

S-PLUS/R : Introdução

1. O que é o S (ou R, a versão livre do S)?
 - (a) S (ou R) é uma linguagem de programação estatística de alto nível e um ambiente de cálculo.
2. S (ou R) é uma linguagem de programação que é:
 - (a) interactiva
 - (b) interpretativa
 - (c) funcional
 - (d) orientada por objectos
3. Para usar S (ou R) de forma eficaz é necessário ter alguns conhecimentos de:
 - (a) métodos estatísticos
 - (b) cálculo numérico
 - (c) técnicas de programação
 - (d) representações estatísticas gráficas
4. Para aprender S (ou R) de forma eficaz será necessário reconhecer e compreender algumas das muitas características e conceitos da linguagem em S (ou R), tais como:
 - (a) functions
 - (b) objects
 - (c) vectorizing
 - (d) coercing
 - (e) lists
 - (f) data frames

(g) methods

5. MANUAIS:

- (a) Manuais on-line em formato PDF disponíveis através do menu Help na versão Windows do S-PLUS 2000 ;
- (b) Manuais on-line em formato PDF: An Introduction to R, the R Reference Manual, e R Language Manual estão disponíveis através do menu Help no menu standard em R;

6. Localizações Web úteis:

- (a) S-PLUS: <https://delta.ist.utl.pt/software/splus.php>
- (b) R site: <http://cran.r-project.org/>

S-PLUS

1. É recomendado que qualquer trabalho realizado no S-PLUS seja colocado em diferentes espaços de trabalho (WORKSPACE). Um “Workspace” é simplesmente uma pasta (directoria) que contém a base de dados numa pasta específica designada por .Data. Esta base de dados conterá o conjunto de todos os objectos S-PLUS que serão criados durante uma sessão de S-PLUS (incluindo programas que poderão ser criados durante essa sessão). Usar “Workspace” (s) diferentes permitirá organizar o trabalho de S-PLUS por projecto de trabalho, diminuindo assim quer a perda de objectos devido a conflitos de nome quer a outros conflitos.
2. Por isso recomendo que criem um “Workspace” cada vez que iniciarem um projecto distinto. Para isso basta seleccionar:

File -> Workspace -> New

a partir do menu principal do S-PLUS, como mostra a Figura 1.

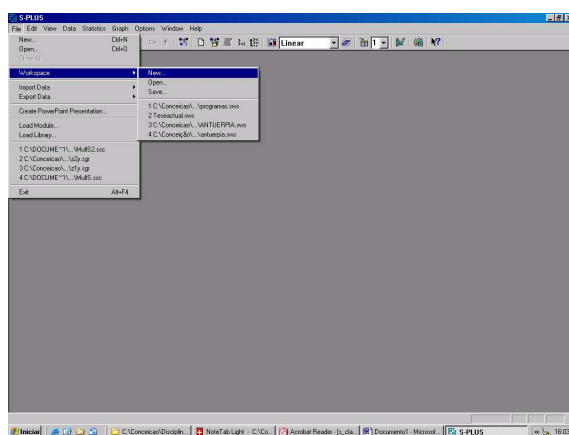


Figura 1: Menu principal do S-PLUS. Criação de um espaço de trabalho.

Aparece uma “dialog box” e aí devem dar o nome (e o caminho) para o novo espaço de trabalho, por exemplo:

```
C:\Conceicao\AML\Projecto1
```

Antes de terminarem a sessão de S-PLUS **DEVEM** guardar o vosso espaço de trabalho, fazendo:

```
File -> Workspace ->Save
```

A próxima vez que começarem uma sessão de S-PLUS devem seleccionar esse **Workspace** que está guardado, bastando fazer:

```
File -> Workspace ->Open
```

escolhendo o espaço de trabalho que pretendem. Estes três passos são muito importantes.

3. O S-PLUS permite efectuar análise de dados de forma padrão bastando utilizar os menus da barra de menu principal do S-PLUS. Basicamente, um conjunto de dados é criado seleccionando

```
File -> New ->Data Set
```

ou pressionando o botão **New Data Set** disponível na barra de tarefas principal. Como resultado de qualquer uma destas operações uma janela (tipo folha de cálculo) abrirá. Os valores dos dados que constitui em o conjunto de dados entrarão por colunas (ou linhas) para cada variável. É possível modificar as propriedades, tais como os nomes e formatos, das variáveis que são criadas. Podem dar um nome ao conjunto de dados e guardá-los numa base de dados de qualquer um dos **Workspace** que previamente criaram.

4. Para calcular estatísticas descritivas num determinado conjunto de dados basta usar o menu principal do S-PLUS seleccionado:

```
Statistics -> Data Summaries -> Summary Statistics...
```

e aí seleccionar o conjunto de dados que se pretende analisar. Esta caixa de diálogo permite ainda ao utilizador seleccionar variáveis específicas assim como produzir estatísticas sumárias por grupos.

5. Para realizar regressão linear simples num determinado conjunto de dados basta escolher no menu principal do S-PLUS:

```
Statistics -> Regression -> Linear...
```

e escolher o nome do conjunto de dados que se pretende analisar, (se ainda não estiver seleccionado). De seguida, selecciona-se a variável dependente usando o menu “drop-down”, e também seleccionar a(s) variáveis explicativas (independentes) de forma análoga à anterior. É ainda possível usar as outras janelas que estão no topo da caixa de diálogo. Por exemplo, seleccionando a janela denominada `Plot`, poder-se-ão mandar fazer um conjunto de gráficos de diagnóstico (como, o gráfico de `Residuals vs. Fit`, `Residuals Normal QQ`). Estas acções criarão uma janela denominada `Report Window` onde os resultados da análise de regressão são apresentados, assim como as janelas dos gráficos pedidos.

6. As análises estatísticas de dados que foram descritas até aqui foram baseadas na utilização da barra de menus do Menu principal do S-PLUS. Esta é uma primeira abordagem. A abordagem que acho mais flexível é a baseada na linguagem de programação S, a qual é um exemplo de uma linguagem de programação interpretativa e funcional. No S-PLUS 2000, é necessário, em primeiro lugar, abrir uma janela (se por defeito ela não abrir automaticamente no início do programa) denominada `Commands Window` fazendo no menu principal do S-PLUS

`Window ->Commands Window`

Em vez disto esta janela pode ser aberta usando o botão `Commands Window` que está na barra de tarefas principal. Nesta janela serão escritos os comandos do S-PLUS, cada um após o símbolo `>`.

Utilização da janela de comandos

1. Os comandos S são expressões que entram após a *prompt* `>`. Em resposta o S mostra o resultado avaliando a expressão. Como um exemplo simples, escreve-se a expressão $2*(5.3 - 9.2)$ após a *prompt* `>`:

```
> 2*(5.3 - 9.2)
```

Esta é uma expressão numérica usando os operadores `*` e `-` e os parêntesis anexam uma sub-expressão. O resultado é mostrado na linha (ou linhas) imediatamente a seguir (a esta acção designa-se em S por `print`). O resultado da avaliação da expressão acima resulta em:

```
[1] -7.8
```

2. Quando a um valor de uma expressão se afecta um nome cria-se um objecto S. Os objectos S (criados ou já existentes) podem ser utilizados noutras expressões S. Um objecto S pode conter valores numéricos, conjunto de caracteres (“character strings”), valores lógicos i.e., (verdadeiro (T) ou falso (F)), resultados de uma análise estatística completa ou ainda um gráfico. Os nomes dos objectos e constantes são combinados com operadores e funções formando expressões mais complexas. Por exemplo,

```
> e<-2*(5.3 - 9.2)
> e
[1] -7.8
> g_2*(5.3 - 9.2)
> 2*g
[1] -15.6
>
```

Como acima ilustrado, o símbolo <- (ou _) é usado em S para associar o nome de um objecto ao valor de uma expressão.

3. S é uma linguagem funcional o que faz com que a maior parte das operações que são produzidas em S são através do uso de funções: quer internas quer construídas pelo utilizador. A função c() é um exemplo de uma função em S por este motivo não se devem denominar quaisquer outros objectos pela letra c. Esta função cria um vector, o qual pode ser atribuído a um objecto da seguinte forma:

```
> c(15:1, 3:11, 7*2, 8)
[1] 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 3 4 5 6 7 8 9
[23] 10 11 14 8
> h<-c(15:1, 3:11, 7*2, 8)
> h
```

h é agora um objecto do tipo vector. O S é capaz de fazer cálculos vectoriais. Por exemplo, operações como determinar logaritmos ou calcular raízes quadradas, podem ser feitas quer em vectores quer em elementos individuais do vector:

```
> log(h)
[1] 2.7080502 2.6390573 2.5649494 2.4849066 2.3978953 2.3025851
```

```

[7] 2.1972246 2.0794415 1.9459101 1.7917595 1.6094379 1.3862944
[13] 1.0986123 0.6931472 0.0000000 1.0986123 1.3862944 1.6094379
[19] 1.7917595 1.9459101 2.0794415 2.1972246 2.3025851 2.3978953
[25] 2.6390573 2.0794415
> sqrt(h)
[1] 3.872983 3.741657 3.605551 3.464102 3.316625 3.162278 3.000000
[8] 2.828427 2.645751 2.449490 2.236068 2.000000 1.732051 1.414214
[15] 1.000000 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
[22] 3.000000 3.162278 3.316625 3.741657 2.828427
> 1/h
[1] 0.06666667 0.07142857 0.07692308 0.08333333 0.09090909 0.10000000
[7] 0.11111111 0.12500000 0.14285714 0.16666667 0.20000000 0.25000000
[13] 0.33333333 0.50000000 1.00000000 0.33333333 0.25000000 0.20000000
[19] 0.16666667 0.14285714 0.12500000 0.11111111 0.10000000 0.09090909
[25] 0.07142857 0.12500000
>

```

Esta funcionalidade permite tirar partido do cálculo vectorial evitando a utilização de ciclos no código de programação, o que torna o cálculo mais eficiente.

4. As funções `ls()` e `rm()` permitem verificar e organizar os objectos S da corrente directoria `Data`.

```

> ls()
[1] ".Last.value" ".Random.seed" "last.dump" "x"
[5] "y"
> rm(x,y)
> ls()
[1] ".Last.value" ".Random.seed" "last.dump"

```

Em alternativa pode usar-se os botões da barra de tarefas padrão. O botão do *object explorer* que se encontra nesta barra pode ser usado para abrir a janela de diálogo do *object explorer*. Esta janela pode ser utilizada para organizar os objectos da base de dados corrente.

5. Um `vector` é um objecto S; `matrix`, `array` e `data.frame` são exemplos de outras classes de objectos em S. Por exemplo, a função S `matrix()` que será utilizada em seguida cria um objecto matriz 3×2 denominado `m` usando os dados especificados como um vector.

```

> m <- matrix(c(1.2, 3.5, 4.7, 1.8, -6.2, 5.3), 3,2)
> m
      [,1] [,2]
[1,]  1.2  1.8
[2,]  3.5 -6.2
[3,]  4.7  5.3
>m[, 2]
[1]  1.8 -6.2  5.3
>m[2:3, 2]
[1] -6.2  5.3
> mode(m)
[1] "numeric"

```

As funções `dim()` e `dimnames()` são úteis para determinar os atributos do objecto `matrix`.

```

> dim(m)
[1] 3 2
>dimnames(m)
NULL

```

Observe-se ainda que a matriz `m` anterior pode ser criada sem usar a função `matrix`, bastando para tal especificar os atributos dimensão do objecto `data` utilizando a função `dim()`.

```

> rm(m)
> m <- c(1.2, 3.5, 4.7, 1.8, -6.2, 5.3)
> m
[1]  1.2  3.5  4.7  1.8 -6.2  5.3
> dim(m)<-c(3, 2)
> m
      [,1] [,2]
[1,]  1.2  1.8
[2,]  3.5 -6.2
[3,]  4.7  5.3
>

```

6. Recorde-se que atrás foi descrito como usar a barra de menu para fazer regressão linear em S-Plus. Em seguida vai utilizar-se a função `S`, `lm()`, para fazer regressão linear simples da variável `y`(Fuel) em `x` (Disp.):

```
> fuel.fit <- lm(Fuel ~ Disp., fuel.frame)
> plot(fuel.fit)
```

O primeiro argumento de `lm()` é `Fuel ~ Disp` o qual é um exemplo de uma *formula* e é a notação usada em S para representar um modelo. Símbolos tais como `+`; `*`; `|`; e `:` são usados para definir as expressões de vários modelos. Por exemplo, se A and B são factores de tratamento e Y é a resposta, um modelo de dois factores com interacção pode ser especificado pela fórmula `Y~A+B +A : B`, ou simplesmente por `Y~A * B`.

7. O uso da função `lm()` é uma oportunidade para discutir um objecto especial do S, denominado lista (`list`). Até ao momento foram dados exemplos de ‘objectos simples’ tais como vectores, matrizes ou arrays, onde as componentes são objectos atómicos. Uma lista em S é um objecto composto, os objectos que a compõem podem ser objectos simples ou também outros objectos compostos. Quando são criadas aos seus componentes são dados números ou nomes. Assim, os componentes da lista podem ser referenciados quer por parêntesis rectos, ou, mais convenientemente usando expressões da forma:

```
nomelista$nomecomponente
```

Muitas funções do S têm como resultado componentes de listas. Estas componentes são descritas na descrição da função. Por exemplo, `lm()` coloca as estimativas dos parâmetros estimados em `coefficients` e os resíduos em `residuals`

```
> fuel.fit$coefficients
(Intercept)      Disp.
  2.691931  0.009984228
```

Os nomes das componentes podem ser abreviados ao número mínimo de letras que as identifique unicamente. Por exemplo, pode diminuir-se a palavra `coefficients` para `coef`:

```
> fuel.fit$coef
(Intercept)      Disp.
  2.691931  0.009984228
```

8. Um data frame é um objecto S muito semelhante a um objecto matrix com a excepção que as colunas podem ter diferentes modos (i.e., umas colunas podem ser factores, outras numéricas, outras lógicas). Os data frames são uma forma de input bastante corrente para entrada de muitas funções S de análise de dados. No exemplo seguinte, `baseball.df` é um data frame onde, as primeiras duas colunas são objectos de modo factor (códigos para os nomes dos jogadores), as colunas seguintes são numéricas, e a última é lógica.

```
> baseball.df
bat.ID pitch.ID event.typ outs.play err.play
r1 pettg001 clemr001 2 1 F
r2 whitl001 clemr001 14 0 F
r3 evand001 clemr001 3 1 F
r4 trama001 clemr001 2 1 F
```

9. Em S chamam-se funções genéricas a funções que fazem determinadas operações dependendo da classe dos objectos a que são aplicadas (tal como a saída da tabela anova a partir dos resultados da análise de regressão). Exemplos destas funções são `print()`, `summary()` e `plot()`. No exemplo acima, cada uma das funções `print()`, `summary()`, etc., utilizam o método corrente para produzir o output desejado a partir do objecto criado por `lm()`.

```
> print(fuel.fit)
Call:
lm(formula = Fuel ~ Disp., data = fuel.frame)
```

```
Coefficients:
 (Intercept)      Disp.
  2.691931  0.009984228
```

```
Degrees of freedom: 60 total; 58 residual
Residual standard error: 0.535053
```

```
>
> summary(fuel.fit)
```

```
Call: lm(formula = Fuel ~ Disp., data = fuel.frame)
Residuals:
      Min       1Q   Median       3Q      Max
```

```
-0.7998 -0.4765 0.04638 0.2653 1.356
```

Coefficients:

	Value	Std. Error	t value	Pr(> t)
(Intercept)	2.6919	0.2074	12.9796	0.0000
Disp.	0.0100	0.0013	7.7630	0.0000

Residual standard error: 0.5351 on 58 degrees of freedom

Multiple R-Squared: 0.5096

F-statistic: 60.26 on 1 and 58 degrees of freedom,
the p-value is 1.53e-010

Correlation of Coefficients:

```
(Intercept)
Disp. -0.9429
```

```
> anova(fuel.fit)
```

Analysis of Variance Table

Response: Fuel

Terms added sequentially (first to last)

	Df	Sum of Sq	Mean Sq	F Value	Pr(F)
Disp.	1	17.25253	17.25253	60.26418	1.530043e-010
Residuals	58	16.60434	0.28628		

O uso da função genérica `plot()` a um objecto criado por `lm()` origina um conjunto de 6 gráficos.

10. Suponha agora que tem um ficheiro de dados, por exemplo, `exemplo1.txt` e `exemplo2.txt` na directoria `C:\users`. A função `read.table()` permite converter esses dados externos directamente num data frame. O separador entre campos é por defeito o usual `\t` ou espaço em branco, mas pode ser definido outro separador.

```
> dat.df1 <-read.table("C:\\users\\exemplo1.txt")
> dat.df1
  V1 V2 V3
1  1  2  3
```

```
2 1 3 4
3 2 12 45
4 1 4 3
>
> read.table("C:\\users\\exemplo2.txt")
      V2 V3
aaa    2  3
bbb    3  4
cc   12 45
jjj    4  3
```

Em alternativa pode usar-se o menu principal do S-PLUS e fazer:

```
File -> Import Data -> From File
```

Gráficos

1. Muitos tipos de gráficos podem ser obtidos usando o botão **Graph** na barra de tarefas do menu principal do S-PLUS. Uma janela de gráfico é então aberta. Aparece ainda um conjunto de botões de desenho, estes podem ser usados para editar quer gráficos existentes quer para criar novos gráficos. Porém, veja-se como se usam funções gráficas tradicionais (`plot()`, `hist()`, `barplot()` etc.) usando o S. Como exemplo, aplique-se a função `plot()` para criar um gráfico de pontos. Coloquemos na base de dados o data frame `fuel.frame` (este frame está disponível numa directoria do S-Plus, basta aplicar a função `attach` para ficar disponível no espaço corrente de trabalho).

```
> attach(fuel.frame)
> plot(Disp.,Fuel)
```

2. Ao observar-se o gráfico que é apresentado na Figura 2 verifica-se que, por defeito, o tipo de gráfico é apenas de pontos. Mas como argumento da função `plot()` pode ser escolhido outro tipo, por exemplo, linhas (`type="l"`), ou pontos e linhas (`type="b"`), etc. Também os eixos, escalas, títulos, símbolos a desenhar e cores podem ser escolhidas. Podem modificar-se esses parâmetros usando os parâmetros gráficos, `pch=`, `col=`, `lty=`, `ylab=` etc., nas funções gráficas, como será ilustrado em seguida. Para descrição detalhada de todos os parâmetros gráficos veja-se a função `par()`. As cores podem ser especificadas

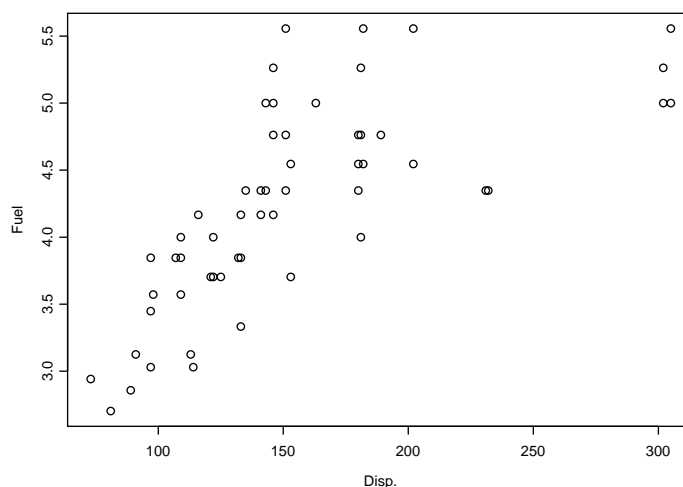


Figura 2: Exemplo de um gráfico de pontos.

usando o parâmetro `col=` as quais dependem do mapa de cores do dispositivo gráfico da máquina. Para se verem as cores por defeito do mapa faça-se:

```
> barplot(rep(1,17), col =1:17)
```

Este comando desenha um gráfico de barras com 17 cores. Usando os inteiros de 1 a 17 podem especificar-se as cores dos elementos do gráfico, tais como das linhas ou símbolos.

Recorde-se o objecto `fuel.fit` resultado da análise de regressão linear e tendo a janela do gráfico da Figura 2 aberta faça-se:

```
> abline(fuel.fit$coef,lty=4)
> plot(Disp.,Fuel,pch=3)
> lines(Disp., fitted(fuel.fit), col=6)
WARNING: Line out of bounds: x1 = 305.000000, y1 = 5.737120, x2 = 302.00
0000, y2 = 5.707168, ymin = 2.588589, ymax = 5.669670
```

Vários exemplos podem ser dados, um outro é a utilização da função `points()`, esta permite adicionar pontos a um gráfico em branco (onde os eixos e escala estão já presentes) ou a um gráfico já desenhado:

```
> plot(Disp.,Fuel,type="n")
> points(Disp., Fuel, col=4, pch=8)
```

As funções `lines()`, `points()`, `abline()`, e `text()`, como ilustrado acima, são funções que permitem usar e modificar várias características a um gráfico existente.

- Para obter gráficos múltiplos numa mesma janela devem ser usadas as funções `mfrow()` ou `mfcol()`. Os seguintes exemplos ilustram a utilização de `mfrow()` e o resultado apresenta-se na Figura 3.

```
> par(mfrow=c(2,2), mar=rep(4,4))
> plot(Disp., Fuel, col=2, pch=4)
> lines(Disp., fuel.fit$fitted, col=5, lty=4)
> qqnorm(fuel.fit$res, col=6)
> boxplot(fuel.fit$res, ylab="Residuals")
> hist(fuel.fit$res, xlab="Residuals", main="Histogram of Residuals")
```

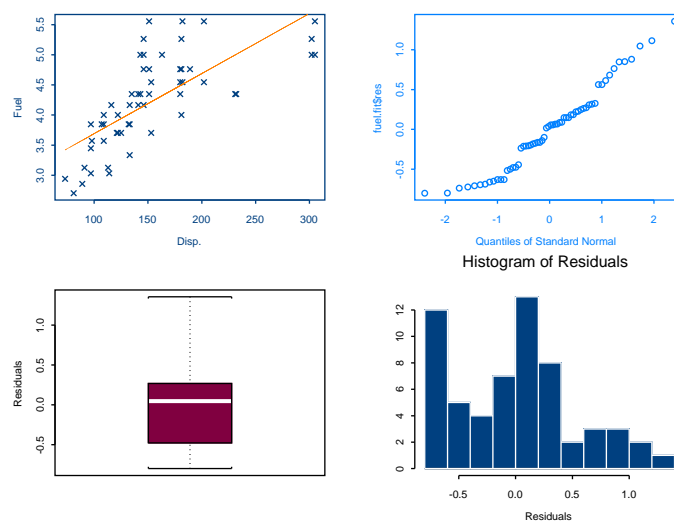


Figura 3: Exemplo de vários gráficos numa página.

- Para visualização de dados multivariados podem usar-se as seguintes funções do S-Plus: gráficos de pontos matriciais (*scatterplot matrices*, *matplots*), gráficos de estrelas (*star plots*), e gráficos de faces (*Chernoff's faces*).

Para construir um *scatterplot matrix* basta fazer o comando:

```
> pairs(longley.x)
```

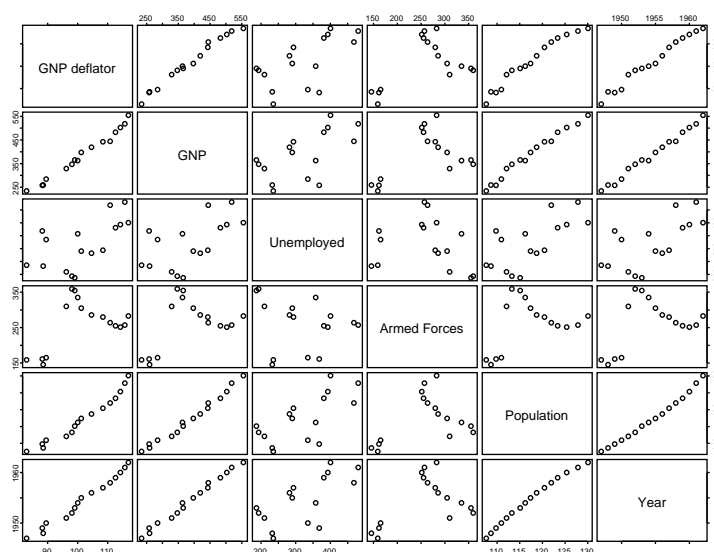


Figura 4: Exemplo de *scatterplot matrix*.

surgindo o gráfico ilustrado na Figura 4. Este tipo de gráfico ilustra a relação entre todos os pares de variáveis num conjunto de dados multivariado.

Se o objectivo é visualizar algum tipo de dados multivariados pode usar-se a função `matplot` para desenhar colunas de uma matrix versus outras colunas. Os comandos seguintes ilustram esta aplicação e o resultado pode ser visualizado na Figura 5.

```
> pet.length <- iris[,3,]
> pet.width <- iris[,4,]
> matplot(pet.length,pet.width)
```

5. Por vezes basta a função `plot` para verificar alguma estrutura nos dados. No exemplo seguinte ilustra-se a situação onde se pode verificar a existência de 4 grupos (*clusters*) distintos no conjunto de dados (veja-se a Figura 6).

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

6. Outras representações gráficas interessantes são os gráficos de estrelas e as faces/caras de Chernoff. No primeiro cada estrela representa um caso (uma linha da matriz de dados, um objecto) e cada ponto da estrela representa uma variável particular, ou coluna. Quer o tamanho, quer a forma de cada estrela

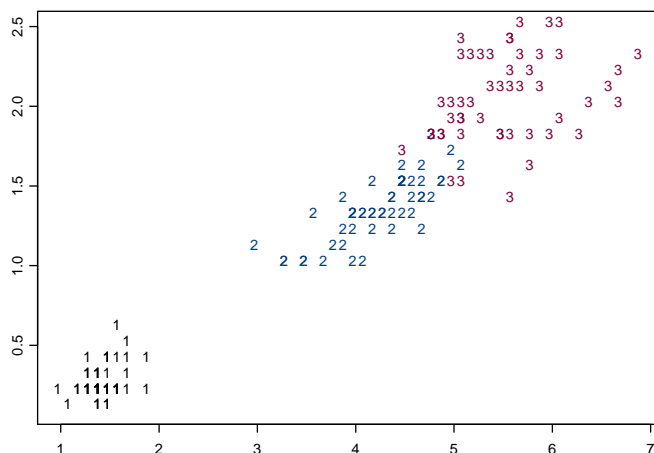


Figura 5: Exemplo de *matplot*.

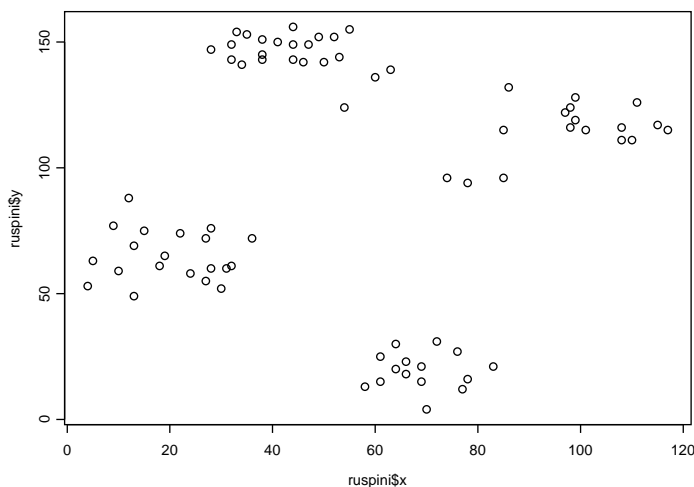


Figura 6: Exemplo de gráfico com *clusters*.

tem significado, o tamanho reflecte a magnitude total do dado e a forma revele as relações entre as variáveis. Comparando duas estrelas pode verificar-se de forma rápida as similaridades ou dissimilaridades entre dois casos (objectos) - estrelas de forma semelhante indicam casos similares. Para criar um gráfico de estrelas basta:

```
>stars(longley.x)
```

obtendo o gráfico da Figura 7.

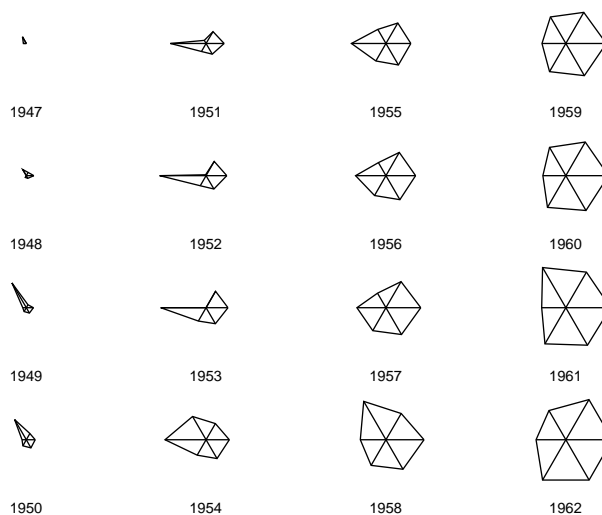


Figura 7: Exemplo de gráfico de estrelas.

As caras ou faces de Chernoff são também uma forma de representar dados multivariados. Cada variável numa dada observação é associada a uma característica da cara. Dois casos (objectos) podem ser comparados fazendo a comparação das diversas características. A função do S-Plus para criar as caras de Chernoff é:

```
>faces(t(cereal.attitude), labels =  
+ dimnames(cereal.attitude)[[2]], ncol=3)
```

E a visualização é a apresentada na Figura 8.

Escrever funções

1. Uma das mais importantes características do S é a capacidade de escrever e modificar funções para efectuar cálculos. A forma geral de uma função em S é:

```
function (arguments) expression
```

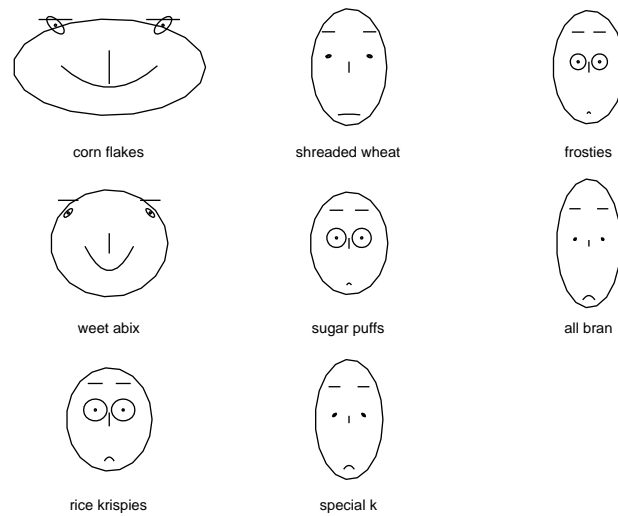


Figura 8: Exemplo de gráfico de faces de Chernoff.

Como um exemplo simples de como escrever funções em S, considere-se a função para obter-se o quadrado de um número: A nova função, a qual será denominada por `quad()`, é definida do seguinte modo:

```
> quad <-function(x) x * x
```

onde `x` é o argumento da função. Esta nova função pode ser usada agora como mais uma função de S:

```
> quad(2)
[1] 4
> m
      [,1] [,2]
[1,]  1.2  1.8
[2,]  3.5 -6.2
[3,]  4.7  5.3
> m[,1]
[1] 1.2 3.5 4.7
> quad(m[,1])
[1] 1.44 12.25 22.09
```

Note-se que a função `quad` é "vectorizável", isto é, no sentido em que também pode ser aplicada a elementos de um vector. Isso deve-se à utilização da operação aritmética "*" que já é uma operação "vectorizável".

2. A função acima usa apenas uma expressão em S. Geralmente é necessário construir funções com um número elevado de expressões S. Quando isto acontece e uma vez que não é muito conveniente escrever uma função na janela de comandos, dever-se-á editá-la numa janela adequada, **Script window**.

A função abaixo descrita e denominada `grande1()`, toma dois vectores como argumento, compara-os, e origina um vector constituído por elementos que correspondem ao máximo dos elementos das respectivas componentes dos dois vectores iniciais. Usa-se a função S, `Edit()`, para abrir numa janela o editor de *scripts*:

```
> Edit(grande1())
```

Aí escreve-se o seguinte código de expressões S que compõem a função `grande1()`.

```
grande1 <-function(x,y)
{
n <-length(x)
z <- rep(0,n)
for (i in 1:n)
{
if(y[i] > x[i]) z[i] <-y[i]
else z[i] <- x[i]
}
z
}
```

O código final da função é apenas `z`. Este código é equivalente a colocar `return(z)`. Após a edição do código deverá ser guardado num ficheiro que poderá ter o mesmo nome da função (é aconselhável para identificação posterior) ou não. Para que a função esteja disponível no ambiente S é necessário “compilá-la”, bastando pressionar a tecla **F10**. Se a função não contiver erros de sintaxe então ela fica automaticamente disponível sendo agora chamada como uma outra função qualquer de S, caso contrário deverão ser corrigidos esses erros repetindo novamente a compilação até não mostrar quaisquer erros de sintaxe. Teste-se a função acima após a criação de dois vectores, `a` e `b`:

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

```

> b<- rep(5,10)
> b
[1] 5 5 5 5 5 5 5 5 5 5
> grande1(a,b)
[1] 5 5 5 5 5 6 7 8 9 10
>

```

3. A função acima descrita usa um ciclo o qual não é um modo de cálculo muito eficiente. Veja-se como se pode usar algumas das técnicas já discutidas anteriormente para minimizar o tempo de cálculo. Reescreva-se a função anterior e denomine-se esta nova função por `grande2()` que evita o uso de ciclos:

```

grande2<-function(x,y)
{
  index <- y > x
  x[index] <- y[index]
  x
}

```

4. A função seguinte, `grande3()` é uma versão mais compacta da função `grande2()`:

```

grande3<-function(x,y)
{
  x[y > x] <- y[y > x]
  x
}

```

5. A função `grande()` usa a estrutura de controlo `ifelse` do S. Isto permite escrever a função numa simples expressão.

```

grande <-function(x,y) ifelse (y > x, y, x )

```

6. Veja-se agora um exemplo de uma função que tem como resultado uma lista. A função `polin()` calcula as duas raízes reais de uma equação de segundo grau.

```
polin <- function(a, b, c)
{
  x1 <- x2 <- NA
  d <- b^2 - 4 * a * c
  if(d < 0)
  cat("Sem raizes reais\n")
  else
  {
    x1 <- (-b + sqrt(d)) / (2 * a)
    x2 <- (-b - sqrt(d)) / (2 * a)
  }
  list(raiz1 = x1, raiz2 = x2)
}
```

Aplique-se esta função ao cálculo das raízes da equação $6x^2 - 5x + 1$. Para isso basta ir à janela de comandos e fazer:

```
> polin(6,-5,1)
$raiz1:
[1] 0.5

$raiz2:
[1] 0.3333333
```