Aulas Práticas

CE219: Controle Estatístico de Qualidade

Adilson dos Anjos

última atualização: 18 de setembro de 2008

1 Introdução

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

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

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

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

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

Alguns links são importantes para conhecer em uma primeira visita:

Download do R: existem vários *mirrors* espalhados pelo mundo. No Brasil, você pode encontrar os arquivos de instalação na UFPR, UFV e ESALQ/USP, IME/USP e FIOCRUZ.

Help pages: são muito úteis. Existe uma lista de discussão no site do **R**, onde os emails são armazenados. Você pode consultá-los através de uma busca no site. Isso é altamente recomendável se estiver pensando em mandar alguma pergunta para a lista.

Contributed documents: ótimos documentos disponíveis. Existem vários arquivos disponíveis que tratam de diferentes assuntos. A maioria está em inglês, mas existem em outros idiomas.

Special projects: algumas packages e alguns grupos de pessoas tiveram um destaque no site. São assuntos e temas específicos que, de tanto serem explorados no **R**, acabaram gerando um conjunto de pacotes e funções que estão organizados

em uma área especial. Apesar de tratar de temas bastante específicos, é uma boa fonte de informação sobre a aplicação do **R** e, claro, de estatística.

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

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

É claro, o próprio **R** possui um material de consulta bastante amplo. No **R**, o arquivo de *Help* pode ser acessado como uma página *html*.

2 Iniciando e finalizando uma sessão no R

No Linux, crie o diretório com o comando `mkdir` , entre no diretório e inicie o **R** com os comandos:

```
$ mkdir inicioR
$ cd inicioR
$ R
```

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

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

```
 getwd()
[1] "/home/adilson/documentos/LIVRO/dados"

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

O símbolo > indica que o programa está pronto para receber um comando. Um comando pode ser uma *expressão* ou uma *atribuição*.

Para finalizar uma sessão digite *q()*.

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

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

Dica: Para cada trabalho diferente, por exemplo, experimentos diferentes, você pode ter um diretório com a sessão do **R** gravada.

Quando entrar novamente você pode recuperar os comandos e objetos que foram gerados na sessão anterior. Use *Load Workspace e Load History* . Isso recupera as informações da última sessão de trabalho no **R**.

Por exemplo, se você está trabalhando em uma sessão do **R**, pode salvar seus comandos utilizando

```
> save.image("a:\minhasessao.R")
```

Para iniciar de onde você parou, utilize

```
> load("a:\minhasessao.R")
```

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

No Windows, inicie o **R**, vá em `file` e depois `new script`. Para começar a trabalhar, digite um comando e as teclas <CTRL> J. O comando será processado no **R**.

No Linux você pode utilizar um editor, como por exemplo o EMACS. Divida a janela do EMACS em dois: <CTRL> x 2. Na janela superior crie um arquivo com a extensão .R. Por exemplo, `comandos.R`. Na janela inferior digite <ESC> <SHIFT> x R. Será perguntado em qual diretório você quer trabalhar. Para facilitar, escolha um diretório onde seus dados esteja armazenados.

Para começar a trabalhar, digite um comando na janela superior e pressione <CTRL> C <CTRL> J. O resultado será processado na janela inferior.

2.1 Início do trabalho

O **R** trabalha com programação orientada à objetos .

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

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

```
> a
[1] 2
```

O número [1] indica a posição do algarismo.

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

```
> A<-3
> A
[1] 3
> a
[1] 2
```

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

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

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

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

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

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

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

Se isso ocorrer, digite o que estiver faltando ou cancele o comando com *ESC*.

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

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

```
> a<-6
> c<-a/(b<-2);c
[1] 3
```

2.2 Linha do Editor

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

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

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

3 O que tenho armazenado

Para ver os objetos armazenados use

```
> ls()
```

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

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

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

Os comandos são armazenados no arquivo *.Rhistory*.

4 O R como calculadora

Nas linhas de comando você pode executar operações como:

```
> 2+2
[1] 4

> 2*3
[1] 6

> 4/2
[1] 2

> 2^2
[1] 4
```

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

```
> sqrt(4)  # raiz quadrada
[1] 2


> sin(x)
[1] 0.9092974
```

Algumas aplicações com vetores (será discutido com mais detalhes na seção 12):

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

> a+b
[1] 3 6 9

> a*b
[1]  2  8 18

> a/b
[1] 2 2 2

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

> y*x
[1]  8 16 20

> y/x
[1] 2 4 5
```

Outros comandos sobre vetores

```
> sum(y)
[1] 22

> length(y)
[1] 3

> media<-sum(y)/length(y)
> media
[1] 7.333333
```

5 Operadores lógicos

Os operadores usuais são $<, <=, >, >=, ==$. O operador $!=$ representa *diferente ou negativa*. Ainda, se a e b são duas expressões, pode-se utilizar o operador & para representar interseção e o operador | para representar união. O símbolo ! representa negação.

Quando uma condição é fornecida, o **R** informa se ela é verdadeira (TRUE) ou falsa (FALSE) ou ainda NA (not available).

Por exemplo,

```
> a<-c(1,2,3,4,5)
> a>3
[1] FALSE FALSE FALSE  TRUE  TRUE
> a!=2
[1]  TRUE FALSE  TRUE  TRUE  TRUE

> b<-c(1,2,3,5,5)
> (b>2) & (b==5)
[1] FALSE FALSE FALSE  TRUE  TRUE

> which(b>3)
[1] 4 5
```

6 Como obter ajuda no R

Como o software é baseado em comandos, você dificilmente conseguirá gravar tudo. De vez em quando precisará consultar os arquivos de ajuda para sanar suas dúvidas.

Ainda, os comandos executados em uma análise poderão ser salvos para serem executados novamente com os mesmos dados ou com dados diferentes. Basta salvá-los em um arquivo.

Existem várias formas de obter ajuda no **R**. Na linha de comando você pode

digitar

```
> help(anova)
```

ou

```
> ?anova
```

O arquivo *help* padrão possui:

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

Ainda é possível fazer uma busca usando

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

ou

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

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

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

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

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

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

7 Demos

O **R**possui algums pacotes de demonstração, que apresentam funcionalidades do software. Por exemplo, digite o comando a seguir e pressione a tecla *ENTER*.

```
> demo(graphics)
```

Digitando apenas

```
> demo()
```

aparecerá uma lista de outras Demos que podem ser visualizadas. Experimente!

8 Tipos de Objetos

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

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

Matrizes: São um generalização de vetores. Os elementos são indexados por índices (linhas e colunas) e são apresentados na forma de uma matriz;

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

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

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

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

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

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

ou simplesmente `class(x)`

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

> is.numeric(x)
[1] TRUE

> class(x)
[1] "numeric"
```

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

Alguns objetos também podem ser convertido para outro com o comando correspondente, por exemplo,

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

> class(x)
[1] "factor"
```

```
> is.numeric(x)
[1] FALSE
```

9 Gerando sequências

Existem várias maneiras de gerar sequências no **R**. Por exemplo,

Gerar a sequência de 1 até 20,

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

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

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

```
> seq(0,10, by = 2)
[1]  0  2  4  6  8 10
```

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

```
> seq(length=12,from=0,by=3)
 [1]  0  3  6  9 12 15 18 21 24 27 30 33
> seq(0,10,length=11)
 [1]  0  1  2  3  4  5  6  7  8  9 10
> seq(0,20,length=11)
 [1]  0  2  4  6  8 10 12 14 16 18 20
```

Usando a função `rep`.

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

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

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

> rep("r",5)
[1] "r" "r" "r" "r" "r"
```

A função `rep` gera um vetor numérico ou alfanumérico. Para ser utilizado em um `data.frame` de um experimento representando um fator, é necessário que se use a classe `factor` utilizando `as.factor()` sobre um vetor existente ou `gl()`.

Para criar um fator, a função `gl` pode ser bastante útil, pois já é criado um vetor da classe *factor*. Veja alguns exemplos:

```
> A<-gl(4,2);A
[1] 1 1 2 2 3 3 4 4
Levels: 1 2 3 4

> class(A)
[1] "factor"
> levels(A)
[1] "1" "2" "3" "4"

> gl(2,4)
[1] 1 1 1 1 2 2 2 2
Levels: 1 2

> gl(3,3,leng=18)
 [1] 1 1 1 2 2 2 3 3 3 1 1 1 2 2 2 3 3 3
Levels: 1 2 3
```

10 Fornecendo nomes

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

```
> notas<-c(7,5,8.5,9)
> names(notas)<-c("João","Pedro","Paulo","José")
> notas
 Joao Pedro Paulo  José
  7.0   5.0   8.5   9.0
> aprovados<-notas>=7
> aprovados
 João Pedro Paulo  José
 TRUE FALSE  TRUE  TRUE
```

11 Gráficos

O **R** possui um grande número de possibilidades gráficas. Por exemplo, a figura de abertura da página do projeto www.r-project.org é um conjunto de vários gráficos construídos no próprio **R**.

Ao longo do livro veremos alguns tipos de gráficos. Nesse momento, veremos como gerar gráficos e como definir alguns parâmetros.

Um gráfico simples pode ser gerado da seguinte maneira:

```
> x<-1:10
> y<-sqrt(x)
> plot(x,y)
```

Toda vez que um gráfico é gerado, uma *janela gráfica é aberta*, exceto quando os comandos salvar do tipo `postscript()` e `pdf()`, por exemplo, são inseridos. Os gráficos podem ser salvos em diferentes formatos. Com o gráfico aberto, basta clicar com o botão direito do mouse (Windows) em cima do gráfico e escolher salvar como: .bmp, .jpeg, .ps, .wmf, png, pdf.

Esse gráfico pode ser modificado acrescentando-se textos e outros elementos, como por exemplo:

```
> plot(x,y,main="Título")
> plot(x,y,main="Título \n inclusive com acento")
> plot(x,y,main="Título",sub="sub título")
> plot(x,y,main="Título",xlab="Eixo x",ylab="Eixo y")
> plot(x,y,main="Título",xlab="Eixo x",ylab="Eixo y",type="l")
> plot(x,y,main="Título",xlab="Eixo x",ylab="Eixo y",type="l",col=2)
> plot(x,y,main="Título",xlab="Eixo x",ylab="Eixo y",type="l",col=2,
  axes=F)
> plot(x,y,main="Título",xlab="Eixo x",ylab="Eixo y",type="b",col=2)
```

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

Você pode acrescentar pontos e linhas em um gráfico,

```
> plot(x,y)
> points(rev(x),y))
> lines(x,3-y)
> lines(x,4-y)
> lines(x,5-y)
```

Linhas também podem ser adicionadas com o comando `abline`,

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

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

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

O argumento `cex=`, aumenta ou diminui o tamanho dos caracteres gráficos. O default é 1.

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

A função `locator()` serve para identificar pontos no gráfico

```
> locator(n=1) # identificando um ponto
> identify(x,y) # identificando vários pontos
```

Clique em um ponto do gráfico! Para terminar, clique com o botão direito do mouse no Linux e no Windows, clique com o botão direito e selecione 'stop'.

Inserindo uma legenda ,

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

```
> plot(x,y,type="l")
> lines(x,y+1,col=2)
> legend(15,5,c("linha 1","linha 2"),pch=19 ,col=1:2,cex=1.5)
```

Inserindo um texto no meio do gráfico

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

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

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

```
> par()
```

para saber qual a configuração de sua janela gráfica.

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

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

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

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

Como salvar um gráfico :

No Windows,

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

No Linux e também no Windows,

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

Nessa opção o gráfico não é mostrado.

12 Trabalhando com números e vetores

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

```
> x.n<-c(89,65,76.5)
> x.l<-c('a','b','c')
> x.l<-letters[1:3]
> x.L<-LETTERS[4:5]
```

outras formas equivalentes:

```
> assign ("x.n",c(89,65,76.5))
> c('a','b','c')->x.l
```

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

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

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

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

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

12.1 Selecionando um subconjunto de um vetor

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

```
>  x<-1:10
>  y<-x[5:10]
>  x; y
 [1]  1  2  3  4  5  6  7  8  9 10
 [1]  5  6  7  8  9 10
> z<-y[-3];z #retira a terceira observação de y
[1]  5  6  8  9 10
> z[1]<-77
> z
```

```
[1] 77  6  8  9 10
> a<-1:5
> c<-a[a>2]
> c
[1] 3 4 5
```

13 Listas e Data frames

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

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

$civil
[1] "casado"

$cachorros
[1] 2
```

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

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

```
> dados[3]
$cachorros
[1] 2

> dados[[3]]
[1] 2

> dados$cachorro
[1] 2
```

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

```
> length(dados)
[1] 3
```

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

```
> dados$n
> dados$c
NULL
> dados$ca
[1] 2
```

13.1 Como alterar valores da lista

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

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

14 Data frame

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

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

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

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

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

```
> search()
```

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

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

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

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

```
> nome
```

Cuidado!

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

16 Matrizes

Matrizes no **R** são tratadas com um array bidimensional, ou seja, linhas e colunas. Uma matriz pode ser criada da seguinte forma:

```
> a<-matrix(1:9,3,3)
> a
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

ou ainda,

```
> matrix(1:12,nrow=3,byrow=T)
```

Pode-se também, juntar vetores de duas formas para formar uma matriz: em colunas, usando a função `cbind` e em linhas usando a função`rbind`:

```
> A<-1:3
> B<-4:6
> C<-7:9
> matrix(cbind(A,B,C),3)
> matrix(rbind(A,B,C),3)
```

Os elementos de uma matriz são selecionados fornecendo a posição do elemento em relação as linhas e colunas.

```
> a[1:3,1]
> a[1,]
```

16.1 Operações com matrizes

As principais operações com matrizes são:

Transposição:

```
> A<-matrix(1:9,3,3)
> B<-matrix(1:9,3,3)
> t(B)
```

Soma e multiplicação:

```
> A+B
> A*B
> A%*%B #elememto por elemento
```

Inversa de matriz:

```
> solve(A) # inversa clássica
> ginv(A) # inversa generalizada (Moore Penrose)
```

A função `solve` pode ser usada para a resolução de sistemas de equações.

Autovalores e autovetores:

```
> eigen(A)
```

17 Obtendo dados de outros "pacotes" (packages)

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

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

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

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

Para ter informações sobre o arquivo de dados use

```
> ?cats
```

18 Lendo dados com read.table()

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

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

Para ver o arquivo basta digitar `dados`.

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

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

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

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

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

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

19 Folha de verificação

Objetivo

O objetivo desta seção é construir tabelas de freqüências para observações de um processo.

A tabela de freqüências corresponde à folha de verificação do processo ou serviço.

Arquivo de dados

Serão utilizados os dados sobre o tempo de parada em um linha de produção: arquivo.

```
> linha<-read.table("linha.txt",header=T)
> linha
```

Estes dados correspondem ao tempo de parada, em minutos, de uma linha de produção. A coluna *cod* indica qual o tipo de problema ocorrido. Os dados foram obtidos em 4 meses de estudo em diferentes turnos e dias.

Folha de verificação

Nesse caso, um resumo dos dados fornece informações sobre o comportamento das observações realizadas durante o período.

Uma análise das freqüências mensais pode ser obtida da seguinte maneira:

```
> mes.tab<-table(linha$cod,linha$mes,dnn=c('cod','mes'))
> mes.tab
```

Esse resultado fornece algumas informações sobre a freqüência de ocorrência dos problemas.

A função `apply()`, mostras os totais da freqüência de ocorrência de cada tipo de problema ou as freqüências mensais:

```
> apply(mes.tab,1,sum)
  1   2   5   6   7   8   9  31  32  33  34  35  36  38  41  43
  7   2   5  71   4  99   2 156   2  38  69 108  16   8  79  16

 44  81  82  84 321 322 421
  9   4  35   5  2  52  99
```

```
> apply(mes.tab,2,sum)
  2   3   4   5
165 279 212 232
```

Observe que alguns códigos *8, 31, 105 e 421*, destacam-se dos demais.

Os mesmos resultados podem ser obtidos com o uso da função `margin.table()`:

```
> margin.table(mes.tab,1) # freq. de paradas por código
> margin.table(mes.tab,2) # freq. de paradas por mês
```

ou simplesmente

```
> table(linha$cod) #conta o num. de paradas por código
```

Observe que nestas análises, não foi considerado o tempo gasto com cada parada.

Exercícios

1. Algum problema ocorre com maior freqüência em algum turno?
2. Os tempos de parada são semelhantes entre turnos?
3. No arquivo é apresentado um conjunto de dados referentes à ocorrências de acidentes com automóveis na cidade de Curitiba no primeiro semestre de 2000. Explore esse conjunto de dados e tente identificar informações que possam ajudar no controle de acidentes.

   Por exemplo, utilizando a função `ftable()` pode-se obter uma tabela com três entradas. Pode-se ter o interesse em saber o número de acidentes por bairro, dia da semana e horário.

   ```
   > tab.1<-ftable(carros[c("bairro","semana","hora")])
   ```

   O valor máximo na tabela pode ser encontrado coma função `max()`.

   ```
   > max(tab.1)
   ```

Como a tabela é extensa, o valor encontrado pode não ser único. Com a função `sort()` obtém-se os valores de contagens ordenados de forma crescente. Nesse caso, o alor encontrado é único.

```
> sort(tab.1)
```

Observe que em muitos bairros a ocorrência de acidentes é pequena. Para diminuir o banco de dados, pode-se fazer a seleção de bairros com maior ocorrência de acidentes, por exemplo.

20 Gráfico de Pareto

Objetivo

O objetivo dessa sessão é construir o gráfico de Pareto.

Arquivo de dados

Utilizaremos, inicialmente, o exemplo das notas de aula:

```
> problemas<-c(13,4,3,2)
> names(problemas)<-c('A','B','C','D')
```

Em seguida, trabalharemos com os dados de tempo de paradas em uma linha de produção, utilizado para exemplificar a folha de verificação.

Gráfico de Pareto

A função que constrói o gráfico de Pareto está no pacote `qcc`. Por isso, antes de começar as análises, deve-se 'carregar' esse pacote.

```
> require(qcc)
```

Nesse pacote, a função `pareto.chart()` constrói o gráfico:

```
> pareto.chart(problemas,col=rainbow(length(problemas)))
```

Se desejar, é possível acrescentar sobre cada barra, as freqüências de cada problema. Execute o comando a seguir, vá até o gráfico e 'clique' acima de cada barra. Os números aparecerão somente após o quarto e último 'clique'.

```
> text(locator(4),c('13','4','3','2'),cex=1.3)
```

Agora, vamos trabalhar com os dados sobre tempo de parada de uma linha de produção.

Considerando todo o período de avaliação, o gráfico de pareto pode ser construído da seguinte maneira:

```
> codigo<-table(linha$cod)
> pareto.chart(codigo,col=rainbow(length(codigo)))
```

Nesse gráfico, o código 31 possui a maior freqüência entre todos os outros.

Considerando apenas o mês 4, teríamos:

```
> codigo.4<-table(linha$cod[which(linha$mes==4)])
> codigo.4<-table(linha$cod[linha$mes==4]) # outra maneira
> pareto.chart(codigo.4)
```

O código 31 ainda apresenta a maior freqüência.

Ainda, pode ser de interesse estudar como se comportam os códigos de parada em cada turno de trabalho. Primeiro, obtém-se os dados para construção dos dados:

```
> turno.1<-table(linha$cod[linha$turno==1])
> turno.2<-table(linha$cod[linha$turno==2])
> turno.3<-table(linha$cod[linha$turno==3])
```

Em seguida, os gráficos de Pareto podem ser obtidos:

```
> old.par<-par() # guardando a config. original
> par(mfrow=c(3,1),mar=c(3,4,3,2)) # alterando margens
```

```
> pareto.chart(turno.1,cex.names=1.35)
> pareto.chart(turno.2,cex.names=1.35)
> pareto.chart(turno.3,cex.names=1.35)

>par(old.par) # retorna para a config. original
```

Veja, por exemplo, que o código 421 possui valores distintos entre os turnos. Tente observar outras mudanças!

Considerando a variável tempo na análise dos dados, pode-se tentar concluir sobre qual problema deve-se começar a atuar. Um simples gráfico de barras para com o tempo total de parada para cada código, pode fornecer essa informação:

```
> x11() # para abrir nova janela gráfica
> barplot(tapply(linha$tempo,linha$cod,sum))
```

Observe que, por exemplo, o código 41 possui uma freqüência menor do que o código 34 mas, o tempo gasto com paradas é muito maior do que o código 34.

Ainda, observe que existem alguns pontos com valores maiores no código 41 em relação ao código 34. Na análise desse problema, isso deve ser levado em consideração antes de uma decisão sobre qual código ou quais códigos serão investigados inicialmente.

```
> x11()
> plot(linha$tempo[linha$cod==c(31,34,41)],
       linha$cod[linha$cod==c(31,34,41)])
```

Exercícios

1. Com os dados dessa seção, procure estudá-los de outras maneiras;

21 Diagrama de Causa e efeito

Objetivo

O objetivo dessa seção é construir um diagrama de causa-e-efeito.

Construção do diagrama

O diagrama de causa-e-efeito é uma função que está dentro do pacote `qcc`. Antes de iniciar, o pacote deve estar instalado.

A função chama-se `cause.and.effect()`. Para construção do diagrama é necessário fornecer uma lista de causas primárias com as respectivas causas secundárias.

```
cause.and.effect(cause=list(Medição=c("Microscópio", "Paquímetro"),
    'Matéria prima'=c("Fornecedores", "Lubrificantes"),
    'Mão de obra'=c("Supervisores", "Treinamento", "Operadores"),
    'Meio ambiente'=c("Temperatura", "Umidade"),
    Métodos=c("Velocidade", "Pressão"),
    Máquinas=c("Tempo de uso", "Marcas")),
    effect="Defeito",title = "Diagrama de Causa e Efeito",
    cex = c(1, 0.9, 1), font = c(1, 3, 2))
```

Para causas primárias com dois ou mais nomes, por exemplo, matéria prima, deve-se utilizar aspas.

22 Gráfico de dispersão

Objetivo

O objetivo dessa seção é utilizar o gráfico de dispersão para identificar possíveis relações de causa e efeito.

Arquivo de dados

Serão utilizados os dados sobre o tempo de parada em um linha de produção: arquivo.

```
> linha<-read.table("linha.txt",header=T)
> linha
```

Gráfico de dispersão

Para fazer um gráfico de dispersão, utiliza-se a função `plot(x,y)`.

Para os dados de tempo de parada da linha, pode-se estar interessado em avaliar se existe alguma relação entre o tempo de parada e turno de trabalho.

```
> attach(linha)
> plot(turno,tempo)
```

Nesse gráfico, observa-se que não há uma estreita relação entre o turno de produção e tempo de parada. Em todos os turnos, o comportamento dos tempos de para possui o mesmo comportamento, considerando todos os códigos de parada.

Pode-se, também, investigar o comportamento dos tempos de parada para cada código. Um gráfico de dispersão entre códigos e tempo de parada mostra o comportamento desses dados.

```
> plot(cod,tempo)
```

No gráfico, aparecem códigos que possuem uma dispersão maior do que outros, além de as freqüências serem diferentes. Visualmente, por existirem muitos códigos de parada, o gráfico não ficou claro. Por isso, pode-se estudá-lo em partes. Por exemplo, visualizando apenas os códigos menores do que 100.

```
> plot(cod[which(cod<100)],tempo[which(cod<100)],cex=.2,pch=19)
```

Pode-se subdividir mais. Vamos experimentar estudar os códigos de 20 a 40.

```
plot(cod[which(cod<=40 & cod>=20)], tempo[which(cod<=40&cod>=20)],
cex=.2,pch=19)
```

Pelo gráfico, pode-se perceber que o código 31 apresenta uma grande freqüência e com valores altos de tempo de parada. Para outros códigos, além da freqüência, o tempo total de parada é uma informação importante para descobrir qual código está causando mais tempo de parada.

```
> tapply(tempo[which(cod<=40 & cod>=20)],
  cod[which(cod<=40 & cod>=20)], sum)
  31   32   33   34   35   36   38
5473  190  326 1081  737  396   47
```

Observe que os códigos 34 e 35 são os que possuem os maiores tempos totais de parada no período estudado.

O comnado `pairs()` faz todos os possíveis gráficos de dispersão entre as colunas de um `data.frame`.

```
> pairs(linha,cex=.3,pch=19)
```

Analise os gráficos!

23 Gráfico de tendência

Objetivo

O objetivo dessa seção é construir um gráfico de tendência. Um gráfico de tendência fornece informações sobre a existência ou não de sazonalidade ou períodos com comportamentos específicos.

Arquivo de dados

Continuaremos a estudar os dados sobre tempo de parada em uma linha de produção.

Gráfico de tendência

Podemos considerar um estudo do código de parada 31.

```
> cod.31<-subset(linha,cod==31,select=c(names(linha)));cod.31
> cod.31$obs<-1:length(cod.31$tempo)
```

Um gráfico de tendência pode ser construído com o uso da função `plot()`.

```
> plot(cod.31$obs,cod.31$tempo,typ="b",pch=19,cex=.7)
> abline(h=mean(cod.31$tempo))
```

Observe que, exite uma variação entre as observações. Em dois períodos há paradas com um tempo maior do que as outras.

24 Métodos para descrever a variação

Objetivo

O objetivo desta seção é apresentar métodos para descrever a variação de dados. Existem muitas alternativas para o estudo da variabilidade de dados. Além de gráficos e medidas como variância e desvio padrão, podem ser aplicados testes para comparação de variâncias, por exemplo.

Arquivo de dados

Ainda os dados sobre tempo de parada da linha de produção.

Em particular, será estudado o código 31 nesse capítulo

```
> cod.31<-subset(linha,cod==31,select=c(names(linha)));cod.31
```

Avaliação gráfica da variação

Um gráfico de ramo-e-folhas:

```
> stem(cod.31$tempo,scale=2)

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

   0 | 1122222233333344445556666667777788999 9
   1 | 000000012222233344444555566677788888
   2 | 00000011122333344455566788999
   3 | 00223378
   4 | 0133445677789
```

```
  5 | 2377778
  6 | 0124578
  7 | 134467
  8 | 9
  9 | 7
 10 | 3
 11 | 55
 12 | 4
 13 |
 14 |
 15 | 0
 16 | 13
 17 | 3
 18 |
 19 | 8
 20 |
 21 |
 22 |
 23 | 9
 24 |
 25 | 9
```

Observe que, a distribuição é bastante assimétrica. Existem muitas paradas com até uma hora de duração e poucas com mais de uma hora de duração.

Um histograma:

```
> hist(cod.31$tempo,nclass=10,col=rainbow(13))
```

Se for de interesse, pode-se traçar a linha de densidade dos dados.

```
> hist(cod.31$tempo,prob=T,nclass=10,ylim=c(0,0.035))
> lines(density(cod.31$tempo) ) # insere a linha
> rug(cod.31$tempo) # insere uma barra com freq. de pontos
```

Um box-plot:

```
> boxplot(cod.31$tempo)
```

A função `boxplot.stats()` fornece informações sobre dados do box plot. Especificamente, `$out` mostra as observações que podem ser consideradas candidatos a outlier (dados discrepantes).

```
> boxplot.stats(cod.31$tempo)
$stats
[1]  1.0 10.0 20.0 45.5 97.0

$n
[1] 156

$conf
[1] 15.50921 24.49079

$out
 [1] 198 163 115 103 124 239 161 173 115 259 150
```

Dados discrepantes podem fornecer muita informação. Por exemplo, o tempo de parada registrado para esse código, pode estar tendo outras causas que levam a esses valores discrepantes. Obviamente, é necessário checar se esses dados foram obtidos de forma correta.

Algumas medidas de variação

Além de gráficos, existem várias medidas para analisar a variabilidade.

1. Variância

   A variância, definida por $\sigma^2 = \frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n-1}$, pode ser obtida com o comando `var()`:

   ```
   > var(tempo)
   [1] 2616.395
   ```

   ```
   > tapply(tempo,cod,var)
             1           2           5           6           7           8
    2557.476190   32.000000 50922.800000 2377.659960 35942.916667   646.598227
             9          31          32          33          34          35
   28084.500000 1866.541398 1800.000000  151.439545  174.078431   12.370630
            36          38          41          43          44          81
   ```

```
303.000000     6.982143  2192.457319  7910.650000   174.500000  3326.000000
        82           84           321           322          421
 14.596639     6.200000     2.000000   818.719080    47.233766
```

2. Desvio padrão

   O desvio padrão, definido como $\sigma = \sqrt{s^2}$ pode ser obtido com o comando `sd`:

```
> sd(tempo)
[1] 51.1507
```

```
> tapply(tempo,cod,sd)
         1          2          5          6          7          8          9
50.571496   5.656854 225.660807  48.761255 189.586172  25.428296 167.584307
        31         32         33         34         35         36         38
43.203488  42.426407  12.306078  13.193879   3.517191  17.406895   2.642374
        41         43         44         81         82         84        321
46.823683  88.941835  13.209845  57.671483   3.820555   2.489980   1.414214
       322        421
28.613268   6.872683
```

3. Coeficiente de variação

   O coeficiente de variação, definido como $CV\% = \frac{s}{\bar{x}}100$, pode ser utilizado para avaliar a variabilidade dos dados. Em geral, é útil para comparar medidas semelhantes.

```
> cod.31<-subset(linha,cod==31,select=c(names(linha)));cod.31
> cod.34<-subset(linha,cod==34,select=c(names(linha)));cod.34
> cv.31<-(sd(cod.31$tempo)/mean(cod.31$tempo))*100;cv.31
[1] 123.1453
> cv.34<-(sd(cod.34$tempo)/mean(cod.34$tempo))*100;cv.34
[1] 84.21625
```

   Nesse exemplo, existe uma maior precisão na variável tempo 34 em relação ao código 31.

Testes sobre variâncias

Além de estatísticas descritivas, pode-se realizar testes de hipóteses para estudar as variâncias.

Por exemplo, pode ser de interesse testar variâncias com uma valor de referência ou ou comparar variâncias entre processos.

No primeiro caso, pode-se utilizar um teste de qui-quadrado para testar a hipótese

$$H_0: \quad \sigma^2 = \sigma_R^2$$
$$H_1: \quad \sigma^2 \neq \sigma_R^2$$

σ_R^2 é o valor da variância referencial.

A estatística de teste é dada por $\chi^2 = \frac{(n-1)s^2}{\sigma_R^2}$

Como exemplo, pode-se ter interesse em comparar a variância do código 1 em relação a variância total da variável tempo.

```
> var(tempo)
[1] 2616.395

> var(tempo[cod==1])
[1] 2557.476
```

Utilizando o teste de qui-quadrado,

```
> var(tempo)
[1] 2616.395
> var(tempo[cod==1])
[1] 2557.476
> length(tempo[cod==1])
[1] 7

> qui<-((length(tempo[cod==1])-1)*var(tempo[cod==1]))/var(tempo);qui
[1] 5.864887

> qchisq(.95,length(tempo[cod==1])-1)
[1] 12.59159

> 1-pchisq(qui,length(tempo[cod==1])-1)
[1] 0.4384947
```

Neste exemplo, não rejeita-se a hipótese nula, ou seja, não há diferença significativa entre a variância do código 1 em relação a variância total.

Se for de interesse testar a hipótese

$$H_0: \ \sigma_1^2 = \sigma_2^2$$
$$H_1: \ \sigma_1^2 \neq \sigma_2^2$$

pode-se utilizar a estatística de teste baseada na razão de variâncias,

$$F = \frac{\hat{\sigma}_1^2}{\hat{\sigma}_2^2}$$

Por exemplo, pode-se ter interesse comparar as variâncias do código 1 e do código 31.

```
> var.test(tempo[cod==1],tempo[cod==31])

F test to compare two variances

data:  tempo[cod == 1] and tempo[cod == 31]
F = 1.3702, num df = 6, denom df = 155, p-value = 0.4599
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.5500939 6.7029025
sample estimates:
ratio of variances
          1.370168
```

Portanto, não rejeita-se a hipótese nula, ou seja, não há evidências para considerar que as variâncias dessas duas amostras diferem.

Exercícios

1. Construa um boxplot para cada um dos códigos de parada. Analise a variabilidade.

2. Escolha um outro código de parada e compare com os resultados estudados nessa seção. Utilize as medidas de variação apresentadas, além da análise gráfica;

3. Estude o comportamento de variação dos turnos de produção. Compare os turnos entre si com relação à variação;

25 Curva Característica de Operação

Objetivo

O objetivo dessa aula é construir e utilizar uma curva característica de operação (CCO).

Erro tipo II

Para construir a CCO, inicialmente precisamos calcular o erro tipo II. Como visto na teoria, podemos criar no **R** um função para calcular o erro tipo II ou β.

```
beta<-function(sigma,delta,n,alpha=0.05)
                {
  z<-qnorm(alpha/2,0,1)
  phi.1= (abs(z)-(((delta*sqrt(n))/sigma)))
  phi.2=     (z-(((delta*sqrt(n))/sigma)))
  phi.1;phi.2
  beta<-pnorm(phi.1)- pnorm(phi.2)
  print(round(beta,4))
}
```

Para utilizar essa função, apenas utilize:

```
> beta(sigma=3,delta=3,n=16) # do exemplo
[1] 0.0207
```

Lembrando que, $1 - \beta$ fornece o poder do teste.

Construindo a CCO

A função `cco.l()`, a seguir, retorna a CCO entre dois valores de n especificados com $\alpha = 0,05$.

```
> cco.l<-function(n.min=10,n.max=12,alpha=0.05)
  {
   z<-qnorm(alpha/2,0,1)
   delta.x<-seq(0,5,length=100)
   beta.y<-seq(0,1,length=100)
   phi.1<-numeric()
   phi.2<-numeric()
   beta <-numeric()
   plot(delta.x,beta.y,type="n")
   abline(h=0)
   for(j in n.min:n.max)
   {
        for(i in 1:length(beta.y))
          {
           phi.1[i]<- (abs(z)-(delta.x[i]*sqrt(j)));
           phi.2[i]<- (z     -(delta.x[i]*sqrt(j)))
          }
     beta<-pnorm(phi.1)-pnorm(phi.2)
     lines(delta.x,beta)
    }
   }
```

Um função, um pouco diferente da anterior, pode ser construída para fornecer as curvas de uma maneira geral. Bastando, apenas, fornecer α e n.

```
cco<- function(alpha=.05, n=2)
   {
   z<-qnorm(alpha/2,0,1)
   delta.x<-seq(0,5,length=100)
   beta.y<-seq(0,1,length=100)
   phi.1<-numeric()
   phi.2<-numeric()
```

```
   beta <-numeric()
   plot(delta.x,beta.y,type="n",xlab="d",ylab=expression(beta))
   abline(h=0)
   for(j in 2:n)
   {
        for(i in 1:length(beta.y))
          {
           phi.1[i]<- (abs(z)-(delta.x[i]*sqrt(j)));
           phi.2[i]<- (z     -(delta.x[i]*sqrt(j)))
           # phi.1; phi.2
          }
     beta<-pnorm(phi.1)-pnorm(phi.2)
     #colorido
     #lines(delta.x,beta,lty=j,col=j)
     #legend(4,.8,leg=2:n,lty=2:n,pch=20,col=2:n,cex=1.2,
     title="n",pt.cex=1.0,ncol=2,pt.lwd=1.3)

     #P&B
     legend(4,.8,leg=2:n,lty=2:n,pch=20,cex=1.2,title="n",pt.cex=1.0,
     ncol=2,pt.lwd=1.3)
     lines(delta.x,beta,lty=j)
    }
 }
```

Após construir a CCO, pode-se encontrar o tamanho de amostra n. Deve-se, então, fornecer o valor de δ e o valor de β

```
> points(2.0,.10,pch=4) # identificando pontos
```

26 Sugestão para exercícios

1. O conjunto de dados apresentado (aqui), refere-se a um estudo sobre problemas encontrados em um fábrica. Durante 10 meses, 6 operadores foram monitorados. Em cada mês, foram obtidas 15 amostras de problemas na fabricação e o custo (em reais) de cada problema encontrado foi calculado.

   O operador 1 é mais treinado do que o operador 2, e este, mais treinado do que o operador 3, e assim sucessivamente.

(a) Indique para a fábrica, qual problema pode ser o mais importante;

(b) Existe algum efeito sazonal nesses dados?

(c) O treinamento pode ser considerado um fator importante para redução de custos?

(d) Algum operador deveria receber mais treinamento para evitar algum problema?

(e) Pode-se dizer, para esses dados, que, quanto mais treinado o operador, menor será a variabilidade dos custos dos problemas?

(f) Sabe-se que o custo médio mensal de cada problema é de R$ 20,00, com uma variância igual a 10. Algum operador possui uma variabilidade diferente da informada?

2. Para um custo médio ao redor de R$ 20,00/problema a fábrica considera uma situação aceitável. Mas, se o custo passar de R$ 25,00/problema, em média, isso torna-se inaceitável. Considerando uma probabilidade de 95%, qual deveria ser o tamanho da amostra para que a fábrica rejeite $H_0 : \mu = 20,00$ quando na verdade a média é $\mu = 25,00$?

27 Gráficos de controle para variáveis

Objetivos

O objetivo desta seção é construir e interpretar gráficos de controle para variáveis. Procure entender os comandos do **R** e interpretar os resultados.

Gráficos de controle para $\bar{x}$ e R

Utilizaremos os dados do exemplo sobre 'anéis de pistão' (Montgomery, 1997), disponível no pacote `qcc`.

```
> require(qcc)
> data(pistonrings)
> attach(pistonrings)
> pistonrings
```

Esse conjunto de dados corresponde a avaliação dos diâmetros dos anéis, obtidos de 40 amostras de tamanho n=5.

As primeiras 25 amostras (trial=TRUE) foram utilizadas para a obtenção do gráfico de controle e as amostras subseqüentes (trial=FALSE) foram utilizadas para monitoramento do processo.

Inicialmente pode-se fazer uma análise exploratória dos dados para conhecer o comportamento da variável resposta.

```
> summary(pistonrings)
> boxplot(diameter ~ sample)
> plot(sample[trial==TRUE], diameter[trial==TRUE], cex=0.7)
> plot(sample, diameter, cex=0.7)
> lines(tapply(diameter,sample,mean))
```

Para utilizar a função `qcc`, é necessário que o objeto a ser analisado esteja no formato exigido pela função. Por isso, utiliza-se a outra função chamada de `qcc.groups` para formatar os dados.

```
> diametro<-qcc.groups(diameter,sample)
> diametro
> class(diametro)
```

Observe que o objeto `diametro` é uma matriz. A função `qcc` reconhece, automaticamente, que o número de colunas é o tamanho de cada amostra.

Gráfico $\bar{x}$

Para fazer o gráfico de controle para $\bar{x}$ utilizando as primeiras 25 amostras utilizamos os seguintes comandos:

```
> qcc(diametro[1:25,],type='xbar')
```

Observe que, para esse conjunto de dados, não há violação de nenhuma regra. Ou seja, nesse caso, diz-se que o processo está sob controle.

Uma vez construído o gráfico de controle para o processo, pode-se monitorar novas amostras, apenas acrescentado os novos dados no gráfico já construído.

```
> qcc(diametro[1:25,],type='xbar',newdata=diametro[26:40,])
```

Observa-se, agora, que algumas observações estão acima do limite de controle superior, indicando que o processo deve estar fora de controle.

Causas especiais de variação devem estar ocorrendo nesse processo, fazendo com que o diâmetro esteja muito acima da média. Nesse caso, ações corretivas devem ser tomadas para corrigir o problema.

Ainda, é possível que pontos discrepantes causem esse comportamento dos dados. Para verificar essa possibilidade, é recomendável analisar as amostras individualmente para saber se existe alguma observação que possa ter originado tal problema. Para um tamanho de amostra pequeno ($n < 10$) pode-se utilizar um gráfico de dispersão e para amostras maiores, pode-se fazer um boxplot, como preferir.

```
> plot(sample[126:200],diametro[126:200])
```

Observando o gráfico, podemos ver que não há indícios de que alguma observação seja considerada um dados discrepante.

Se for de interesse, limites de aviso ou alerta podem ser inseridos no gráfico. Por exemplo, pode-se inserir uma linha com dois desvios.

```
> qcc(diametro[1:25,],type='xbar',newdata=diametro[26:40,])
> names(qcc.xbar)
> qcc.xbar<-qcc(diametro[1:25,],type='xbar',
  newdata=diametro[26:40,],nsigmas=2,plot=FALSE)

> abline(h=qcc.xbar$lim[1],col="red",lty=2)
> abline(h=qcc.xbar$lim[2],col="red",lty=2)
```

Gráfico R

Para construir o gráfico de controle R, basta definir o argumento `type` como `R`.

```
> qcc(diametro[1:25,], type="R")
> qcc(diametro[1:25,], type="R", newdata=diametro[26:40,])
```

Observe que na primeira parte do gráfico, há um ponto em destaque que representa uma seqüência de observações abaixo da amplitude média. O **R** identificou essa observação como um possível ponto fora de controle.

Se for de interesse, pode-se obter a amplitude média utilizando os seguintes comandos:

```
> dm.25<-diametro[1:25,]
> R<-numeric()
> for(i in 1:25){R[i]<-diff(range(dm.25[i,]))}
> mean(R)
```

Desenvolvimento de d_2, D_3 e D_4

Como visto na teoria, para construir os limites de controle dos gráficos de controle são utilizadas algumas constantes. Essas constantes podem ser obtidas por simulação computacional.

Inicialmente, vamos obter o valor da constante d_2. Para tanto, consideraremos o valor de $n = 4$. Assim, deve-se gerar 4 amostras da distribuição Normal padrão. Depois de geradas, tomam-se amostras de 4 observações e calcula-se a amplitude de cada amostra. A amplitude média dessas amostras é que vai originar a constante d_2.

```
> N1<-rnorm(1000)
> N2<-rnorm(1000)
> N3<-rnorm(1000)
> N4<-rnorm(1000)
> N<-cbind(N1,N2,N3,N4)
> d<-numeric()
> for(i in 1:1000)
          {
           d[i]<- diff(range(N[i,]))
          }
mean(d)
```

Esta simulação deve ser realizada com um n grande. Para um valor de n=4, o valor tabelado para d_2 é 2,059.

Para obter o valor de d_3, basta calcular o desvio padrão de d_2. Os valores de D_3 e D_4 são obtidos calculando-se as expressões desenvolvidas na parte teórica.

```
> d2<-mean(d);d2
> d3<-sd(d2);d3

> D4<-1+(3*d3/d2);D4
> D3<-1-(3*d3/d2);D3
```

Como D_3 é um valor negativo (e não existe amplitude negativa!) ele é substituído por zero. Para $n = 4$, $d_4 = 2,282$

Os valores de d_2, A_2, D_3 e D_4 são tabelados para diferentes valores de n (1).

Tabela 1: Constantes par o gráfico $\bar{x}eR$.

n	2	3	4	5	6	7
d_2	1,128	1,693	2,059	2,326	2,534	2,704
A_2	1,880	1,023	0,729	0,577	0,483	0,419
D_3	0	0	0	0	0	0,076
D_4	3,267	2,574	2,282	2,114	2,004	1,924

Gráficos de controle para S

Para construir o gráfico de controle S, basta definir o argumento `type` como `S`.

```
> qcc(diametro[1:25,], type="S")
> qcc(diametro[1:25,], type="S", newdata=diametro[26:40,])
```

Observe que, os limites do gráfico de $\bar{x}$ baseado no desvio padrão é idêntico ao gráfico de $\bar{x}$ baseado em R. Para amostrs pequenas isso é esperado.

Exercícios

1. Calcule os limites de controle para o gráfico $\bar{x}$ dos anéis de pistão utilizando os valores tabelados;

2. Calcule os limites de controle para o gráfico R dos anéis de pistão utilizando os valores tabelados;

3. Construa e interprete os gráficos de controle $\bar{x}$ e R para os seguintes conjuntos de dados:

   (a) Gere os dados com os seguintes comandos do **R**:

   ```
   > set.seed(1) # semente para simulação
   > a1<-rnorm(30,25,.5);a1 #30 obs. média 25; DP 0.5
   > set.seed(2) # semente para simulação
   > a2<-rnorm(30,25,.5);a2
   > set.seed(3) # semente para simulação
   > a3<-rnorm(30,25,.5);a3
   > dados<-cbind(a1,a2,a3);dados
   ```

   Experimente eliminar uma observação da segunda amostra ou alterar um valor:

   ```
   > dados[2,2]<-NA # ou
   > dados[2,2]<-50
   ```

   Refaça os gráficos.

   Mude a semente e os parâmetros da Normal e analise os dados. Experimente também variar o tamanho das amostras do processo.

   (b) Analise os dados desse processo.

Exercícios opcionais

(a) Explore a função `qcc` modificando argumentos e estude as opções de construção de um gráfico de controle para $\bar{x}$ e R;

(b) O pacote `qAnalyst` também possui algumas funções aplicadas ao controle estatístico de qualidade. Experimente fazer alguns gráficos de controle e compare-os com os gráficos do pacote `qcc`.

Veja alguns exemplos

```
require(qAnalyst)

data(cranks)

xbarchart=spc(x=cranks$crankshaft, sg=cranks$workingDay,
```

```
 type="xbar", name="crankshaft")

rbarchart=spc(x=cranks$crankshaft, sg=cranks$workingDay,
 type="r", name="crankshaft")

plot(xbarchart)
plot(rbarchart)
```

Veja a aplicação nos dados dos anéis de pistão (pistonrings):

```
#pistonrings

xbarchart.piston=spc(x=pistonrings$diameter, sg=pistonrings$sample,
 type="xbar", name="pistonrings")

rbarchart.piston=spc(x=pistonrings$diameter, sg=pistonrings$sample,
 type="r", name="pistonrings")

plot(xbarchart.piston)
plot(rbarchart.piston)
```

Outro exemplo:

```
data(brakeCap)
x=brakeCap$hardness
sg=brakeCap$subgroup
go=spc(x=x,sg=sg,type="xbar")
plot(go)
```

E o gráfico de Pareto:

```
 require(MASS)
data(Cars93)
paretoData=Cars93$Manufacturer[1:45]
paretoChart(x=paretoData,mergeThr=.8)
```

28 Gráficos de controle para atributos

Objetivos

O objetivo desta seção é construir e interpretar gráficos de controle para atributos. Procure entender os comandos do **R** e interpretar os resultados.

Gráfico de controle p

Utilizaremos os dados do exemplo sobre 'suco de laranja' (Montgomery, 1997), disponível no pacote `qcc`.

```
> require(qcc)
> data(orangejuice)
> attach(orangejuice)
> orangejuice
```

Esse conjunto de dados corresponde a latas de suco de laranja concentrado. As latas são fabricadas por uma máquina. Nesse caso, amostras são inpscionadas para verificar se existe algum defeito.

O gráfico de controle inicial (calibração) é baseado em 30 amostras de tamanho $n = 50$. Para esses dados, não existe um padrão para a fração de nãoconformes. Deve-se utilizar $\bar{p}$.

Para fazer o gráfico p, utilizando as 30 primeiras amostras. seguintes comandos são necessários:

```
> qcc(D[trial==TRUE], sizes=size[trial==TRUE], type="p")

Call:
qcc(data = D[trial == TRUE], type = "p", sizes = size[trial==TRUE])

p chart for D[trial == TRUE]

Summary of group statistics:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0800  0.1600  0.2100  0.2313  0.2950  0.4800
```

```
Group sample size:  50
Number of groups:  30
Center of group statistics:  0.2313333
Standard deviation:  0.421685

Control limits:
        LCL       UCL
 0.05242755 0.4102391
```

Observe no gráfico, que existem dois pontos fora de controle. São as observações 15 e 23. Nesse caso, existem duas informações adicionais que justificam os problemas ocorridos no processo. Mudança de matéria prima e um erro do operador.

Por isso, essas observações devem ser retiradas da amostra e novos limites devem ser calculados.

```
> inc <- setdiff(which(trial), c(15,23)) # elimina obs
> q1 <- qcc(D[inc], sizes=size[inc], type="p")
```

Observe no gráfico, que ainda existe uma observação que está fora de controle. Nesse caso, não há informações sobre esse ponto (ou amostra). Pode-se decidir em retirá-lo e recalcular os limites ou deixar esse ponto e reajustar a máquina.

Pode-se observar pelas estatísticas do gráfico que o percentual de defeitos é de 21,5%. Por esse motivo, nesse exemplo, decidiu-se realizar ajustes na máquina e coletar mais 24 amostras para saber se as mudanças foram eficientes.

```
> qcc(D[inc], sizes=size[inc], type="p", newdata=D[!trial],
  newsizes=size[!trial])
```

Observe que o percentual de produtos defeituosos é menor na segunda fase do que na primeira (isso pode ser testado, veja a seguir). Isso indica que o processo, após receber ajustes tornou-se melhor.

```
> prop.test(c(133,301),n=c(1200,1400))

2-sample test for equality of proportions with continuity correction
```

```
data:  c(133, 301) out of c(1200, 1400)
X-squared = 49.6724, df = 1, p-value = 1.817e-12
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.13284351 -0.07548983
sample estimates:
   prop 1    prop 2
0.1108333 0.2150000
```

Por isso, faz sentido aqui utilizar os limites de controle da segunda fase e não os limites da primeira fase.

Assim, após utilizar os novos dados para obtenção dos limites de controle, pode-se fazer o monitoramento do processo.

```
> detach(orangejuice)
> data(orangejuice2);orangejuice2
> names(D) <- sample
> attach(orangejuice2)
> qcc(D[trial], sizes=size[trial], type="p")
> q2 <- qcc(D[trial], sizes=size[trial], type="p",
 newdata=D[!trial], newsizes=size[!trial])
```

Gráfico de controle c

Utilizaremos os dados do exemplo sobre 'placas de circuito impresso' (Montgomery, 1997), disponível no pacote `qcc`.

```
> require(qcc)
> data(circuit)
> attach(circuit)
> circuit
```

Estes dados correspondem a 26 amostras de tamanho 100, ou seja, em cada amostra foram avaliados 100 placas de cicuito impresso e o número de não conformidades foi obtido.

Para construir o gráfico de controle,

```
> qcc(x[trial], sizes=size[trial], type="c")
```

Nesse gráfico, observa-se que nas amostras 6 e 20 ocorreram pontos fora dos limites. Sabe-se, como no exemplo anterior, que nessas amostras ocorreram problemas que foram corrigidos durante o processo. Por isso, pode-se optar por eliminar essas amostras e recalcular os limites de controle do gráfico.

```
> inc <- setdiff(which(trial), c(6,20))
> qcc(x[inc], sizes=size[inc], type="c", labels=inc)
```

Agora, não há pontos fora de controle. Assim, o processo pode ser monitorado considerando esses limites de controle.

```
> qcc(x[inc], sizes=size[inc], type="c", labels=inc,
newdata=x[!trial], newsizes=size[!trial], newlabels=which(!trial))
```

Ainda há um número razoável de não conformidades. Por isso, é necessário que o processo seja reavaliado e corrigido. Mas, isso é um trabalho para o pessoal da engenharia!

Gráfico de controle μ

Utilizaremos os dados do exemplo sobre 'fabricação de PC's' (Montgomery, 1997), disponível no pacote `qcc`.

```
> detach(circuit)
> require(pcmanufact)
> data(pcmanufact)
> attach(pcmanufact)
> pcmanufact
```

Estes dados correspondem a 20 amostras de tamanho $n = 5$. Nesse caso, 5 computadores foram avaliados e o número de não conformidades foi obtido. Aqui, interessa o número médio de não conformidades por computador.

```
> qcc(x, sizes=size, type="u")
```

Neste exemplo, não há pontos fora de controle.

Exercícios

1. Para todos os gráficos gerados, realize os cálculos e encontre os limites de controle e linha central;

2. Veja a teoria para o gráfico np e também encontre os limites de controle para o exemplo do suco de laranja (orangejuice).

3. Faça um quadro resumo de todos os gráficos de controle estudados, indicando a linha central, e os limites de controle;

Exercícios opcionais

1. Utilize comandos básicos para gráficos e com os cálculos dos limites de controle gere os gráficos de controle;

29 Gráficos de controle: tamanho de amostra

Objetivos

O objetivo dessa seção é encontrar o erro β para alguns gráficos de controle para variáveis e atributos e construir a curva característica de operação (CCO).

Valor de β e CCO para o gráfico $\bar{x}$.

O valor de β para o gráfico $\bar{x}$ depende da constante L, que é o número de desvios para os limites de controle, do valor de k, que é a constante a ser avaliada, ou seja, e é ela quem determina qual a chance de um ponto ultrapassar esse limite em função do desvio padrão e n é o tamanho da amostra utilizada.

Nesse exemplo, vamos utilizar o valor padrão de $L = 3$, assumindo $k = 2$ e um tamanho de amostra $n = 6$. A função `beta.xbar` calcula o valor de β nessas condições.

```
> beta.xbar<-function(L=3,k=2,n=6)
                {
  phi.1= (L-k*sqrt(n))
```

```
  phi.2= (-L-k*sqrt(n))
  beta<-pnorm(phi.1)- pnorm(phi.2)
  result<-round(c(beta,phi.1,phi.2),4)
  nomes<-c("beta","phi.1","phi.2")
  names(result)<-nomes
  print(result)
}
```

Agora, basta utilizá-la:

```
> beta.xbar()
   beta   phi.1   phi.2
 0.0288 -1.8990 -7.8990
```

O valor de 0.0288 é o risco de que uma mudança na ordem de 2σ não seja percebida no gráfico de controle.

Fazendo

$$CMP = \frac{1}{1 - 0.0288} = 1,029654$$

obtém-se o número de amostras necessárias para se detectar uma mudança na média na ordem de 2σ.

De forma geral, pode-se construir uma CCO para esse tipo de gráfico. Nesse caso, é possível definir o tamanho de amostra em função do risco assumido e da definição da variação na média.

```
> cco.xbar<-function(n.min=2,n.max=4,L=3)
  {
   k.x<-seq(0,4.5,length=100)
   beta.y<-seq(0,1,length=100)
   phi.1<-numeric()
   phi.2<-numeric()
   beta <-numeric()
   plot(k.x,beta.y,type="n",xlab="k",ylab=expression(beta))
   abline(h=0)
   for(j in n.min:n.max)
```

```
   {
        for(i in 1:length(k.x))
          {
           phi.1[i]<- L-k.x[i]*sqrt(j)
           phi.2[i]<- -L-k.x[i]*sqrt(j)
          }
     beta<-pnorm(phi.1)-pnorm(phi.2)
     legend(3,.8,leg=n.min:n.max,lty=n.min:n.max,pch=20,cex=1.2,
     title="n",pt.cex=1.0,ncol=2,pt.lwd=1.3)
     lines(k.x,beta,lty=j)
    }
}
```

Para identificar um ponto, e encontrar o tamanho de amostra, utilize a função `points`:

```
> points(1.0,.10,pch=4)
```

Valor de β e CCO para o gráfico p.

Para o gráfico de controle p, pode-se, também, encontrar o valor de β. Nesse caso, a distribuição utilizada, é a distribuição Binomial.

Para criar essa função, devemos fornecer o tamanho da amostra e os limites de controle do gráfico, além do valor da linha central.

```
> beta.p<-function(n=50,p=0.3,LSC=0.3697,LIC=0.0303)
               {
  D1<- pbinom(n*LSC, n, p, lower.tail = TRUE, log.p = FALSE)
  D2<- pbinom(n*LIC, n, p, lower.tail = TRUE, log.p = FALSE)
  beta<-D1-D2
  result<-round(c(beta,D1,D2),4)
  nomes<-c("beta","D1","D2")
  names(result)<-nomes
  print(result)
               }
```

```
> beta.p()
  beta     D1     D2
0.8594 0.8594 0.0000
```

Nesse caso, se $p = 0.3$ (um processo fora de controle), tem-se o valor de $\beta = 0.8594$, ou seja, teremos um CMP=7, que indica o número médio de amostras para se identificar uma mudança dessa magnitude, através de um ponto fora de controle.

Para diminuir o valor de β, pode-se aumentar o tamanho da amostra e/ou diminuir o intervalo de obtenção das amostras. Ao invés de coletar uma amostra a cada hora (sriam necesárias 7 horas para encontrar o problema), poderia-se reduzir para meia hora, ou seja, teríamos um intervalo de 3,5 horas para encontrar o problema.

A curva característica de operação pode ser obtida com a seguine função:

```
> cco.p<-function(n=50,LSC=0.3697,LIC=0.0303)
  {
   p.x<-seq(0,.8,length=100)
   beta.y<-seq(0,1,length=100)
   D1<-numeric()
   D2<-numeric()
   beta <-numeric()
   plot(p.x,beta.y,type="n",xlab="p",ylab=expression(beta))
   abline(h=0)
           for(i in 1:length(p.x))
     {
     D1[i]<- pbinom(n*LSC, n, p.x[i])
     D2[i]<- pbinom(n*LIC, n, p.x[i])
     }
    beta<-D1-D2
    text(c(.5,.5,.5),c(.8,.75,.7),c("n"=n,LSC,LIC),adj=c(0,05))
    text(c(.48,.48,.48),c(.8,.75,.7),c('n=','LSC=','LIC='),adj=c(1,05))
    lines(p.x,beta)
    result<-list(beta,p.x)
    nomes<-c("beta","p")
    names(result)<-nomes
    return(result)
 }
```

A curva pode ser gerada com

```
> cco.p()
```

Observe que ela representa a curva apenas para os parâmetros fornecidos.

Valor de β e CCO para o gráfico c.

Para o gráfico de controle c, utilizamos a distribuição Poisson para encontrar o valor de β:

```
> beta.c<-function(c=15,LSC=33.22,LIC=6.48)
                {
  D1<- ppois(LSC, c)
  D2<- ppois(LIC, c )
  beta<-D1-D2
  result<-round(c(beta,D1,D2),4)
  nomes<-c("beta","D1","D2")
  names(result)<-nomes
  print(result)
                }
```

na função `beta.c`, devemos fornecer o parâmetro c (número de não-conformes) e os limites de controle. Como os limites necessitam ser números inteiros, o **R** automaticamente faz o arredondamento.

```
> beta.c()
  beta     D1     D2
0.9924 1.0000 0.0076
```

Para c=15, teremos um $\beta = 0,9924$.

Também, para o gráfico c, pode-se construir a CCO. Basta apenas fornecer os limites de controle:

```
> cco.c<-function(LSC=33.22,LIC=6.48)
  {
```

```
  c.x<-seq(0,49.5,by=.5)
  beta.y<-seq(0,1,length=100)
  D1<-numeric()
  D2<-numeric()
  beta <-numeric()
  plot(c.x,beta.y,type="n",xlab="c",ylab=expression(beta))
  abline(h=0)
          for(i in 1:length(c.x))
         {
          D1[i]<- ppois(LSC, c.x[i])
          D2[i]<- ppois(LIC, c.x[i])
         }
  beta<-D1-D2
  text(c(40,40),c(.85,.8),c(LSC,LIC),adj=c(0,05))
  text(c(40,40),c(.85,.8),c('LSC=','LIC='),adj=c(1,05))
  lines(c.x,beta)
  result<-list(beta,c.x)
  nomes<-c("beta","c")
  names(result)<-nomes
  return(result)
}
```

Utilize, então a função `cco.c()`. Se for de interesse, pode-se encontrar um valor de β para um determinado parâmetro c, fazendo:

```
> betas<-cco.c()
> names( betas)
> betas$beta[which(betas$c==49.5)] # por exemplo
```

Exercícios

1. Para os exemplos de gráficos de controle $\bar{x}$, p e c estudados no pacote `qcc` (exemplos: pistonrings, orangejuice2, circuit), determine a chance de se cometer o erro tipo II.

30 Índices de Capacidade

Objetivos

O objetivo desta seção é encontrar índices de capacidade para para processos. Procure entender os comandos do **R** e interpretar os resultados.

Obtenção dos índices

No pacote `qcc` há duas funções que realizam a análise de capacidade de um processo. A primeira é a função `process.capability()`, que faz um histograma dos dados, calcula os índices ce capacidade e ainda estima o número de observações que deverão ultrapassar os limites de especificação.

Inicialmente, vamos chamar o pacote `qcc` e trabalhar com o conjunto de dados sobre anéis de pistão (pistonrings):

```
> require(qcc)
> data(pistonrings)
> attach(pistonrings)
```

Para utilizar a função `process.capability()`, deve-se obter um gráfico de controle do tipo 'xbar'. É a partir deste objeto que a função `process.capability()` irá obter as informações para calcular os índicess de capacidade.

Observe os argumentos dessa função e veja que várias informações podem ser fornecidas para a obtenção dos índices.

```
> args(process.capability)
function (object, spec.limits, target, std.dev, nsigmas,
confidence.level = 0.95, breaks = "scott", add.stats = TRUE,
print = TRUE, restore.par = TRUE)
```

Aqui, *object* deve ser um gráfico de controle do tipo 'xbar'. *spec.limits* são os limites de especificação. *target* é o valor nominal, em geral, o ponto médio entre os limites de especificação, e *std.dev* é o desvio padrão. O desvio padrão pode ser obtido diretamente do gráfico de controle (mais comum) ou então, ser fornecido.

Para usar a função devemos criar um objeto 'qcc'. Utilizaremos os dados das 25 primeiras amostras:

```
> diameter <- qcc.groups(diameter, sample)
> q <- qcc(diameter[1:25,], type="xbar", nsigmas=3, plot=T)
```

Para obter os índices, inicialmente, vamos fornecer os limites de especificação corretos (essa informação é obrigatória) e o desvio padrão obtido no gráfico R:

```
> process.capability(q, spec.limits=c(73.95,74.05),std=0.0099)
```

Observe no gráfico que o processo não está perfeitamente centrado, por isso as diferenças entre C_p e C_{pk}.

Agora, vamos experimentar algumas mudanças nos argumentos da função e ver como isso afeta os índices de capacidade do processo.

Por exemplo:

1. Alterando os limites de especificação para menos:

   ```
   > process.capability(q, spec.limits=c(73.93,74.07),std=0.0099)
   ```

   Observe que agora, muitos pontos ficam fora dos limites de especificação, ou seja, muitos produtos não-conformes ou defeituosos. Veja, ainda, que, aparecem os percentuais de observações esperadas e observadas, respectivamente, que ultrapassam os limites de especificação.

2. Alterando os limites de especificação para mais:

   ```
   > process.capability(q, spec.limits=c(73.94,74.06),std=0.0099)
   ```

   O processo melhorou bastante. Nesse caso, temos um processo dentro do padrão Seis Sigma!

3. Modificando o *target*:

   ```
   > process.capability(q, spec.limits=c(73.95,74.05), target=74.02)
   ```

   Desse modo, o processo não está centrado no valor nominal desejado. Por isso, o valor de C_{pm} é pequeno, como era de se esperar.

Experimente fazer outras mudanças, inclusive do desvio padrão para ver como se comportam os índices. Não esqueça de interpretá-los!

Uma análise mais geral

No **R** há uma função chamada `process.capability.sixpack()` que cria um conjunto de gráficos referentes a análise do processo, além de fornecer os índices de capacidade estudados.

Para usar essa função, basta seguir os mesmos procedimentos utilizados na função `process.capability()`:

```
>   process.capability.sixpack(q, spec.limits=c(73.95,74.05))
```

Como novidade, este ressultado apresenta um Gráfico Normal de Probabilidade para avaliar se realmente os dados apresentam um distribuição Normal.

Um site para visualizar índices

Visite o site e veja como se comportam os índices de capacidade em função da distribuição dos dados.

Exercícios

1. Utilize os comandos abaixo para criar e analisar dados de processos.

   ```
   > x <- matrix(rnorm(100, 0, 1), ncol=5)
   > q <- qcc(x, type="xbar", plot=FALSE)
   > process.capability.sixpack(q, spec.limits = c(-2,2), target=0)
   ```

   Você pode:

   - mudar os parâmetros da Normal (média e/ou variância);
   - mudar o tamanho e número de amostras;
   - mudar os limites de especificação;
   - mudar o valor nominal (target)

2. Procure na internet, em livros ou na sua empresa, conjuntos de dados para serem analisados e interpretados. Busque, preferencialmente, dados analisados para você poder comparar os resultados.

Exercício opcional

1. Utilize outra distribuição de probabilidade, além da Normal, para avaliar os índices de capacidade;

2. Utilize uma transformação de dados e analise os resultados.

31 Amostragem por aceitação

Objetivos

O objetivo desta seção é encontrar a probabilidade de aceitação para uma amostradege por aceitação simples de lotes. Também será dado enfoque para a construção da CCO. Procure entender os comandos do **R** e interpretar os resultados.

Probabilidade de aceitação - Pa

Como visto na teoria, a probabilidade de aceitação de um lote - Pa - é dada pelo uso da expressão da Distribuição Binomial. Nesse caso, desejamos encontrar a $P(d \leq c)$.

$$P_A = P(d \leq c) = \sum_{d=0}^{c} \frac{n!}{d!(n-d)!} p^d (1-p)^{n-d}$$

No **R** pode-se utilizar a distribuição Binomial, considerando o exemplo fornecido na teoria, da seguinte maneira:

```
> dbinom(0,89,0.01)+dbinom(1,89,0.01)+dbinom(2,89,0.01)
[1] 0.93969
> sum(dbinom((0:2),89,0.01))
[1] 0.93969
```

Essa é a probabilidade de aceitação de um lote com $n = 89$, $c = 2$ e $p = 0,01$.

Construção da CCO

De uma maneira geral, pode-se construir uma curva característica de operação e avaliar qual é o comportamento da probabilidade de aceitação Pa em função das características da amostragem e do lote.

Considerando que estamos tratando com a distribuição Binomial, pode-se construir uma curva considerando diferentes proporções de defeituosos em um lote, para um determinado valor de n e c.

De forma semelhante a realizada anteriormente, pode-se construir uma função da seguinte maneira:

```
> cco.s<-function(n=89,c=2){
   p.x<-seq(0,0.08,length=100)
   pa.y<-seq(0,1,length=100)
   p <-numeric()
   plot(p.x,pa.y,type="n",xlab="Proporção de defeituosos - p",
   ylab="Probabilidade de aceitação - Pa",cex.lab=1.5)
   abline(h=0)
           for(i in 1:length(p.x))
          {
           p[i]<-sum(dbinom((0:c),n,p.x[i]))

          }
     lines(p.x,p)
    text(c(0.05,0.047,0.05,0.047),c(0.6,0.6,0.5,0.5),
    c("n"=n,'n =',"c"=c,'c ='),cex=1.3)
    }
```

Para utilizar a função `cco.s()` deve-se fornecer o tamanho da amostra e o valor de c.

Construída a curva, podemos, então, encontrar qual o valor de Pa para um determinado valor de p. Como exemplo, se o valor de $p = 0,02$, para encontrar Pa, pode-se desenhar um linha vertical

```
> cco.s()
> abline(v=0.02,col='red',lty=2)  # ou
> segments(0.02,0,0.02,0.99, col= 'blue',lty=2)
```

Para encontrar o valor de Pa, usa-se a função `locator()`. Utilize a função da seguinte maneira:

1. faça o gráfico e mantenha a janela gráfica aberta aberta;
2. na linha de comando do **R**utilize a função para localizar $n = 1$ ponto:

```
> locator(n=1,type="p",col='red')
```

3. vá para a janela gráfica e clique no ponto onde a linha vertical cruza a curva;
4. volte para a linha de comando do **R**e observe o valor de y.

Exercícios

1. Defina as condições de amostragem de um lote (n, p, c). Encontre o valor de Pa;
2. Suponha uma situação factível (por exemplo, no seu local de trabalho, um exemplo de livro ou mesmo hipotético). Defina o valor de p e/ou c. Tente encontrar um tamanho de amostra n para essa condição.