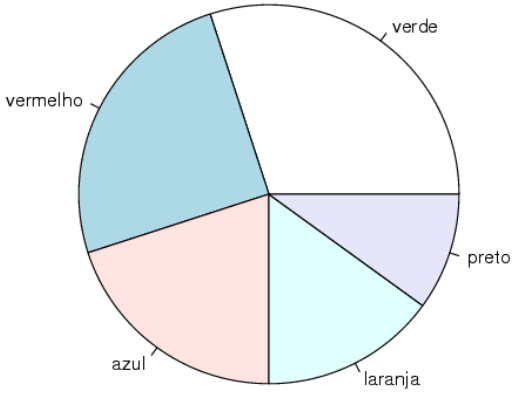
Alguns outros exemplos podem ser vistos a seguir. Por exemplo:

```
> cores <- c(0.3, 0.25, 0.2, 0.15, 0.1)
> names(cores) <- c("verde", "vermelho", "azul", "laranja",
+       "preto")
```

Para esses dados, um gráfico de setores pode ser construído da seguinte forma:

Figura 2.4: Gráfico de Setores.

```
> pie(cores)
```

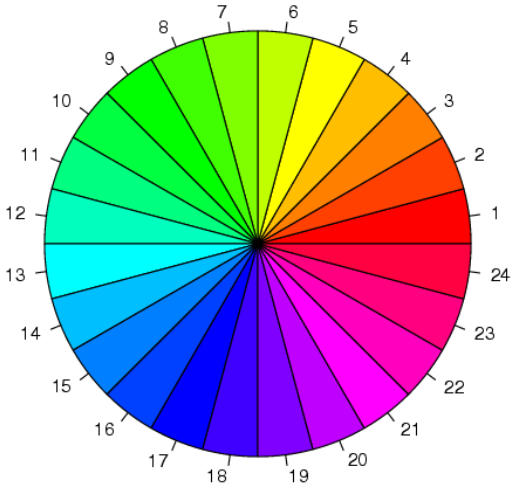


Pode-se trabalhar com cores da seguinte forma:

Experimente trabalhar com cores:

Figura 2.5: Gráfico de Setores.

```
> pie(rep(1, 24), col = rainbow(24), radius = 0.9)
```



Existem outros parâmetros gráficos que podem ser utilizados. Veja os argumentos da função `pie()` e outros exemplos utilizando `?pie`.

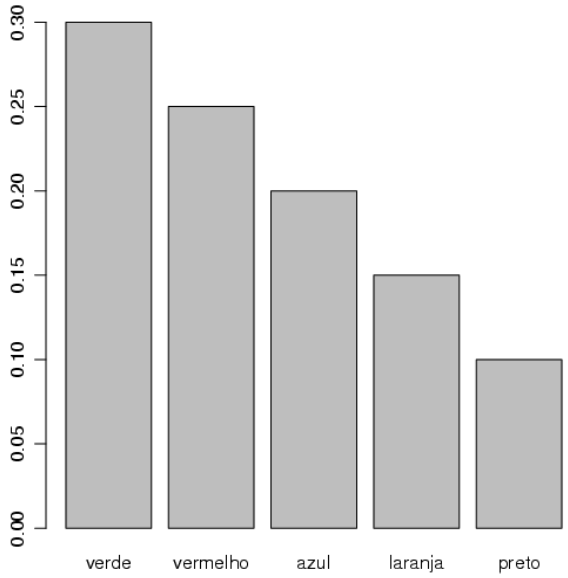
2.3.4 Gráfico de barras

Assim como o gráfico de setores, o gráfico de barras é utilizado para representar a freqüência absoluta ou percentual de diferentes categorias.

Basicamente, são construídas barras proporcionais as freqüências. No R, utiliza-se a função `barplot()`. Como exemplo veja a figura 2.6,

Figura 2.6: Gráfico de Barras.

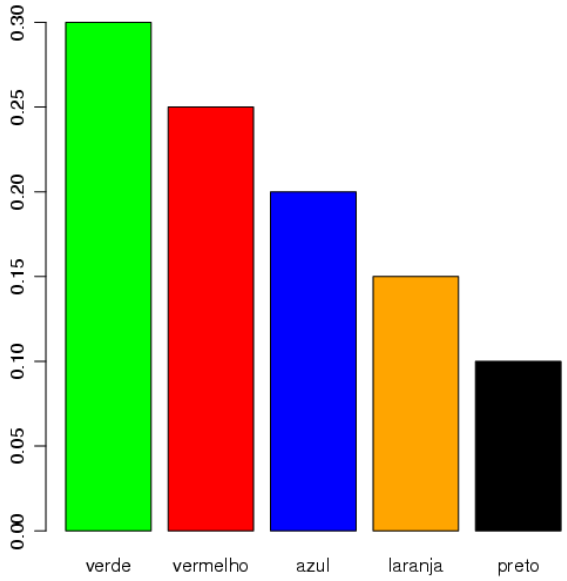
```
> barplot(cores)
```



Também, pode-se trabalhar com cores (figura 2.7):

Figura 2.7: Gráfico de Barras.

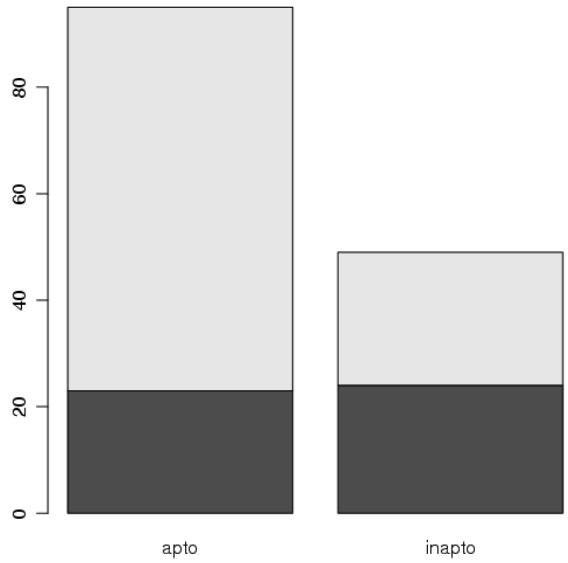
```
> barplot(cores, col = c("green", "red", "blue", "orange",
+       "black"))
```



Quando trata-se de dados dispostos em uma tabela, a função `barplot()` 'enxerga' os dados da seguinte maneira (figura 2.8):

Figura 2.8: Gráfico de Barras a partir de uma tabela.

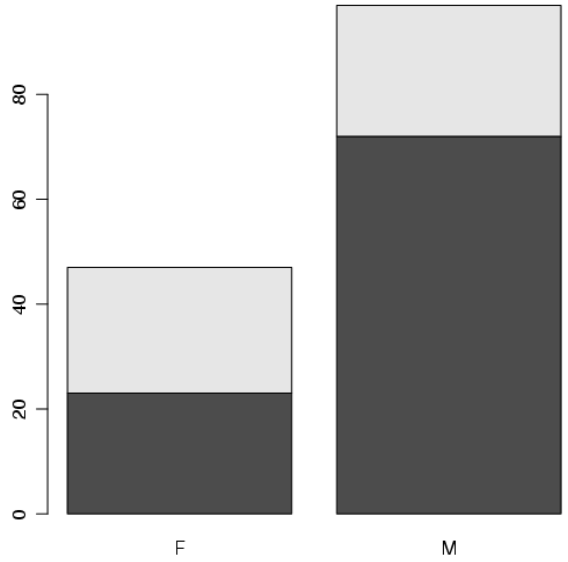
```
> barplot(sex.t)
```



Para inverter as posições, basta utilizar a função `t()`, que transpõe a matriz de dados (figura 2.9):

Figura 2.9: Gráfico de Barras a partir de uma tabela - inversão .

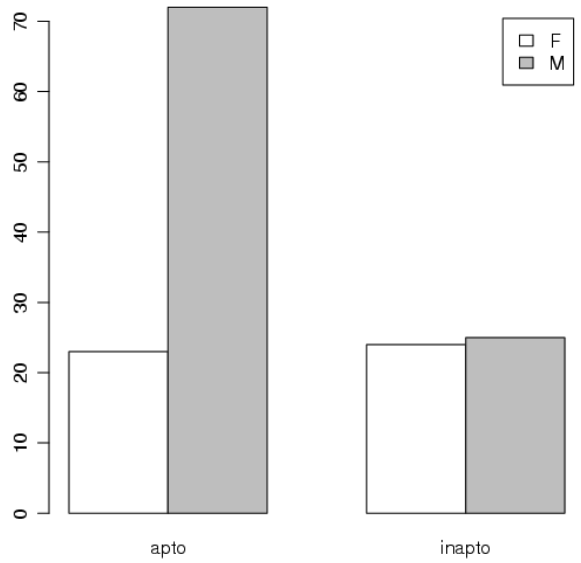
```
> barplot(t(sex.t))
```



Ainda, pode-se colocar as barras lado a lado e inserir uma legenda (figura 2.10) :

Figura 2.10: Gráfico de Barras a partir de uma tabela - legenda.

```
> barplot(sex.t, beside = TRUE, legend.text = rownames(sex.t),
+       col = c("white", "gray"))
```



Se os dados estão no formato de uma matriz, as barras são agrupadas para representar os dados de maneira adequada. Por exemplo, considere o seguite conjunto de dados:

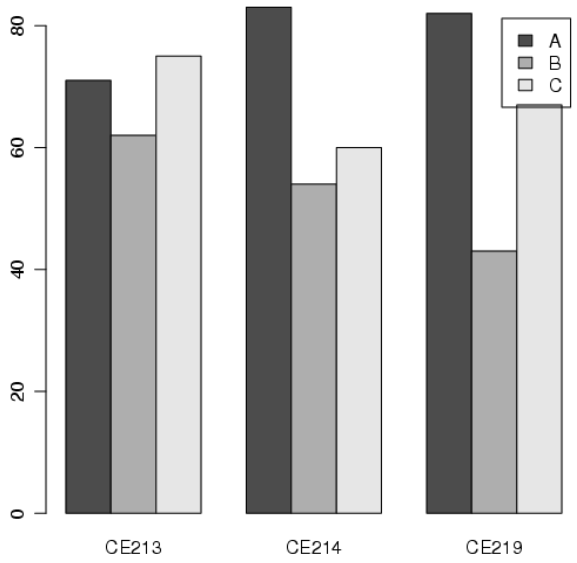
```
> disciplina <- c("CE213", "CE214", "CE219")
> alunos <- c("A", "B", "C")
> notas <- matrix(c(71, 62, 75, 83, 54, 60, 82, 43, 67), nc = 3,
+       dimnames = list(alunos, disciplina))
> notas
```

```
  CE213 CE214 CE219
A    71    83    82
B    62    54    43
C    75    60    67
```

Para esses dados, a função `barplot()` retorna o seguinte gráfico:

Figura 2.11: Gráfico de Barras a partir de uma 'matriz'.

```
> barplot(notas, beside = TRUE, legend = rownames(notas))
```



Veja o arquivo de ajuda da função `barplot()` para ver mais exemplos e outras funcionalidades.

```
> detach(cats)
```

2.3.5 Histograma

Um histograma é utilizado para representar a distribuição de uma variável aleatória contínua. Basicamente, as freqüências observadas são representadas por classes de ocorrência. As freqüências absolutas podem ser substituídas por freqüências relativas ou proporcionais.

Um dos principais aspectos na construção de um histograma é a definição do número de classes. Existem várias proposições sobre esse aspecto. No R algumas dessas proposições estão implementadas. Inicialmente, vamos estudar os comandos básicos do histograma e a seguir, aplicar os diferentes métodos para definição do número de classes.

No R, a função `hist()` constrói o histograma. Para ver os argumentos dessa função, utilize `?hist` ou `args(hist)` para ver quais argumentos são possíveis para a função `hist()`.

```
function (x, ...)
NULL
```

Como exemplo, considere o seguinte conjunto de dados:

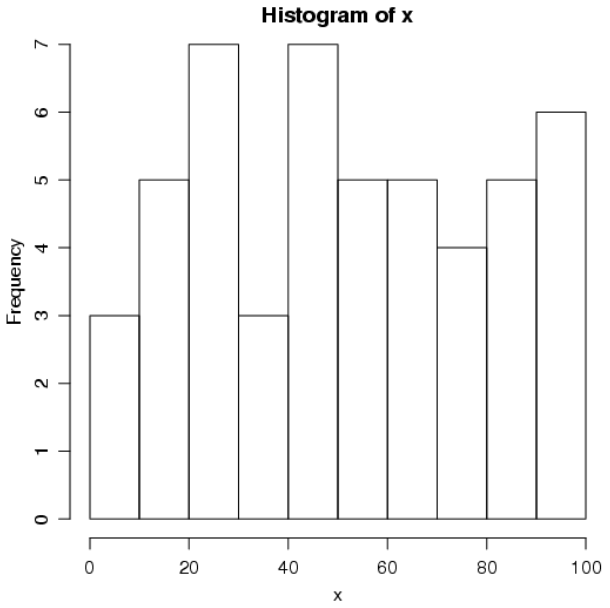
```
> set.seed(123)
> x <- sample(1:100, 50, rep = TRUE)
> x
```

```
 [1]  29  79  41  89  95   5  53  90  56  46  96  46  68  58  11  90  25
[18]   5  33  96  89  70  65 100  66  71  55  60  29  15  97  91  70  80
[35]   3  48  76  22  32  24  15  42  42  37  16  14  24  47  27  86
```

Um simples histograma pode ser gerado da seguinte maneira:

Figura 2.12: Histograma simples.

```
> hist(x)
```

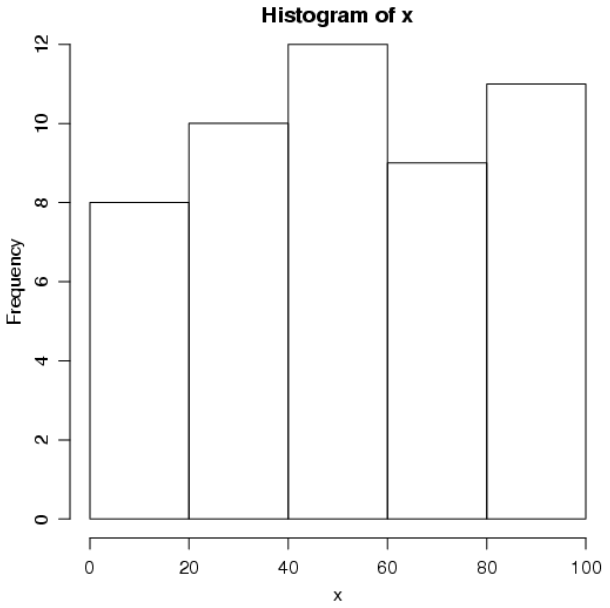


O número de classes pode ser alterado com o comando `breaks`. Nesse caso são aceitos um simples número indicando quantas classes devem ser formadas. Internamente, há uma função de tolerância que pode não responder ao valor informado. Dependendo da configuração dos dados,o R, automaticamente, faz um ajuste diferente do solicitado.

Veja alguns exemplos

Figura 2.13: Histograma simples.

```
> hist(x, 5)
```



As classes podem ser explicitamente definidas pelo usuário de várias maneiras:

```
> hist(x, br = c(0, 15, 30, 45, 60, 75, 90, max(x)))
> hist(x, br = seq(0, 100, len = 8))
```

As classes também podem ter tamanhos diferentes:

```
> hist(x, br = c(0, 10, 35, 45, 60, 70, 90, max(x)))
```

O argumento `right=TRUE`, indica que o intervalo de classe é definido da forma (a,b], ou seja, aberto à esquerda e fechado à direita.

O argumento `include.lowest=TRUE` indica como o R trata das observações que estão nos extremos dos dados. Esse argumento é dependente da opção `breaks` e `rigth`.

O número de classes é importante na construção do histograma, pois isso afeta a forma do gráfico e por sua vez, a maneira como os dados ou informações serão interpretadas.

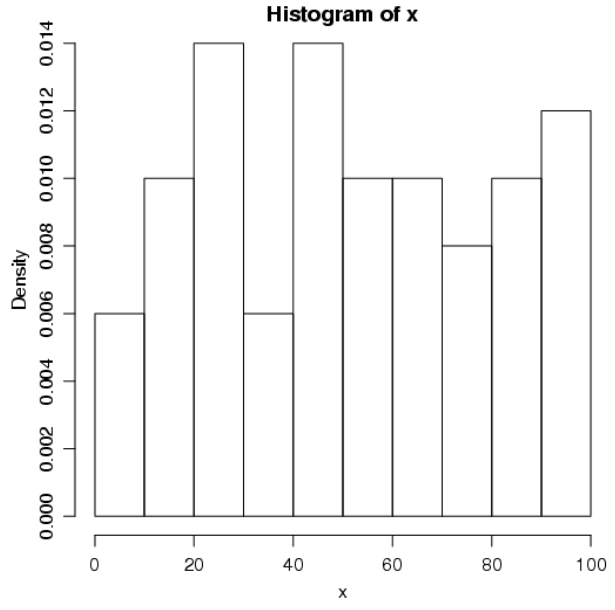
Na função `hist()`o método de Sturges é o padrão. Esse método define o número de classes, k, através da seguinte expressão $k = 1 + \log_2(n)$. Outra opção é utilizar o método ou regra de Scott, onde o número de classes é dado por: $k = (2n)^{1/3}$.

Histograma com probabilidades

O argumento `freq=FALSE` define que ao invés de serem inseridas as freqüências no histograma deve-se inserir as proporções ou probabilidades. Dessa maneira, o R insere as barras do histograma de modo que elas tenha uma área de valor 1.

Figura 2.14: Histograma com probabilidades.

```
> hist(x, freq = FALSE)
```



Observe nesse gráfico, que no eixo y aparecem as probabilidades e não as freqüências.

Outros elementos

Quando a função `hist()` é utilizada, algumas informações são geradas. Por exemplo:

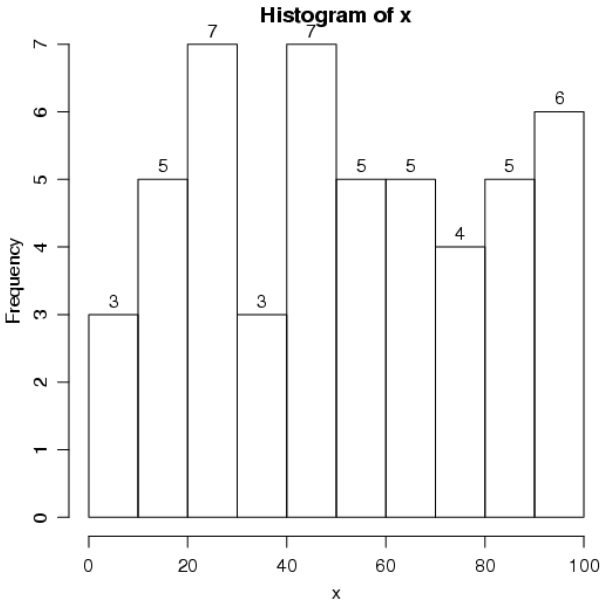
```
> hist.meu <- hist(x)
> hist.meu
```

```
$breaks
 [1]   0  10  20  30  40  50  60  70  80  90 100

$counts
 [1] 3 5 7 3 7 5 5 4 5 6

$intensities
 [1] 0.005999999 0.010000000 0.014000000 0.006000000 0.014000000
 [6] 0.010000000 0.010000000 0.008000000 0.010000000 0.012000000

$density
 [1] 0.005999999 0.010000000 0.014000000 0.006000000 0.014000000
 [6] 0.010000000 0.010000000 0.008000000 0.010000000 0.012000000

$mids
 [1]  5 15 25 35 45 55 65 75 85 95

$xname
[1] "x"

$equidist
[1] TRUE

attr(,"class")
[1] "histogram"
```

Essas informações podem ser utilizadas, por exemplo, na identificação ou na inserção de freqüências sobre as barras (figura 2.15).

Figura 2.15: Histograma com identificação.

```
> hist.meu <- hist(x)
> text(hist.meu$mid, hist.meu$counts + 0.2, hist.meu$counts)
```



2.3.6 Funções rug e jitter

Em um histograma, pode-se ter interesse em visualisar a densidade de dados por cada classe. A função `rug()` insere no gráfico pequenas linhas, indicando a posição dos pontos em daca barra do histograma. Pode acontecer que dados iguais ocorram em um classe. Nesse caso, pode-se utilizar a função `jitter()` que adiciona uma certa quantidade a cada observação, permitindo que dados iguais possam ser diferenciados.

Em alguns casos, principalmente se existirem muitos dados, essas funções podem não ser muito úteis.

```
> rug(jitter(x, 0.15))
```

Experimente, também, trabalhar com cores, linhas (density), legendas etc.

Sugestão de estudo

Dentro do pacote `MASS` há uma função que faz histogramas e também dentro do pacote `lattice`. Veja quais são os argumentos de cada função, compare com a função `hist` e tente construir alguns gráficos.

```
> require(MASS)
```

```
[1] TRUE
```

```
> truehist(x)
> require(lattice)
```

```
[1] TRUE
```

```
> histogram(x)
```

2.3.7 Gráficos de dispersão ou Scatter plots

Gráficos que mostram a dispersão de dados são úteis para identificar muitas características de dados. Além da dispersão, outliers, tendências entre outros aspectos, podem ser explorados com gráficos de dispersão ou Scatter plots.

No R, além da função `plot()`, outras funções como `stripchart()` e `dotchart()` podem ser utilizados para representar pontos.

Dependendo do tipo de objeto que está sendo utilizado, um simples vetor ou uma matriz, cada função gerará gráficos de acordo com a antureza dos dados.

stripchart

A função `stripchart()` produz um gráfico unidimensional, ou seja, considerando apenas uma escala. Um gráfico de pontos pode ser uma alternativa a um boxplot, principalmente quando o número de pontos é pequeno.

Inicialmente, explore os argumentos da função `stripchart()`:

```
> args(stripchart)
```

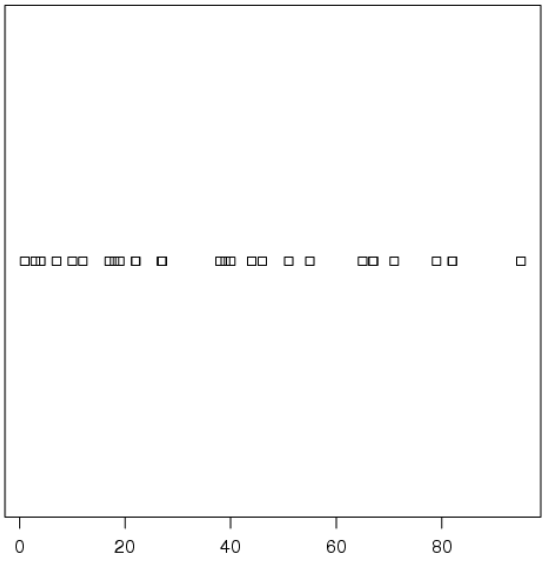
```
function (x, method = "overplot", jitter = 0.1, offset = 1/3,
    vertical = FALSE, group.names, add = FALSE, at = NULL, xlim = NULL,
    ylim = NULL, main = "", ylab = "", xlab = "", log = "", pch = 0,
    col = par("fg"), cex = par("cex"))
NULL
```

Como exemplo, vamos considerar o seguinte conjunto de dados:

```
> set.seed(12)
> dados <- sample(100, 30, rep = TRUE)
```

Figura 2.16: Gráfico de dispersão - Stripchart.

```
> stripchart(dados, method = "overplot")
```



Outras opções podem ser inseridas. Em especial, o método de posicionamento dos pontos é relevante, pois, por exemplo, observações com o mesmo valor podem passar despercebidas em um gráfico desse tipo. Assim, existem três

opções nesse aspecto: *overplot*, que insere os pontos sobrepostos, *jitter*, insere os pontos com um "ruído"e *stack*, que insere os pontos um sobre o outro, como se fosse uma pilha de tijolos.

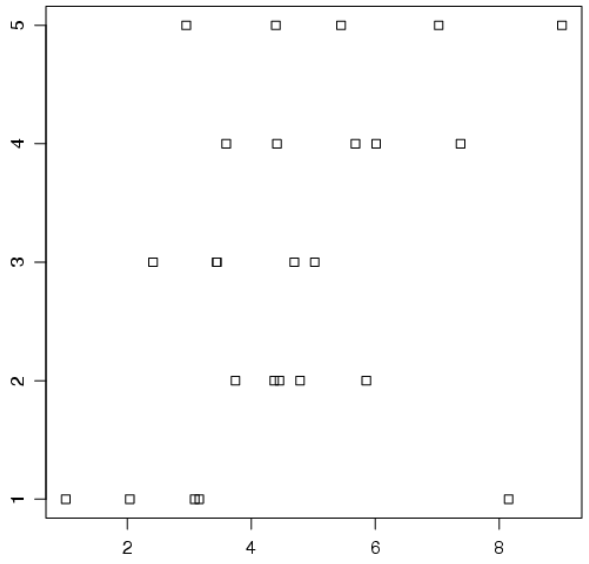
```
> stripchart(dados, method = "jitter", jitter = 0.1)
> stripchart(dados, method = "stack")
```

Quando os dados estão agrupados por grupos ou categorias, a função `stripchart()` constrói um gráfico, separando as categorias, da seguinte maneira

---

Figura 2.17: Stripchart com grupos.

```
> set.seed(12)
> x <- rnorm(25, 5, 2)
> grupo <- gl(5, 5)
> stripchart(x ~ grupo)
```



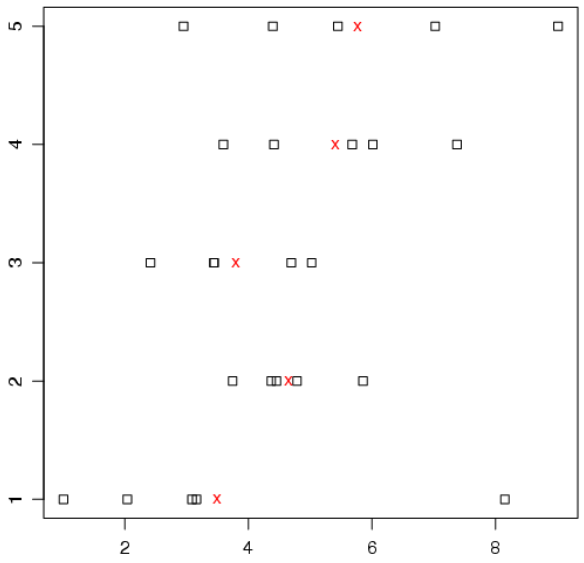
Experimente, por exemplo, inserir as médias no gráfico:

Figura 2.18: Stripchart com grupos e médias.

```
> grupo.m <- tapply(x, grupo, mean)
> grupo.m
```

```
       1        2        3        4        5
3.488884 4.642260 3.803347 5.415836 5.766047
```

```
> stripchart(x ~ grupo)
> stripchart(grupo.m ~ 1:5, col = 2, add = T, pch = "x")
```



Dotchart

A função `dotchart()` é utilizada visualizar os dados, na ordem em que são fornecidos. Antes de iniciar, veja quais são os argumentos dessa função.

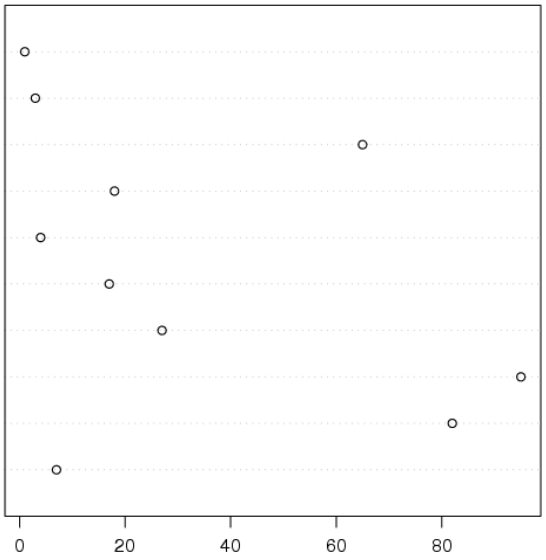
Como exemplo, considere o seguinte conjunto de dados:

```
> set.seed(12)
> dados <- sample(100, 10, rep = TRUE)
```

Um gráfico pode ser construído da seguinte maneira:

Figura 2.19: Dotchart.

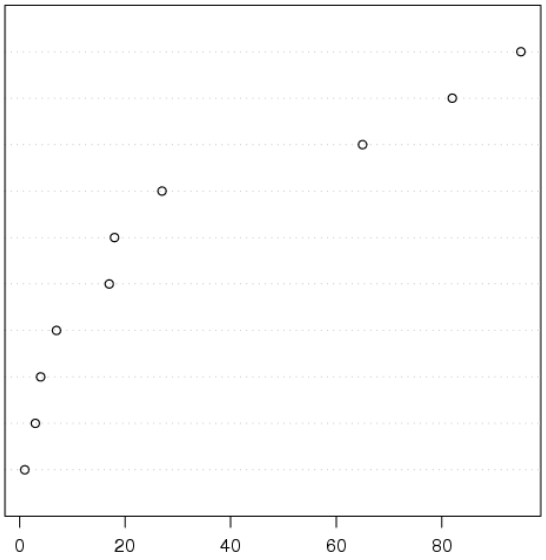
```
> dotchart(dados)
```



Também, se os dados estiverem em uma ordem, o mesmo conjunto de dados é visualizado de outra maneira:

Figura 2.20: Dotchart.

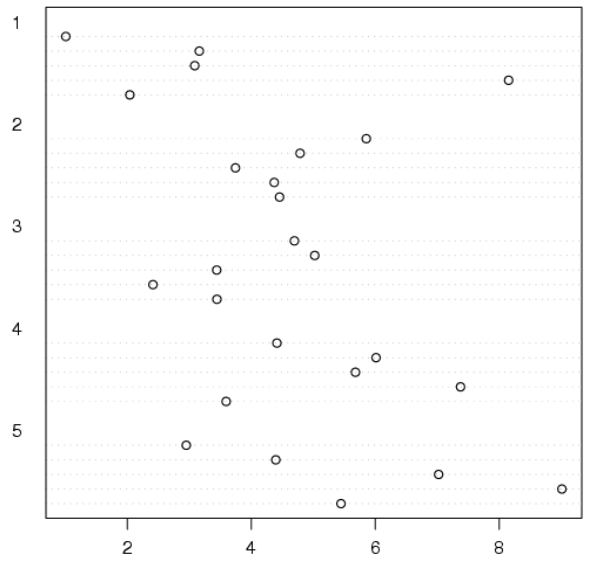
```
> set.seed(12)
> dados <- sample(100, 10, rep = TRUE)
> dados

 [1]  7 82 95 27 17  4 18 65  3  1

> dotchart(sort(dados))
```



Se os dados estiverem na forma de grupos

Figura 2.21: Dotchart.

```
> set.seed(12)
> x <- rnorm(25, 5, 2)
> grupo <- gl(5, 5)
> grupos <- data.frame(x, grupo)
> dotchart(x, groups = grupo)
```



Para finalizar, procure alterar os argumentos fornecidos para as funções `dotchart()` e `stripchart()`. Experimente também, estudar outros argumentos e trabalhar com outros dados.

2.4 Mais gráficos exploratórios

Existem muitos gráficos para análsie exploratória de dados. Nesta seção, veremos alguns outros gráficos que podem ser utilizados para resumir informações. No **R**, alguns desses gráficos são inseridos em pacotes (packages).

Por exemplo, no pacote `gplots` existem algums funções gráficas para explorar dados.

2.4.1 ballonplot

A função `baloonplot()` resume as informações em uma tabela mas, há algumas características que podem ser acrescentadas nessa tabela.

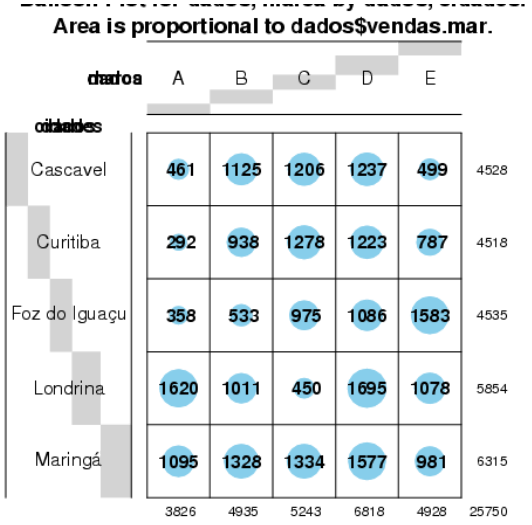
Primeiro, é necessárioa carregar o pacote `gplots`.

```
> require("gplots")
```

Veja o exemplo:

Figura 2.22: Baloonplot.

```
> set.seed(12)
> vendas.mar <- sample(100:1000, 50)
> vendas.abr <- sample(100:800, 50)
> marca <- rep(LETTERS[1:5], each = 10)
> cidade <- c("Curitiba", "Maringá", "Londrina", "Cascavel",
+       "Foz do Iguaçu")
> cidades <- rep(cidade, 10)
> dados <- data.frame(cidades, marca, vendas.mar, vendas.abr)
> balloonplot(dados$marca, dados$cidades, dados$vendas.mar)
```



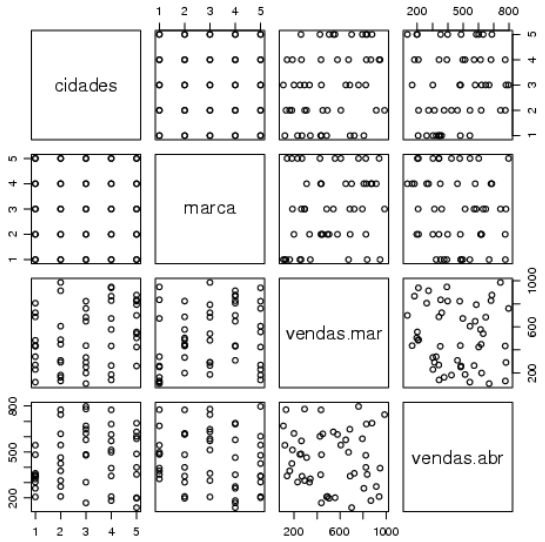
Experimente verificar quais são os argumentos da função e tente modificar alguns argumentos do exemplo acima.

2.4.2 pairs

Outro gráfico exploratório, pode ser construído com a função `pairs()`. Essa função relaciona as variáveis contidas em um 'data frame' ou matriz, formando uma matriz de dispersão de dados. Veja o resultado da função aplica aos dados de vendas:

Figura 2.23: Múltiplos gráficos de dispersão.

```
> pairs(dados)
```



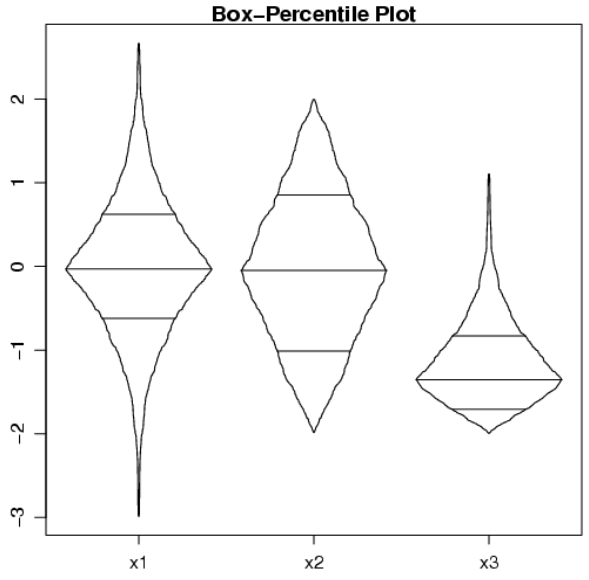
2.4.3 bpplot

Dentro do pacote `Hmisc`, experimente utilizar a função `bpplot()`. Essa função cria um boxplot, considerando os percentis de cada variável. Essaa função retorna um gráfico para a varíavel parecido com o boxplt mas, as observações da

variável são orepresentadas pela largura do boxplot.

Figura 2.24: Gráfico boxplot com percentis.

```
> set.seed(12)
> x1 <- rnorm(500)
> x2 <- runif(500, -2, 2)
> x3 <- abs(rnorm(500)) - 2
> bpplot(x1, x2, x3)
```

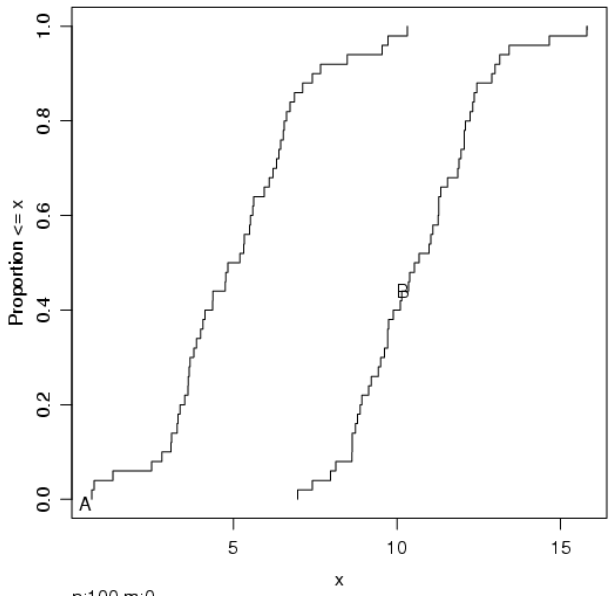


2.4.4 ecdf

A função `ecdf`, dentro do pacote `Hmisc`, constrói um gráfico com a função de distribuição empírica acumulada dos dados. É um gráfico onde a ordem de cada observação é inserida no gráfico em função de cada observação na amostra.

Figura 2.25: Gráfico ecdf.

```
> A <- rnorm(50, 5, 2)
> B <- rnorm(50, 10, 2)
> x <- c(A, B)
> g <- c(rep("A", length(A)), rep("B", length(B)))
> ecdf(x, group = g)
```



2.4.5 O gráfico Normal de Probabilidade

Em muitas análises estatísticas, um dos pressupostos mais comuns é de que a variável aleatória em estudo tenha distribuição Normal. Além de vários testes de hipóteses existentes para realizar essa avaliação, existem, também, maneiras gráficas de se avaliar se uma variável aleatória possui ou não distribuição Normal.

Um desses métodos gráficos, bastante utilizado, é o Gráfico Normal de Probabilidade. No **R**, a função `qqnorm()` constrói o gráfico.

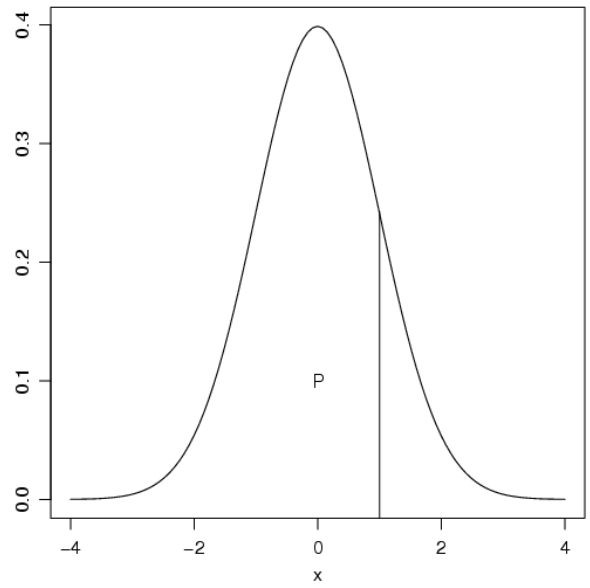
Esse gráfico também pode ser utilizado para avaliar se existem outliers ou efeitos significativos de fatores em experimentos planejados.

Base teórica do método

Considere a figura 2.26, representando uma Distribuição Normal.

Figura 2.26: Gráfico da distribuição Normal.

```
> curve(dnorm(x), from = -4, to = 4, ylab = "")
> segments(1, -0.2, 1, 0.243)
> text(0, 0.1, "P")
```

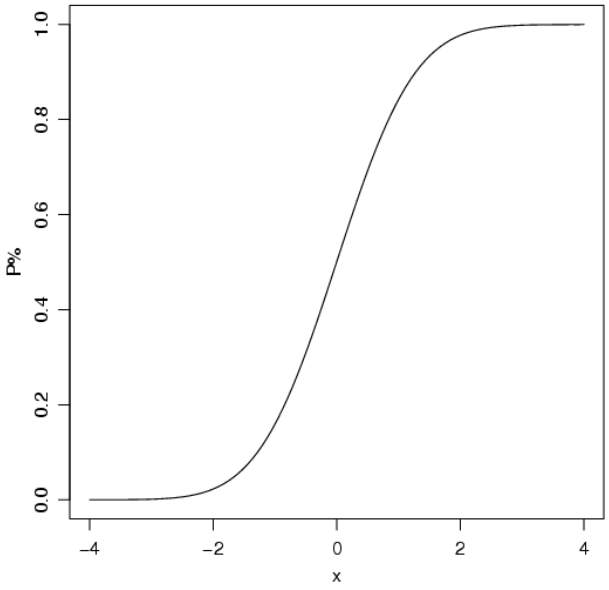


A probabilidade percentual *P* de ocorrência de algum valor menor do que *x* é dada pela área a esquerda de *x*.

Fazendo um gráfico para todos os possíveis valores de *P* vs *x* obtém-se a curva normal acumulada, que tem a característica sigmóide (forma de S, figura 2.27).

Figura 2.27: Gráfico da distribuição Normal acumulada.

```
> curve(pnorm(x), from = -4, to = 4, ylab = "P%")
```



Para gerar o gráfico normal de probabilidade, faz-se um ajuste na escala vertical (P) de modo a se obter uma linha reta. Considere uma amostra de 10 pontos (n=10). A primeira observação (ordenada) apresenta os primeiros 10% da distribuição acumulada da variável. A segunda representa 20% e assim por diante. Considerando que a para estabelecer uma reta a partir da curva acumulada o ponto central (50% ou 0.5) permanece o mesmo, o ajuste pode ser realizado em relação ao centro ou 0.5. Ou ainda, a cada 10% pode-se considerar o ponto intermediário da classe para refazer o gráfico de *P* vs *x*. Assim, de forma geral tem-se

$$P_i = 100 * \frac{i - 0.5}{n}$$

onde $i = 1, 2, 3, \ldots, n$ representa a posição da observação na amostra ordenada e n o tamanho da amostra ou no caso de um experimento, o número de efeitos.

Um gráfico das observações ordenadas vs os valores percentuais P_i indica o comportamento da variável.

No caso da avaliação da Normalidade de uma variável, espera-se que a maioria dos pontos esteja sobre a reta e concentrados na região central. Nas caudas, como se espera para uma distribuição Normal, podem aparecer pontos não muito concentrados e um pouco afastados da reta. Pontos muito distantes e afastados da reta indicam um possível outlier ou dado discrepante.

Em se tratando de um experimento, nesse gráfico, tem-se que tentar avaliar quais são os efeitos que não devem estar ocorrendo ao acaso. Isso pode ser identificado de forma *subjetiva* pelos pontos que se afastam de uma linha reta.

Essa linha reta é colocada subjetivamente sobre os pontos. Uma sugestão (Montgomery, 2001) é estabelecer uma reta entre os pontos nos quantis 25% e 75%.

Para saber quais são os efeitos *significativos* basta observar quais são os pontos no gráfico que se afastam da reta e identificar os efeitos correspondentes.

Construindo o gráfico no R

Exemplo de dados de um experimento:

Para construir o gráfico, inicialmente necessitamos das estimativas dos efeitos dos fatores do experimento. Considere o exemplo de um experimento fatorial 2^4 com uma repetição. Os efeitos estimados são apresentados na tabela 2.1:

Tabela 2.1: Estimativas de efeitos de um experimento 2^4

Efeitos	Estimativas
A	-8,00
B	24,00
C	-2,25
D	-5,50
AB	1,00
AC	0,75
AD	0,00
BC	-1,25
BD	4,50
CD	-0,25
ABC	-0,75
ABD	0,50
ACD	-0,25
BCD	-0,75
ABCD	-0,25

As probabilidades de cada um dos efeitos são obtidas por $P = 100 * (i - 0, 5)/15$. Assim, os resultados para construção do gráfico são apresentados na tabela 2.4.5

Tabela 2.2: Cálculo dos Percentuais P_i

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
efeitos	-8	-5,5	-2,25	-1,25	-0,75	-0,75	-0,25	-0,25	-0,25	0	0,5	0,75	1	4,5	24
Identificação	A	D	C	BC	ABC	BCD	CD	ACD	ABCD	AD	ABD	AC	AB	BD	B
P_i	3,3	10	16,7	23,3	30	36,7	43,3	50	56,7	63,3	70	76,7	83,3	90	96,7

As probabilidades de cada um dos efeitos são obtidas por $P = 100 * (i - 0,5)/15$.

O gráfico é obtido quando os efeitos *vs* P são colocados em um gráfico de dispersão. Uma linha reta entre os valores dos efeitos ajuda a verificar quais são os pontos que podem ser considerados como significativos.

Essa linha reta é colocada subjetivamente sobre os pontos. Uma regra (Montgomery, 2001) é ligar os pontos nos quantis 25% e 75%.

Os efeitos são:

```
> efeitos <- c(-8, 24, -2.25, -5.5, 1, 0.75, 0, -1.25, 4.5,
+       -0.25, -0.75, 0.5, -0.25, -0.75, -0.25)
```

Ordenando

```
> efeitos.o <- sort(efeitos)
> fatores <- c("A", "B", "C", "D", "AB", "AC", "AD", "BC",
+       "BD", "CD", "ABC", "ABD", "ACD", "BCD", "ABCD")
> names(efeitos) <- fatores
```

Aplicando os percentuais acumulados para os efeitos tem-se os seguintes valores:

```
> p <- c(3.3, 10, 16.7, 23.3, 30, 36.7, 43.3, 50, 56.7, 63.3,
+       70, 76.7, 83.3, 90, 96.7)
```

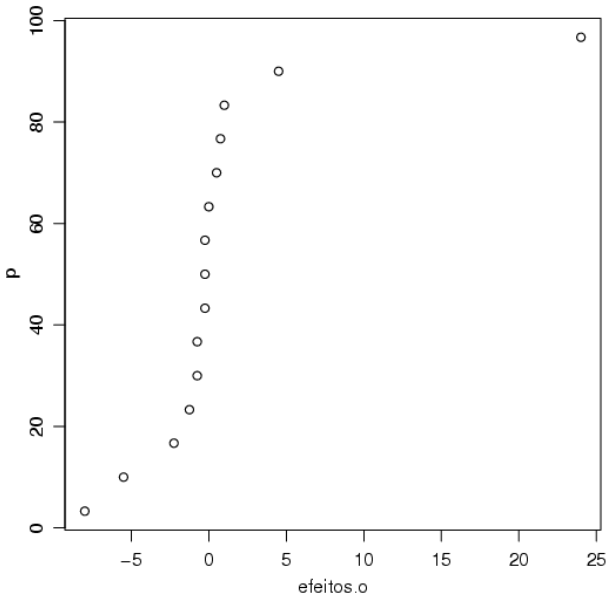
Esses valores podem ser obtidos através da seguinte função no **R**:

```
> P <- function(efeitos.o) {
+       p <- numeric(0)
+       n <- 1:length(efeitos)
+       for (i in n) {
+            p[i] <- 100 * (i - 0.5)/ length(efeitos)
+       }
+       print(p)
+ }
> P()

 [1]  3.333333 10.000000 16.666667 23.333333 30.000000 36.666667 43.333333
 [8] 50.000000 56.666667 63.333333 70.000000 76.666667 83.333333 90.000000
[15] 96.666667
```

O gráfico (figura 2.28) é obtido pelo comando `plot()`:

Figura 2.28: Gráfico dos efeitos.

```
> plot(efeitos.o, p)
```



Para fazer a linha, podemos encontrar os quantis dos efeitos,

```
> quantile(efeitos.o)
```

```
    0%    25%    50%    75%   100%
-8.000 -1.000 -0.250  0.625 24.000
```

Utilizando o comando `locator()` pode-se fazer uma linha entre os pontos dos quantis 25% e 75%. Depois do comando, clique nos pontos correspondentes no gráfico e uma linha será criada.

```
> points(quantile(efeitos.o, 0.25), 25, col = 2, pch = 19,
+     cex = 0.5)
> points(quantile(efeitos.o, 0.75), 75, col = 2, pch = 19,
+     cex = 0.5)
> segments(quantile(efeitos.o, 0.25), 25, quantile(efeitos.o,
+     0.75), 75, col = "blue")
```

Você pode usar, também, o comando `locator`:

```
> locator(n = 2, type = "l")
```

Os efeitos significativos podem ser identificados com o comando `identify()`. Lembre-se que os efeitos estão ordenados do menor para o maior.

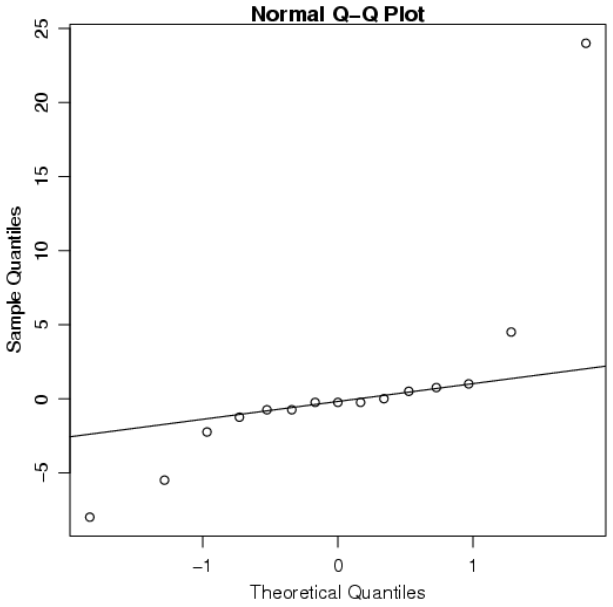
```
> identify(efeitos.o, p)
```

Para terminar, aperte o botão direito do mouse (Linux). No Windows, aperte o botão direito e selecione `stop`.

Faça agora o Gráfico Normal de Probabilidade gerado pelo **R**. Compare os gráficos.

Figura 2.29: Gráfico Normal de Probabilidade.

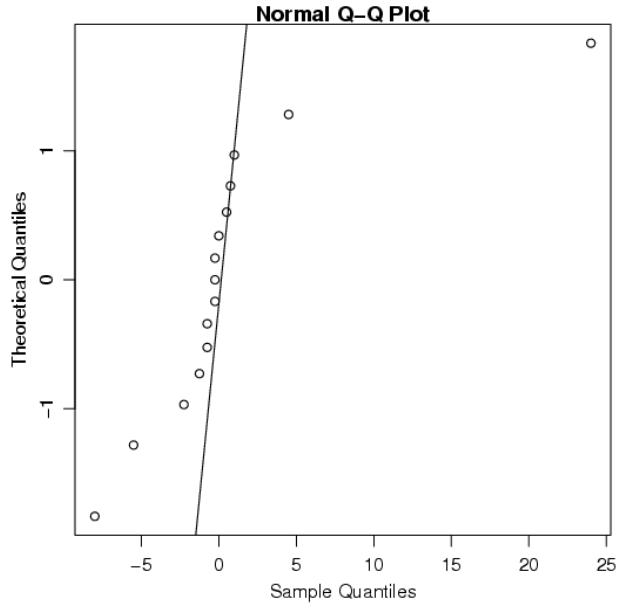
```
> qqnorm(efeitos)
> qqline(efeitos)
```



Experimente a opção

Figura 2.30: Gráfico Normal de Probabilidade - datax=T.

```
> c <- qqnorm(efeitos, datax = T)
> qqline(efeitos)
```



```
> identify(c$x, c$y, labels = names(c$x))
```

E agora?

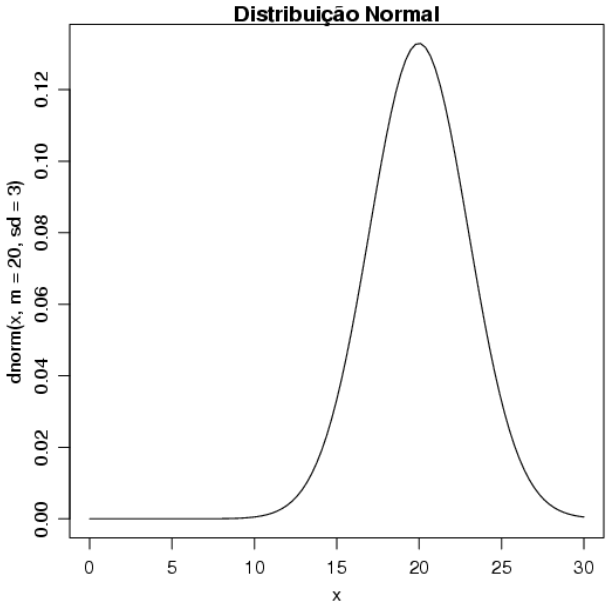
2.4.6 Curvas de densidades

Em algumas situações, pode ser de interesse fazer gráficos de funções relacionadas à algumas distribuições. Nesta seção, veremos como utilizar a função `curve()` para gerar alguns gráficos dessa natureza.

Por exemplo, vamos utilizar inicialmente a distribuição Normal.

Figura 2.31: Uso da função curve para Distribuição Normal.

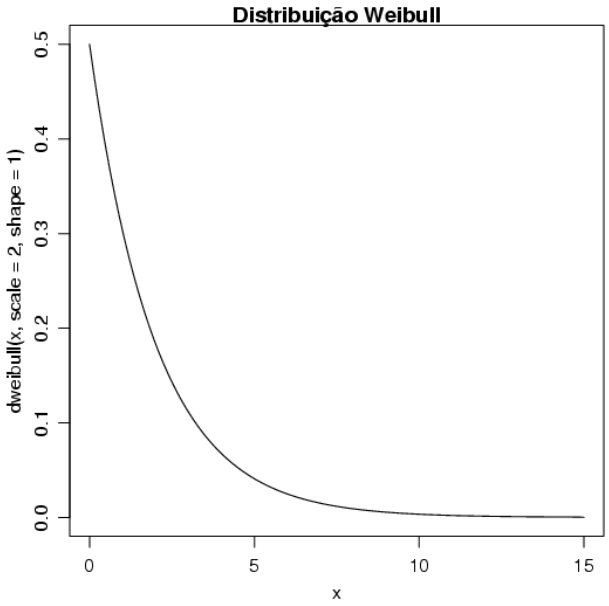
```
> curve(dnorm(x, m = 20, sd = 3), from = 0, to = 30, main = "Distribuição Normal")
```



A distribuição Weibull, pode ser representada da seguinte forma:

Figura 2.32: Uso da função curve para Distribuição Weibull.

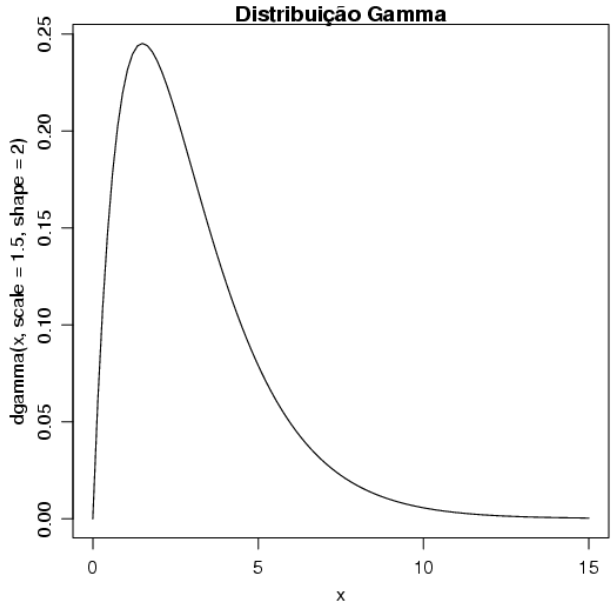
```
> curve(dweibull(x, scale = 2, shape = 1), from = 0, to = 15,
+       main = "Distribuição Weibull")
```



E a distribuição Gama, da seguinte maneira:

Figura 2.33: Uso da função curve para Distribuição Gamma.

```
> curve(dgamma(x, scale = 1.5, shape = 2), from = 0, to = 15,
+       main = "Distribuição Gamma")
```



Em cada uma destas distribuições, modifique os parâmetros para entender o comportamento de cada uma.

Sugestão de exercícios

Experimente construir o gráfico de outras distribuições. Como sugestão, experimente a distribuição *F*, χ^2 e *t*.

2.4.7 coplot

A função `coplot()` pode ser utilizada para representar dados que podem ser arranjados da forma *respâ‖b*, onde *a* e *b* são variáveis do tipo fator. Em geral, modelos estatísticos podem ser escritos sob essa forma. Uma plicação está na análise de dados experimentais, por exemplo.

Como exemplo, considere o seguinte conjunto de dados

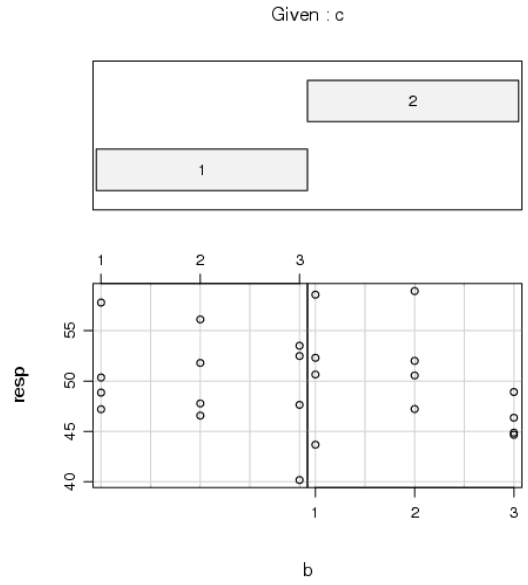
```
> set.seed(123)
> resp <- rnorm(24, 50, 5)
> b <- as.factor(rep(1:3, each = 8))
> c <- as.factor(rep(1:2, each = 4, len = 24))
> dados <- data.frame(resp, b, c)
> dados
```

```
       resp b c
1  47.19762 1 1
2  48.84911 1 1
3  57.79354 1 1
4  50.35254 1 1
5  50.64644 1 2
6  58.57532 1 2
7  52.30458 1 2
8  43.67469 1 2
9  46.56574 2 1
10 47.77169 2 1
11 56.12041 2 1
12 51.79907 2 1
13 52.00386 2 2
14 50.55341 2 2
15 47.22079 2 2
16 58.93457 2 2
17 52.48925 3 1
18 40.16691 3 1
19 53.50678 3 1
20 47.63604 3 1
21 44.66088 3 2
22 48.91013 3 2
23 44.86998 3 2
24 46.35554 3 2
```

A função `coplot()` pode ser utilizada da seguinte maneira:

Figura 2.34: Uso do coplot.

```
> coplot(resp ~ b | c)
```

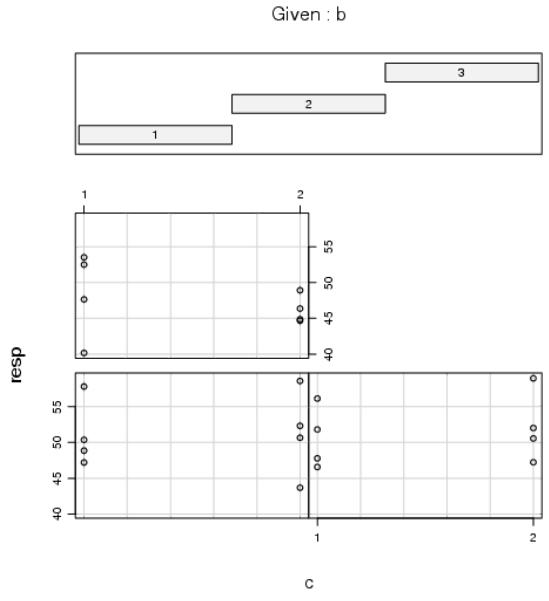


Interprete o gráfico!

Experimente, também, inverter os fatores:

Figura 2.35: Uso do coplot.

```
> coplot(resp ~ c | b)
```



2.4.8 Gráficos para representação de superfície de resposta

Em algumas análises, gráficos tridimensionais podem ser gerados para visualização do comportamento de modelos estatísticos.

Superfície de resposta

Entre os mais comuns está o gráfico de superfície de resposta. Este tipo de gráfico é gerado a partir de modelos que expressam o comportamento de uma variável resposta em função de dois ou mais fatores.

Não abordaremos aqui o ajuste de modelos. Faremos apenas o uso de modelos pré-existentes

Para construir o gráfico de superfície de resposta são necessárias três variáveis, ou seja, *x* e *y*, representando os níveis dos fatores e uma variável *z*, representando o comportamento da variável resposta.

Como exemplo, considere uma variação de *x* e *y* da seguinte forma

```
> x <- seq(-1.5, 1.5, length = 30)
> y <- seq(-1.5, 1.5, length = 30)
```

Os valores preditos pelo modelo, nessa escala de variação, podem ser preditos por uma função ou um modelo

```
> z.f <- function(x, y) {
+     y <- 31.76 + 1 * x + 0.46 * y - 0.74 * x^2 - 0.51 * y^2 +
+         0.275 * x * y
+ }
```

Alternativamente, os coeficientes do modelo podem ser obtidos diretamente do modelo (nesse caso ele não existe, apenas apresentamos a forma)

```
> z <- function(x = x, y = y) {
+     y <- modelo$coef[1] + modelo$coef[2] * x + modelo$coef[3] *
+         y + modelo$coef[4] * x^2 + modelo$coef[5] * y^2 +
+         modelo$coef[6] * x * y
+ }
```

Antes de construirmos o gráfico, veja como as três variáveis *x*, *y* e *z* estão organizadas.

Para gerar o gráfico, os dados precisam ser agrupados de forma a serem utilizados pela função `persp`.

A função `outer` prepara os dados para serem utilizados pela função `persp`.

```
> z <- outer(x, y, z.f)
```

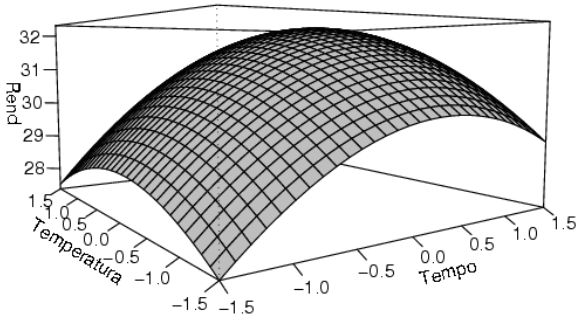
Para mais detalhes dessa função utilize

```
> "?"(outer)
```

Para construir a superfície, utilize a função `persp` da seguinte maneira:

Figura 2.36: Gráfico de uma superfície de resposta.

```
> persp(x, y, z, theta = -35, phi = 5, expand = 0.5, col = "gray",
+     xlab = "Tempo", ylab = "Temperatura", zlab = "Rend",
+     scale = T, ticktype = "detailed")
```



Experimente alterar alguns argumentos e entender sua funcionalidade.

Os mesmos dados podem ser representados de duas outras formas. Através de curvas de nível com a função `countour()` ou de um gráfico representando a variável *z* por cores com a função `image()`.

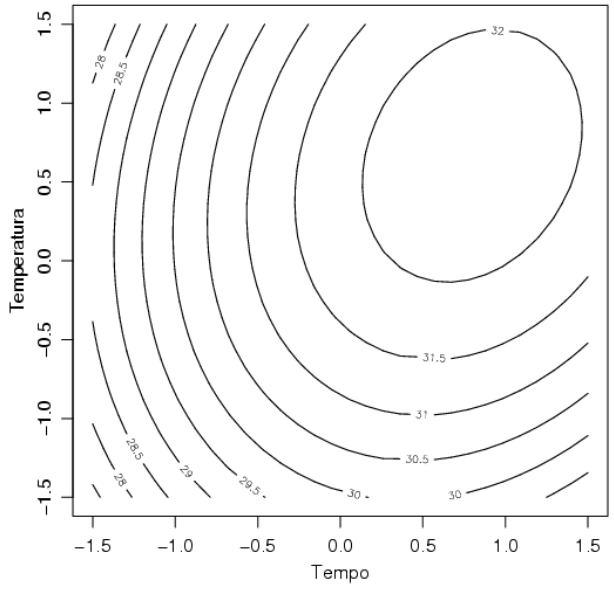
Curva de nível

Uma curva de nível pode ser construída com a função `countour()`. Observe que cada curva corresponde a uma 'altura' da variável resposta

---

Figura 2.37: Gráfico de uma curva de nível.

```
> contour(x, y, z, xlab = "Tempo", ylab = "Temperatura")
```



Investigue a função `curve()` e experimente alterar algumas opções.

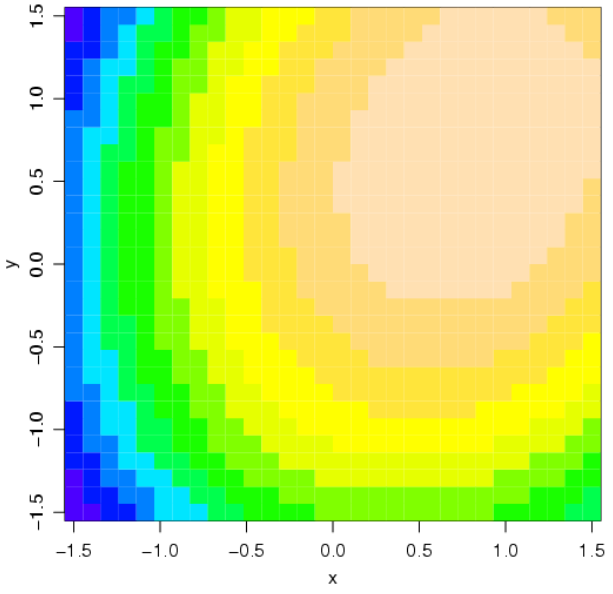
Gráfico image()

Uma outra alternativa aos gráficos anteriores é construir um gráfico de cores para representar o modelo. A função `image()` pode ser utilizada nesse caso.

VEja o seguinte exemplo:

Figura 2.38: Gráfico de cores.

```
> image(x, y, z, col = topo.colors(12))
```

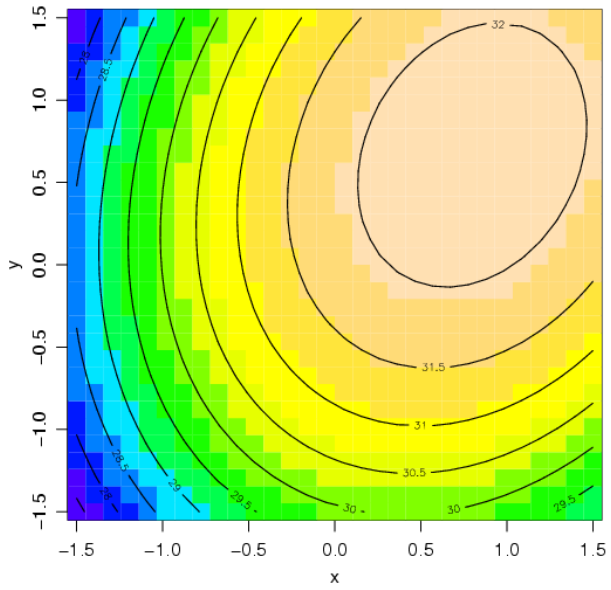


Para finalizar, considere que o ponto máximo (nesse caso) corresponde a 168 minutos e 151,5 graus. A representação pode ser feita da seguinte forma

---

Figura 2.39: Gráfico com representação do ponto de máximo.

```
> image(x, y, z, col = topo.colors(12))
> contour(x, y, z, xlab = "Tempo", ylab = "Temperatura", add = T)
> points(168, 151.5, col = "red")
```



Para finalizar, experimente utilizar alguns exemplos do pacote `lattice`:

```
> require(lattice)
> x <- seq(-pi, pi, len = 20)
> y <- seq(-pi, pi, len = 20)
> g <- expand.grid(x = x, y = y)
> g$z <- sin(sqrt(g$x^2 + g$y^2))
> print(wireframe(z ~ x * y, g, drape = TRUE, aspect = c(3,
+     1), colorkey = TRUE))

> wireframe(volcano, shade = TRUE, aspect = c(61/87, 0.4),
+     light.source = c(10, 0, 10))
```

Experimente também a utilizar a função `scatterplo3d()` do pacote do mesmo nome:

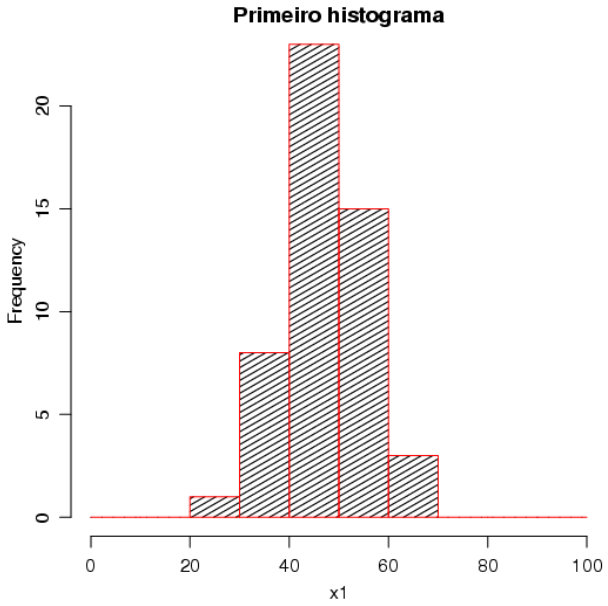
```
> require(scatterplot3d)
> z <- seq(-10, 10, 0.01)
> x <- cos(z)
> y <- sin(z)
> scatterplot3d(x, y, z, highlight.3d = TRUE, col.axis = "blue",
+     col.grid = "lightblue", main = "scatterplot3d - 1", pch = 20)
```

2.5 Exercícios

1. Construa o histograma da figura 2.40, baseado nos seguintes dados:

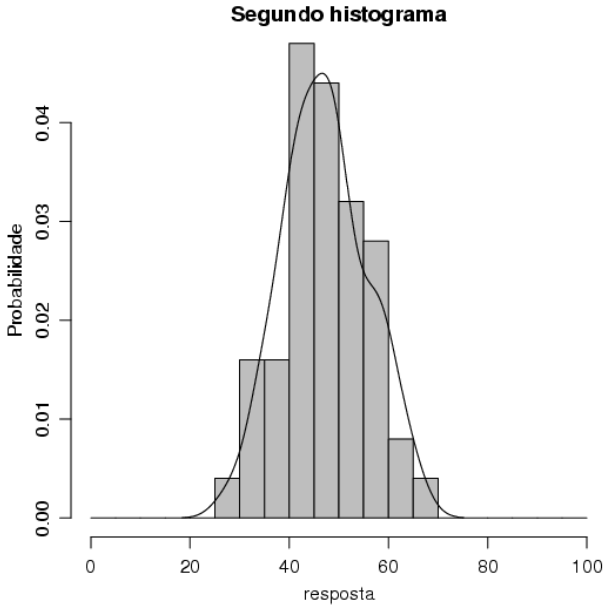
```
> set.seed(11)
> x1 <- rnorm(50, 50, 10)
```

Figura 2.40: Primeiro histograma.



2. Construa o histograma da figura 2.41, baseado nos seguintes dados de 1:

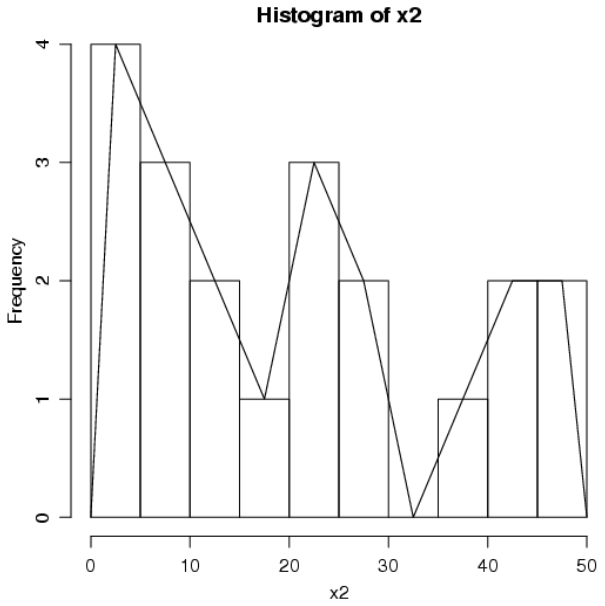
Figura 2.41: Segundo Histograma.



3. Construa o histograma e o polígono de ferqüência da figura 2.42, baseado nos seguintes dados:

```
> set.seed(11)
> x2 <- sample(1:50, 20, rep = TRUE)
```

Figura 2.42: Polígono de Freqüência.



4. Construa o gráfico boxplot da figura 2.43, baseado nos seguintes dados:

```
> set.seed(123)
> x3 <- rnorm(50)
```

Figura 2.43: Gráfico boxplot com rug.