R aplicado à Engenharia

David Souza

21 Outubro 2019

III Semana de Engenharia | Universidade Federal Fluminense *Campus* Rio das Ostras

Introdução pessoal

- Engenheiro de Produção pela UFF
- Mestrando em Engenharia de Produção pela PUC-Rio
- Lean Six Sigma Black Belt
- Áreas de atuação:
  - Séries Temporais
  - Simulação
  - Machine learning
  - Gerência de Operações
  - Otimização

R para Engenharia

Agenda

1. Introdução ao R
2. Análise de dados
3. Testes de Hipótese
4. Cartas de Controle
5. Cadeias de Markov

Introdução ao R

Introdução ao R: Objetivos

- Apresentar uma breve história do **R** e seus usos
- Apresentar a IDE RStudio
- Introduzir a sintaxe e boas práticas
- Apresentar as principais estruturas usadas
- Mostrar como carregar e visualizar dados
- Mostrar como usar bibliotecas

Por que progamar?

- Desempenho
- Escala
- Reprodutibilidade
- Evitar ou eliminar retrabalho

Por que usar R?

- Livre e open-source
- Multiplataforma
- Ferramentas disponíveis
- Desempenho
- Extensibilidade

Sobre o R, RStudio

O R é uma linguagem e um ambiente para computação estatística, similar à linguagem S desenvolvida pela AT&T no Bell Labs.

A linguagem disponibiliza uma série de ferramentas estatísticas (testes estatísticos, modelos lineares e não-lineares, análise de séries temporais), gráficas e numéricas através do CRAN (*Comprehensive R Archive Network*).

O RStudio é um ambiente de desenvolvimento open-source para R.

Boas práticas: formatação de códigos

Na hora de escrever qualquer código, é importante considerar:

- Nomes significantes para variáveis e arquivos,
- Indentação,
- Comentários.

Estes três elementos facilitam a compreensão de como o código funciona ao ser lido por terceiros ou ao ser revisto em um período posterior.

Exemplo: Para realizar alguns experimentos, uma tabela com 50 elementos e amostras de três distribuições (normal, *t*, uniforme) é criada.

```r
ii <- data.frame(1:50,
rnorm(50), rt(50, 10),
runif(50))
ii
```

```
##    X1.50   rnorm.50.  rt.50..10.   runif.50.
## 1      1 -0.93941930  0.07761799 0.272134630
## 2      2 -1.04622294  0.08150379 0.772305530
## 3      3 -0.80397307  0.24045371 0.643055339
## 4      4 -2.04955350  0.22240177 0.645298280
## 5      5  0.16203959  1.60427786 0.955291097
## 6      6 -1.28098895 -0.33970039 0.068889499
## 7      7 -0.92571393 -0.45980869 0.968683878
## 8      8 -0.46859327  0.11722284 0.691870225
## 9      9  0.47074751 -0.43087717 0.828368400
## 10    10  2.07442477  0.89536722 0.764852999
## 11    11  1.49923523 -0.04047716 0.067322129
## 12    12  1.76694577  0.43431901 0.549419900
## 13    13 -0.11981554  1.39550055 0.473901355
```

tabela_experimentos.R

```r
# DataFrame das distribuições para um experimento
experimentos <- data.frame(iteracao = 1:50,
                           exp_normal = rnorm(50),
                           exp_t = rt(50, 10),
                           exp_uniforme = runif(50))

# Observando 5 primeiros itens
head(experimentos, 5)
```

```
##   iteracao exp_normal      exp_t exp_uniforme
## 1        1  1.7827026 -0.2208650   0.02522732
## 2        2 -0.1453349  1.1104381   0.75737549
## 3        3 -0.2832498 -0.1702858   0.05925983
## 4        4 -0.2040276  0.5740936   0.95462740
## 5        5 -0.6564102 -0.4094373   0.49066487
```

Boas práticas: projetos

Na hora de realizar uma análise, é uma boa prática criar uma estrutura de projeto para organizar os arquivos necessário. A organzação auxilia na reprodutibilidade, e permite que o código seja distribuído como um pacote.

```
Nome do Projeto
├── R/ ..... Código escrito em R
├── data/ ..... Dados a serem lidos (leitura apenas)
├── docs/ ..... Documentação/artigo gerado sobre as análises
├── figs/ ..... Figuras geradas pelo código
├── output/ ..... Saídas geradas pelo código
├── src/ ..... Código escrito em C/C++
└── main.R ..... Arquivo que executa a análise
```

Tipos e estruturas básicos

Dentro do R alguns tipos numéricos já foram pré-definidos: `numeric`, `integer`, `logical`, `complex`, `character`. A declaração de tipo não é obrigatória: a própria linguagem faz os ajustes durante a execução.

O R também conta com uma série de representações de estruturas para armazenar e manipular dados:

Dimensões	Homogêneo	Heterogêneo
1	Vetor	Lista
2	Matriz	Data Frame
n	Array	—

Acessando dados nas estruturas

Os dados podem ser acessados por índices (posição) através da seguinte notação:

1. Vetores: `ex_vetor[x]`
2. Matrizes e data frames: `ex_tabular[i, j]`
3. Arrays: `ex_array[i, j, k]`
4. Listas: `ex_lista[[x]]`

Importante lembrar que todo os índices em **R** começam com **1**.

```
experimentos[2:4, ]
```

```
##   iteracao exp_normal      exp_t exp_uniforme
## 2        2 -0.1453349  1.1104381   0.75737549
## 3        3 -0.2832498 -0.1702858   0.05925983
## 4        4 -0.2040276  0.5740936   0.95462740
```

Acessando dados nas estruturas

Caso existam etiquetas de identifiação nas estruturas, é possível fazer o acesso de duas formas:

1. Usando o acesso de índice, que permite o uso de múltiplos identificadores:

```
experimentos[18:21, c("exp_t", "exp_uniforme")]
```

```
##          exp_t exp_uniforme
## 18 -0.6606374    0.3354311
## 19  0.8919628    0.4796399
## 20 -1.9126845    0.1164111
## 21  0.4626017    0.9497714
```

Acessando dados nas estruturas

2. Usando o operador $, que só permite o acesso a apenas uma coluna por vez:

```
experimentos$exp_normal[1:2]
```

```
## [1]  1.7827026 -0.1453349
```

Paradigmas

1. Orientado a objeto:

   - Tudo no **R** é um objeto, inclusive escalares.
   - Todo objeto possui alguma propriedade (e.g. tipo dos dados, dimensões).

```
typeof(3L)
## [1] "integer"
dim(airquality)
## [1] 153   6
lista <- list(3L, 1:5, datasets::AirPassengers)
head(lista[[3]])
##      Jan Feb Mar Apr May Jun
## 1949 112 118 132 129 121 135
```

2. Funcional:

- Maneira de interagir com os objetos, de acordo com suas propriedades.
- Não alteram os inputs.

```
du <- runif(100)
length(du)
## [1] 100
sum(du)
## [1] 49.65727
mean(du)
## [1] 0.4965727
head(du, 5)
## [1] 0.4240161 0.3791791 0.7548422 0.4157105 0.5728738
```

R como Calculadora

```r
# Quatro operações básicas
15 + (2 * 2.5) - (6 / 3)
## [1] 18

# Raiz quadrada
sqrt(81)
## [1] 9

# Exponenciação
3 ** 3 # ou 3 ^ 3
## [1] 27

# Logaritmo natural
log(3.6)
## [1] 1.280934
```

R como Calculadora

```r
# Produto vetorial
c(1, 1) %*% c(2, 1)
##      [,1]
## [1,]    3

# Transposição de matrizes
A <- matrix(c(1, 2, 0, 5), byrow = TRUE, ncol = 2)
t(A)
##      [,1] [,2]
## [1,]    1    0
## [2,]    2    5

# Determinante
det(A)
## [1] 5
```

R como Calculadora

```
# Produto matricial
B <- matrix(c(2, 0, 1, 2), byrow = TRUE, ncol = 2)
A %*% B
##      [,1] [,2]
## [1,]    4    4
## [2,]    5   10

# Inversão de matrizes
solve(A)
##      [,1] [,2]
## [1,]    1 -0.4
## [2,]    0  0.2

# Solução de sistemas lineares
b <- c(1, 1)
solve(A, b)
## [1] 0.6 0.2
```

Vetorização numérica

Soma de um escalar com um vetor

```
# Criando um vetor de números aleatórios
randn <- rnorm(4)
randn
```

```
## [1] -0.85701729  0.41462712 -0.79229017  0.06670949
```

```
# Somando 5 a todos
5 + randn
```

```
## [1] 4.142983 5.414627 4.207710 5.066709
```

Vetorização numérica

```r
# Criando um vetor e um data frame de números aleatórios
vec_y <- 3 * rbeta(4, 0.5, 0.2)
df_randn <- data.frame(x = rnorm(4), y = rchisq(4, 10))

# Somando o vetor com o data frame
vec_y + df_randn
```

```
##             x         y
## 1 2.51325173 13.724030
## 2 0.03934756 16.133718
## 3 3.68710094  6.368595
## 4 1.22870506 10.879006
```

Declarando funções

Podemos construir funções no **R** com a palavra-chave `function()`, colocando entre os parênteses os argumentos de entrada.

```
fsoma <- function(x, y, z = 2) {
  h <- 3*x + 2*y - z
  return(h)
}

fsoma(3, 5)
## [1] 17
fsoma(4, 15, 6)
## [1] 36
```

Análise de dados

Análise de dados: Objetivos

- Carregar dados no sistema
- Tirar medidas sobre os dados
- Visualizar os dados

Carregando dados: dados em pacotes

O R já conta com alguns dados espalhados por diversos pacotes, que são carregados de duas formas:

1. De maneira implícita: ao carregar o pacote, os dados já ficam disponíveis no ambiente.

```
# library(datasets)
head(airquality)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

Carregando dados: dados em pacotes

2. De maneira explícita: após carregar o pacote é necessário usar o comando `data(nome_do_dataset)` para deixá-lo disponível no ambiente.

Exemplo:

```
# library(qcc)
data(pistonrings)
head(pistonrings)
```

```
##   diameter sample trial
## 1   74.030      1  TRUE
## 2   74.002      1  TRUE
## 3   74.019      1  TRUE
## 4   73.992      1  TRUE
## 5   74.008      1  TRUE
## 6   73.995      2  TRUE
```

Lendo dados: dados no HD

É necessário verificar o formato do arquivo:

1. Arquivos em texto (`.csv`, `.tsv`) são lidos com a função `read.csv()`.
2. Arquivos de Excel (`.xls`, `.xlsx`) são lidos com funções das bibliotecas `readxl` (ambos formatos) ou `openxlsx` (apenas `.xlsx`).
3. Para outros formatos, é necessário buscar a biblioteca adequada.

Determinado o formato de leitura, cria-se o objeto com as funções adequadas.

Importante: Deve-se considerar a localização do arquivo (caminho, `path`) na hora de usar as funções.

Lendo arquivos `.csv`, `.tsv`

1. Diferença entre arquivos `.csv` e `.tsv`: o primeiro usa vírgula (`,`) ou ponto-vírgula (`;`) para separar os valores, o outro usa um espaço tabular (`\t`).
2. Caso seja necessário ler arquivos com texto como texto de fato, usar `stringsAsFactors = FALSE`.

```
arquivo_csv <- read.csv("C:/projeto/data/amostra_lotes.csv")
```

Lendo arquivos .xlsx

```
library(openxlsx)
wb <- read.xlsx("C:/projeto/data/ss.dados.xlsx",
                sheet = 1,
                colNames = TRUE)
```

Lendo dados da Web

Para ler arquivos da web há duas opções:

1. Arquivos de texto podem ser lidos diretamente pela função `read.csv`.
2. Arquivos de Excel precisam ser baixados de uma forma ou de outra.

   Mas é possível criar uma estrutura temporária para ler os dados.

```
f_link <- "https://souzapd.github.io/datasets/ss.dados.xlsx"
f_temp <- tempfile(fileext = ".xlsx")
download.file(f_temp, f_link, mode = "wb")
wb <- read.xlsx(f_temp, sheet = 1, colNames = TRUE)
```

Informações sobre a distribuição dos dados

O programa já tem na sua instalação base uma série de funções para avaliar a distribuição de dados numéricos.

- `mean(dados), median(dados)`: média e mediana
- `var(dados), sd(dados)`: variância e desvio-padrão
- `max(dados), min(dados)`: máximo e mínimo
- `quantile(dados, p)`: sem um valor `p` dá os quantis 0, .25, .50, .75 e 1.
- `IQR(dados)`: distância interquartílica (entre o 1º e 3º quartis)
- `summary(dados)`

Com exceção da função `summary()`, todas as outras têm a opção para evitar dados não disponíveis (`NA`): basta acrescentar `na.rm = TRUE`.

Vetorização com funções

Se tentarmos aplicar as funções acima para todo o Data Frame, obteremos um erro.

Mas é possível vetorizar a aplicação através da função `apply()`. Vamos usar o dataset `stackloss` para demonstrar.

```
# Aplicando a média nas colunas (índice = 2)
apply(stackloss, 2, mean, na.rm = TRUE)
##   Air.Flow Water.Temp Acid.Conc. stack.loss
##   60.42857   21.09524   86.28571   17.52381

# Obtendo o quantil 25%
apply(stackloss, 2, quantile, probs = 0.25, na.rm = TRUE)
##   Air.Flow Water.Temp Acid.Conc. stack.loss
##         56         18         82         11
```

Gráficos no R

A linguagem possui 3 engines gráficas.

1. `base`: gráficos simples, expressivos e personalizáveis.
2. `lattice`: funções de alto nível para gráficos 2D e 3D.
3. `ggplot`: Baseado na Gramática de Gráficos (*Grammar of Graphics*), versátil e com alto nível de personalização.

Gráficos com `lattice`

O `lattice` é uma biblioteca já disponível na instalação padrão, com funções de alto nível:

- `xyplot`: gráficos para duas variáveis.
- `bwplot`: boxplot.
- `histogram`: histogramas.
- `densityplot`: gráfico de densidade.
- `qqmath`: gráfico quantil-quantil para avaliar normalidade.

É uma ótima ferramenta para visualizar dados durante o desenvolvimento de uma análise. É possível criar gráficos adequados para publicação, embora sua customização seja um pouco complexa por vezes.

Scatterplot

```
xyplot(Petal.Length ~ Petal.Width,
       groups = Species,
       data = iris,
       main = "Scatterplot: largura-comprimento de pétalas",
       xlab = "Largura",
       ylab = "Comprimento")
```

```r
# Dados de séries temporais podem ser inseridos diretamente
xyplot(LakeHuron,
       main = "Nível do Lago Huron",
       xlab = "Ano",
       ylab = "Nível (ft.)")
```

Boxplot

```
bwplot(height ~ voice.part,
       data = singer,
       main = "Boxplot de altura para tipo de voz",
       xlab = "Voz",
       ylab = "Altura (pol.)")
```

Histograma

```
amostras <- data.frame(sorteio = rnorm(1000))
histogram( ~ sorteio,
       data = amostras,
       type = "density",
       main = "Histograma: Sorteio aleatório ~ N(0, 1)",
       xlab = "Valores")
```

Análises estatísticas

Análises estatísticas: Objetivos

- Revisitar conceitos estatísticos:
  - Variável aleatória
  - Inferência estatística
- Como realizar testes de hipótese no R

Definição: Variável aleatória

Uma variável aleatória é uma regra que relaciona números reais com resultados experimentais.

- `Discreta`: número de não-conformidades em uma amostra; condição operacional de uma máquina.

- `Contínua`: vida útil de uma lâmpada (em horas), quantidade de liquído em uma lata de 330ml.

Inferência estatística

A inferência estatística tem como objetivos concluir algo sobre uma população, ou tomar uma decisão. Para tal, a informação de uma amostra aleatória dessa população é empregada.

Para tal, a inferência trabalha com dois conceitos: a estimação de parâmetros e os testes de hipótese.

Estimação de parâmetros

A estimação de parâmetros é a ação de usar uma amostra para estimar os valores de parâmetros populacionais desconhecidos.

Parâmetro θ	Estatística	Estimativa pontual $\hat{\theta}$
μ	$\frac{\sum X_i}{n}$	$\bar{x}$
σ^2	$\frac{\sum (X_i - \bar{X})^2}{n-1}$	s^2
p	$\frac{X}{n}$	$\hat{p}$
$\mu_1 - \mu_2$	$\frac{\sum X_{1i}}{n_1} - \frac{\sum X_{2i}}{n_2}$	$\bar{x}_1 - \bar{x}_2$
$p_1 - p_2$	$\frac{X_{1i}}{n_1} - \frac{X_{2i}}{n_2}$	$\hat{p}_1 - \hat{p}_2$

Teste de hipótese

- Hipótese: uma afirmação sobre os dados que queremos avaliar.
- **Teste de hipótese**: processo de tomada de decisão, composto por uma hipótese nula (o *status quo*), e uma hipótese alternativa.
  Normalmente apresentam o formato:

$$H_0: \mu = \mu_0$$
$$H_a: \mu \neq \mu_0$$

- Com base em uma estatística de teste, podemos ter indícios para (1) rejeitar a hipótese nula; ou (2) não rejeitar a hipótese nula.

Hipótese alternativa

A hipótese alternativa pode ser expressa de três formas:

1. $\sigma_1^2 \neq \sigma_2^2$
2. $\sigma_1^2 > \sigma_2^2$
3. $\sigma_1^2 < \sigma_2^2$

Independente da hipótese alternativa, temos os seguintes tipos de erro em testes de hipótese:

Decisão	H_0 é verdadeira	H_0 é falsa
Não rejeitar H_0	-	Erro Tipo II (β)
Rejeitar H_0	Erro Tipo I (α)	-

Inferência para Six Sigma

Ao trabalhar com Six Sigma, assumimos que os dados amostrados (1) seguem uma distribuição normal $X \sim N(\mu, \sigma^2)$ e (2) são *iid* (independentes e identicamente distribuídos).

Esses pressupostos são críticos: ignorar um coloca qualquer análise em xeque.

Testando normalidade: Gráfico $q - q$

```
teste_esp <- read.xlsx("ss.dados.xlsx", sheet = 1)
qqmath(~ mm,
       teste_esp,
       main = "q-q Plot",
       prepanel = prepanel.qqmathline,
       panel = function(x, y, ...) {
         panel.qqmath(x, ...)
         panel.qqmathline(x, col = 2)})
```

Testando normalidade: Shapiro

Hipóteses:

- H_0: os dados são normais.
- H_a: os dados não são normais.

```
teste_esp <- read.xlsx("ss.dados.xlsx", sheet = 1)
shapiro.test(teste_esp$mm)
```

Testes de hipótese para médias

Teste t para 1 amostra

Considere um processo de manufatura, cuja especificação requer que as peças possuam em média 50mm. Como estipulado pela Qualidade, uma amostra de peças recém produzidas foi coletada para inspeção. A batelada se encontra em conformidade com a especificação?

```
teste_esp <- read.xlsx("ss.dados.xlsx", sheet = 1)
t.test(teste_esp$mm, mu = 50, alternative = "two.sided")
```

ANOVA: Comparação de várias médias

Considere o mesmo processo de manufatura. Recentemente a Engenharia sugeriu o uso de outros materiais para o processo, como uma alternativa para tornar o processo mais eficiente e barato. 5 alternativas de ligas foram sugeridos e testados. Existe alguma diferença entre eles?

```
teste_liga <- read.xlsx("ss.dados.xlsx", sheet = 2)
fit <- aov(mm ~ liga, data = teste_liga)
summary(fit)
TukeyHSD(fit, ordered = TRUE)
plot(TukeyHSD(fit, ordered = TRUE))
```

Testes para variâncias

Teste *F*

Usamos o Teste *F* para comparar a razão de duas variâncias (ou, alternativamente, se duas variâncias são iguais).

A qualidade agora busca avaliar dois fornecedores do mesmo insumo para o processo produtivo das peças. Foram realizadas 10 coletas de peças feitas com materiais de cada. Existe variação significante entre as peças produzidas?

```
teste_fornecedores <- read.xlsx("ss.dados.xlsx", sheet = 3)
var.test(mm ~ fornecedor, data = teste_fornecedores)
```

Teste de Barlett

O teste de Barlett é uma forma de comparar a variância de várias amostras.

Vamos retomar ao dados usados na ANOVA. Foi verificado se havia diferença em relação à especificação da média. Mas deseja-se agora saber se existe diferença na variância das peças produzidas com cada.

```
teste_liga <- read.xlsx("ss.dados.xlsx", sheet = 2)
bartlett.test(mm ~ liga, data = teste_liga)
```

Cartas de Controle

Cartas de Controle: Objetivos

- Revisão das cartas de controle
- Realizar o plot dessas cartas em R

Por que usar Cartas de Controle?

Qualquer processo, por mais bem projetado que seja, está sujeito a alguma variabilidade. Dessas, podemos considerar causas comuns (aquelas que são parte inerente do processo) ou causas especiais (oriundas de alguma situação extraordinária).

Podemos usar Cartas de Controle como ferramentas de diagnóstico que nos permitam avaliar se um processo está sobre controle ou se está se comportando de maneira diferente da projetada.

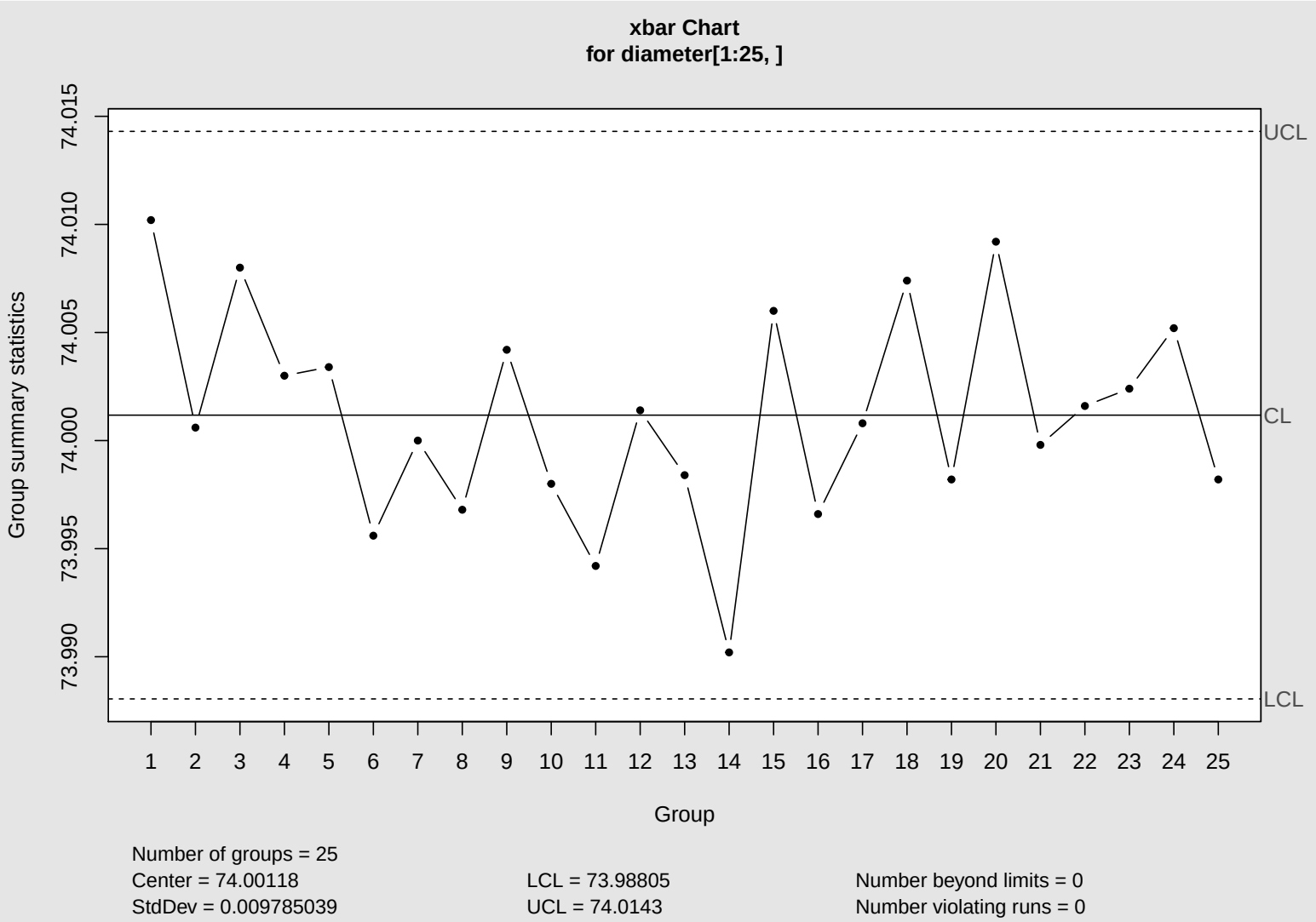
Também são ferramentas testadas em campo para melhorar a produtividade, bem como ajudam a evitar ajustes desnecessários dos processo produtivos.

Tipos de Cartas

Podemos pensar em dois tipos de cartas: para **variáveis** ou para **atributos**.

Carta	Tipo	Descrição
$\bar{X}$	V	Média amostral de uma variável contínua.
R	V	Amplitude amostral de uma variável contínua.
S	V	Desvio-padrão de uma variável contínua.
p	A	Proporção de não-conformidades.
np	A	Número de não-conformidades.
c	A	Número de defeitos por unidade.
u	A	Número médio de defeitos por unidade.

Elementos de uma Carta de Controle



Cálculo dos limites de controle

Carta	Linha central	Limites
$\bar{X}$	$\bar{\bar{x}}$	$\bar{\bar{x}} \pm A_2 \bar{r}$
R	$\bar{r}$	$D_3 \bar{r}$, $D_4 \bar{r}$
S	$\bar{s}$	$B_3 \bar{s}$, $B_4 \bar{s}$
p	$\bar{p}$	$\bar{c} \pm 3\sqrt{\bar{p}(1-\bar{p})/n}$
np	$n\bar{p}$	$\bar{c} \pm 3\sqrt{n\bar{p}(1-\bar{p})}$
c	$\bar{c}$	$\bar{c} \pm 3\sqrt{\bar{c}}$
u	$\bar{u}$	$\bar{u} \pm 3\sqrt{\bar{u}/n}$

Biblioteca `qcc`

A principal função do pacote (`qcc()`) apresenta três argumentos-chave:

- `data`: um vetor, matriz ou data frame. No caso de dados tabulares, cada linha se refere a um grupo racional.
- `type`: o tipo de carta a ser gerado. Para os gráficos de média, há dois tipos: `xbar.one` e `xbar`.
- `sizes`: o tamanho de cada amostra para cada grupo. Obrigatório para as cartas de atributos

A função também permite que comparemos um processo com novos dados através do argumento `newdata`.

Cartas de Controle para Variáveis

Vamos olhar novamente para a produção da peça. Dessa vez, vamos olhar para seu processo produtivo e ver se o mesmo está sob controle. Assim, foram coletadas amostras de tamanho $n = 5$ de hora em hora para análise. Como o processo está se comportando.

```
dados_variavel <- read.xlsx("qc.dados.xlsx", sheet = 1)
dados_variavel <- with(dados_variavel, qcc.groups(mm, amostra))
carta_xbar <- qcc(dados_variavel, type = "xbar")
carta_r <- qcc(dados_variavel, type = "R")
```

Cartas de Controle para Variáveis

Há uma solicitação da qualidade para verificar um outro processo dentro da fábrica – a produção de chapas de metal. No processo foram coletadas 30 amostras a cada 30 minutos para verificar a presença de mossas e riscos. Sabe-se também que para mossas, só são aceitas até duas por peça, e que no caso de riscos, o limite é de 2% de prevalência.

```
dados_atributo <- read.xlsx("qc.dados.xlsx", sheet = 2)
carta_u <- with(dados_atributo, qcc(mossas,
                                    size = n,
                                    type = "c",
                                    limits = c(0, 2)))

carta_p <- with(dados_atributo, qcc(riscos,
                                    size = n,
                                    type = "p",
                                    limits = c(0, 0.02)))
```

Cadeias de Markov

Cadeias de Markov: Objetivos

- Revisitar conceitos de processos estocásticos
- Revisitar conceitos de cadeias de Markov:
  - Matriz de transição
  - Forma canônica
- Plotar os grafos associados

O que é um processo estocástico?

Podemos nos perguntar: qual a difereança entre uma `variável aleatória` e um `processo estocástico`?

O que é uma Cadeia de Markov (CM)?

Cadeias de Markov tem esse nome por causa da **propriedade de Markov**.

Seja um processo estocástico X_t discreto. A probabilidade de transição de um estado para outro é dada por:

$$P\big(X_t = i_t \big| X_{t-1} = i_{t-1}, ..., X_1 = i_1, X_0 = i_0\big) = P\big(X_t = i_t \big| X_{t-1} = i_{t-1}\big) \quad \forall i_t \in S$$

Matriz de transição

Como a probabilidade de uma mudança de estado só depende do estado anterior, podemos usar uma notação matricial para representar essas transições.

Exemplo: Um Táxi transita entre três localidades (Aeroporto, Centro, Hotel).

$$P = \begin{array}{c} \\ A \\ C \\ H \end{array} \begin{array}{c} \begin{array}{ccc} A & C & H \end{array} \\ \begin{pmatrix} 0 & 1/2 & 1/2 \\ 1/5 & 2/5 & 2/5 \\ 3/5 & 2/5 & 0 \end{pmatrix} \end{array}$$

Outra propriedade importante é que as linhas da matriz somam 1.

$$\sum_{j=1}^{n} P_{i,j} = 1 \quad \forall i = 1, \ldots, n$$

Estados de uma cadeia

Estado Recorrente Uma vez que se entra neste estado, um eventual retorno é assegurado.

Estado Absorvente Uma vez que se entra neste estado, nunca mais o deixa.

Estado Transiente Uma vez que se entra neste estado, o retorno não é garantido.

Estado Periódico O estado que pode somente ser alcançado nos passos $m, 2m, 3m$, onde $m > 1$ e $m \in \mathbb{N}$.

Estado Ergódico Uma vez que se entrou neste estado, um retorno é assegurado dentro de um número finito de passos, porém o estado não é periódico e pode voltar antes de qualquer passo n.

Tipos de cadeia

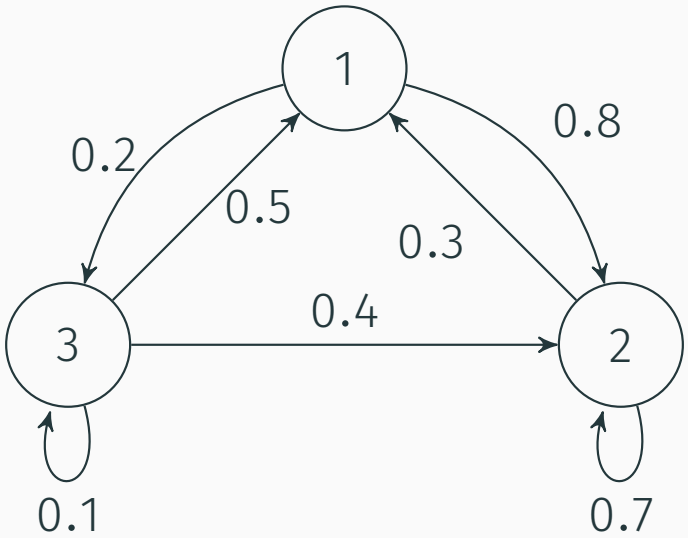
Cadeia Ergódica Todos os estados são recorrentes e aperiódicos.

Cadeia Irredutível Cada estado pode ser alcançado a partir de qualquer outro estado – todos os estados são comunicantes.

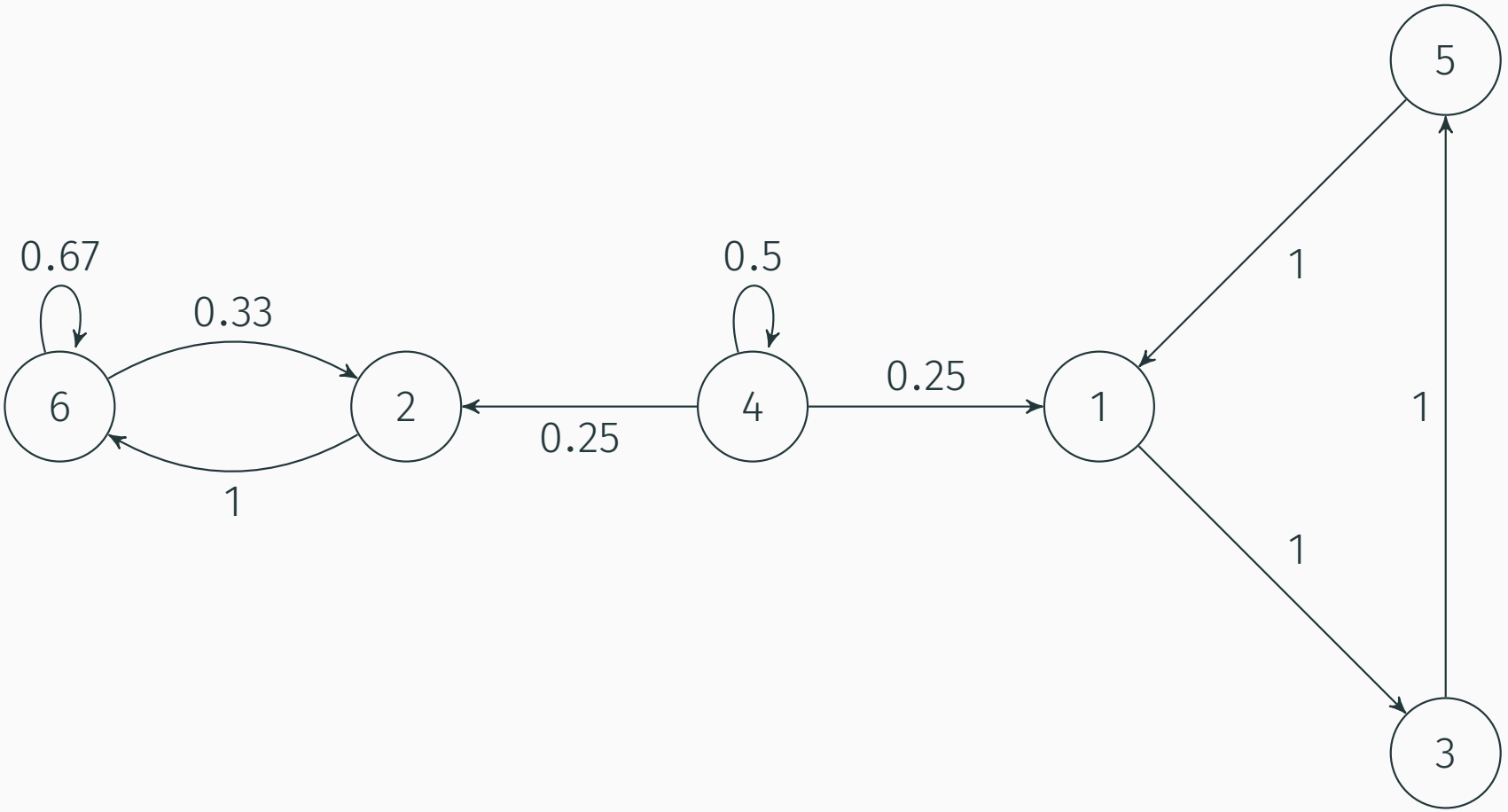
Cadeia Absorvente A cadeia contém um ou mais conjuntos fechados e o processo poderá eventualmente ser absorvido em um dos conjuntos fechados.

Conjunto Fechado Nenhum estado, a não ser algum pertencente ao conjunto, pode ser alcançado de qualquer de qualquer outro estado.

Exemplo: cadeia ergódica



Exemplo: cadeia não-ergódica



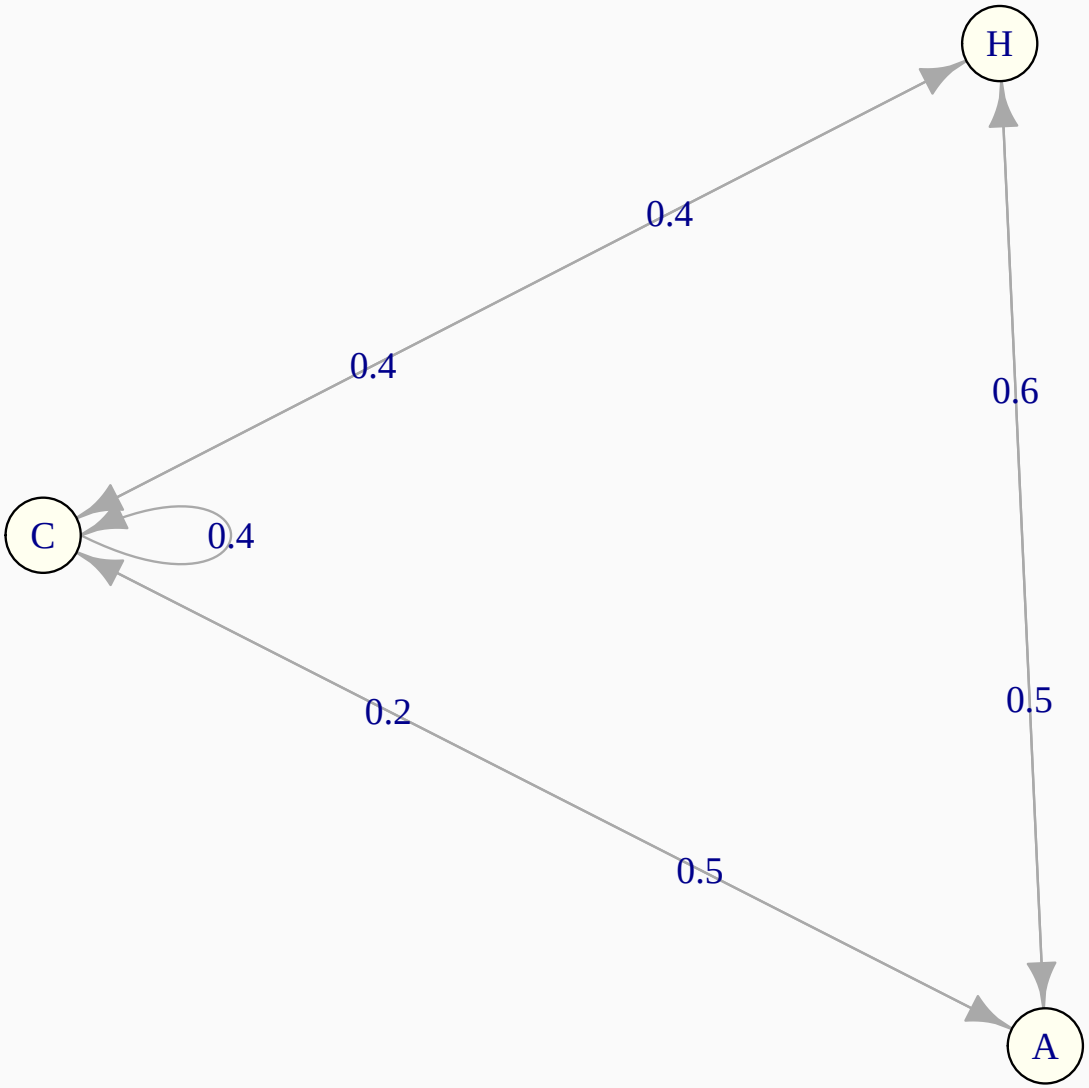
Trabalhando com CMs no R

Faremos uso do pacote **markovchain** para criar a representação adequada.

```
matriz_taxi <- matrix(c(0,   1/2, 1/2,
                        1/5, 2/5, 2/5,
                        3/5, 2/5, 0),
                      byrow = TRUE, ncol = 3)
cadeia_taxi <- new("markovchain",
                   transitionMatrix = matriz_taxi,
                   states = c("A", "C", "H"),
                   name = "Táxi")
cadeia_taxi
## Táxi
##  A  3 - dimensional discrete Markov Chain defined by the following states
##  A, C, H
##  The transition matrix  (by rows)  is defined as follows:
##     A   C   H
## A 0.0 0.5 0.5
## C 0.2 0.4 0.4
## H 0.6 0.4 0.0
```

Visualização da matriz de transição

```
plot(cadeia_taxi)
```



Estados

O pacote permite avaliar a natureza dos estados de uma cadeia com as seguintes funções:

- `transientStates()`: retorna quais os estados transientes na cadeia.
- `recurrentStates()`: retorna quais os estados recorrentes existem na cadeia.
- `absorbingStates()`: retorna quais os estados absorvetens existem na cadeia.

Estado estacionário

Uma pergunta que podemos fazer é sobre a condição do sistema em um tempo futuro. O que acontece quando:

$$\lim_{t \to \infty} P^t$$

Podemos fazer a operação com o auxílio do R para observar que para valores de t muito altos, as linhas se estabilizam.

Estado estacionário

De maneira equivalente, podemos resolver o seguinte sistema com uma restrição.

$$\pi = \pi P$$

$$\sum_{j=1}^{s} \pi_j = 1$$

O pacote disponibiliza a função `steadyStates()`, que retorna o vetor de estado estacionário de uma cadeia.

Tempo Médio de Recorrência, de Primeira Passagem

Outras duas coisas que podemos nos perguntar são:

1. Quanto tempo leva para retornarmos a um dado estado?
2. Sendo a cadeia ergódica, em quantos passos chegaremos a um outro estado?

Para tal, temos as funções `meanRecurrenceTime()` - que calcula o tempo para retornar a um dado estado. Também pode ser obtido pela equação $\frac{1}{\pi}$; e a função `meanFirstPassageTime` - que retorna o valor esperado de etapas para chegar a outros estados.

Exemplo: Táxi

```
# Estado estacionário
steadyStates(cadeia_taxi)
##              A         C        H
## [1,] 0.2682927 0.4268293 0.304878

# Tempo médio de recorrência
meanRecurrenceTime(cadeia_taxi)
##        A        C        H
## 3.727273 2.342857 3.280000

# Tempo médio de primeira passagem
meanFirstPassageTime(cadeia_taxi)
##          A        C   H
## A 0.000000 2.142857 2.2
## C 3.181818 0.000000 2.4
## H 2.272727 2.285714 0.0
```

Exemplo: Vendas por um call centre

Considere um processo de vendas por um call center.

1. Considerando os clientes sem interesse pelo produto: após uma ligação, 60% destes demonstram um baixo interesse pela compra, 30% um alto interesse, e 10% pedem o descadastro.
2. Considerando os clientes com baixo interesse: após uma ligação, 30% continuam com baixo interesse, 20% passam a ter um alto interesse, 30% fazem a aquisição do produto, e 20% solicitam o descadastro.
3. Considerando os clientes com alto interesse : 10% perdem um pouco do interesse, 40% mantém um alto nível de interesse, e 50% realizam a aquisição do produto.

Pergunta-se: Qual a probabilidade que um cliente venha a comprar o produto? Qual a probabilidade que um cliente solicite o descadastramento?

Exemplo: Processo de vendas de um calcenter

```
matriz_cc <- matrix(c(1,  0,  0, 0,  0,
                      .1, 0, .6, .3, 0,
                      .2, 0, .3, .2, .3,
                      0,  0, .1, .4, .5,
                      0,  0,  0,  0,  1),
                    byrow = TRUE,
                    ncol = 5)

cadeia_cc <- new("markovchain",
                 states = c("D", "I0", "IB", "IA", "Aq"),
                 transitionMatrix = matriz_cc,
                 name = "Call Center")
```

Tratando Cadeias Absorventes

Como a cadeia que representa o processo de vendas por um call centre não é ergódica, é necessária uma outra abordagem. Podemos reescrever a matriz de transição com a seguinte forma:

$$P^* = \begin{array}{c} \\ Tr \\ Ab \end{array} \begin{array}{c} \begin{array}{cc} Tr & Ab \end{array} \\ \begin{pmatrix} \boldsymbol{Q} & \boldsymbol{R} \\ \mathbf{0} & \boldsymbol{I} \end{pmatrix} \end{array}$$

Onde:

- $\boldsymbol{Q}$ é a matriz de transição (MT) entre os estados transientes.
- $\boldsymbol{R}$ é a MT dos estados transientes para os absorventes.
- $\boldsymbol{I}$ é a matriz identidade de transição entre os estados absorventes.
- $\mathbf{0}$ é uma matriz nula representando a não-comunicação entre os estados absorventes e transientes.

Cadeias Absorventes: Matriz fundamental

Com as novas matrizes acima, podemos calcular a matriz fundamental N.

Definimos N como $\left(I_n - Q\right)^{-1}$.

Multiplicando a matriz N pela R, obtemos a matriz de probabilidade B de cada estado transiente ser absorvido.

Respondendo às perguntas do Call Centre

```
cf_cc <- canonicForm(cadeia_cc)
transIdx <- which(states(cf_cc) %in% transientStates(cf_cc))
absIdx <- which(states(cf_cc) %in% absorbingStates(cf_cc))
mQ <- as.matrix(cf_cc@transitionMatrix[transIdx,transIdx])
mR <- as.matrix(cf_cc@transitionMatrix[transIdx,absIdx])
```

Respondendo às perguntas do Call Centre

```
# Matriz N
(N <- solve(diag(ncol(mQ)) - mQ))
##    I0    IB    IA
## I0  1 0.975 0.825
## IB  0 1.500 0.500
## IA  0 0.250 1.750

# Matriz B
(B <- N %*% mR)
##        D    Aq
## I0 0.295 0.705
## IB 0.300 0.700
## IA 0.050 0.950
```

Obrigado

Contato

- linkedin: `linkedin.com/in/souzapd`
- site: `souzapd.github.io`
- email: `souza.pd@outlook.com`

Referências


CRAN. 2019. URL: https://www.r-project.org/about.html. Acesso em 15 out. 2019.


James R Kirkwood. *Markov Processes*. CRC Press, 2015, p. 321.


Douglas R Montgomery, George C Runger e Norma Faris Hubele. *Engineering Statistics*. 5ª ed. John Wiley & Sons, 2010, p. 544.


R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2018. URL: https://www.R-project.org/.


RStudio. 2019. URL: https://rstudio.com/about/. Acesso em 15 out. 2019.


Deepayan Sarkar. *Lattice: Multivariate Data Visualization with R*. ISBN 978-0-387-75968-5. New York: Springer, 2008. URL: http://lmdvr.r-forge.r-project.org.


Luca Scrucca. “qcc: an R package for quality control charting and statistical process control”. Em: *R News* 4/1 (2004), pp. 11–17. URL: https://cran.r-project.org/doc/Rnews/.


Giorgio Alfredo Spedicato. “Discrete Time Markov Chains with R”. Em: *The R Journal* (jul. de 2017). R package version 0.6.9.7. URL: https://journal.r-project.org/archive/2017/RJ-2017-036/index.html.


Michael W Trosset. *An introduction to statistical inference and its applications with R*. Chapman e Hall/CRC, 2009.


Hadley Wickham. *R Packages*. 1ª ed. O’Reilly Media, 2015, p. 318.