

**WAGNER SILVA**

**O TESTE DE TUKEY NO SOFTWARE R: UMA NOVA  
ABORDAGEM VIA LINGUAGEM C++**

Monografia de conclusão de curso apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências deste departamento para obtenção do título de Bacharel em Ciência da Computação.

Orientador

Prof. Dr. Marcelo Silva de Oliveira

LAVRAS  
MINAS GERAIS - BRASIL

2003

**WAGNER SILVA**

**O TESTE DE TUKEY NO SOFTWARE R: UMA NOVA  
ABORDAGEM VIA LINGUAGEM C++**

Monografia de conclusão de curso apresentada  
ao Departamento de Ciência da Computação da  
Universidade Federal de Lavras como parte das  
exigências deste departamento para obtenção do  
título de Bacharel em Ciência da Computação.

APROVADA em \_\_\_ de \_\_\_\_\_ de \_\_\_\_\_.

---

UFLA  
Prof. Dr. Marcelo Silva de Oliveira  
Orientador

---

UFLA  
Prof. Dr. André Luís Zambalde

---

UFLA  
Prof. Dr. José Monserrat Neto

LAVRAS  
MINAS GERAIS - BRASIL

## RESUMO

**Título: O Teste de Tukey no Software R: Uma nova abordagem via Linguagem C++**

Neste trabalho apresenta-se a importância do software livre, para análises estatística, enfocando em especial o software denominado R. Pode-se constatar que o R tem sua potência consideravelmente aumentada, pois o R possui uma imensa capacidade de se conectar com outras linguagens, tais como Fortran, C e C++. Os principais softwares estatísticos, inclusive o software R, apresentam uma opção para a realização do Teste de Tukey, mas a saída convencional proporcionada por este software é, no entanto, confusa para um usuário inexperiente. Este trabalho propõe uma forma de apresentação dos resultados via descrição convencional do teste, ou seja, o uso de letras para identificar diferenças entre médias.

**Palavras-chave:** C++, R, Teste de Tukey.

## ABSTRACT

**Title: The Test of Tukey in the Software R: A new presentation using C++**

### **Language**

It's presented in this work the importance of free software, for statistical analysis, with a special focus on the software denominated R. We consider that the R has its potency increased considerably because the R has an immense capacity to connect to other programming languages as Fortran, C and C++. The principal statistical softwares including the R software, gives just one option for the realization Test of Tukey, but the conventional output of these softwares are confusing to an inexperienced user. This work proposes a form of result presentation through conventional test description, or say the use of letters to identify the differences between the averages.

**Keywords:** C++, R, Teste de Tukey.

## **DEDICO**

*A Deus*

*“Que morreu por nós, para que, quer vigiemos, quer durmamos, vivamos juntamente com ele.”*

*1 Tess 5:10*

*A minha querida mãe*

Se hoje conquisto algo para o meu futuro é porque você, minha querida mãe, foi a todo instante a força que precisei por meio de suas palavras de incentivo e otimismo.

*Aos meus queridos irmãos: Eny, José Carlos, Shirley e Sueli.*

“Fácil é demonstrar raiva e impaciência quando algo o deixa irritado. Difícil é expressar o seu amor a alguém que realmente te conhece, te respeita e te entende. Amo todos vocês.”

## **AGRADECIMENTOS**

À Universidade Federal de Lavras que proporcionou-me ensino superior público de qualidade possibilitando, dessa forma, minha formação acadêmica.

Aos professores desta Universidade, em especial aos dos Departamentos de Ciência da Computação e de Ciências Exatas.

Ao Professor Marcelo Silva de Oliveira pelo apoio didático e espiritual.

## LISTA DE TABELAS

<b>TABELA 1. CUSTO COM A LICENÇA DO SAS – UFLA NO PERÍODO DE 1997 A 2003. ....</b>	<b>11</b>
<b>TABELA 2. DADOS EXPERIMENTAIS ARRANJADOS DE ACORDO COM O MODELO DE BLOCOS CASUALIZADOS COM EFEITOS PARAMÉTRICOS DE TRATAMENTOS E BLOCOS, CONSIDERANDO ERROS NORMAIS, MÉDIA E VARIÂNCIA IGUAL A 100. ....</b>	<b>15</b>
<b>TABELA 3. PROCEDIMENTO PARA O CÁLCULO DOS VALORES INFERIOR E SUPERIOR DO ALGORITMO.....</b>	<b>15</b>
<b>TABELA 4. DESCRIÇÃO DOS VALORES INFERIOR E SUPERIOR PARA CADA MÉDIA .....</b>	<b>16</b>
<b>CONFORME O PROCEDIMENTO DESCRITO NA TABELA 7 .....</b>	<b>16</b>
<b>TABELA 5. REAJUSTE DO VALOR INFERIOR - FUNÇÃO ADJUSTLOWER.....</b>	<b>16</b>
<b>TABELA 6. INTERVALOS VÁLIDOS.....</b>	<b>17</b>
<b>TABELA 7. PROCEDIMENTOS PARA COMPILAR E LIGAR O CÓDIGO C++ COM O CÓDIGO R. ....</b>	<b>39</b>
<b>TABELA 8. EXEMPLO DE CÓDIGO C++ PARA EXECUÇÃO NO CÓDIGO R.....</b>	<b>39</b>
<b>TABELA 9. EXEMPLO DE CÓDIGO R PARA EXECUÇÃO DO CÓDIGO C++ DA TABELA 8.....</b>	<b>40</b>
<b>TABELA 10. NÚMERO DE COCHINILHAS VIVAS APÓS A APLICAÇÃO DE FUNGICIDA. ....</b>	<b>41</b>

## LISTA DE FIGURAS

<b>FIGURA 1. SUBSISTEMAS (PACOTES).....</b>	<b>17</b>
<b>FIGURA 2. DIAGRAMAS DE CLASSES.....</b>	<b>19</b>
<b>FIGURA 3. O AMBIENTE DO R NO SISTEMA OPERACIONAL WINDOWS.....</b>	<b>20</b>
<b>FIGURA 4. SAÍDA IMPLEMENTADA PARA O TESTE DE TUKEY... </b>	<b>42</b>



## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO .....</b>	<b>1</b>
<b>2</b>	<b>ESTATÍSTICA E COMPUTAÇÃO .....</b>	<b>3</b>
<b>2.1</b>	<b>Estatística Experimental.....</b>	<b>6</b>
2.1.1	Testes de médias.....	7
2.1.2	Teste de Tukey .....	8
<b>2.2</b>	<b>Software Estatísticos .....</b>	<b>9</b>
2.2.1	Software Comercial .....	9
2.2.2	Software Livre.....	12
<b>3</b>	<b>MATERIAIS E MÉTODOS.....</b>	<b>14</b>
<b>3.1</b>	<b>Materiais .....</b>	<b>14</b>
<b>3.2</b>	<b>Métodos.....</b>	<b>14</b>
<b>4</b>	<b>O SOFTWARE R.....</b>	<b>20</b>
<b>4.1</b>	<b>Considerações Iniciais.....</b>	<b>20</b>
4.1.1	O Ambiente do R .....	20
4.1.2	O CRAM.....	21
4.1.3	Aspectos Gerais do R.....	22
<b>4.2</b>	<b>Estatística no R.....</b>	<b>24</b>
<b>4.3</b>	<b>Programação no R.....</b>	<b>32</b>
4.3.1	Scripts .....	32
4.3.2	Funções .....	32
4.3.2.1	Controle de Fluxo.....	34
4.3.2.2	Controle de Repetição.....	36
<b>4.4</b>	<b>Compilação e ligação do código C++ com o código R.....</b>	<b>38</b>
<b>5</b>	<b>UM EXEMPLO PRÁTICO.....</b>	<b>41</b>

<b>5.1</b>	<b>O Problema .....</b>	<b>41</b>
<b>5.2</b>	<b>A Solução com o R.....</b>	<b>41</b>
<b>6</b>	<b>CONCLUSÕES .....</b>	<b>43</b>
<b>7</b>	<b>REFERÊNCIAS BIBLIOGRÁFICAS .....</b>	<b>44</b>
	<b>ANEXOS .....</b>	<b>46</b>

## 1 INTRODUÇÃO

Podemos perceber muitos casos em que o computador e a informática nos permitem abordar de um novo ponto de vista. Exemplos que surgem são as muitas aplicações em estatística (tratando com vasta quantidade de dados, análises estatísticas que demandam grande quantidade de cálculos, como por exemplo, análises geoestatísticas, análise exploratória de dados, análise multivariada, controle estatístico de processos, análise estatística por meio de gráficos) e em probabilidades (com todas as possibilidades abertas para a simulação através da geração de números pseudoaleatórios).

Mas, é sobretudo na análise que as oportunidades para utilizar a informática são mais ricas e mais numerosas. Exemplos, são o estudo das funções, na observação e estudo das sucessões e séries, cálculo integral, equações diferenciais e desenvolvimentos assintóticos. Em adição a estes campos onde o uso do computador aparece naturalmente, temos visto desenvolvimentos em novos campos que se desenvolveram amplamente devido aos computadores. O computador é não só uma ajuda para o cálculo e demonstração, mas também uma força para o desenvolvimento.

O acesso a bons softwares comerciais está profundamente limitado ao poder aquisitivo do usuário o que tem sido um problema de relevância sensível nos dias atuais, principalmente levando-se em consideração as leis de direitos autorais e suas respectivas autuações monetárias.

A opção do software livre tem sido cada vez mais considerada não somente para pessoas físicas, mas também para pessoas jurídicas. Na prática de análises estatísticas, sejam em pesquisa científicas ou em aplicações empresariais, uma das principais (senão a principal) alternativas de software livre na atualidade tem sido o Software R. Trata-se de um software voltado para a estatística disponibilizado sob os termos da GNU na forma de código aberto.

Neste contexto, o presente trabalho pretende demonstrar o uso do R para a estatística, além de implementar uma nova forma de apresentação dos resultados para o Teste de Tukey através da linguagem de programação C++.

## 2 ESTATÍSTICA E COMPUTAÇÃO

O Software R é o resultado da idéia de se aliar a teoria Estatística e o poder dos modernos computadores. Para se entender plenamente a junção destes dois campos (Estatística e Computação), é necessário antes recuperação de algumas considerações feitas em anos anteriores.

Segundo Dachs (1988),

**O papel do computador nos trabalhos de estatística é, hoje, muito grande. Tem crescido muito nos últimos vinte anos, acelerou-se mais nos cinco últimos, devido ao aparecimento e difusão dos microcomputadores, e a tendência é a de aumentar ainda mais.**

**Hoje é quase impossível desenvolver análises estatísticas de dados sem o uso do computador e de software adequado. Mesmo para conjuntos de dados de tamanhos pequeno e médio, é necessário recorrer ao computador, pois grande parte das técnicas inclui processos iterativos e são numericamente sofisticadas.**

**Como ferramenta de trabalho em pesquisa em estatística, o papel do computador também tem crescido muito. Seja em estudos de Monte Carlo, seja como ferramenta para investigações heurísticas, o computador esta cada vez mais presente no processo de busca de novas técnicas e metodologias, e até na formulação e solução de problemas teóricos. Talvez o trabalho mais interessante que apareceu na literatura sobre esse aspecto seja o de Efron (1979). Eddy (1986) coordenou um grupo de trabalho sobre o uso do computador na pesquisa em estatística, com a participação ativa de outros nove especialistas. Os trabalhos desse grupo, incluindo as discussões, são uma fonte importante de idéias sobre o assunto. Num apêndice são também apresentados os recursos disponíveis em diversos departamentos de estatística de universidades americanas. No livro editado por Kahn (1980) há vários trabalhos mostrando o uso do computador como ferramenta de pesquisa em áreas de probabilidades e atuária.**

Existem hoje vários periódicos científicos dedicados exclusivamente (ou com grande ênfase) à área de estatística computacional, como o *Journal Of Statistical Computation* e o *Communications in Statistics*, série B. Outros, ainda, tem sistematicamente seções dedicadas à estatística computacional, ou a algoritmos estatísticos como: *The American Statistician e Applied Statistics (Journal of the Royal Statistical Society*, serie C).

O número de pacotes computacionais para análise estatística tem crescido muito e a tendência é aumentar ainda mais nos próximos anos. Uma nova perspectiva está começando a abrir-se com a possibilidade de se desenvolverem sistemas especialistas. Existe hoje uma linguagem computacional voltada para a solução de problemas estatísticos, o S, parente da linguagem C, desenvolvida originalmente nos laboratórios da Bell e atualmente em uso em várias universidades e instituições de pesquisa.

É muito difícil apresentar um panorama histórico do desenvolvimento do uso do computador em estatística. O início foi lento, como quase sempre, e somente no final dos anos 60 o computador começou de fato a representar um papel importante no uso e desenvolvimento da estatística. Alguns artigos mostram o panorama nessa época, como os de Box (1969) e Tukey (1968). Francis (1981) apresenta uma resenha e comparação de pacotes existentes até o fim da década de 70.

A partir dos anos 70 o desenvolvimento acelerou-se rapidamente, e começaram a aparecer livros sobre estatística computacional. Apenas para citar alguns: Afifi; Azen (1979). Livro sobre o uso do computador e pacotes de forma genérica em várias áreas de estatística, 1979.

Chambers. Texto geral, com discussão de projeto e estrutura de dados e programas em estatística, bem como de avaliação e uso de programas e pacotes, 1977.

Cooke et al. Livro elementar com programas em BASIC. Apesar de ser introdutório apresenta bons conceitos e tem algumas sugestões e informações interessantes, 1981.

Groeneveld. Apareceu dois ou três anos antes do que deveria. Trata-se de um livro para ensinar probabilidade e estatística com exemplos e exercícios em BASIC. Se tivesse

surgido depois da grande difusão dos microcomputadores seu impacto poderia Ter sido maior, 1979.

Hemmerle. Foi o pioneiro. Vale a pena examiná-lo para compará-lo com os livros mais novos, 1967.

Kennedy e Gentle. Texto amplo, cobrindo muito aspectos da área. Em geral, difícil de ler, com muitas demonstrações incompletas, estilo desagradável e pedante. Ainda assim, indispensável em muitos aspectos de estatística computacional, apesar dos vários erros, 1980.

Maindonald. Dedicado em grande parte a problemas e algoritmos em regressão , modelos lineares em geral, e um pouco de metodologia multivariada. Muitos exemplos de implementação em FORTRAN. 1980.

Krishnaiah e Gentle(a aparecer). Um dos volumes da enciclopédia editada pelo primeiro autor. Deve trazer muitas novidades.

Uma indicação da importância que adquiriu a área de estatística computacional são os freqüentes simpósios sobre o assunto. Nas reuniões do *International Statistical Institute* e da *American Statistical Association* há sempre sessões sobre o tema e , anualmente, realiza-se o *Computer Science & Statistics: Symposium on the Interface*. No ano de 1987 ocorreu o 19º.

O valor do computador na Estatística foi realçado de um modo marcante por Bradley Efron (1979) , em seu famoso artigo “Computadores e Estatística: pensando o impensável”:

**O propósito da Teoria Matemática é de fato, de toda teoria científica é simplificar situações complexas ... O advento de computadores velozes tem redefinido o significado da palavra “simples” nas ciências matemáticas. Por exemplo, um problema de otimização que é reduzido a um problema de programação linear, é freqüentemente, considerado resolvido, desde que o Método Simplex é numericamente eficiente... O mesmo processo trabalha na Estatística Matemática ... Um a teoria auxiliada por computador não é menos “matemática” do que as teorias do passado, ela é apenas menos restringida pelas limitações do cérebro humano... O caminho para o sucesso é claro... Uma mistura de pensamento matemático tradicional**

## **combinado com o poder numérico e organizador do computador.**

Pode-se perceber a importância do computador na estatística conforme Dachs e Bradley Efron. A utilização da computação na estatística, seja em simulações ou em investigações heurísticas, onde a busca exaustiva é inviável, a computação cumpri seus objetivos que é o de trazer soluções viáveis, com baixo custo e tempo reduzido.

### **2.1 Estatística Experimental**

Um dos mais fundamentais interesses da análise de dados é a comparações entre tratamentos. Um tratamento é uma condição homogênea aplicada distintamente sobre um grupo de indivíduos (pessoas, plantas, animais, peças, etc), que pode alterar-lhes suas características. Por exemplo, considere quatro rações diferentes para engorda de frangos. O tratamento é o tipo de ração, pois cada uma delas pode levar a uma taxa de peso diferenciado. O objetivo de um experimento deste tipo seria comparar as rações, para verificar qual delas proporcionam maior ganho de peso no mesmo tempo.

A técnica fundamental para proceder nesta comparação é a Análise de Variância (ANAVA), a qual consiste de basicamente duas etapas: o teste F é um teste de comparações múltiplas.

O Teste F é um teste estatísticos que informa se há, ou não há, pelo menos um tratamento diferente dentre todos os tratamentos aplicados.

Assim, tomando o exemplo anterior, o teste F dirá apenas se:.

- Todos as rações são iguais entre si, ou
- Pelo menos uma é diferente das outras.



O teste F não diz qual é diferente de qual: ele apenas informa se há, ou se não há, alguma diferença entre os tratamentos. A discriminação entre os tratamentos é feita pela etapa seguinte.

### **2.1.1 Testes de médias**

Para discutir sobre a filosofia das comparações múltiplas, o aspecto mais importante é Ter sempre em mente que toda inferência realizada esta sujeita a erros. Esses erros podem ser classificados em três categorias. O primeiro deles é o denominado erro tipo I, que refere-se a probabilidade ( $\alpha$ ) de rejeitar uma hipótese quando ela é verdadeira O segundo, erro tipo II , refere-se à probabilidade de ( $\beta$ ) de aceitar uma hipótese como verdadeira quando de fato, ela é falsa. Para o erro tipo I, nos procedimentos de comparações múltiplas entre outras, existem duas formas de medir esse erro. A primeira refere-se à avaliação da probabilidade de se rejeitar uma hipótese verdadeira em todas as possíveis combinações dos níveis dos tratamentos tomados dois a dois, sendo conhecida por taxa de erro tipo I por comparação (comparisonwise ou per-comparison error rate). A Segunda forma é a mediada do erro tipo I como a probabilidade de se realizar pelo menos uma inferência errada por experimento e é conhecida por taxa de erro tipo I por experimento (experimentwise error rate).

Finalmente, o terceiro tipo de erro ao se realizar uma inferência, conhecido como erro tipo III refere-se à probabilidade de classificar um nível de tratamento como superior ao outro, quando de fato o segundo nível supera o primeiro.

Existem vários testes para comparações múltiplas. Eles diferem fundamentalmente na filosofia de controle do erro tipo I, ou seja, testes tais, como o de Duncan e o LSD (baseado na distribuição de t de student) não controlam a taxa de erro por experimento, mas controlam a taxa de erro por comparação. Por outro lado, os testes como o de Tukey e Scheffé controlam

adequadamente as taxas de erro por experimento e por comparação, preservando o nível nominal de significância ( $\alpha$ ).

Outros testes utilizados são o Teste de Bonferroni, Student-Kenewman-Keuls (SNK) e o Scott e Knott (SK).

Cada um destes testes tem algumas características que devem ser observados para que eles sejam aplicados, começando pelo controle dos erros tipo I e II citados anteriormente.

O Teste de Tukey é um dos mais utilizados, devido ao bom controle dos erros citados e a relativa simplicidade de aplicação.

### 2.1.2 Teste de Tukey

O Teste de Tukey, baseado na amplitude total estudentizada (“studentized range”) pode ser utilizado para comparar todo e qualquer contraste ente duas médias de tratamentos. Considere o caso em que há quatro tratamentos, existe, portanto seis contrastes distintos que podem ser estudados.

$$C_{4,2} = \frac{4!}{2!2!} = 6$$

O teste é exato e de uso muito simples quando o número de repetições é o mesmo para todos os tratamentos, o que admitiremos.

Começaremos por calcular o valor de  $\Delta = q \cdot \frac{s}{\sqrt{r}}$ , onde q é o valor da amplitude total estudentizada ao nível de 5% ou de 1% de probabilidade; s é a estimativa do desvio padrão residual, e r é o numero de repetições, suposto 6 para todos os tratamentos. Neste caso tem-se n = 4 tratamentos, n' = 20 graus de liberdade para o resíduo, e o valor q ao nível de 5% de probabilidade é, pois, 3,96 (este valor pode ser obtido por tabelas disponíveis em livros, ou calculado pelo software).

Tem-se:

$$\Delta = 3,96 \cdot \frac{1,20}{\sqrt{6}} = 1,94$$

Então todo contraste entre duas media, isto é, do tipo

$$Y = m_i - m_u$$

cuja estimativa exceder o valor  $\Delta = 1,94$  será significativo ao nível de 5% de probabilidade. Tal acontece com a diferença entre a maior e a menor das medias amostrais ( $\hat{m}$ ) achadas.

Considere que

$$\hat{Y} = \hat{m}_1 - \hat{m}_3 = 26 - 22,8 = 3,2$$

Se adotarmos o nível de 1% de probabilidade, teremos:

$$\Delta = 5,02 - \frac{1,2}{\sqrt{6}} = 2,46$$

E este mesmo contraste será ainda significativo neste nível.

## 2.2 Software Estatísticos

### 2.2.1 Software Comercial

**Aptech Systems** - Home Page do GAUSS, um pacote de software matemático e estatístico incluindo uma poderosa língua de programação, para o DOS, o Windows, o OS/2 e plataformas Unix. Homepage: <http://www.aptech.com/>

**Cytel software** - Software estatístico para inferência exata como StatXact, Proc-StatXact para o SAS, LogXact e EaSt. Homepage: <http://www.cytel.com/>

**GraphPad** - GraphPad Prism para Windows.

**Idea Works** - Vários sistemas de software, incluindo o Statistical Navigator para DOS/Win. Homepage: <http://www.graphpad.com/welcome.htm>

**Ivation Datasystems** - Beyond 20/20 software para Windows. Homepage:  
<http://www.ivation.com/>

**MathSoft** - Produtos de Softwares: MathCAD, Axum, S-Plus. Homepage:  
<http://www.mathsoft.com/>

**MathWorks** - Software MATLAB. Homepage: <http://www.mathworks.com/>

**Minitab** - Software Minitab, FAQs, macros gratuitas, suporte técnico e algum material para a comunidade estatística como mailing lists. Homepage:  
<http://www.minitab.com/>

**NAG - The Numerical Algorithms Group** - Software numérico e bibliotecas, Fortran 90, GenStat, Glim, MLP, AXIOM, Iris Explorer. Homepage:  
<http://www.nag.co.uk/>

**NCSS Statistical Software** - Cópia gratuita de avaliação e calculador de probabilidade da NCSS gratuito. Homepage: <http://www.ncss.com/>

**NWP Associates** - STATLETS - Java applets para análise estatística e gráficos, gratuito, mas com versões acadêmicas disponíveis limitadas. Homepage:  
<http://www.statlets.com/>

**SAS** - Sistema SAS em muitas plataformas, JMP, compilador C. Leque variado de material do SAS, vista geral do software, e serviços de suporte incluindo uma base de dados pesquisável por SAS Notes e documentos de suporte técnico. Homepage: <http://www.sas.com/>

A Tabela 1 mostra os gastos do DEX- Departamento de Exatas referentes ao pagamento da licença entre o período de 1997 à 2003 para alguns módulos do Software.

**Tabela 1. Custo com a Licença do SAS – UFLA no período de 1997 a 2003.**

**PACOTES/ SAS**

<b>Data</b>	<b>IML</b>	<b>GRAPH</b>	<b>ACCPCFF</b>	<b>BASE</b>	<b>STAT</b>	<b>ETS</b>	<b>TOTAL</b>
*	379,5	506	506				<b>1.391,50</b>
**	563,64	563,64	563,64	577,06	563,64	563,64	<b>3.395,26</b>
***	900,6	900,6	900,6	921,5	900,6	900,6	<b>5.424,50</b>
****	1185	1185	1185	1212,5	1185	1185	<b>7.137,50</b>
*****	498,07	498,07	498,07	509,63	498,07	498,07	<b>2.999,98</b>

\* Novembro 1997 à Novembro 1998

\*\* Novembro 1998 à Novembro 1999

\*\*\* Novembro 2000 à Novembro 2001

\*\*\*\* Novembro 2001 à Outubro 2002

\*\*\*\*\* Novembro 2002 à Outubro 2003

**SGC - Statistical Graphics Corporation** - STATGRAPHICS - pacote da análise estatística para MS Windows. Homepage: <http://www.sgcorp.com/>

**SHAZAM** - SHAZAM, software disponível para variados PC, estações de trabalho e plataformas mainframe. Homepage: <http://www.sgcorp.com/>

**SPSS** - Software SPSS proprietário de produtos como o Systat e BMDP. Homepage: <http://www.spss.com/>

**Stata** - Stata para DOS/Win/Mac/Unix. Homepage: <http://www.stata.com/>

**Stat-Ease** - Design-Ease e Design-Expert para Windows, software para o planejamento de experiências. Homepage: <http://www.statease.com/>

**Statistical Solutions** - Distribuidor de software estatístico, incluindo Solas para análise de dados omisso enQuery Advisor para planejamento de experiências (versões de demonstração disponíveis). Homepage: <http://www.statsol.ie/>

**StatSci** - S-Plus software. Homepage: <http://www.statsci.com/>

**StatSoft** - Statística, pacote de software estatístico para DOS/Win/Mac.  
Homepage: <http://www.statsoftinc.com/>

**Unistat** - Pacote Estatístico UNISTAT do Windows com características add-in para MS-Excel ou MS-Office, demo disponível.

Homepage: <http://www.unistat.com/>

**Visual Numerics** - Produtos: PV-WAVE, IMSL (bibliotecas numéricas e gráficas de C e Fortran), Stanford Graphics.

Homepage: <http://www.vni.com/index.html>

### 2.2.2 Software Livre

**ARfit** - Coleção gratuita de rotinas atlab, funcionando em Unix, por de T.Schneider e A.Neumaier.

Homepage: <http://solon.cma.univie.ac.at/~neum/software/arfit/>

**BMDP Statistical Software** - Algum freeware como o SOLO probability calculator (o mesmo que o calculador da probabilidade de NCSS) e o BMDP/DIAMOND LE. Homepage: <http://www.bmdp.com/>

**Decision Analyst** - STATS -Software Estatístico gratuito para pesquisa de mercado, funcionando em IBM PC Windows.

Homepage: <http://www.decisionanalyst.com/download.htm>

**Click & Learn Regression** - Software iterativo para análise de regressão linear, versão de demonstração disponível.

Homepage: <http://www.learnregression.com/>

**MD Anderson Cancer Center Biomathematics** - Variado software estatístico especializado, freeware disponível, programas DOS ou Mac, incluído software  
Homepage: STPLAN. <http://odin.mdacc.tmc.edu/anonftp/>

**R** - é um ambiente para computação estatística. É um projeto de GNU muito similar à linguagem e ao ambiente de S. R fornece uma ampla variedade de técnicas estatísticas e gráficas e é altamente extensível. Compila e funciona em uma ampla variedade de plataformas de UNIX e Linux. Também compila e funciona em Windows 9x/NT/2000 e em MacOS. Homepage: <http://www.r-project.org>

**SISVAR** – Programa de análises estatísticas (Statistical Analysis Software) e planejamento de experimentos. O programa Sisvar foi desenvolvido principalmente com finalidades didáticas. Atualmente, serve como suporte para as disciplinas de estatística experimental e estatística básica dos cursos regulares de graduação da Universidade Federal de Lavras. A versão atual do Sisvar para Windows vem sendo largamente utilizada pelos pesquisadores da Universidade Federal de Lavras, bem como, de outras universidades e instituições de pesquisa. Os alunos do curso de tutoria (especialização), bem como outros alunos e pesquisadores da UFLA e de outras instituições podem baixar uma cópia do programa. Juntamente com o programa pode se efetuar o download do manual em PDF. Homepage: <http://www.dex.ufla.br/danielff/>

**XTREMES** - software estatístico para análise de dados extremos (versões para Windows), incluído no livro "Statistical Analysis of Extreme Data", by R.-D. Reiss and M.Thomas (1997).

Homepage: <http://www.xtremes.math.uni-siegen.de/>

### **3 MATERIAIS E MÉTODOS**

#### **3.1 Materiais**

O presente trabalho foi desenvolvido no DEX - Departamento de Ciências Exatas da Universidade Federal de Lavras utilizando-se o software livre R instalado em dois computadores com a seguinte descrição:

- Computador com processador Pentium III, 128 MB de RAM e Sistema Operacional Windows 98, 2ª edição;
- Computador com processador AMD ATHON 1.8, 512 MB de RAM e Sistemas Operacionais Linux Red Hat 7.3 e Windows 98, 2ª edição.

O delineamento de pesquisa utilizado foi preponderantemente pesquisa bibliográfica em livros de Estatística , livros de programação C++ e manuais do R, obtidos principalmente na Internet.

O ambiente de programação utilizado foram C++ e R, ambos para Linux e Windows. A condução da pesquisa baseou-se em seminários semanais com o orientador, onde discussões sobre o software e aplicações específicas na Estatística foram realizadas.

#### **3.2 Métodos**

A Tabela 2 refere-se a dados simulados segundo Ramalho, Ferreira & Oliveira (2000).



**Tabela 2. Dados experimentais arranjados de acordo com o modelo de blocos casualizados com efeitos paramétricos de tratamentos e blocos, considerando erros normais, média e variância igual a 100.**

Tratamentos	Blocos			
	1	2	3	4
1	87.266	70.833	73.995	101.898
2	98.233	81.401	98.238	85.106
3	66.402	81.157	90.298	83.986
4	78.002	75.248	106.960	85.740
5	78.788	90.352	101.393	115.510
6	90.574	81.478	95.684	116.209
7	98.646	101.155	117.566	113.388
8	114.673	108.203	134.400	124.469
9	109.815	128.579	117.145	140.229
10	109.025	130.849	121.789	132.305

**Fonte: Ramalho, Ferreira & Oliveira (2000).**

Foi criada uma função que calcula um intervalo baseado na média e na diferença mínima significativa com base na amplitude estudentizada (DMS de Tukey). O procedimento desta função está apresentado na Tabela 3.

**Tabela 3. Procedimento para o cálculo dos valores inferior e superior do algoritmo**

<b>X inferior</b>	lower = mean – DMS
<b>X superior</b>	upper = mean

**Fonte: Dados do trabalho**

A função InsertOrdered é responsável por inserir, em uma estrutura de dados lista duplamente encadeada, em ordem decrescente tomando a média como referência, conforme a descrito na tabela 4.

**Tabela 4. Descrição dos valores inferior e superior para cada média conforme o procedimento descrito na Tabela 7**

<b>Mean</b>	<b>Lower</b>	<b>Upper</b>
123.942	99.7558	123.942
123.492	99.3058	123.492
20.436	96.25	120.436
107.689	83.5025	107.689
96.5108	72.3245	96.5108
95.9862	71.8	95.9862
90.742	66.5558	90.742
86.4875	62.3013	86.4875
83.498	59.3118	83.498
80.4608	56.2745	80.4608

**Fonte: Dados do trabalho**

A função AdjustLower(), resultado na Tabela 5, faz um reajuste no valor lower percorrendo a estrutura de dados lista do final para o início até encontrar uma média maior que o lower avaliado.

**Tabela 5. Reajuste do valor inferior - função AdjustLower.**

<b>mean</b>	<b>Lower</b>	<b>Upper</b>
123.942	107.689	123.942
23.492	107.689	123.492
120.436	96.5108	120.436
107.689	86.4875	107.689
96.5108	80.4608	96.5108
95.9862	80.4608	95.9862
90.742	80.4608	90.742
86.4875	80.4608	86.4875
83.498	80.4608	83.498
80.4608	80.4608	80.4608

**Fonte: Dados do trabalho**

Em seguida uma função de nome MeetNode() é responsável por marcar o intervalo como válido ou inválido, conforme a Tabela 6, sendo considerado um intervalo válido todo aquele que não está dentro de um outro intervalo.

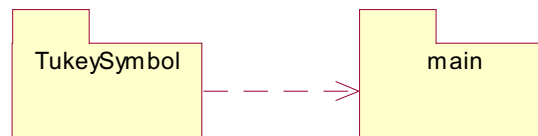
**Tabela 6. Intervalos válidos.**

Mean	Lower	Upper
123.942	107.689	123.942
120.436	96.5108	120.436
107.689	86.4875	107.689
96.5108	80.4608	96.5108

**Fonte: Dados do trabalho**

A função InsertSymbol() tem o objetivo de inserir o mesmo símbolo para os intervalos válidos. Finalmente, PrintTableTukey() é responsável por imprimir a saída implementada.

A figura 1 e a figura 2 utiliza os eventos de modelagem de modelagem da UML segundo BOOCH (1999). A figura 2 representa a relação de subsistemas. O subsistema TukeySymbol representa o código-fonte R e o subsistema main é a implementação do código-fonte C++. Entre eles há uma relação de dependência.



**Figura 1. Subsistemas (Pacotes).**

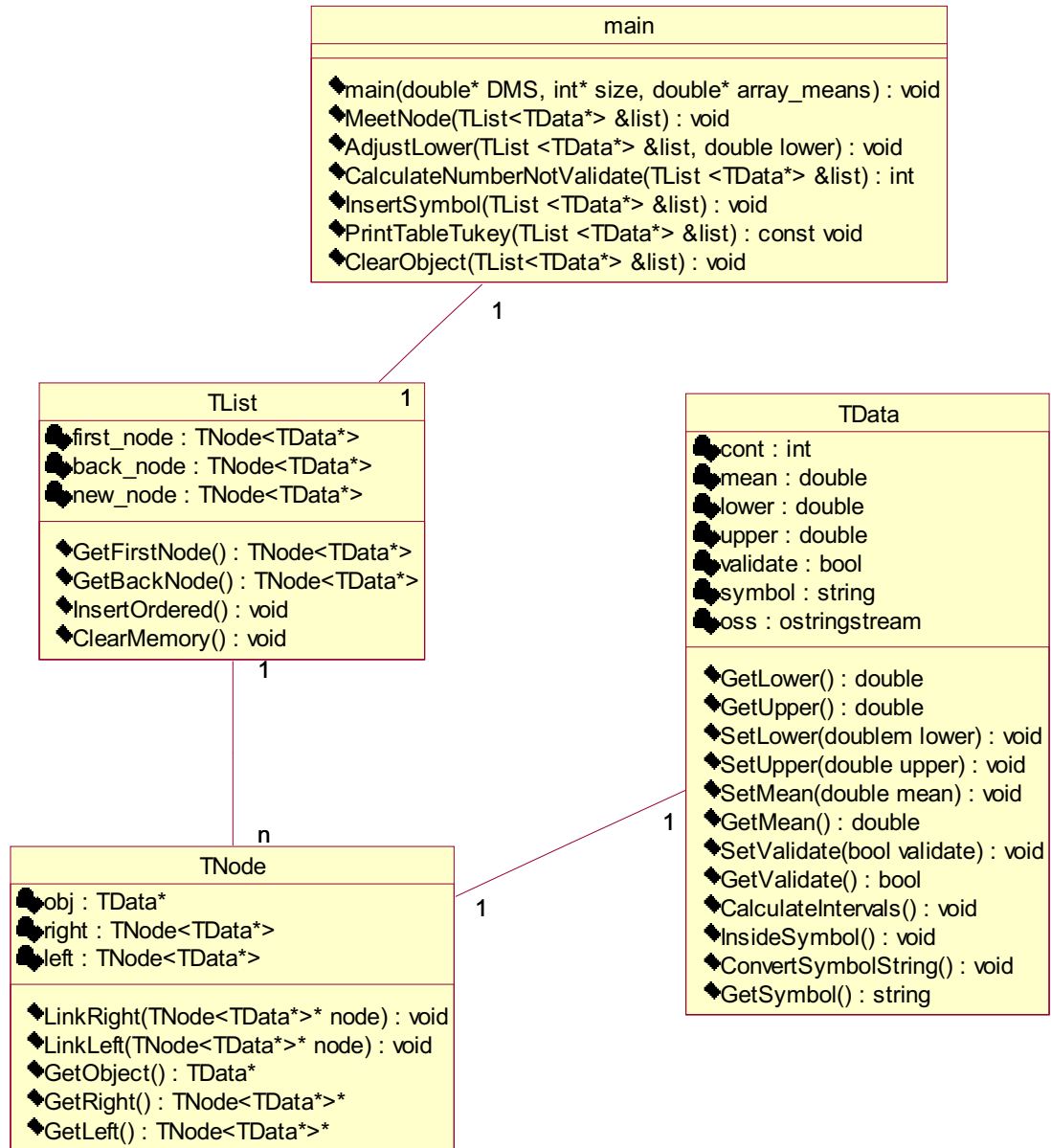
**Fonte: Software Rational Rose**

Na classe main estão listadas as seguintes funções: void main(double\* DMS, int\* size, double\* array\_means), void MeetNode(TList<TData\*> &list), double AdjustLower (TList <TData\*> &list, double lower), int CalculateNumberNotValidate(TList <TData\*> &list), void InsertSymbol(TList <TData\*> &list), const void PrintTableTukey(TList <TData\*> &list), void ClearObject(TList<TData\*> &list).

Uma classe main possui uma única instância do tipo TList relacionadas na classe TList com as seguintes funções: TNode<TData\*>\* GetFirstNode(), TNode<TData\*>\* GetBackNode(), void InsertOrdered(TData\* value), void ClearMemory().

Uma classe TList pode definir n instancias da classe TNode com as seguintes funções: TNode: TData\* GetObject(); TNode< TData\*>\* GetRight(), TNode<TData\*>\*GetLeft(), void LinkRight(TNode< TData\*>\* node), void LinkLeft(TNode< TData\*>\* node).

TNode possui uma relação de 1 para 1 com a classe TData que possui as seguintes funções: double GetLower(), double GetUpper(), void SetLower(double lower), void SetUpper( double upper ); void SetMean( double mean ), double GetMean(), void SetValidate(bool validate); bool GetValidate(); void CalculateIntervals(double mean, double\* DMS ), void InsideSymbol(string character, int digit), void ConvertSymbolString(), string GetSymbol().



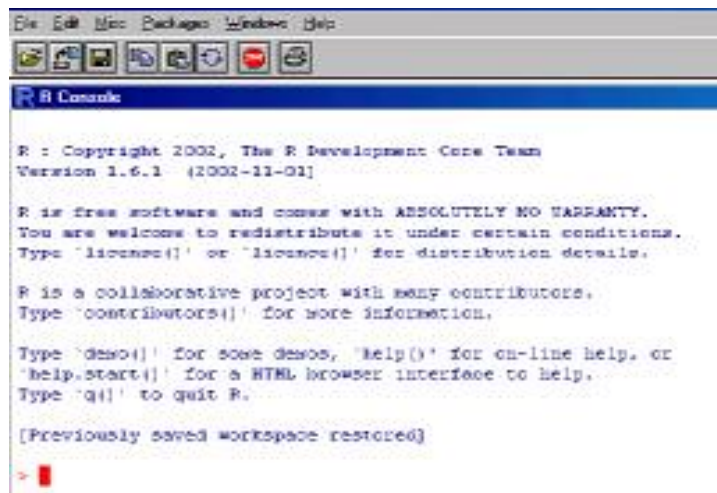
**Figura 2. Diagramas de Classes**  
**Fonte: Software Rational Rose**

## 4 O SOFTWARE R

### 4.1 Considerações Iniciais

#### 4.1.1 O Ambiente do R

Muitos usuários tratam o R simplesmente como um software estatístico. Na verdade, o R é um ambiente de programação propício as mais diversificadas áreas. Neste contexto, pode-se citar atualmente a estatística, não só, como alvo de pesquisa, mas talvez um dos principais meios de divulgação deste software. O conjunto de técnicas estatísticas, tais como: geoestatística, modelos generalizados etc... formam as bibliotecas (library), popularmente conhecidas como pacotes, na instalação do programa R aproximadamente oito bibliotecas são fornecidas, porém, o usuário tem total flexibilidade em instalar outras bibliotecas. No caso das plataformas a serem executadas o R, atualmente se encontra as mais diversificadas, ou seja, desde as plataformas Unix e de sistemas similares como o Linux e também na versão Windows NT/9x/2000.



```
R : Copyright 2002, The R Development Core Team
Version 1.6.1 (2002-11-01)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for a HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

>
```

**Figura 3. O ambiente do R no sistema operacional Windows.  
Fonte: Software R Versão 1.6.1**

### 4.1.2 O CRAM

CRAN é uma rede do ftp<sup>1</sup> (*File Transfer Protocol*) e dos usuários da correia fotorreceptora em torno do mundo que armazenam idênticas e modernas versões do código e documentação para o pacote estatístico do R. A rede detalhada do arquivo do R está disponível nos seguintes URLs (*Uniform Resource Locator*)<sup>2</sup>.

Austrália	<a href="http://cran.au.r-project.org">http://cran.au.r-project.org</a>	
	<a href="http://mirror.aarnet.edu.au/pub/CRAN">http://mirror.aarnet.edu.au/pub/CRAN</a>	
Áustria	<a href="http://cran.at.r-project.org">http://cran.at.r-project.org</a>	
Brasil	<a href="http://cran.br.r-project.org">http://cran.br.r-project.org</a>	(UFPR)
	<a href="http://www.termix.ufv.br/CRAN">http://www.termix.ufv.br/CRAN</a>	(UFV)
Hungria	<a href="http://cran.hu.r-project.org">http://cran.hu.r-project.org</a>	
Japão	<a href="ftp://ftp.u-aizu.ac.jp/pub/lang/R/CRAN/">ftp://ftp.u-aizu.ac.jp/pub/lang/R/CRAN/</a>	
África Sul	<a href="http://cran.za.r-project.org">http://cran.za.r-project.org</a>	
Switzerland	<a href="http://cran.ch.r-project.org">http://cran.ch.r-project.org</a>	
Reino Unido	<a href="http://cran.uk.r-project.org">http://cran.uk.r-project.org</a>	
Estados Unidos	<a href="http://cran.us.r-project.org">http://cran.us.r-project.org</a>	
Los Angeles, CA	<a href="http://www.bioconductor.org/CRAN/">http://www.bioconductor.org/CRAN/</a>	
Boston	<a href="http://lib.stat.cmu.edu/R/CRAN/">http://lib.stat.cmu.edu/R/CRAN/</a>	

---

<sup>1</sup> Protocolo de Transferência de Arquivos. Formalizado em 1973, é o processo pelo qual o usuário pode ter acesso a inúmeros “depósitos” de arquivos (texto, imagens, sons e programas) situados em computadores remotos de instituições públicas e privadas.

<sup>2</sup> Tipos de endereços de recursos da Internet, entre eles estão os endereços dos sites da Web.

### 4.1.3 Aspectos Gerais do R

O R armazena seus dados em um arquivo chamado *.RData* que permanece gravado em disco entre sessões do R. Pode-se ter múltiplos arquivos *.RData* em diferentes diretórios. Basta rodar o R em um determinado diretório para que o arquivo seja criado.

Desta forma aconselha-se que sempre que iniciar um novo projeto seja criado um novo diretório e neste diretório seja iniciado o programa R.

No WINDOWS: inicie o R e defina o diretório de trabalho clicando em FILE → CHANGE WORKING DIRECTORY.

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

```
$ mkdir Project
$ cd Project
$ R
```

O programa R será inicializado mostrando a seguinte mensagem:

```
R : Copyright 2003, The R Development Core Team
Version 1.6.2 (2003-01-10)
```

R is free software and comes with ABSOLUTELY NO WARRANTY.

You are welcome to redistribute it under certain conditions.

Type ``license()'` or ``licence()'` for distribution details.

R is a collaborative project with many contributors.

Type ``contributors()'` for more information.

Type ``demo()'` for some demos, ``help()'` for on-line help, or



`help.start()' for a HTML browser interface to help.

Type `q()' to quit R.

>

O sinal > é o prompt da linha de comando, indicando que o R está pronto para começar a receber comandos.

No R, ao contrário de outros softwares estatísticos para Windows, as tarefas requeridas pelo analista não estão todas comodamente dispostas nos menus ao alcance do mouse. Essas tarefas, como manipular seus dados, fazer determinada análise estatística ou desenhar um gráfico, devem ser requeridas na linha de comando na janela *Commands*. Embora em princípio isso possa parecer uma desvantagem, o uso mais avançado do R mostrará que essa orientação é vantajosa, pois permite imensa flexibilidade de uso e expansão das capacidades do software.

O R tenta interpretar tudo o que é digitado na linha de comando (seguido de ENTER ).

Veja estes testes:

```
> 2+2          # Assumiui a tarefa de calculadora
[1] 4
> 2 < 4 # Avaliou a expressão e respondeu que é verdadeira (T)
[1] TRUE
> 2 > 4 # Avaliou a expressão e respondeu que é falsa (F)
[1] FALSE
> pi          # Retornou o valor da constante matemática pi
[1] 3.141593
```

Um objeto R é criado como o resultado de uma expressão R. Para salvar o objeto resultante, simplesmente associe um nome ao objeto usando operadores de atribuição <-, \_ ou ->.

Por exemplo, os comandos seguintes criam um objeto x que contém o inteiro 10:

```
> x <- 10
ou
> x_ 10
ou
10 -> x
```

Para ver o conteúdo de x:

```
> x
[1] 10
```

Uma vez criados e associados a nomes (usando o operador de atribuição), os objetos R são permanentes, isto é, escritos no disco sem que você precise salvá-los ao sair do programa.

O conteúdo do diretório pode ser visto através do comando *ls()* ou usando o *Object Manager* do menu *Tolls*. Pode-se remover um objeto do diretório através do comando *rm(nome dos objetos separados por vírgula)*:

```
> rm(x)
O objeto x deixa de existir:
> x
Error: Object "x" not found
```

No R existe distinção entre letras maiúsculas e minúsculas, para os nomes de objetos ou de suas funções (comandos).

## 4.2 Estatística no R

Para poder realiza as análises estatística será necessário que se salve os dados X e Y em um arquivo (“arquivo.txt”).

Carregue os dados do arquivo

```

dados <- read.table("arquivo.txt",h=T)

```

	X	Y
	9.44	9.28
	17.61	8.67
	8.89	6.28
	16.94	12.67
	10.39	6.67
	11.78	7.28
	15.06	15.39
	7.06	5.61
	19.56	11.94
	8.32	5.11
	23.17	17.33

### Obtendo Estatísticas Descritivas

A maneira mais direta de se obter estatísticas descritivas de um conjunto de dados é através da função **summary**:

```

> summary(dados)          # aparecerá somente as estatísticas de posição
  X      Y
Min. : 7.060 Min. : 5.110
1st Qu.: 9.165 1st Qu.: 6.475
Median :11.780 Median : 8.670
Mean   :13.475 Mean   : 9.657
3rd Qu.:17.275 3rd Qu.:12.305
Max.   :23.170 Max.   :17.330

```

É possível também obter estatísticas descritivas de cada variável individualmente:

```

> mean(dados$X)          # calcula a média dos dados X
[1] 13.47455
> median(dados$Y)       # calcula a mediana de dados Y
[1] 8.67
> var(dados$X)          # calcula a variância dos dados X
[1] 28.00569
> sd(dados$Y)           # calcula o desvio padrão dos dados Y
[1] 4.125271
> quantile(dados$X)     # quantil

```

```
0% 25% 50% 75% 100%
7.060 9.165 11.780 17.275 23.170
```

```
> IQR(dados$Y) # Amplitude Interquartilica
[1] 5.83
> cor(dados$X,dados$Y) # Calcula o coeficiente de correlação entre duas
colunas de números
[1] 0.8399821
> max(dados$X) # O maior valor da massa de dados X
[1] 23.17
> min(dados$Y) # O menor valor da massa de dados Y
[1] 5.11
> range(dados$X) # Amplitude total - diferença entre o maior e o menor
valor X
[1] 7.06 23.17
> rank(dados$Y) # Utilizado para apresentar o posto(rank) dos dados
[1] 7 6 3 9 4 5 10 2 8 1 11
```

Não existe no R uma função para calcular o coeficiente de variação, mas ele pode ser obtido facilmente utilizando-se a expressão:

```
>sd(dados$X)/mean(dados$Y)*100 # calcula o coeficiente de variação do X
[1] 54.79849
```

### Geradores de números aleatórios

O R pode gerar números aleatórios (pseudo-aleatórios) de um grande número de distribuições: uniforme, normal, binomial, poisson, gamma, chi-quadrado, etc. Os nomes das funções para gerar números aleatórios iniciam-se com a letra r e são bastante intuitivos: runif, rnorm, rbinom, rpois, rgamma, rchisq etc. Junto a cada função geradora de cada distribuição há funções para calcular probabilidade, densidade e quantis. Por exemplo, para distribuição normal temos:

```
rnorm(n,mean=0,sd=1) # Gera n números
dnorm(q,mean=0,sd=1) # Densidade para o q-ésimo quantil
```

```
pnorm(q,mean=0,sd=1)      Probabilidade para o q-ésimo quantil
qnorm(p,mean=0,sd=1)     Quantil para a probabilidade p
```

## **Análise Exploratória de Dados**

Tabelas de frequência podem ser obtidas combinando duas funções do R.

- Função cut: “corta” uma variável quantitativa em classes, sendo necessário fornecer o argumento breaks que define os limites das classes desejadas;
- Função table: faz uma tabela de frequência com base numa variável já dividida em classes.

```
>range(dados$X)          # verifica a amplitude dos dados de X
[1] 7.06 23.17
```

```
>seq(5,55,by=5)         # verificando os limites das classes
[1] 5 10 15 20 25 30 35 40 45 50 55
```

```
>table(cut(dados$X,breaks=seq(5,55,by=5))) # obtém a tabela de freq.
(5,10] (10,15] (15,20] (20,25] (25,30] (30,35] (35,40] (40,45] (45,50] (50,55]
  4     2     4     1     0     0     0     0     0     0
```

## **Histograma de Frequência**

A tabela de frequência é um resumo numérico importante, mas para estudarmos a distribuição dos dados é mais interessante construirmos o gráfico histograma.

```
>hist(dados$X,col="blue")
>hist(dados$Y,col="red")
```

Para facilitar a comparação entre os dois gráficos de histograma, seria interessante construí-los lado-a-lado:

```
>par(mfrow=c(1,2))
>hist(dados$X,col="blue")
>hist(dados$Y,col="red")
```

### **Bordando o Histograma**

```
>hist(dados$X, main = NULL)
>hist(dados$Y, main = "Histograma de frequência Y", lab = "Classes", ylab =
"Freqüência")
```

### **Análise de Regressão**

O objetivo da análise de regressão é ajustar uma equação de primeiro grau a duas colunas de números (variáveis).

Exemplo:

Foi verificada em cinco regiões da cidade de São Paulo uma relação entre o conteúdo de monóxido de carbono no ar (em  $\mu\text{g}/\text{m}^3$ ) e o número de atendimentos médicos na unidade de saúde de respectiva região

Onde : <b>X</b> – ( $\mu\text{g}/\text{m}^3$ )	<b>x</b>	<b>y</b>
<b>Y</b> – número de atendimentos médicos	<b>7</b>	<b>191</b>
	<b>13</b>	<b>443</b>
	<b>14</b>	<b>530</b>
Carregue os dados do arquivo	<b>17</b>	<b>615</b>
	<b>20</b>	<b>884</b>

```
> dados <- read.table("a:\\monoxido.txt",h=T)
```

Ajuste de modelos de regressão linear simples e múltipla. Antes do ajuste de qualquer modelo, deve-se criar um objeto que conterà todas as informações referente ao ajuste daquele modelo. Supondo que iremos ajustar 10 diferentes modelos para uma mesma base de dados. O primeiro objeto será chamado de MOD1, que se refere ao ajuste do primeiro modelo. O primeiro modelo é linear simples ( $Y = a + bX$ ). A linha de comando será a seguinte:

```
> mod1 <- lm(y~x)
```

OBS.: - **y** refere-se à coluna da variável dependente que tem no arquivo;  
- **x** refere-se à coluna da variável independente que tem no arquivo;

Para verificar os valores dos coeficientes a e b, basta digitar a seguinte linha de comando:

```
> mod1
```

Call:

```
lm(formula = reg$y ~ reg$x)
```

Coefficients:

(Intercept)	reg\$x
-191.84	51.02

```
> attach(dados)
```

```
> reg<-lm(y~x)
```

```
> layout(matrix(1:4,2,2))
```

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

```
> abline(lm(y~x))
```

```
> summary(reg)
```

Call:

```
lm(formula = y ~ x)
```

Residuals:

```
  1    2    3    4    5
25.722 -28.380  7.603 -60.447  55.502
```

Coefficients:

```
      Estimate Std. Error t value Pr(>|t|)
(Intercept) -191.840    80.040  -2.397  0.0962 .
x             51.017     5.389   9.467  0.0025 **
```

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 52.47 on 3 degrees of freedom

Multiple R-Squared: 0.9676, Adjusted R-squared: 0.9568

F-statistic: 89.62 on 1 and 3 DF, p-value: 0.002498

```
> res<-residuals(reg)
```

```
> pre<-predict(reg)
```

```
> res
```

```
  1    2    3    4    5
25.721519 -28.379747  7.603376 -60.447257  55.502110
```

```
> pre
```

```
  1    2    3    4    5
165.2785 471.3797 522.3966 675.4473 828.4979
```

```
> plot(res,pre)
```



## Funções de Probabilidade

Cerca de 30% de um rebanho suíno estão infectados pela peste suína. Qual a probabilidade de que em uma amostra de 45 animais apresentar:

a) 15 animais contaminados

```
> dbinom(15,45,0.3)
[1] 0.1115353
```

b) No máximo 15

```
> pbinom(15,45,0.3)
[1] 0.7462153
```

c) 15 ou mais

```
> 1- pbinom(15,45,0.3)
[1] 0.2537847
```

## Derivadas

```
>E <- expression(x^4 + 2*x^5)
```

```
> D(E, "x")
```

```
4 * x^3 + 2 * (5 * x^4)
```

```
> D(D(E, "x"), "x")
```

```
4 * (3 * x^2) + 2 * (5 * (4 * x^3))
```

```
> F <- expression((x^4)*y + y^5)
```

## Integrais

```
> integrate(dnorm,-Inf,Inf) // calcula integrais para funcoes de 1 variavel
```

```
1 with absolute error < 9.4e-05
```

```
> func <- function(x){x^3}
```

```
> integrate(func,1,15)
```

```
12656 with absolute error < 1.4e-10
```

## 4.3 Programação no R

### 4.3.1 Scripts

Uma série de comandos a serem processados pelo R podem ser descritos em um arquivo e processados em seqüência, de uma só vez (quantas vezes quiser), usando a função *source()*. Digite o nome do arquivo com a extensão \*.R, por exemplo, use o nome script.R.

Digite o seguinte no arquivo:

```
dat <- c(23, 34, 56, 55, 43, 22, 39, 50)
print(mean(dat))
print(median(dat))
print(var(dat))
print(sd(dat))
print(min(dat))
print(max(dat))
print(summary(dat))
```

Agora vá ao prompt do R e digite:

```
> source("script.r")
[1] 40.25
[1] 41
[1] 177.0714
[1] 13.30682
[1] 22
[1] 56
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 56.00
```

### 4.3.2 Funções

Uma das grandes vantagens do R é que ele nos permite escrever nossas próprias funções. Isso o torna uma ferramenta poderosa para testar novas metodologias e realizar simulações.

A nova função R que você construir poderá ser completamente nova ou apenas uma modificação personalizada de uma função R existente. Você pode desejar ainda usar as funções já existentes de modo repetido no seu conjunto de dados - seu trabalho será facilitado incorporando estas tarefas em uma única função.

Vamos ver agora os conceitos básicos da construção de funções R e alguns exemplos simples. Para uma discussão completa, veja o [R-lang.pdf](#) disponível para download no CRAM.

### Sintaxe geral

A sintaxe geral para definir uma nova função a partir da linha de comando é:

```
nome <- function( argumentos ) {corpo }
```

**nome:** nome que você escolhe para a função

**argumentos:** os argumentos necessários para a função, separados por vírgula.

**corpo:** conjunto de comandos (tarefas) necessários para que a função retorne o valor desejado

Vamos construir uma função que calcula as raízes de uma equação do 2º grau ao fornecermos os parâmetros a,b e c e retorna um vetor com os valores das raízes.

```
calcularRaizes <- function(a,b,c)  
{  
  delta <- b^2-4*a*c  
  x1 = (-b + sqrt(delta))/(2*a)  
  x2 = (-b - sqrt(delta))/(2*a)  
  raizes<- c(x1,x2)  
}
```

Salve o código raízes em um arquivo separado como “funcao.r”

Testando:

```
> source("funcao.r")  
> resp<-calculaRaizes(1,-7,10)  
> resp  
[1] 5 2
```

**Obs.:** O último objeto que se faz referência antes de “fechar a função” com } é o objeto que será retornado pela função. Neste caso o objeto raízes.

Tendo visto esse exemplo bem simples, vamos generalizar algumas observações sobre os elementos da sintaxe geral

nome <- **function**( argumentos ) {corpo }.

O nome do objeto a ser referenciado deve aparecer (sozinho) {na última linha antes do } final. Se nenhum nome aparecer no final do corpo da função, será retornado valor da última expressão avaliada no corpo da função.

#### 4.3.2.1 Controle de Fluxo

##### **if-else**

O par de instruções *if-else* possibilita a execução condicional de enunciados, fazendo com que os enunciados após a instrução if sejam executados desde que a condição testada seja verdadeira. A instrução else é opcional. Os enunciados após a instrução else são executados desde que a condição testada seja false.

**if** (condição1) {tarefa a ser realizada se condição1 = TRUE }

**else** {tarefa a ser realizada se condição = FALSE}

Podemos testar múltiplas condições usando os operadores & e | representam E e OU, respectivamente:

**if**(condição1 & condição2)

{

tarefa a ser realizada se as duas condições são TRUE

}

**if**(condição1 | condição2)

{

tarefa a ser realizada se ao menos uma das condições é TRUE

}

A seguir, exemplo onde as variáveis serão somadas se o teste for verdadeiro e serão subtraídas se o teste for falso.

```
aritmética <- function(x,y)
{
  if(x > y) { resultado <-x+y}
  else{resultado < x-y}

  return(resultado)
}
```

Salve o código anos em um arquivo separado como “comandoIf-Else.r”

Testando:

```
> source("comandoIf-Else.r")
```

```
> aritmética(2,3)
```

```
[1] -1
```

```
> aritmética(3,2)
```

```
[1] 5
```

### switch

Discutimos a estrutura de seleção *if-else*, mas ocasionalmente um algoritmo conterá uma série de decisões em que uma variável ou expressão será separadamente testada para cada um dos valores que ela pode assumir e ações diferentes serão executadas. O programa R oferece a estrutura de seleção múltipla *switch* para tratar tais tomadas de decisões.

A estrutura *switch* do exemplo a seguir consta de uma serie de rótulos que serão avaliados para verificar qual sentença executar.

```
opcoes <- function(x,escolha)
{
  switch(escolha,
  media = mean(x),
  mediana = median(x),
```

```

    variancia = var(x),
    minimo = min(x),
    maximo = max(x)
}

```

Salve o código em um arquivo separado como “comandoSwitch.r”

```

> source("comandoSwitch.r")
> x<-c(1,2,3,4,5)
> opcoes(x,"media")
[1] 3
> opcoes(x,"mediana")
[1] 3
> opcoes(x,"variancia")
[1] 2.5
> opcoes(x,"minimo")
[1] 1
> opcoes(x,"maximo")
[1] 5

```

#### 4.3.2.2 Controle de Repetição

##### **for**

O comando **for** permite que uma tarefa seja repetida à medida que uma variável assume valores em uma seqüência específica:

```
for(variável in seqüência) {tarefas }
```

Vamos escrever uma função que calcula a média aritmética dos elementos armazenados em um vetor.

```

media <- function(vet)
{
    soma<-0 # inicializacao de soma
    y <- 0 # inicializacao de y

    n <- length(vet) # função que retorna o tamanho do vetor

    for (i in 1:n) # Percorrendo cada elemento do vetor x

```

```

    {
        soma <- soma + vet[i]
    }

    media<- soma/n

    return (media)
}

```

Salve o código anos em um arquivo separado como “comandoFor.r”

Testando:

```

> x<-c(1,2,3,4)
> source("comandoFor.r")
> resp <- media(x)
> resp
[1] 2.5

```

### **while**

O comando **while** permite que uma tarefa seja repetida enquanto uma condição (expressão com resultado lógico) é verdadeira:

```
while(condição) { tarefas }
```

```

media <- function(vet)
{
    contador <- length(vet)
    soma <- 0

    while(contador > 0)
    {
        soma = soma + vet[contador]
        contador <- contador-1
    }

    media = soma/length(vet)

    return(media)
}

```

Salve o código anos em um arquivo separado como “comandoWhile.r”

```
> x<-c(1,2,3,4)
> comandoWhile(x)
[1] 2.5
```

### **repeat**

O comando **repeat** permite que uma tarefa seja repetida indefinidamente, a não ser que a condição no comando **break** seja satisfeita:

```
media <- function(vet)
{
  contador <- 1
  soma <- 0

  repeat
  {
    soma <- soma + vet[contador]
    contador <- contador + 1
    if(contador == length(vet)+1) break
  }
  media <- soma/length(vet)

  return(media)
}
```

Salve o código em um arquivo separado como “comandoRepeat.r”

```
> x<-c(1,2,3,4)
> comandoRepeat(x)
[1] 2.5
```

## **4.4 Compilação e ligação do código C++ com o código R**

Segundo RDCT (2003) os procedimentos na Tabela 3 serão necessários para gerar o arquivo de extensão so que fará a ligação com o código R .



**Tabela 7. Procedimentos para compilar e ligar o código C++ com o código R.**

**No Terminal digite:** Rcmd SHLIB Nome\_do\_arquivo.cpp  
 Chame o R: **R**  
 Carregue o arquivo de extensão so:  
**Dyn.load (paste("Nome\_arquivo\_extensão\_so", .Platform\$dynlib.ext, sep = ""))**  
 Execute o código C++: **.C("Nome\_da\_função")**

**Fonte: Dados do trabalho**

Veja o exemplo da Tabela 8.

**Tabela 8. Exemplo de código C++ para execução no código R**

<pre>// principal.cpp #include &lt;iostream.h&gt; #include "fatorial.h"  extern "C" {     void main(double* num)     {         Fatorial fat;          cout &lt;&lt; "\nO Fatorial de "               &lt;&lt; *num &lt;&lt; " e': " &lt;&lt;               fat.calcFat(*num) &lt;&lt; endl;     } } //extern "C"</pre>	<pre>// fatorial.h class Fatorial {     public:         double calcFat(double n); };  double Fatorial::calcFat(double n) {     if(n==0    n==1)         return 1;     else         return(n* calcFat(n-1)); }</pre>
--	---

**Fonte: Dados do trabalho**

**Tabela 9. Exemplo de código R para execução do código C++ da  
Tabela 8.**

```
Fatorial<- function(num)
{
  dyn.load(paste("principal",.Platform$dynlib.ext, sep = ""))
  .C("main",as.numeric(num))
}
```

**Fonte: Dados do trabalho**

**Procedimentos:**

**No terminal digite:** Rcmd SHLIB principal.cpp

**Chame o R: R**

```
> source("fat.r")
> fatorial(4)
O Fatorial de 4 e': 24
[[1]]
[1] 4
```

## 5 UM EXEMPLO PRÁTICO

### 5.1 O Problema

Os dados seguintes referem-se ao números de cochonilhas vivas em cada parcela (médias de 3 plantas), 8 dias após a aplicação de 5 inseticidas:

A - FOLIDOL      B – MALATOL+ TRIONA  
C – TRIONA      D – DIMETOATO      E CYPROLANE

O tratamento F corresponde a uma testemunha (sem aplicação de fungicida)

**Tabela 10. Número de cochonilhas vivas após a aplicação de fungicida.**

Repetições	Tratamentos					
	A	B	C	D	E	F
1	15,7	23,5	20,5	25,7	15,2	40,4
2	10	28,3	18,3	26,7	17,1	43,2
3	12,2	23,4	20,1	25,4	18,2	45,1
4	13,2	27,6	22,5	27,2	16,6	45

### 5.2 A Solução com o R

A seguir é apresentada a saída do Teste de Tukey no software R para os dados referente a Tabela 6.

```
> attach(dados)
> b <- factor(bloco)
> t <- factor(trat)
> modelo <- aov(y~b+t)
> teste <- as.matrix(TukeyHSD(modelo,"t",ordered=T))
> teste[1,]
[[1]]
      diff      lwr      upr
2-1 0.4333333 -2.782926 3.649593
3-1 0.5666667 -2.649593 3.782926
```

4-1 1.8500000 -1.366260 5.066260  
3-2 0.1333333 -3.082926 3.349593  
4-2 1.4166667 -1.799593 4.632926  
4-3 1.2833333 -1.932926 4.499593

A Figura 4 mostra o resultado da saída implementada para o R onde as médias seguidas da mesma letra não diferem estatisticamente pelo Teste de Tukey ao nível de 5%. Observa-se que a saída atual do R, versão 1.6.2 (até mesmo na versão 1.8) não apresenta a formatação em letra(s). É esta característica que a proposta desta monografia propõe.

```
root@localhost:~$ Rgui
Arquivo  Editar  Parâmetros  Ajuda

R is a collaborative project with many contributors.
Type 'contributors()' for more information.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for a HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> dados <- read.table("exemplos/cochinillas/cochinillas.txt", header=T)
> attach(dados)
> t <- factor(treat)
> b <- factor(bloco)
> model <- aov(y~b+t)
> TukeySymbol(model, "t")

12.78 e4
16.78 e3 e4
20.35 e3
25.70 e2
26.23 e2
43.42 e1
> |
```

**Figura 4. Saída Implementada para o Teste de Tukey.  
Fonte: Dados do trabalho**

## 6 CONCLUSÕES

A utilização de softwares livres de alta qualidade e multiplataforma<sup>3</sup> é uma tendência mundial, ainda mais depois do advento da Internet, pelo seu baixo custo e alta performance .

A implementação do código C++ para alterar a saída do R é bastante simples e eficaz. No exemplo, a nova saída mostrou-se adequada para facilitar o entendimento e interpretação dos resultados referente ao Teste de Tukey, pois fornece ao usuário uma rápida visão das igualdades entre médias.

Foi constatado também que o uso da linguagem de programação C++ é adequada para desenvolvimento de aplicativos no R. O C++ é mais um recurso de aumento no potencial do R pela sua eficácia, pois esta linguagem é capaz de resolver qualquer tipo de problema computacional e não há limites para o que um bom programador C++ é capaz de fazer.

---

<sup>3</sup> Capacidade de ser executado em diversos Sistemas Operacionais

## 7 REFERÊNCIAS BIBLIOGRÁFICAS

- Banzatto, D.A.; Kronka, S.N. **Experimentação Agrícola. Jaboticabal:** FUNEP, 1992. 247 p.
- Boock,G; Rumbaugh, J; Jacobson, Ivar. **The Unified Modeling Language User Guide.** Addison-Wesley, 1999.
- Box,G.E.P. **The Challenge of Statistical Computation.** Em Milton, R.C e T.A. Nelder(editores). **Statistical Computation.** Academic Press, Nova Iorque, 1969.
- Deitel, H.M.; Deitel, P.J. **C++ Como Programar. Porto Alegre:** Bookman, 2001. 401p.
- Dachs, J.N.W. **Estatística Computacional.** Rio de janeiro: Livros Técnicos e Científicos editora. 1988 , 236p.
- Eddy, W.F. Computers in Statistical Research. Workshop on the Use of Computers in Statistical Research / Willian F. Eddy, Chairman. **Statistical Science**, 1, 419-453, 1986.
- Efron, B. **Computers and the Theory of Statistics: Thinking the Unthinkable.** SIAM Review, 21,460-480
- Francis,I. **Statistical Software: A comparative Review.** North Holland, Nova Iorque, 1981.
- Gomes, F.P. **Estatística Experimental.** Piracicaba: USP/ESALC, 2000, 477.
- Ikara, R.; Gentleman, R.R: A language for data analysis and graphics. **Journal of computation and graphical statistics.** Alexandria, v. 5, n. 3, p. 299-314, Sept. 1996.
- Kahn, P.M.(editor). **Computational Probability.** Academic Press, Nova Iorque, 1980.
- R Development Core team. **Writing R. Extensions,** 2003.
- Ramalho, M.A.P; Ferreira, D.F.; Oliveira, A.C. **Experimentação em Genética: e melhoramentos de plantas.** Lavras:UFLA, 2000. 326 p.

Tukey, J.W. **Is Statistics a Computing Science?** Em E. Watts, (editor). **The Future of Statistics.** Academic Press, Nova Iorque, 1968.

## **ANEXOS**



## ANEXO A

### A organização de dados no R

Antes de fazer a análise de seus dados no R, você deve convertê-los, dentro do programa, para a forma de objetos de dados R. O tipo de dado irá depender da natureza dos seus dados. Vamos concentrar nossa atenção em quatro tipos:

**Vetor:** Conjunto de elementos de mesmo modo em uma ordem especificada (unidimensional)

**Matriz:** Disposição bidimensional de elementos de mesmo modo.

**Data frame:** Disposição bidimensional cujas colunas que podem representar dados de tipos diferentes

**Lista:** combinar diferentes coisas em um mesmo objeto. Estas coisas podem ser vetores, matrizes, números e/ou caracteres e até mesmo outras listas.

### O uso do software R

#### Vetores

Um vetor é um conjunto de elementos em uma ordem específica. A ordem é especificada quando você cria o vetor (usualmente a ordem na qual você digita os dados) e isto é importante porque você pode se referir a um elemento unicamente pela sua posição no vetor.

Todos os elementos de um vetor devem ser de um único modo. Como os dados são mais freqüentemente números, os vetores numéricos são os mais usuais.

A função `c()` é usada para criar um vetor a partir de seus argumentos. Por exemplo:

```
> x <- c(2,3,5,7,11)
> x
[1] 2 3 5 7 11 # digitando o nome do objeto é exibido o seu conteúdo
```

Os argumentos de `c()` podem ser escalares ou vetores.

### **Operações com vetores**

Operações aritméticas em vetores são efetuadas para cada um de seus elementos.

Veja alguns exemplos:

```
> x <- 1:10
> x
[1] 1 2 3 4 5 6 7 8 9 10
> x + 2
[1] 3 4 5 6 7 8 9 10 11 12
> x * 2
[1] 2 4 6 8 10 12 14 16 18 20
```

Se for realizada uma operação em dois vetores os cálculos são também feitos em um elemento de cada vez.

```
> x <- 1:10
> y <- 21:30
> x+y
[1] 22 24 26 28 30 32 34 36 38 40
```

### **E se os vetores tiverem tamanhos diferentes?**

Neste caso o R usa a "lei da reciclagem", o que significa que os elementos do menor vetor serão repetidos até atingir o tamanho do maior vetor.

Veja o exemplo:

```
> x <- 1:10
> y <- c(1,2)
> x+y
[1] 2 4 4 6 6 8 8 10 10 12
```

Neste exemplo y foi repetido cinco vezes e adicionado a x da seguinte forma,

```
x : 1 2 3 4 5 6 7 8 9 10
y : 1 2 1 2 1 2 1 2 1 2 (note como foi repetido)
-----
x+y : 2 4 4 6 6 8 8 10 10 12
```

Entretanto, se o comprimento do maior vetor não é um múltiplo do tamanho do menor vetor o R vai processar o comando e emitir uma mensagem de alerta ("warning")

### **Acessando partes de um vetor**

Índices ("subscripts") são a maneira utilizada pelo R para extrair partes de um vetor. São utilizados colchetes e os índices são contados a partir de 1.

```
> x <- 10:1
> y <- x[2]
> y
[1] 9
```

Pode-se extrair partes de um vetor usando um vetor de índices

```
> y <- x[c(1,3,5)]
> y
[1] 10 8 6
> x[4:7]
[1] 7 6 5 4
```

e note que o valor retornado é um vetor. Se tentarmos extrair um elemento com um índice maior que a dimensão de um vetor.

```
> x[12]
[1] NA
```

O resultado será NA (que significa "Not Available"). Este símbolo é usado pelo R para indicar valores perdidos ("missing data") ou qualquer outro valor sem sentido.

Outros exemplos:

# inverte a ordem dos elementos de x

Há também uma função para fazer esta inversão: rev(x)

O número de elementos de um vetor é o seu length :

```
> length(x)
```

## Listas

Listas são usadas para combinar diferentes coisas em um mesmo objeto. Estas coisas podem ser vetores, matrizes, números e/ou caracteres e até mesmo outras listas. Aqui vai um exemplo simples:

```
> lista <- list(name1=21,name2='Wagner',name3=c(65,78,55))
> lista
$name1:
[1] 21
$name2:
[1] "Wagner"
$name3:
[1] 65 78 55
```

Listas são construídas com a função *list()*. Os componentes da lista são introduzidos usando a forma usual (*nome = arg*) de atribuir argumentos em uma

função. Quando se digita o nome de um objeto que é uma lista cada componente é mostrado com seu nome e valor.

Cada elemento da lista pode ser acessado individualmente por seu nome antecedido pelo símbolo \$:

```
> lista$name1  
[1] "Wagner"
```

Pode-se ainda acessar cada elemento pelo seu número de ordem na lista utilizando-se colchetes duplos:

```
> lista[[1]]  
[1] 21  
> lista[[3]]  
[1] 65 78 55
```

## **Matrizes**

Uma matriz é uma disposição bidimensional dos dados, em linhas e colunas. Os elementos de uma matriz no R devem ser de um único modo, sendo mais comum termos dados numéricos.

### **Criando Matrizes**

Há várias formas de criar uma matriz. A função *matrix()* recebe um vetor como argumento e o transforma em uma matriz de acordo com as dimensões especificadas:

```
> x <- 1:16  
> x  
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16  
> xmat <- matrix(x,ncol=4)  
> xmat
```

```
      [,1] [,2] [,3] [,4]
[1,]  1   5   9  13
[2,]  2   6  10  14
[3,]  3   7  11  15
[4,]  4   8  12  16
```

Neste exemplo foi construída uma matriz de 4 linha 4 colunas usando os números de 1 a 16. Note que a matriz é preenchida ao longo das linhas. Para inverter este padrão deve-se adicionar o argumento *byrow = T*, para dizer que a matriz deve ser preenchida por colunas:

```
> matrix(x,ncol=4,byrow=T)
      [,1] [,2] [,3] [,4]
[1,]  1   2   3   4
[2,]  5   6   7   8
[3,]  9  10  11  12
[4,] 13  14  15  16
```

### **Informações sobre a matriz**

Podemos verificar a dimensão de uma matriz com a função *dim()*:

```
> dim(xmat)
[1] 4 4
```

### **Operações com matrizes**

Todas as funções usuais operam em matrizes da mesma forma que operam com vetores - elemento por elemento.

Portanto se multiplicarmos duas matrizes com *\** você terá o produto de cada elemento de uma matriz pelo elemento correspondente na outra matriz, e não a multiplicação de matrizes usual.

Para fazer a multiplicação usual de matrizes deve-se usar o símbolo `%*%:`

```
> xmat %*% xmat
  [,1] [,2] [,3] [,4]
[1,]  90 202 314 426
[2,] 100 228 356 484
[3,] 110 254 398 542
[4,] 120 280 440 600
```

### Índices das matrizes

Para extrair um único elemento da matriz use colchetes com dois números separados por vírgula. O primeiro número indica o número da linha enquanto o segundo indica o número da coluna.

```
> xmat[3,4]
[1] 15
```

Pode-se extrair uma linha inteira ou uma coluna inteira usando apenas um número e a vírgula. Para extrair uma coluna coloque o número da coluna desejada depois da vírgula. Para extrair uma linha coloque o número da linha desejada depois da vírgula. Quando se extrai uma linha ou uma coluna o resultado é um vetor.

```
> xmat[,4]      # extraindo toda coluna
[1] 13 14 15 16
> xmat [3,]     # extraindo toda linha
[1] 3 7 11 15
```

### Lendo matrizes de arquivos

Suponha que tenhamos o conjunto de notas de cinco alunos em três provas armazenado em um arquivo texto chamado *notas.txt* com o seguinte conteúdo:

```
95 94 96
90 89 90
98 95 99
99 97 100
87 86 88
```

Sendo que as linhas representam os alunos e as colunas representam as provas.

Podemos usar a função *matrix* para converter um vetor em uma matriz, especificando o número de linhas (ou colunas) e se o preenchimento da matriz deve ser feito por colunas (default) ou por linhas.

A função *matrix* tem a seguinte estrutura: **matrix(dados, nrow= , ncol= , byrow=F)**

Sendo: *byrow* tem valor lógico, com F (false) para preenchimento por colunas e T (true) para preenchimento por linhas.

Para ler matrizes em dados externos como *notas.txt*, a função *matrix* deve ser usada em conjunto com a função *scan*:

```
> notas <- matrix(scan('notas.txt'), ncol=3, byrow=T)
Read 15 items
> notas
  [,1] [,2] [,3]
[1,] 95 94 96
[2,] 90 89 90
[3,] 98 95 99
[4,] 99 97 100
[5,] 87 86 88
```



Uma matriz também pode ser construída pela combinação de dois ou mais vetores de mesmo *length*. Por exemplo, se temos três vetores com as notas em cada prova:

```
> notas.p1 <- c(95,90,98,99,87)
> notas.p2 <- c(94,89,95,97,86)
> notas.p3 <- c(96,90,99,100,88)
```

Usamos a função *cbind* para combiná-los como colunas da matriz *notas*:

```
> notas <- cbind(notas.p1, notas.p2, notas.p3)
> notas
      notas.p1 notas.p2 notas.p3
[1,]      95      94      96
[2,]      90      89      90
[3,]      98      95      99
[4,]      99      97     100
[5,]      87      86      88
```

Usamos a função *rbind* para combiná-los como linhas da matriz *notas*.

```
> notas <- rbind(notas.p1,notas.p2,notas.p3)
> notas
      [,1] [,2] [,3] [,4] [,5]
notas.p1  95  90  98  99  87
notas.p2  94  89  95  97  86
notas.p3  96  90  99 100  88
```

## Data Frames

Freqüentemente, os dados são uma mistura de variáveis numéricas (como idade), variáveis categóricas (como sexo) e caracteres (como nome do indivíduo). Uma tabela deste tipo não pode ser lida como uma matriz, já que mistura dados de diferentes modos, mas pode ser lida como um *data frame*.

Um *data frame* também é uma disposição bidimensional dos dados, mas pode ter elementos de diferentes modos em diferentes colunas, desde que cada coluna tenha o mesmo tamanho. Assim, um *data frame* é uma estrutura tabular na qual as colunas representam variáveis e as linhas representam os indivíduos.

Um *data frame* pode ser criado de duas maneiras:

**read.table:** lê os dados de um arquivo externo

**data.frame:** Combina objetos de vários modos

### Lendo um data frame de um arquivo texto

Um formato comum de dados é na forma de arquivo texto com uma linha para cada registro, com elementos separados por espaços ou vírgulas. Considere o arquivo, "alunos.txt":

	Alunos	Sexo	Prova1	Prova2	Prova3
1	Adriana	Fem	80	90	75
2	Gilson	Masc	75	90	60
3	Plínio	Masc	85	94	95
4	Tofo	Masc	90	94	55
5	Valdir	Masc	100	70	55
6	Wagner	Masc	70	89	80

Usamos a função `read.table()` para ler estes dados e armazená-los em um objeto. Fornecemos o nome do arquivo (entre aspas), o caracter de separação dos elementos (vírgula). O comando é:

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

	Alunos	Sexo	Prova1	Prova2	Prova3
1	Adriana	Fem	80	90	75
2	Gilson	Masc	75	90	60
3	Plinio	Masc	85	94	95
4	Tofo	Masc	90	94	55
5	Valdir	Masc	100	70	55
6	Wagner	Masc	70	89	80

O argumento `header=T` indica que a primeira linha contém os nomes das colunas. Se não for o caso, apenas não forneça argumento `header`, pois seu default é `FALSE`.

O número de linhas e o número de colunas de um *data frame* podem ser obtidos usando-se a função `dim`:

```
> dim(alunos)
[1] 6 5
```

### Índices como em matrizes

O arquivo foi lido em algo que se parece um pouco com uma matriz. Pode se usar índices para selecionar linhas e colunas da mesma forma que em matrizes:

```
> alunos[,2]
[1] Fem Masc Masc Masc Masc Masc
Levels: Fem Masc
```

Os nomes das colunas podem ser definidos como em listas e as colunas são selecionadas usando o símbolo `$`:

```
> alunos$Alunos
[1] Adriana Gilson Plinio Tofo Valdir wagner
Levels: Adriana Gilson Plinio Tofo Valdir wagner
> alunos$Prova1
[1] 80 75 85 90 100 70
```

## Adicionando Colunas

Assim como em matrizes pode-se usar a função `cbind()` para se adicionar colunas a um data-frame. Se um nome for definido no argumento de `cbind()` este nome será associado à nova coluna.

```
> notas <- cbind(alunos,Media = c(81,75,91,79,75,79))
> notas
```

	Alunos	Sexo	Prova1	Prova2	Prova3	Media
1	Adriana	Fem	80	90	75	81
2	Gilson	Masc	75	90	60	75
3	Plínio	Masc	85	94	95	91
4	Tofo	Masc	90	94	55	79
5	Valdir	Masc	100	70	55	75
6	Wagner	Masc	70	89	80	79

## ANEXO B

```
#ifndef TDATA_H
#define TDATA_H

#include<sstream>
#include<string>

using namespace std;

class TData
{
public:

    TData();
    double GetLower(){ return lower; }
    double GetUpper(){ return upper; }
    void SetLower( double lower ){ this->lower = lower; }
    void SetUpper( double upper ){ this->upper = upper; }
    void SetMean( double mean ){ this->mean = mean;}
    double GetMean(){return mean;}
    void SetValidate(bool validate){this->validate = validate;}
    bool GetValidate(){ return validate;}
    void CalculateIntervals( double mean, double* DMS );
    void InsideSymbol(string character, int digit){ oss << character <<
digit << " ";}
    void ConvertSymbolString(){ symbol = oss.str();}
    string GetSymbol(){return symbol;}

private:

    int cont;
    double mean;
    double lower;
    double upper;
    bool validate;
    string symbol;
    ostringstream oss;
};

TData::TData()
```

```

{
    validate = true;
    double lower = 0;
    upper = 0;
    mean = 0;
    symbol=" ";
}

void TData::CalculateIntervals(double mean, double* DMS)
{
    lower = mean - *DMS;
    upper = mean;
}
#endif

```

---

```

#include "node.h"
#ifndef TLISTA_H
#define TLISTA_H
template <class G>

class TList
{
public:

    TList();
    ~TList();
    TNode<G>* GetFirstNode();
    TNode<G>* GetBackNode();
    void InsertOrdered(G value);

private:

    TNode<G>* first_node;
    TNode<G>* back_node;
    TNode<G>* new_node;
    void ClearMemory();
};

template <class G>
TList<G>::TList()
{

```

```
    first_node = back_node = new_node = NULL;
}
```

```
template <class G>
TList<G>::~~TList()
{
    ClearMemory();
}
```

```
template <class G>
void TList<G>::ClearMemory()
{
    TNode<G>* temp;

    temp = first_node;

    while (first_node != NULL)
    {
        first_node = first_node->GetRight();

        delete temp;

        temp = first_node;
    }
}
```

```
template <class G>
TNode<G>* TList<G>::GetFirstNode()
{
    return(first_node);
}
```

```
template <class G>
TNode<G>* TList<G>::GetBackNode()
{
    return(back_node);
}
```

```
template <class G>
void TList<G>::InsertOrdered(G value)
{
    TNode<G>* aux1;
```

```

TNode<G>* aux2;
bool condicao;

new_node = new TNode<G>(value);

if (first_node==NULL) //lista vazia
    first_node = back_node = new_node;
else
{
    if(first_node==back_node) //Lista com apenas um node
    {
        if(back_node->GetObject()->GetMean() > value->GetMean())
        {
            back_node->LinkRight(new_node);
            new_node->LinkLeft(back_node);
            back_node = new_node;
        }
        else
        {
            new_node->LinkRight(back_node);
            back_node->LinkLeft(new_node);
            first_node = new_node;
        }
    }
}
else // Lista com n TNodos
{
    condicao = true;
    aux1 = NULL;
    aux2 = first_node;

    while((aux2!=NULL)&&(condicao))
    {
        if(aux2->GetObject()->GetMean() > value->GetMean())
        {
            aux1 = aux2;
            aux2 = aux2->GetRight();
        }
        else
            condicao = false;
    }

    if(aux1==NULL) // inserir no inicio

```



```

        {
            new_node->LinkRight(aux2);
            aux2->LinkLeft(new_node);
            first_node = new_node;
        }
    else if(aux2==NULL) // inserir no final
    {
        aux1->LinkRight(new_node);
        new_node->LinkLeft(aux1);
        back_node = new_node;
    }
    else // inserir entre
    {
        aux1->LinkRight(new_node);
        new_node->LinkLeft(aux1);
        new_node->LinkRight(aux2);
        aux2->LinkLeft(new_node);
    }
    }
}
}
#endif

```

---

```

#ifndef TNODE_H
#define TNODE_H

template <class G>
class TNode
{
public:

    TNode(G val){obj = val; right = NULL; left = NULL;}
    G GetObject(){return obj;}
    TNode<G>* GetRight(){return right;}
    TNode<G>* GetLeft(){return left;}
    void LinkRight(TNode<G>* node){right = node;}
    void LinkLeft(TNode<G>* node){left = node;}

private:

    G obj;

```

```

    TNode<G>* right;
    TNode<G>* left;
};
#endif

// classe Principal
#include<iostream.h>
#include<string>
#include<iomanip.h>
#include<ctype.h> // for atoi
#include"list.h"
#include"data.h"

using namespace std;

extern "C"
{
    //funcao para marcar o no como valido ou nao valido
    void MeetNode(TList<TData*> &list);
    //funcao para ajustar o lower
    double AdjustLower (TList <TData*> &list, double lower);
    // funcao que calcula o numero de intervalos validos
    int CalculateNumberNotValidate(TList <TData*> &list);
    //funcao para inserir o simbolo na lista
    void InsertSymbol(TList <TData*> &list);
    //funcao para imprimir a saida de tukey
    const void PrintTableTukey(TList <TData*> &list);
    //deletar todos os objetos TData que estao nos nodos
    void ClearObject(TList<TData*> &list);

    void main(double* DMS, int* size, double* array_means)
    {
        TData* dat;
        TList<TData*> list;
        TNode<TData*>* node;
        TData* aux;
        double new_lower;
        double new_upper;

        //inserir intervalos
        for (int j=0; j < *size; j++)
        {

```

```

        dat = new TData;
        dat->SetMean(array_means[j]);
        dat->CalculateIntervals(array_means[j], DMS);
        list.InsertOrdered(dat);
    }

    node = list.GetFirstNode();
    aux = node->GetObject();

    while(node != NULL)
    {
        new_lower = AdjustLower (list, aux->GetLower());
        aux->SetLower (new_lower);
        node = node->GetRight();
        if (node != NULL)
            aux = node->GetObject();
    }
    MeetNode(list);
    InsertSymbol(list);
    PrintTableTukey(list);
    ClearObject(list);
}

void ClearObject(TList<TData*> &list)
{
    TNode<TData*>* node;
    TData* temp;

    node = list.GetFirstNode();

    while(node!=NULL)
    {
        temp = node->GetObject();
        delete temp;
        temp = NULL;
        node = node->GetRight();
    }
}

//funcao para marcar o no como valido ou nao valido
void MeetNode(TList<TData*> &list)
{

```

```

TNode<TData*>* previous;
TNode<TData*>* cursor;
double lower;
double upper;

previous = list.GetFirstNode ();
cursor = previous->GetRight();

while(previous!=list.GetBackNode())
{
    lower = previous->GetObject()->GetLower();

    upper = previous->GetObject()->GetUpper();
    while (cursor!=NULL)
    {
        if ((cursor->GetObject()->GetLower () >= lower) &&
            (cursor->GetObject()->GetUpper () < upper))
        {
            cursor->GetObject()->SetValidate(false);
        }
        cursor = cursor->GetRight();
    }
    previous = previous->GetRight();
    cursor = previous->GetRight();
}

//funcao para ajustar o lower
double AdjustLower (TList <TData*> &list, double lower)
{
    TNode <TData*> *tmp;

    tmp = list.GetBackNode(); // ultimo node da lista

    while (tmp->GetObject()->GetMean() < lower)
        tmp = tmp->GetLeft();

    return (tmp->GetObject()->GetMean());
}

int CalculateNumberNotValidate(TList <TData*> &list)
{

```

```

int number = 0;
TNode<TData*>* node;

node = list.GetFirstNode();

while(node!=NULL)
{
    if(node->GetObject()->GetValidate())
    {
        number++;
    }
    node = node->GetRight();
}
return(number);
}
void InsertSymbol(TList <TData*> &list)
{
    string symbol = "a";
    double lower;
    double upper;
    int size,cont;
    int i,j;
    double* array_intervals;
    double mean;
    TNode<TData*>* node;
    TNode<TData*>* previous;
    TNode<TData*>* temp;

    cont = 1;
    i = 0;
    size = CalculateNumberNotValidate(list);

    array_intervals = new double[2*size];

    previous = list.GetFirstNode();

    while(previous!=NULL)
    {
        if(previous->GetObject()->GetValidate())
        {
            array_intervals[i] = previous->GetObject()->GetLower();
            array_intervals[i+1] = previous->GetObject()->GetUpper();

```

```

        i = i + 2;
    }
    previous = previous->GetRight();
}
cout << endl;

for(j=0;j < 2*size;j = j+2)
{
    temp = list.GetFirstNode();

    while(temp!=NULL)
    {
        mean = temp->GetObject()->GetMean();

if((array_intervals[j]<=mean)&&(mean<=array_intervals[j+1]))
        {
            temp->GetObject()->InsideSymbol(symbol,cont);
        }
        temp = temp->GetRight();
    }
    cont++;
}

temp = list.GetBackNode();

while(temp!=NULL)
{
    temp->GetObject()->ConvertSymbolString();
    temp = temp->GetLeft();
}
}

const void PrintTableTukey(TList <TData*> &list)
{
    TNode<TData*>* temp;
    string token;
    char* ptr;
    int size;
    int number;

    temp = list.GetBackNode();

```

```

while(temp!=NULL)
{
    token = temp->GetObject()->GetSymbol();

    size = token.length() + 1; //incluindo o nulo
    ptr = new char[size];
    token.copy(ptr,token.length(),0);
    ptr[size-1] = 0;

    number = atoi(&ptr[1]);

    cout << setiosflags(ios::left|ios::showpoint)
        << setprecision(4) << temp->GetObject()->GetMean() <<
"\t";

    for(int j = 0; j < number-1; j++)    cout << " ";

    cout << setiosflags(ios::right) << temp->GetObject()-
>GetSymbol() << endl;
    temp = temp->GetLeft();
}

delete[] ptr;
}
} //extern "C"

```