Universidade de Brasília
IE – Departamento de Estatística

PACOTE R: MÉTODO SCOTT-KNOTT AJUSTADO PARA DADOS DESBALANCEADOS

Felipe da Rocha Ferreira

Trabalho de Conclusão de Curso apresentado para o Departamento de Estatística, Instituto de Ciências Exatas, Universidade de Brasília, como parte dos requisitos necessários para o grau de Bacharel em Estatística.

Brasília
2021

Felipe da Rocha Ferreira

PACOTE R: MÉTODO SCOTT-KNOTT AJUSTADO PARA DADOS DESBALANCEADOS

Orientador: Prof. Dr. Eduardo Monteiro de Castro Gomes
Universidade de Brasília - UnB

Trabalho de Conclusão de Curso apresentado para o Departamento de Estatística, Instituto de Ciências Exatas, Universidade de Brasília, como parte dos requisitos necessários para o grau de Bacharel em Estatística.

Brasília
2021

AGRADECIMENTO

Agradeço aos meus pais, Marcia e Silvano, por serem os maiores responsáveis pela minha formação; ao professor Eduardo Monteiro pela orientação neste trabalho; aos amigos de curso que mais me apoiaram nessa jornada: Anna Vilasboas, Fabiana de Souza, Hannah Franco, Larissa Silva e Luana Barreto; e a minha grande amiga, Luiza Veras.

Este trabalho é dedicado ao meu pai, Silvano Ferreira Rocha.

"Eu posso não ter ido para onde eu pretendia ir, mas eu acho que acabei terminando onde eu pretendia estar".

Douglas Adams

RESUMO

Rocha, Felipe. Pacote R: Método Scott-Knott Ajustado para Dados Desbalanceados. 2021. 70 f. Trabalho de Conclusão de Curso – Curso de Bacharelado em Estatística, Universidade de Brasília. Brasília-DF, 2021.

Este trabalho visa disponibilizar, através de uma implementação em pacote R, a modificação do teste de Skott-Knott (SCOTT; KNOTT, 1974) para teste de comparações múltiplas com desbalanceamento das observações proposta por Conrado (2017). Foram realizados experimentos de simulações de Monte Carlo para avaliação empírica do erro nominal do Tipo I e o Poder do Teste. Para tal, foram considerados cenários distintos para comparar duas abordagens em relação ao Skott-Knott: Scott-Knott Modificado com perdas de unidades observacionais; e Scott-Knott após Imputação Múltipla.

Palavras-chave: Desbalanceado. Scott-Knott. R. Package. Imputação Múltipla. Monte Carlo.

ABSTRACT

This undergraduate thesis aims to make available, through an implementation in R Package, the modification of the Skott-Knott test (SCOTT; KNOTT, 1974) for multiple comparisons test with unbalanced observations proposed by Conrado (2017). Monte Carlo simulation experiments were carried out for the empirical evaluation of the Type I error and the Power of the test. For this purpose, different scenarios were considered to compare two approaches in relation to Skott-Knott: Modified Scott-Knott with losses of observational units; and Scott-Knott after Multiple Imputation.

Keywords: Unbalanced. Scott-Knott. R. Package. Multiple Imputation. Monte Carlo.

Sumário

1 Introdução

Dentre os vários métodos estatísticos da literatura, a análise de variância (ANOVA) tornou-se uma das mais importantes na análise de experimentos, informando ao pesquisador quando existe ou não diferença estatisticamente significante entre seus diversos tratamentos. Entretanto, na maioria dos casos, tal informação não é suficiente para determinar uma tomada de decisão, sendo necessária uma maior investigação sobre quais tratamentos ou quais grupos de tratamentos diferem entre si. Surgindo, assim, diversas análises de comparações múltiplas, dentre elas, o método de aglomeração de Scott-Knott (SCOTT; KNOTT, 1974), que sofre uma limitação importante: só pode ser aplicado em tratamentos balanceados.

A principal vantagem de se utilizar métodos de aglomeração a métodos de comparações múltiplas consiste na maior facilidade de interpretação: em experimentos com muitos tratamentos, a utilização de métodos de comparações múltiplas, como o *Teste Tukey*, é extremamente complicada. Por fazer comparações 2-a-2, esses métodos criam conflitos de interpretação, como no exemplo: em um experimento com os tratamentos A, B e C, pode ocorrer de A diferir de B, mas B e A não diferirem de C (Ou seja, B e C são estatisticamente similares, mas A difere de B e não difere de C). Conflito este que, por não haver sobreposição dos grupos, não ocorre com os métodos de aglomeração.

Então, para usufruir do ganho de interpretação em experimentos com parcelas perdidas, pode-se optar por efetuar imputações das parcelas ausentes e, dentre os métodos existentes para efetua-las, a utilização de algoritmos de Imputações Múltiplas se destacam por levarem em conta a incerteza e a variabilidade da estimação. Por outro lado, podem influenciar no poder do teste à medida que o número de unidades faltantes aumenta. Por isso, outra alternativa a ser analisada seria a submissão dos dados originais ao Scott-Knott Modificado por Conrado (2017).

Para avaliar a performance de diferentes modelos e técnicas, pode-se utilizar Simulações de Monte Carlo, que serão usadas para mostrar o comportamento, de forma empírica, do α (Erro Tipo I) e $1 - \beta$ (Poder do Teste) nestas distintas abordagens em diversos cenários.

Neste trabalho foi utilizada a linguagem e o ambiente de programação R, que se destaca pela gratuidade e diversidade de fóruns e blogs de compartilhamento de dúvidas e algoritmos. Dentre os objetivos propostos neste trabalho está a criação de um pacote em linguagem R para compartilhar as implementações computacionais desenvolvidas. Por meio do pacote desenvolvido, pode-se utilizar a proposta de Conrado (2017) para seleção de tratamentos desbalanceados. Por meio das funções disponibilizadas no pacote, foram realizados experimentos de simulação para comparar o desempenho do método Scott-

Knott adaptado em relação à técnica de imputação de dados para utilização do Scott-Knott Tradicional.

Este trabalho está organizado de forma que na Seção 2 é apresentada uma revisão das principais teorias necessárias; na Seção 3 tem-se a implementação computacional do Pacote R, juntamente com sua disponibilização no repositório *GitHub* e a determinação do número de iterações a serem realizadas nas Simulações de Monte Carlo; e finalmente, na Seção 4, são apresentados os resultados e as considerações finais.

2 Revisão Bibliográfica

2.1 ANOVA

2.1.1 Modelo de Efeitos

Supondo um experimento com k tratamentos cujo interesse consiste em observar o comportamento destes em relação a uma variável quantitativa Y, torna-se de grande utilidade representar cada observação y_{ij} por meio de um modelo:

$$y_{ij} = \mu + \tau_i + \varepsilon_{ij} \begin{cases} i & = 1, 2, ..., k; \\ j & = 1, 2, ..., n_i. \end{cases} \tag{2.1.1}$$

onde,

y_{ij} é a j-ésima resposta do tratamento i;

ε_{ij} o erro associado a observação j do tratamento i;

μ o efeito comum;

τ_i o efeito do tratamento i com $\sum_{i=1}^{k} \tau_i = 0$;

k o número de tratamentos;

n_i o número de repetições do tratamento i com $\sum_{i=1}^{k} n_i = N$.

Este denominado *Modelo de Efeitos Fixos* (Único Fator), apresenta μ como uma constante e τ_i como os desvios em relação a esta quando os tratamentos específicos são aplicados. Portanto, pode-se utilizar a Análise de Variância (ANOVA) proposta por Ronald Aylmer Fisher (1923) para testar se os efeitos dos tratamentos são ou não significativos, ou seja, para testar as hipóteses:

$$\begin{cases} H_0 : \tau_1 = \tau_2 = ... = \tau_k = 0; \\ H_1 : \tau_i \neq 0, \text{ para pelo menos um } i. \end{cases} \tag{2.1.2}$$

2.1.2 Pressupostos para a Realização do Teste ANOVA

Antes de aplicar o teste, deve-se observar se os dados apresentam as seguintes características:

(i) erros independentes e identicamente distribuidos;

(ii) erro segue uma Distribuição Normal com média 0;

(iii) erros com variâncias comuns, ou seja, homocedasticidade.

Estes pressupostos implicam que as observações devam ser mutualmente independentes e $y_{ij} \sim N(\mu + \tau_i, \sigma^2)$.

2.1.3 Tabela de Análise de Variância

Desenvolvida para sumarizar e simplificar a apresentação dos dados, a tabela de Análise de Variância é amplamente difundida e utilizada (Tabela 1).

Tabela 1: Tabela de Análise de Variância para o Modelo de Efeitos Fixos (Único Fator).

Fonte de Variação	**Graus de Liberdade**	**Soma de Quadrados**	**Quadrado Médio**	**F**
Tratamento	k-1	$SS_{Tratamento}$	$MS_{Tratamento}$	$\frac{MS_{Tratamento}}{MS_E}$
Erro	N-k	SS_{Erro}	MS_E	
Total	N-1	SS_{Total}		

Fonte: Adaptado de (MONTGOMERY, 2017, p. 73).

Observe que a soma total dos quadrados (SS_{Total}) é dividida entre a soma dos quadrados devido aos tratamentos ($SS_{Tratamento}$) e a soma dos quadrados devido ao erro (SS_{Erro}):

$$\begin{gathered} SS_{Total} = SS_{Tratamento} + SS_{Erro}; \\ \sum_i \sum_j (y_{ij} - \bar{y}_{..})^2 = \sum_i n_i (\bar{y}_{i.} - \bar{y}_{..})^2 + \sum_i \sum_j (y_{ij} - \bar{y}_{i.})^2. \end{gathered} \tag{2.1.3}$$

Com Soma de Quadrados dividido pelo seu respectivo Grau de Liberdade,

$$\begin{gathered} MS_{Tratamento} = \frac{SS_{Tratamento}}{k-1}; \\ MS_E = \frac{SS_{Erro}}{N-k}. \end{gathered} \tag{2.1.4}$$

Derivou-se a Estatística do Teste:

$$F_{Observado} = \frac{MS_{Tratamento}}{MS_E}, \text{ Sob } H_0 \text{ verdadeiro, } F \sim F_{\alpha;\,(k-1,\,N-k)}. \tag{2.1.5}$$

Com $|F_{obs}| > |F_{crit}|$, conclui-se haver evidências que nos levam a rejeitar H_0, ou seja, que pelo menos um tratamento produz efeito significativo em Y.

Quando o número de observações em cada tratamento é diferente (*Design* Desbalanceado), duas modificações são necessárias:

$$\begin{aligned} SS_{Total} &= \sum_{i=1}^{a}\sum_{j=1}^{n_i} y_{ij}^2 - \frac{y_{..}^2}{N}; \\ SS_{Tratamento} &= \sum_{i=1}^{a} \frac{y_{i.}^2}{n_i} - \frac{y_{..}^2}{N}. \end{aligned} \tag{2.1.6}$$

Deve-se priorizar a realização de *designs* equilibrados por duas razões (MONTGOMERY, 2017, p. 78): O poder do teste é maximizado quando as amostras são de tamanhos iguais e a estatística do teste é relativamente insensível a pequenos desvios da suposição de homocestaticidade de variâncias, exceto se as amostras forem de tamanhos diferentes.

2.2 Scott-Knott

O teste de Scott-Knott utiliza a razão de verossimilhança para testar se subgrupos distintos, divididos de forma a maximizar a soma de quadrados, são significantes por meio da distribuição χ^2.

Após o teste de ANOVA rejeitar H_0, deve-se seguir estes passos para realizar o Teste Scott-Knott:

1. hipóteses;

$$\begin{cases} H_0 : G_1 = G_2 \text{ (Subgrupos de tratamentos não diferem);} \\ H_1 : G_1 \neq G_2 \text{ (Subgrupos de tratamentos diferem).} \end{cases}$$

2. calcular a média de y para cada um dos k tratamentos e ordena-las;

3. gerar, por meio da divisão das médias ordenadas em duas partições sucessivas, os $k-1$ subgrupos possíveis (Por exemplo, os tratamentos A, B, C e D podem ser particionados em 3 subgrupos: A vs BCD, AB vs CD e ABC vs D);

4. para cada subgrupo, calcular a soma de quadrados B_0;

$$B_0 = \frac{T_1^2}{k_1} + \frac{T_2^2}{k_2} - \frac{(T_1 + T_2)^2}{k_1 + k_2}. \tag{2.2.1}$$

onde,

T_g é a soma das médias no sugbrupo $g,\ g = 1, 2$;

k_g o número de médias no sugbrupo $g,\ g = 1, 2$.

5. determinar o subgrupo que maximiza a soma de quadrados;

6. calcular o estimador de máxima verossimilhança σ^2;

$$\widehat{\sigma}_0^2 = \frac{[\sum_{i=1}^{k}(y_i - \bar{y})^2 + vs^2]}{k + v};\ s^2 = \frac{MS_E}{n}. \tag{2.2.2}$$

onde,

v é o graus de liberdade do erro;

n o número de observações nos tratamentos (ou blocos);

y_i a média do tratamento k;

$\bar{y}$ a média de todos os tratamentos envolvidos na comparação.

Ou seja, os números de repetições (ou blocos) em cada tratamento precisam ser os mesmos.

7. calcular a Estatística do Teste λ para a partição que maximizou B_0;

$$\lambda = \frac{\pi}{2(\pi - 2)} * \frac{B_0}{\widehat{\sigma}_0^2}. \tag{2.2.3}$$

8. se $\lambda > \chi^2_{\alpha, \frac{k}{(\pi-2)}}$, tem-se evidências para rejeitar a hipótese nula, ou seja, evidências de que os dois subgrupos diferem;

9. em caso de rejeição de H_0, deve-se efetuar o teste para cada subgrupo gerado, de forma independente, até que os subgrupos não rejeitem a hipótese nula ou que fiquem com apenas uma média.

Observe que os grupos são separados de forma a classificar determinada média em apenas uma partição; são cálculos extensos para serem feitos à mão; e, pela estrutura do teste, o controle da Taxa de Erro Tipo I aparenta ser problemático, principalmente para grandes números de tratamentos significativos.

Entretanto, após simulações computacionais em duzentos e cinquenta e dois mil (252.000) experimentos inteiramente casualizados, Silva, Ferreira e Bearzoti (1999, p. 694)

concluíram que a

> [...] utilização do teste de Scott-Knott é recomendada por possuir poder elevado, taxas de erro tipo I quase sempre de acordo com os níveis nominais e por apresentar resultados com ausência de ambigüidade.

2.2.1 Exemplo de Aplicação do Teste de Skott-Knott

Considerando a Tabela 2 como resultado de um experimento fictício, o pesquisador, após realizar o teste de ANOVA (Tabela 3), realizou os passos do teste de Skott-Knott para investigar quais grupos de tratamentos diferem entre.

Tabela 2: Dados fictício para a aplicação do Scott-Knott.

Tratamento	**Observações**			
A	8	9	7	6
B	0	1	-1	2
C	-2	-4	-4	-1

Fonte: Elaboração própria.

Após realizar a ANOVA, concluiu-se haver pelo menos 2 tratamentos que diferem entre si.

Tabela 3: Tabela de Análise de Variância para o Modelo de Efeitos Fixos (Único Fator) do Exemplo de Aplicação do Teste de Skott-Knott.

Fonte de Variação	**Graus de Liberdade**	**Soma de Quadrados**	**Quadrado Médio**	**F**
Tratamento	2	219.50	109.75	$\frac{109.75}{1.86} = 6.73e - 06$
Resíduos	9	16.75	1.86	
Total	11	236.25		

Fonte: Elaboração própria.

Teste de Scott-Knott:

1. calcular a média de y para cada um dos $k = 3$ tratamentos e ordena-las;

$$\bar{y}_A = \frac{8+9+7+6}{4} = 7,5;$$
$$\bar{y}_B = \frac{0+1-1+2}{4} = 0,5;$$
$$\bar{y}_C = \frac{-2-4-4-1}{4} = -2,75.$$

2. gerar os $3 - 1 = 2$ subgrupos possíveis;

$$G_1 = A \ vs \ BC;$$
$$G_2 = AB \ vs \ C.$$

3. para cada subgrupo, calcular a soma de quadrados B_0;

   - $A \ vs \ BC$:

$$B_0 = \frac{7,5^2}{1} + \frac{(0,5+(-2,75))^2}{2} - \frac{(7,5+0,5+(-2,75))^2}{1+2} = 49,59375.$$

   - $AB \ vs \ C$:

$$B_0 = \frac{(7,5+0,5)^2}{2} + \frac{(-2,75)^2}{1} - \frac{(7,5+0,5+(-2,75))^2}{2+1} = 30,375.$$

4. o subgrupo que maximiza a soma de quadrados é $A \ vs \ BC$;

5. calcular o estimador de máxima verossimilhança σ^2;

$$v = 9;$$
$$MS_E = 1,86;$$
$$\bar{y} = \frac{7,5+0,5+(-2,75)}{3} = 1,75;$$
$$s^2 = \frac{1,86}{4} = 0,465.$$

Tabela 4: Cálculos auxiliares.

Tratamento	$\bar{y}_i$	$\bar{y}$	$(\bar{y}_i - \bar{y})^2$
A	7,5	1,75	33,0625
B	0,5	1,75	1,5625
C	-2,75	1,75	20,25
Soma	-	-	54,875

Fonte: Elaboração própria.

$$\widehat{\sigma}_0^2 = \frac{(54,875 + 9 * 0.465)}{3+9} = 4,921667.$$

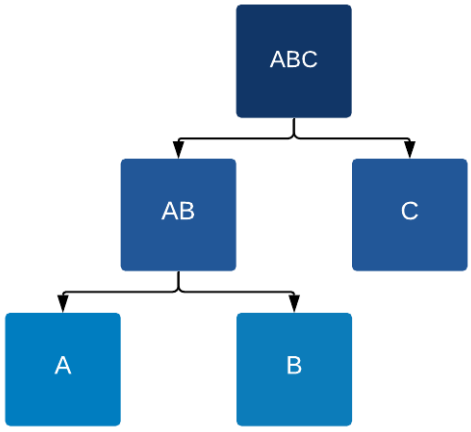
6. calcular a Estatística do Teste λ para a partição que maximizou B_0;

$$\lambda = \frac{\pi}{2(\pi - 2)} * \frac{49,59375}{4,921667} = 13,86511.$$

7. como $\lambda > \chi^2_{0,05,\frac{3}{(\pi-2)}} = 7,158978$, tem-se evidência para não aceitar a hipótese nula, ou seja, evidências de que os subgrupos AB vs C diferem para um $\alpha = 0,05$.

Fazendo os passos 3 a 6 no subgrupo A vs B também rejeitaremos a hipótese nula, chegando à conclusão de que todos os tratamentos diferem entre si (Fluxograma).

Figura 1: Fluxograma do Exemplo de Aplicação do Teste de Skott-Knott.



Fonte: Elaboração própria.

2.2.2 Scott-Knott Modificado

A modificação proposta por Conrado consiste na utilização de s_c^2 como alternativa a s^2 para calcular o σ na Equação 2.2.2.

$$s_c^2 = \frac{1}{k}\sum_{i=1}^{k}(SE_{\bar{y}_i});\ SE_{\bar{y}_i} = \text{desvio-padrão de } \bar{y}_i. \tag{2.2.4}$$

Com essa modificação, a restrição de tratamentos balanceados é contornada, fazendo com que o Teste possa ser utilizado também em experimentos de dados faltantes.

2.3 Imputação Múltipla (IM)

Nos anos 70, D.B. Rubin propôs o Método de Imputação Múltipla (IM) como uma alternativa mais eficaz e robusta de imputação para dados faltantes em relação aos métodos de imputação simples. Por se tratar de um método mais trabalhoso, sua realização só ganhou popularidade com o advento computacional.

Para realizar o IM e garantir que a imputação leve em conta a incerteza sobre o dado faltante, variabilidade e maior precisão da estimação, deve-se realizar os 3 passos descritos por Nunes, Klück e Fachel (2009):

1. gerar m banco de dados, onde os dados ausentes devem ser imputados de acordo com uma técnica adequada de imputação;
2. utilizar um método estatístico tradicional para analisar os m bancos de dados completos separadamente (com dados ausentes imputados no passo 1);
3. obter a inferência da imputação repetida.

Os passos 1 e 2 requerem grande atenção do pesquisador, pois a utilização de um método inadequado diminui a precisão da estimativa. Segundo Stef Van Buuren (2018), o modelo de imputação deve preservar as relações nos dados e a incerteza sobre as relações para que resulte em inferências estatísticas válidas.

Já o passo 3 consiste no cálculo da estimativa combinada $\bar{Q} = \frac{1}{m}\sum_{\ell=1}^{m}\hat{Q}_\ell$ onde $\hat{Q}_\ell$ é a ℓth imputação repetida; e no cálculo da variância combinada $T = \bar{U} + B + \frac{B}{m} = \bar{U} + \left(1 + \frac{1}{m}\right)B$, onde $\bar{U} = \frac{1}{m}\sum_{\ell=1}^{m}\bar{U}_\ell$ é a variância dentro das imputações, $B = \frac{1}{m-1}\sum_{\ell=1}^{m}(\hat{Q}_\ell - \bar{Q})(\hat{Q}_\ell - \bar{Q})'$ a variância entre imputações, e $\bar{U}_\ell$ é a matriz de variância-covariância de $\hat{Q}_\ell$.

2.3.1 Imputação por Regressão Linear Bayesiana com Pacote `mice`

O pacote R `mice`, desenvolvido por Stef Van Buuren e Groothuis-Oudshoorn em 2011, que aplica todos os passos de IM descritos por Rubin através de diversos métodos de imputação, realiza, por padrão, o Método de Imputação por Correspondência Média Preditiva (parâmetro *pmm*). Entretanto, os dados a serem utilizados na simulação seguem uma distribuição Normal, e "[...] se for sabido que a variável está próxima da distribuição normal, usar *norm* em vez do padrão (*pmm*) pode ser mais eficiente."(VAN BURREN, 2018, cap. 6.3, tradução nossa)[1].

[1]No original: [...]if it is known that the variable is close to normally distributed, using *norm* instead of the default *pmm* may be more efficient.

O pacote `mice` realiza o método de Imputação por Regressão Linear Bayesiana, também conhecida como modelo normal (parâmetro *norm*), pelos seguintes passos (VAN BURREN, 2018, cap. 3.2, tradução nossa)[2]:

1. calcula a matriz de produtos cruzados $S = X'_{obs}X_{obs}$;
2. calcula $V = (S + diag(S)\kappa)^{-1}$, com algum pequeno parâmetro de cume κ;
3. calcula pesos de regressão $\hat{\beta} = VX'_{obs}y_{obs}$;
4. sorteia uma variável aleatória $\dot{g} \sim \chi^2_\nu$ com $\nu = n_1 - q$;
5. calcula $\dot{\sigma}^2 = (y_{obs} - X_{obs}\hat{\beta})'(y_{obs} - X_{obs}\hat{\beta})/\dot{g}$;
6. sorteia q variáveis $N(0,1)$ independente no vetor $\dot{z}_1$;
7. calcula $V^{1/2}$ por decomposição de Cholesky;
8. calcula $\dot{\beta} = \hat{\beta} + \dot{\sigma}\dot{z}_1V^{1/2}$;
9. sorteia n_0 variáveis $N(0,1)$ independentes no vetor $\dot{z}_2$;
10. calcula os n_0 valores $y_{imp} = X_{mis}\dot{\beta} + \dot{z}_2\dot{\sigma}$.

Além disso, optou-se por utilizar $m = 20$, pois, de acordo com Stef Van Buuren (2018, cap. 2.8, tradução nossa)[3]

> [...], definir m alto pode não valer a pena a espera extra se o interesse principal estiver nas estimativas pontuais (e não no erro-padrão, p-valor e assim por diante). Nesse caso, usar *m=5-20* será suficiente [...].

2.4 Simulação de Monte Carlo

O Método de Monte Carlo foi formalizado oficialmente em 1949 através da publicação do artigo *Monte Carlo Method*, por John Von Neumann e Stanislav Ulam, mas seu surgimento ocorreu em 1946, durante um jogo de paciência: interessado em descobrir

[2]No original: 1. Calculate the cross-product matrix $S = X'_{obs}X_{obs}$; 2. Calculate $V = (S+diag(S)\kappa)^{-1}$, with some small κ; 3. Calculate regression weights $\hat{\beta} = VX'_{obs}y_{obs}$; 4. Draw a random variable $\dot{g} \sim \chi^2_\nu$ with $\nu = n_1 - q$; 5. Calculate $\dot{\sigma}^2 = (y_{obs} - X_{obs}\hat{\beta})'(y_{obs} - X_{obs}\hat{\beta})/\dot{g}$; 6. Draw q independent $N(0,1)$ variates in vector $\dot{z}_1$; 7. Calculate $V^{1/2}$ by Cholesky decomposition; 8. Calculate $\dot{\beta} = \hat{\beta} + \dot{\sigma}\dot{z}_1V^{1/2}$; 9. Draw n_0 independent $N(0,1)$ variates in vector $\dot{z}_2$; 10. Calculate the n_0 values $y_{imp} = X_{mis}\dot{\beta} + \dot{z}_2\dot{\sigma}$.

[3]No original: *[...], setting m high may not be worth the extra wait if the primary interest is on the point estimates (and not on standard errors, p -values, and so on). In that case using m=520 will be enough [...].*

a probabilidade de sucesso de determinada jogada, Ulam percebeu que realizar inúmeras jogadas seria mais simples que realizar os cálculos de análise combinatória.

Esse método consiste na criação de um determinado número de amostras, segundo uma função densidade de distribuição de probabilidade definida, e observar o resultado obtido por meio da média, desvio-padrão ou outra técnica estatística de interesse.

Entretanto, devido a se tratar de simulações computacionais, as amostras são geralmente construídas por meio de números pseudoaleatórios através de algoritmos computacionais capazes de gerar números de forma próxima à aleatoriedade natural. Neste trabalho, utilizou-se a função `rnorm(...)` do R, capaz de gerar números pseudoaleatórios com Distribuição de Probabilidade Normal.

2.4.1 Encontrando Valores Esperados Usando Simulação de Monte Carlo

De acordo com Jan Palczewski e Andrzej Palczewski ([21- -?]), tomando $X_1,\ X_2,\ ...$ uma sequência de variáveis aleatórias independentes distribuídas de forma idêntica com valor esperado μ e variação σ^2. Definindo-se Y_n como a média dessa sequência, ou seja,

$$Y_n = \frac{X_1 + X_2 + ... + X_n}{n},\ n = 1, 2, ... \tag{2.4.1}$$

Tem-se que, de acordo com a *Lei Forte dos Grandes Números*, Y_n converge quase certamente para μ quando $n \to \infty$.

Portanto, supondo que tenhamos uma variável aleatória X com esperança e variância desconhecidas $a = E[X]$ e $b^2 = Var[X]$.

Estamos interessados em calcular uma aproximação para a. Para tal, podemos obter amostras independentes de X por meio de um gerador de números pseudo-aleatórios, pois já vimos que, pela *Lei Forte dos Grandes Números*, a média de um grande número de amostras pode dar uma boa aproximação do valor esperado. Portanto,

$$a_M = \frac{1}{M}\sum_{i=1}^{M} X_i, \tag{2.4.2}$$

é um bom estimador de a, onde X_i representa o resultado da i-ésima iteração, com $i = 1, 2, .., M$.

2.4.2 Determinando o Número de Iterações

Paula (2014, p. 52) descreve, em sua dissertação, 4 passos básicos para definir o número de iterações necessárias para se obter a com determinado nível de confiança:

1. especificar a difirença máxima entre a_M e a (optou-se, nesse trabalho, por representar essa diferença por meio do símbolo ϱ) para um nível α e encontrar o valor $d = \frac{\varrho}{z_{(1-\frac{\alpha}{2})}}$. Por exemplo, com 95% de confiança e esperando uma diferença máxima de $\varrho = 0,01$, tem-se que $d = \frac{0,01}{z_{(0,975)}} = \frac{0,01}{1,96} = 0,0051$;

2. gerar, no mínimo, 100 iterações;

3. avaliar se $\frac{S}{\sqrt{M}} < d$, onde S é o desvio-padrão baseado nas M interações. Caso não satisfaça a condição, continuar gerando novas iteração até que se obtenha um valor M que a satisfaça;

4. estimar a por meio do a_M.

O objetivo desses passos é encontrar um valor M que permita ao pesquisador calcular, com $(1-\alpha)\%$ de confiança, uma estimativa a_M que não difira de a mais que $(z_{(1-\frac{\alpha}{2})})d$.

3 Metodologia

Nessa seção, serão apresentadas a implementação computacional do pacote R (Subseção 3.1), juntamente com sua disponibilização no repositório *GitHub* (Subseção 3.2 e Subseção 3.3) e a determinação do número de iterações a serem realizadas nas Simulações de Monte Carlo (Subseção 3.4).

Na implementação do pacote que realiza o teste modificado por Conrado, desenvolveu-se uma função através do *software R* utilizando dois pacotes do conglomerado *Tidyverse*, implementado por Lionel Henry e Hadley Wickham com o apoio do *Rstudio*.

O primeiro, `purr`, é uma alternativa às funções em *loop*, não por questões de desempenho, mas como uma alternativa visualmente mais simples para executar determinada função em todos os elementos de um vetor (WICKHAM; GROLEMUND, 2017). Neste trabalho, será utilizado para aplicar funções a todos os elementos de determinada lista.

`Dplyr`, o segundo pacote *Tidyverse* utilizado, apresenta uma ótima estrutura gramatical para a manipulação e visualização de banco de dados, e foi utilizado na apresentação da tabela com os resultados (por motivos estéticos, optou-se pela supressão das mensagens e avisos através do pacote `pkgcond`). E, para uma apresentação mais visual, disponibilizou-se duas opções gráficas: via `ggplot2`, também do conglomerado *Tidyverse* e `plotly`, desenvolvido pela empresa *Plotly*.

Para disponibilizar a implementação computacional em forma de pacote, utilizou-se de diversas funções do pacote `devtools`[4] com intuito de criar o "esqueleto" com as pastas e arquivos, para personalizar o site e executar algumas verificações; testes unitários foram realizados por meio do pacote `testthat`; e utilizou-se do *GitHub* como repositório.

Por fim, foram feitos experimentos em Simulações de Monte Carlo (METROPOLIS; ULAN, 1949) em diferentes cenários para a obtenção, de forma empírica, do Erro Tipo I e do Poder do Teste utilizando duas metodologias distintas: (i) Scott-Knott Modificado para dados ausentes; (ii) Imputação dos dados faltantes através do algoritmo MICE para Imputação Múltipla (desenvolvido por Stef Van Buuren) implementado no pacote `mice` (criado por Stef Van Buuren e Groothuis-Oudshoorn) e posteriormente a aplicação do Scott-Knott Tradicional, com banco de dados completos. Todos esses resultados são apresentados graficamente por meio do *Microsoft Office 365*.

[4] Alternativamente, pode-se utilizar a função `package.skeleton` do pacote `utils`.

3.1 Algoritmo

O pacote é composto por sete funções, sendo quatro (`ANOVA(...)`, `scott(...)`, `number_to_letters(...)` e `RUNscott(...)`) de cálculos internos e três (`usktest(...)`, `plot_usk(...)` e `plotly_usk(...)`) com impressão ao usuário.

Apesar de cada uma possuir seus parâmetros de entrada, o usuário só precisa especificar aqueles referentes a função `usktest(...)`:

1. **formula**: fórmula utilizada para fazer o teste de ANOVA via `aov(...)` (No momento, este pacote só realiza o Scott-Knott desbalanceado para análise de variância de fator único, por isso a **formula** deverá ser do tipo *observação* $\sim$ *tratamento*);
2. **dataset**: indicação do banco de dados a ser utilizado;
3. **alpha**: Erro Tipo I que o pesquisador está apto a aceitar (Por padrão, tem-se 0,05);
4. **ANOVA**: informação sobre se o usuário quer ou não a saída da tabela da ANOVA (Por padrão, tem-se *TRUE*).

3.1.1 `ANOVA(...)`

O R efetua a ANOVA para experimentos inteiramente casualizados através da função `aov(...)`, que foi indexada na função `ANOVA(...)` para utilização dos resultados sem a necessidade de refazer os cálculos sempre que necessário.

Algorithm 1: Função que calcula a ANOVA

$$\begin{aligned}
\textbf{ANOVA} \leftarrow\ & \textit{function}(dataset, var1, var2)\{ \\
& modelo \leftarrow aov(dataset[[var1]] \sim dataset[[var2]]) \\
& \textit{return}(modelo) \\
& \}
\end{aligned}$$

3.1.2 `scott(...)`

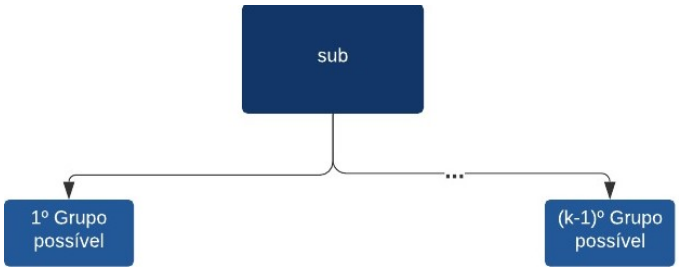
Essa função contém todas as fórmulas descritas em **2.2 Scott-Knott** (Subseção 2.2), onde optou-se, em alguns momentos, por trabalhar com listas e, para aplicar funções em cada elemento destas, utilizou-se a função `map(...)` do pacote `purr`.

Primeiro, a função calcula as estatísticas básicas, como médias e o número de tratamentos.

Sub e *SubG* são listas contendo todos os subgrupos possíveis: *Sub* (Figura 2), é uma lista vazia contendo $k-1$ componentes, que será preenchida com os tratamentos

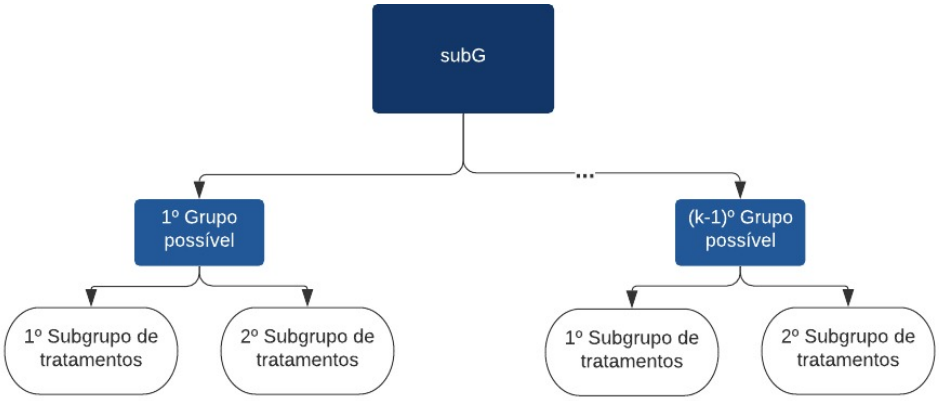
e suas médias posteriormente, através da função `map(...)`, gerando a *sugG* (Figura 3). Observe que esse é o passo 3 do Skott-Knott (Gerar os $k-1$ subgrupos possíveis).

Figura 2: Lista *sub*



Fonte: Elaboração própria.

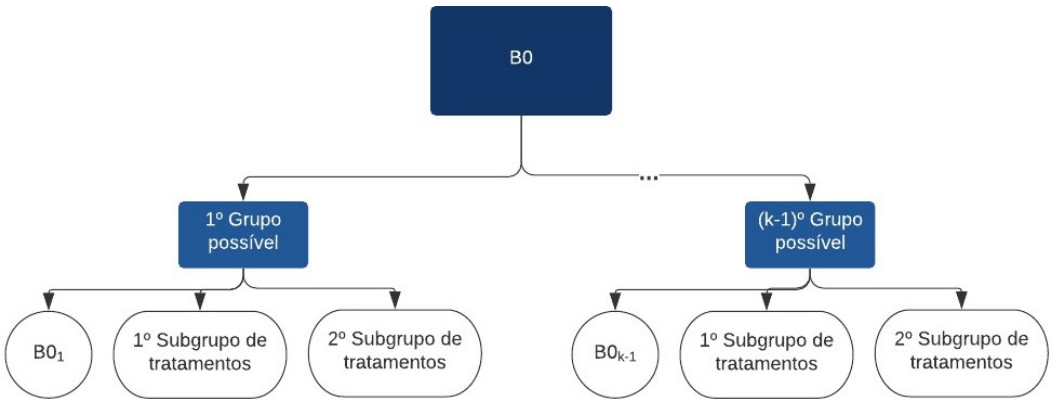
Figura 3: Lista *subG*



Fonte: Elaboração própria.

Na Lista *B0*, novamente utilizou-se `map(...)` aplicado à *sub* para calcular *B0* de cada um dos componentes, retornando a lista na estrutura da Figura 4.

Figura 4: Lista *B0*



Fonte: Elaboração própria.

Os próximos passos, que concluem os cálculos de `scott(...)`, são intuitivos e a descrição de cada etapa presente no algoritmo é o suficiente para seu entendimento.

Algorithm 2: PARTE I - Função com as fórmulas do Scott-Knott Modificado.

```
scott ← function(dataset, var1, var2, alpha, v, MSE){
        yi ← sort(tapply(dataset[[var1]], dataset[[var2]],
                mean, na.rm = TRUE), decreasing = TRUE)
        bar_y ← mean(yi)
        k ← length(yi)
#Function to obtain the 'n' of each Treatment without counting the 'NA'
        length.na.rm ← function(x){sum(!is.na(x))}
#s^2 modified by Conrado : s_c^2 = 1/k Σ_{i=1}^{k} (SE_{ȳ_i})
        s2_c ← mean(MSE/tapply(dataset[[var1]], dataset[[var2]],
                length.na.rm))
#List with the number of possible subgroups (k − 1)
        sub ← as.list(1 : (k − 1))
#Creating the subgroups
        subG ← map(sub, function(sub){list(yi[c(1 : sub)],
                yi[−c(1 : sub)])})
#Calculating B0 for each of the subgroups
        B_0 ← as.list(map(as.numeric(sub), function(x){
        #Defining
            T1 ← sum(subG[[x]][[1]])
            T2 ← sum(subG[[x]][[2]])
            k1 ← length(subG[[x]][[1]])
            k2 ← length(subG[[x]][[2]])
        #Formula of B_0 = T_1^2/k_1 + T_2^2/k_2 − (T_1+T_2)^2/(k_1+k_2)
            list((T1^2/k1) + (T2^2/k2)−
                (sum(c(T1, T2))^2)/sum(c(k1, k2)), subG[[x]])
        }))
#Getting the biggest B_0
        B0max ← B0[which.max(map(B0, 1))][[1]][[1]]
#Getting the set that gives the biggest B_0
        SubB0max ← B0[which.max(map(B0, 1))][[1]][[−1]]
#Calculating the sigma : σ̂_0^2 = [Σ_{i=1}^{k} (y_i − ȳ)^2 + vs^2]/(k+v); s^2 = MS_E/n
        sigma ← (sum((yi − bar_y)^2) + (v * s2_c))/(k + v)
#Calculating the lambda : λ = π/(2(π−2)) * B_0/σ̂_0^2
        lambda ← (pi/(2 * (pi − 2))) * (B0max/sigma)
```

Algorithm 3: PARTE II - Função com as fórmulas do Scott-Knott Modificado.

$\#Calculating\ the\ Chi - Square : \chi^2_{\alpha,\frac{k}{(\pi-2)}}$

$limite \leftarrow qchisq((1 - alpha), df = k/(pi - 2))$

$\#reject\ H_0\ if\ \lambda > \chi^2_{\alpha,\frac{k}{(\pi-2)}}$

$Rejeita \leftarrow ifelse(lambda > limite, TRUE, FALSE)$

$\#Output$

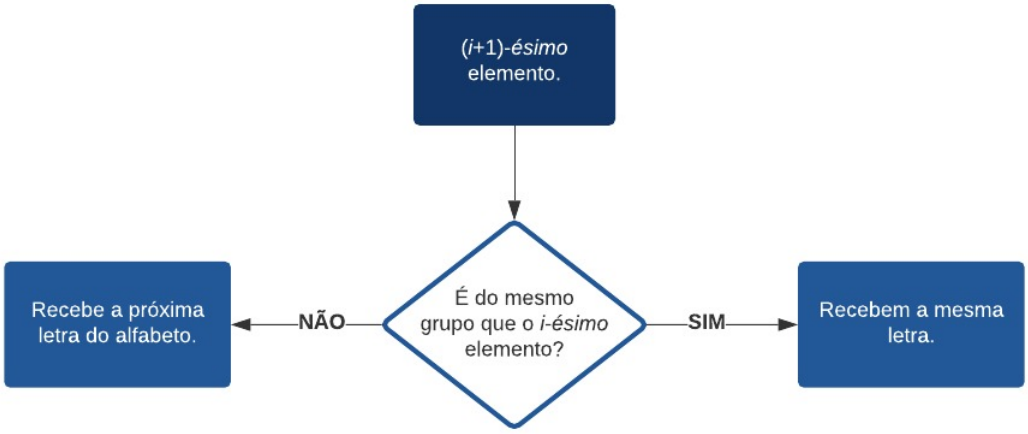
$return(list(rejeita = rejeita, SubB0max = SubB0max))$

}

3.1.3 `number_to_letters(...)`

Sendo uma função útil em `RUNscott(...)`, `number_to_letters(...)` tem por objetivo converter a classificação dos grupos de números para letras: a princípio, com esta função, pega-se um vetor, onde o primeiro elemento é classificado com a letra "a" e vai-se percorrendo todos os i elementos para classifica-los, onde o $(i+1)$-*ésimo* elemento recebe a mesma letra que o *i-ésimo* caso sejam do mesmo grupo e, se forem de grupos distintos, recebe a próxima letra do alfabeto (Figura 5).

Figura 5: Fluxograma com a regra para conversão de letra em número do $(i+1)$-*ésimo* elemento.



Fonte: Elaboração Própria.

Por exemplo, ao se entrar com um vetor contendo os elementos *{1, 2, 2, 3, 3, 3}*, nesta ordem, a função retornará *{"a", "b", "b", "c", "c", "c"}*; e se for na ordem *{3, 1, 2, 2, 3, 3}* retornará *{"a", "b", "c", "c", "d", "d"}*. Ou seja, a função não leva em consideração o valor, apenas se o $(i+1)$-*ésimo* é igual ao seu antecessor.

Algorithm 4: Função que converte números em letras.

```
number_to_letters ← function(a){
    j ← 1; n ← length(a)
    modificado ← c(); letras ← c(); repet ← ceiling(n/26)

#Set the maximum number of letters, where the alphabet repeats every 26 numbers
    for(r in 1 : repet){
        letras ← letras; reset ← 1
#1 to 26 : "a``, ..., "z``; 27 to 52 : "aa``, ..., "zz``; 53 to 78 : "aaa``, ..., "zzz``; ...
        for(l in (26 * r − 25) : (26 * r)){
            letras[l] ← paste0(c(as.character(rep(letters[reset], r))),
                collapse = "``)
            reset ← reset + 1
        }
    }
#Sorting the (i + 1) − th element
    for(i in 1 : (n − 1)){
#The first group receives the first letter ("a``)
        modificado[1] ← letras[1]
#The following treatments receive the same letter as the previous one
#(if they are from the same group)
        if(a[i + 1] == a[i]){
            modificado[i + 1] ← modificado[i]
            j ← j
#The following treatments receive the next letter
#(if they are not from the same group as the previous one)
        }else{
            modificado[i + 1] < −letras[j + 1]
            j ← j + 1
        }
    }
    return(modificado)
}
```

3.1.4 `RUNscott(...)`

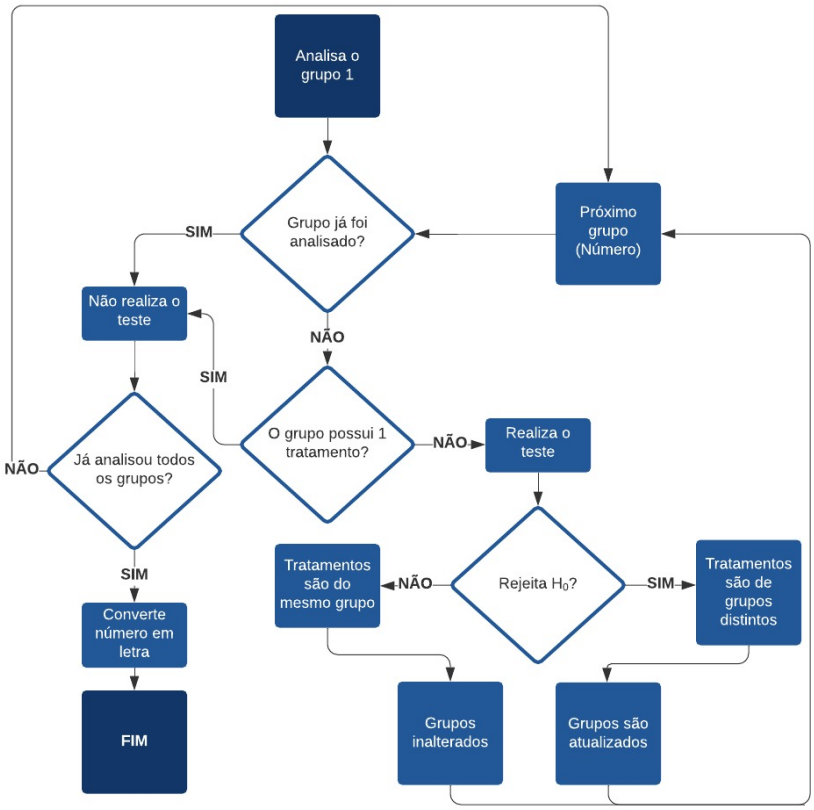
Sendo a etapa de execução, esta função aplica as regras de parada, padroniza e retorna, no formato *tibble*, ordenado de forma decrescente pela média de cada tratamento,

uma tabela contendo 6 colunas:

1. **Treatment**: tratamentos ao qual as outras colunas se referem;
2. **Group**: indicação do Grupo;
3. **Mean**: média de Y;
4. **min**: menor valor de Y;
5. **max**: maior valor de Y;
6. **Ordem**: coluna utilizada na implementação gráfica com o *rank*.

Para uma melhor organização, criou-se duas funções externas contendo implementações importantes para a execução em `RUNscott(...)`: `scott(...)` e NUMBER_TO_LETTERS(...); a sumarização dos dados foi feita através da estrutura gramatical do pacote `Dplyr`; para suprimir as mensagens e avisos retornadas por esse pacote utilizou-se a função `suppress_messages(...)` do pacote `pkgcond`; e, como auxílio ao entendimento das regras de execução, tem-se o fluxograma (Figura 6) e comentários no código.

Figura 6: Fluxograma com as regras utilizadas na função `RUNscott(...)`.



Fonte: Elaboração Própria.

Algorithm 5: PARTE I - Função que executa o teste de Scott-Knott Modificado.

```
RUNscott ← function(dataset, var1, var2, alpha, v, MSE){
    dados ← dataset
    dados$Treatment ← dataset[[var2]]
    dados$y ← dataset[[var1]]
    i ← 1; Group ← 1
    dados$Group ← 1
#Running the test for all possible partitions and sorting the groups
#Group 1, 2, ..., up to how many groups there are
    while(i <= max(Group)){
#If there is more than 1 treatment in the Group: take the test
        if(length(unique(dados[which(dados$Group == i), ]$Treatment)) > 1){
#Running the modified scott − knott test for the i−th group
            Segundo ←
            scott(dados[which(dados$Group == i), ], var1, var2, alpha, v, MSE)
#If H0 is rejected: assign the Groups
            if(Segundo[[1]][[1]] == TRUE){
#It will be assigned with different number than any existing one
                Group ← c(as.numeric(max(dados$Group))
                        +1, as.numeric(max(dados$Group)) + 2)
#The code splits into 2 groups (Makes to 1 and then to 2)
                for(j in 1 : 2){
#Get from the first to the last treatment belonging to this group
                    for(k in 1 : length(Segundo[[2]][[j]])){
#Look at all the lines of the data to see if in the nr − th line we have the treatment
                        for(nr in 1 : nrow(dados)){
#Put the group on the line of your treatment
                            if(is.element(dados$Treatment[nr],
                            names(Segundo[[2]][[j]])) == TRUE){
                            dados$Group[nr] ← Group[j]
                            }
                        }
                    }
                }
#If H0 was rejected, the 'while' is updated
                Group ← as.numeric(levels(as.factor(dados$Group)))
            }
        }
```

Algorithm 6: PARTE II - Função que executa o teste de Scott-Knott Modificado.

```
    #Updating the 'i' of the 'while'
            i ← i + 1
        }
    #Table with the data
    #Looking at the result with suppress messages
        suppress_messages(
            R ← dados% > %group_by(Treatment, Group)% > %
                summarise("Mean" = round(mean(y, na.rm = TRUE), 2),
                min = min(y, na.rm = TRUE),
                max = max(y, na.rm = TRUE))% > %
                    arrange(desc("Mean")))
    #'Ordem' will only be used in the chart (do not print to the user)
        R$Ordem ← as.numeric(row.names(R))
    #Transforming from number to letter
        R$Group ← c(number_to_letters(as.numeric(R$Group)))
    #In the graph it is important that the group is a factor
        R$Group ← as.factor(R$Group)
        return(R)
    }
```

3.1.5 `usktest(...)`

Além de receber os parâmetros do usuário, esta função realiza as checagens de erro e padroniza a saída: exibe o resultado do teste de Skott-Knott Modificado juntamente, caso o usuário deseje, com a tabela de ANOVA e retorna os valores em formato de lista.

3.1.5.1 Configurando as Mensagens de Erros da Função

Para um bom funcionamento, o usuário precisa seguir alguns requisitos básicos que, caso descumpridos, retornarão os seguintes avisos:

1. **Fórmula não está no formato** *'observation ∼ treatment'*: *"At the moment, this package only does the Unbalanced Scott-Knott for single factor analysis of variance, so your 'formula' must be 'observation ∼ treatment'"* (No momento, este pacote só faz o Scott-Knott desbalanceado para análise de variância de fator único, então sua 'fórmula' deve ser *'observation ∼ treatment'*);

2. **variável indicando os tratamentos não é do tipo fator:** *"The variable [...]*

has been changed to format 'factor'" (A variável [...] foi alterada para o formato 'factor');

3. **variável indicando os resultados métricos do experimento não é do tipo numérico:** *"The variable [...] must be numeric"* (A variável [...] deve ser numérica);

4. **testando apenas 1 tratamento:** *"The variable [...] must have more than 1 type of treatment"* (A variável [...] deve ter mais de 1 tipo de tratamento);

5. **tratamentos contam com apenas 1 observação:** *"All [...] must have more than 1 observations"* (Todas [...] devem ter mais de 1 obervação).

Com exceção do segundo aviso, que informa ao pesquisador que o teste foi realizado após transformar a variável responsável por representar os tratamentos em um fator, as demais retornam apenas a mensagem de erro, sem que nenhum teste seja executado.

3.1.5.2 Exibição dos Resultados da `usktest(...)`

O resultado é no formato de lista e contém 8 componentes, quando o parâmetro $ANOVA$ é atribuído com $FALSE$ e 9, caso contrário:

1. [[**Variable of observations**]]**:** Nome da variável contendo os resultados métricos do experimento;

2. [[**Variable of treatment**]]**:** nome da variável contendo os tratamentos;

3. [[**ANOVA**]]**:** lista com 5 elementos contendo o resultado do teste de ANOVA (Apenas se $ANOVA == TRUE$);

   I **DF:** graus de liberdade;

   II **Sum Sq:** somas dos quadrados;

   III **Mean Sq:** quadrados médios;

   IV **F value:** estatística do teste;

   V **Pr(>F):** p-valor.

4. **treatments:** fator contendo os nomes dos tratamentos;

5. **Group:** fator contendo os nomes dos grupos;

6. **Mean:** média de Y para cada tratamento;

7. **min:** menor valor de Y para o tratamento;

8. **max:** maior valor de Y para o tratamento;

9. **Ordem:** "Rank" do tratamento de acordo com a média;

Algorithm 7: PARTE I - Função responsável por receber os parâmetros do usuário para execução da ANOVA e do Teste de Skott-Knott Modificado.

```
usktest ← function(formula, dataset, alpha = 0.05, ANOVA = TRUE){
    var1 ← as.character(formula[[2]])
    var2 ← as.character(formula[[3]])]]
#Checking for possible errors
    if(length(as.character(formula[[3]])) > 1){
        verifications ← list(c(length(as.character(formula[[3]])) > 1,
            "At the moment, this package only does the Unbalanced
            Scott − Knott for single factor analysis of variance, so your
            'formula' must be 'observation ~ treatment'``))
    }else{
        if(is.factor(dataset[[var2]]) == FALSE){
            warning(paste0("The variable '``, as.character(formula[[3]]),
            "' has been changed to format 'factor'``))
            dataset[[var2]] < −as.factor(dataset[[var2]])
        }
        verifications ← list(
            c(is.numeric(dataset[[var1]]) == FALSE,
                paste0("The variable '``, as.character(formula[[2]]),
                "' must be numeric``)),
            c(length(unique(dataset[[var2]])) < 2,
                paste0("The variable '``, as.character(formula[[3]]),
                    "' must have more than 1 type of treatment``)),
            c(sum(tapply(dataset[[var1]], dataset[[var2]], function(x){
                sum(is.na(x))}) == table(dataset[[var2]])) > 0,
                    paste0("All '``, as.character(formula[[3]]),
                        "' must have more than 1 observations``))
        )
    }
    message ← as.character("``); count_errors ← 0
        for(i in 1 : length(verifications)){
            message ← ifelse(verifications[[i]][[1]] == TRUE, paste(message,
                as.character(verifications[[i]][[2]]), sep = "``), message)
            count_errors ← ifelse(verifications[[i]][[1]] == TRUE,
                count_errors + 1, count_errors)
        }
```

Algorithm 8: PARTE II - Função responsável por receber os parâmetros do usuário para execução da ANOVA e do Teste de Skott-Knott Modificado.

```
        if(count_errors > 0){
        stop(paste(message))
        }else{
    #Running the Modified Skott - Knott Test
            Result ← RUNscott(dataset, var1, var2, alpha, ANOVA(formula, dataset)
                    ["df.residual``]], summary(ANOVA(formula, dataset))
                    [[1]][["MeanSq``]][[2]])
            colnames(Result)[1] ← var2
            Resultado ← as.list(c(var1, var2, Result))
            names(Resultado)[1 : 2] ← c("Variable of observations``,
                                        "Variable of treatment``)
    #Running the ANOVA Test (if the user wants to)
            if(ANOVA == TRUE){
                Anova ← summary(ANOVA(formula, dataset))
                print("##############ANOVA############``)
                print(Anova)
                print("############Scott - Knott##########``)
                Resultado ← as.list(c(var1, var2, Anova, Result))
                names(Resultado)[1 : 3] ← c("Variable of observations``,
                                            "Variable of treatment``,"ANOVA``)
            }
    #Printing the results in the form of tables
        print(subset(Result, select = −c(Ordem)))
    #Returning results in unprinted 'list' format
        invisible(Resultado) } }
```

3.1.6 `plot_usk(...)`

Apesar deste pacote contar com uma função de recursos gráficos mais avançados (`plotly_usk(...)`), optou-se pela habilitação de `plot_usk(...)` devido à grande familiaridade que os usuários da linguagem R possuem com o `ggplot2`.

`plot_usk(...)` recebe como parâmetro de entrada a `usk_test(...)` devidamente preenchida e utiliza seus resultados, no formato de lista, para elaborar um gráfico estático.

Algorithm 9: Função para visualização gráfica em *ggplot2*.

```
plot_usk ← function(test){
#Transforming the modified Skott-Knott result into data.frame
    data <-as.data.frame(test[c(test[[2]],"Group`,"Mean``,"min``,
        "max``,"Ordem``)])
#Setting the default Treatments name to 'Treatment'
        colnames(data)[1] <-c("Treatment`)
#Generating the graph
        graphic <-data% > %
    #Empty grid containing coordinates
            ggplot(aes(Treatment = Treatment, y = Ordem,
                    x = "Mean``, color = Group))+
    #Adding the points (averages) and caption block
            geom_point()+
    #Customizing the labels
            scale_y_continuous(breaks = c(data$Ordem),
                    labels = c(as.character(data[, 1])), name = test[[2]])+
    #Adding range (minimum to maximum)
            geom_errorbar(aes(xmin = min, xmax = max),
                    width = 0.2, size = 0.7)+
    #Setting the theme
            theme_bw()+
    #Title
            labs(title = "Scott - Knott`)
#Returning the graphic
        return(graphic)
    }
```

3.1.7 `plotly_usk(...)`

Retornando um gráfico dinâmico, com opções de *zoom*, *download* em diversos formatos e com recursos que facilitam a visualização da informação, está função utiliza como entrada a `usk_test(...)`, devidamente preenchida, mas utiliza `usk_plot(...)` para transformar o gráfico de `ggplot2` em `plotly`.

Algorithm 10: Função para visualização gráfica em *plotly*.

```
plotly_usk ← function(test){
#Using ggplot2 as a base
    ggplot_usk <−plot_usk(test)
#Transforming ggplot2 into plotly
    graphic <−ggplotly(ggplot_usk, tooltip = c("Treatment`,"x``,"Group`))% > %
    #Centering the title
        layout(title = "Scott − Knott`,
    #Customizing the labels and caption
            xaxis = list(title = "Mean``, titlefont = list(
                family = "Courier New, monospace``, size = 18,
                color = "#7f7f7f`)),
            yaxis = list(title = "``),
            showlegend = TRUE)
#Returning the graphic
        return(graphic)
    }
```

3.2 *Git* e *GitHub* no *RStudio*

Nessa seção, serão apresentadas a implementação computacional do pacote R (Subseção 3.1) juntamente com sua disponibilização no repositório *GitHub* (Subseção 3.2 e Subseção 3.3) e a determinação do número de iterações a serem realizadas nas Simulações de Monte Carlo (Subseção 3.4).

Sendo o mais popular sistema de controle de versão para desenvolvimento de pacotes *R*, a utilização do *Git* com o site *GitHub* - a maior e mais avançada plataforma de desenvolvimento do mundo (GITHUB INC. b., 2021) - é de fácil configuração.

Neste trabalho, utilizou-se o *software RStudio* no sistema operacional *Windowns 10 home* e, para possibilitar a utilização do *Git* com *GitHub* nesse conjunto, é necessário seguir os seguintes passos (considerando que o usuário já tenha a linguagem *R* e o *software RStudio* instalados em sua máquina):

3.2.1 Criando uma Conta no *GitHub*

Para a criação de uma conta gratuita, necessitou-se de um endereço de e-mail válido e foi solicitada a escolha de um ID de usuário único.

3.2.2 Instalando o *Git*

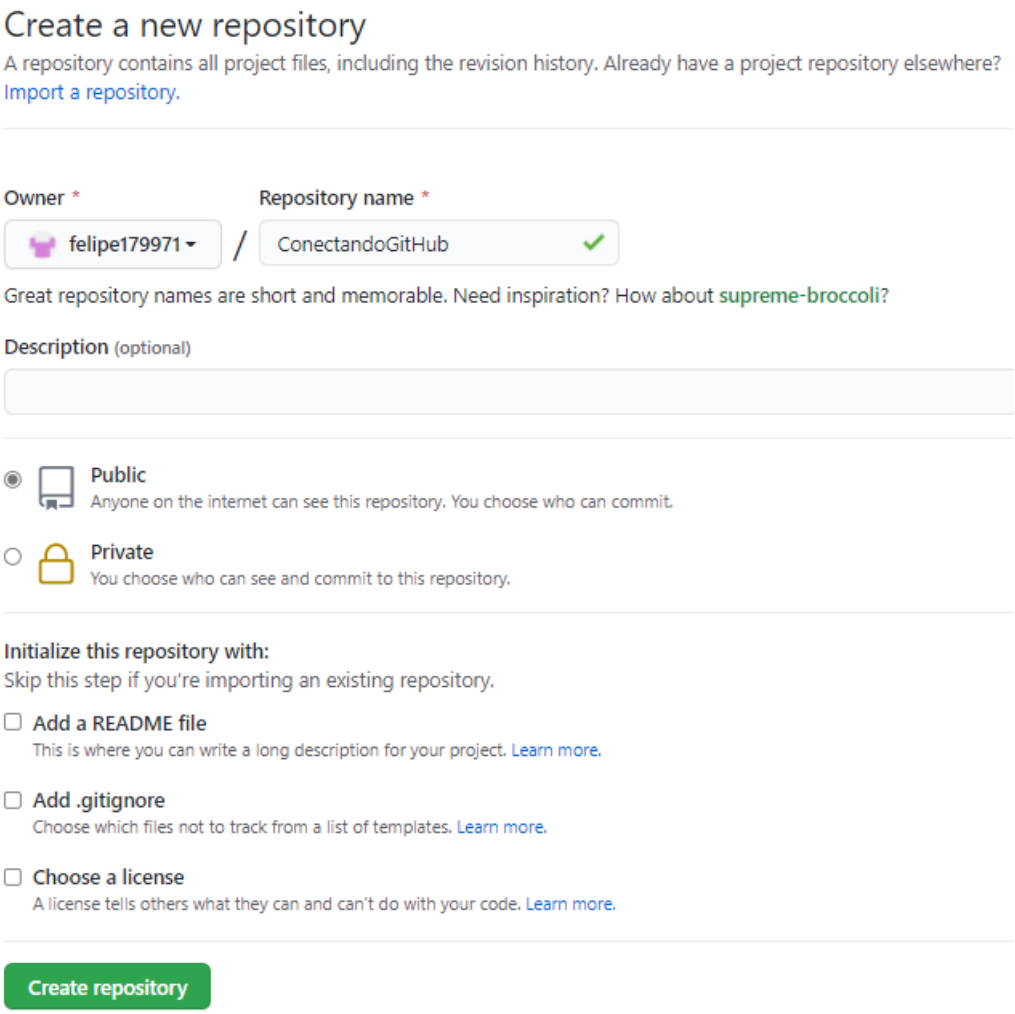
Através do site *Git - Downloading Package* <https://git-scm.com/> escolheu-se a versão do *Bit* compatível com a máquina para baixar o instalador: neste trabalho, o *Git* foi instalado por meio do instalador "64-bit Git for Windows Setup".

3.2.3 Configurando o *RStudio*

3.2.3.1 Entrando no *GitHub*

Nessa etapa, foi necessário entrar no site do *GitHub* para criar um novo repositório de nome único (Figura 7).

Figura 7: Criando repositório nomeado *ConectandoGitHub*.

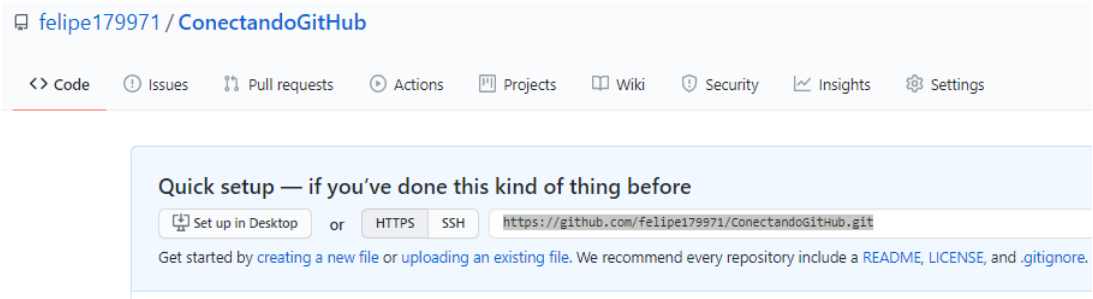


Fonte: Elaboração Própria.

Ao clicar em CREATE REPOSITORY, copiou-se o *link* do repositório gerado na

página seguinte (Figura 8).

Figura 8: Copiando *link* do repositório.

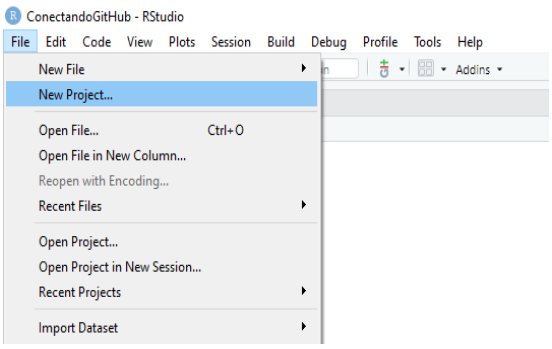


Fonte: Elaboração Própria.

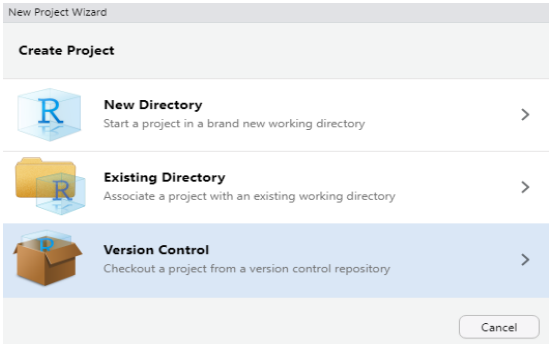
3.2.3.2 Abrindo o *RStudio*

Para criar um projeto na máquina local que estivesse conectado ao repositório, foi necessário seguir o caminho FILE → NEW PROJECT → VERSION CONTROL → GIT. Após chegar em *Clone Git Repository*, colou-se o *link* do repositório, selecionou-se o nome do diretório e escolheu-se um local, na máquina, para a clonagem do projeto (Figura 9).

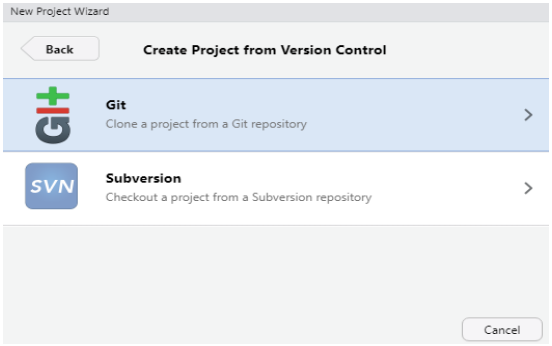
Figura 9: Iniciando um novo projeto conectado ao diretório.



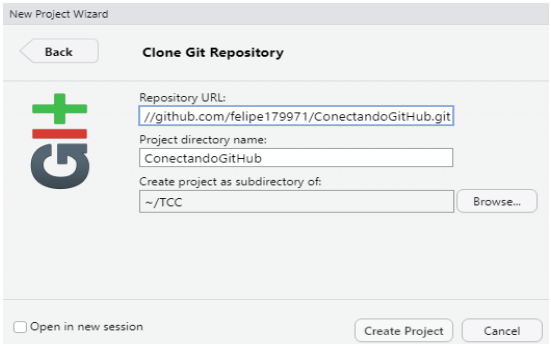
(a) New Project;



(b) Version Control;



(c) Git;



(d) Colando o *link* e escolhendo o local.

Fonte: Elaboração Própria.

O próximo passo foi identificar a máquina inserindo o seguinte código em `Terminal` (Figura 10):

Código 1: Identificando a máquina.

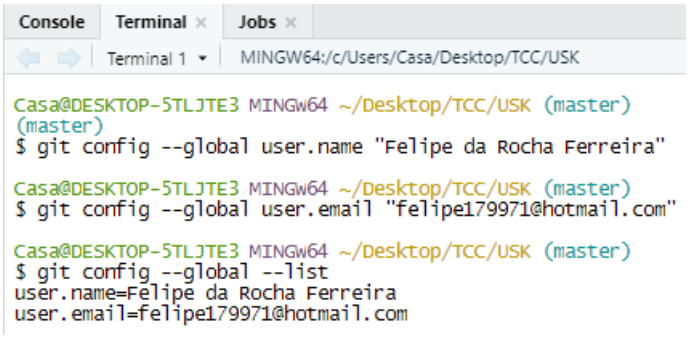
```
1 git config --global user.name "SEU NOME"
2 git config --global user.email "SEU E-MAIL"
```

Após, realizou-se a verificação das informações registradas, para que nenhum dado fosse inserido de forma incorreta:

Código 2: Verificando as informações.

```
1 git config --global list
```

Figura 10: Identificando a máquina.



Fonte: Elaboração Própria.

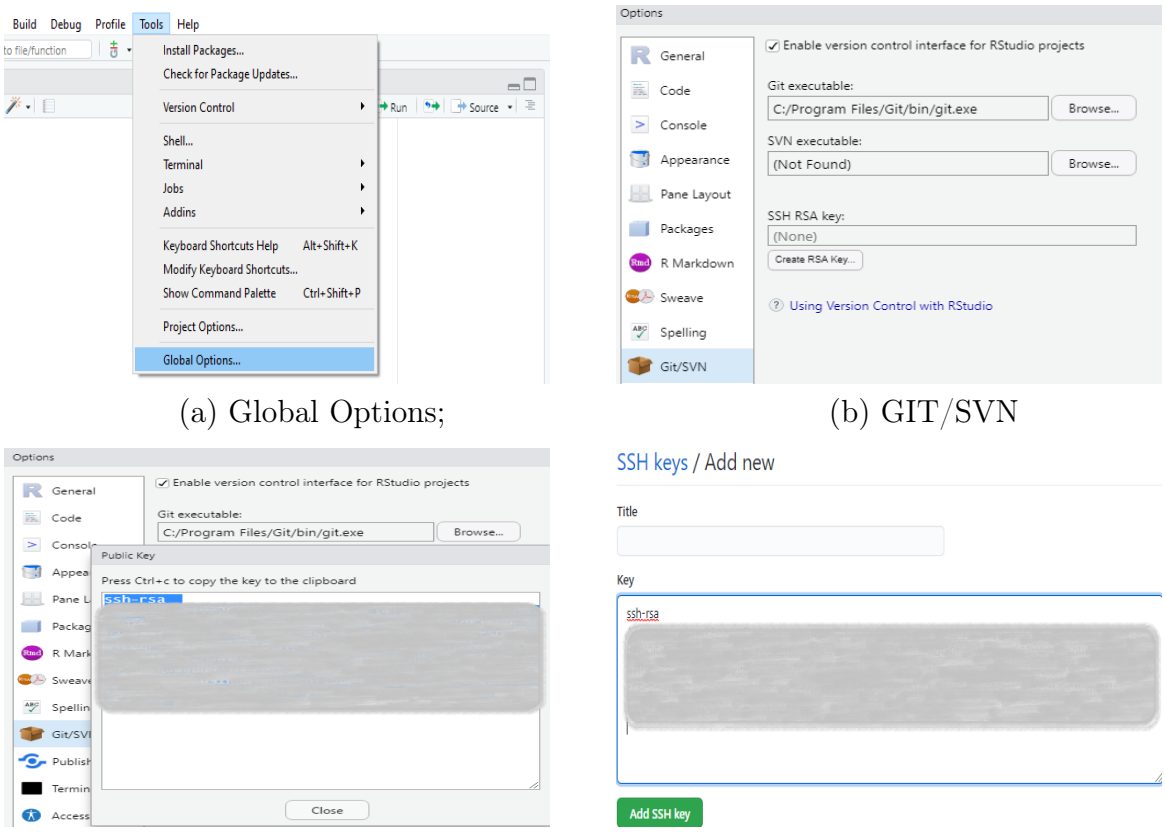
Neste trabalho utilizou-se a chave *SSH* para permitir que a máquina e o *GitHub* se comunicassem de forma segura, sem a necessidade de senha. Para verificar se o computador já possuia uma senha *SSH* executou-se, no `Console`, o Código 3.

Código 3: Verificando se já existe uma senha *SSH*.

```
1 file.exists("~/.ssh/id_rsa.pub")
```

Devido a inexistência da chave, foi preciso seguir o caminho TOOLS → GLOBAL OPTIONS → GIT/SVN → CREATE RSA KEY...; em seguida, visualizou-se a chave clicando em "View public key" para copia-la; por fim, acessou-se o site "*SSH and GPG keys*"<https://github.com/settings/keys> e a colou em "New SSD key" (Figura 11).

Figura 11: Iniciando um novo projeto conectado ao diretório.



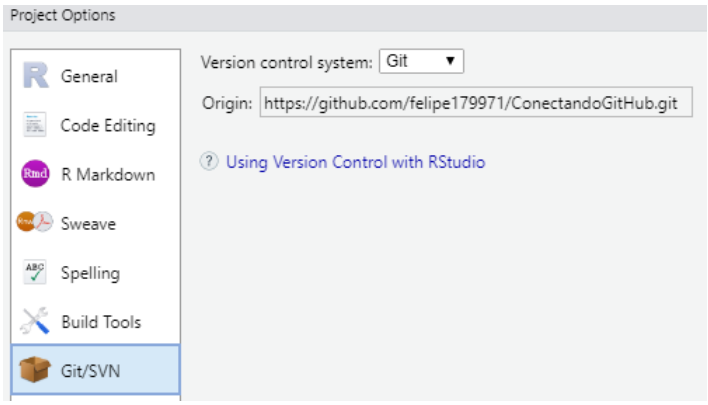
(a) Global Options; (b) GIT/SVN

(c) *Create RSA key...* e copie a chave; (d) Colando a chave.

Fonte: Elaboração Própria.

Com esses passos, o *Git* e o *RStudio* foram conectados, mas, para ter um *Repo* – diretório para armazenar arquivos de configuração e registrar as alterações no código - foi necessário clicar em TOOLS → PROJECT OPTIONS → GIT/SVN e alterar "version control system" para "Git" (Figura 12).

Figura 12: Criando um *Repo*.



Fonte: Elaboração Própria.

3.3 Implementação do Pacote R

Além dos arquivos e pastas obrigatórias para a composição e criação de um pacote R, pode-se elaborar complementares, todos com estruturas e regras rígidas cujo descumprimento inviabilizam um bom funcionamento, tornando a implementação custosa e cansativa.

Tendo isso em vista, Hadley Wickham, Jim Hester e Winston Chang com o apoio do *RStudio* e *R Core team* criaram o pacote `Devtools` (*Tools to Make Developing R Packages Easier* - Ferramentas para facilitar o desenvolvimento de pacotes R). Tal pacote cria toda a estrutura obrigatória e aplica as regras exigidas em cada um de seus componentes de forma automatizada, centralizando a energia do usuário unicamente no desenvolvimento de sua função.

3.3.1 Estrutura do Pacote

3.3.1.1 Componentes Obrigatórios

Todo pacote deverá ter, obrigatoriamente, os seguintes componentes:

1. **R**/: pasta que hospedará as funções;
2. **man**/: pasta que hospedará toda a documentação;
3. **DESCRIPTION**: arquivo contendo as informações sobre as funções do pacote;
4. **NAMESPACE**: arquivo informando as dependências do pacote e quais funções o usuário terá acesso.

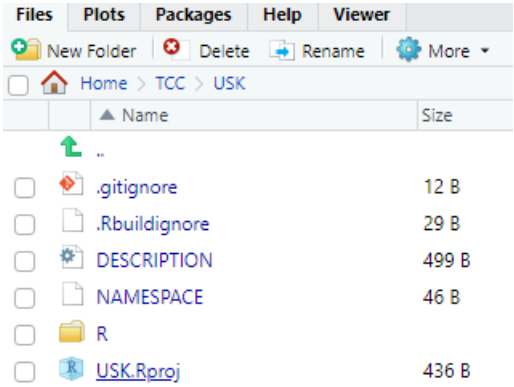
A maioria dos itens obrigatórios foram criados de forma automatizada ao se abrir uma seção no *RStudio* e digitar o seguinte código:

Código 4: Carregando o pacote `devtools` e criando o projeto *USK*.

```
#Carregando o pacote
library(devtools)
#Criando o projeto
create('USK')
```

Com este comando, o *RStudio* iniciou uma nova seção com quase toda a estrutura obrigatória, além dos arquivos *Git* (Figura 13). A criação da pasta **man** e os arquivos opicionais serão tratados nos próximos tópicos.

Figura 13: Alguns itens obrigatórios gerados com o *Código 4*.



Fonte: Elaboração Própria.

3.3.1.2 Componentes Opcionais

Mas, além desses, este pacote conta com os seguintes itens:

1. **README**: dois arquivos responsáveis pelo site;
2. **tests**: pasta que hospedará a estrutura dos testes unitários.

3.3.2 Configuração do *GitHub*

Para possibilitar que qualquer pessoa baixe e usufrua do pacote, criou-se um repositório no *GitHub* contendo todas as pastas deste projeto e um controle de versão *Git* para facilitar as alterações ao longo do desenvolvimento.

Após a execução de `usethis::use_git()`, seguiu-se os devidos passos verificadores (Figura 14) e a seção foi reiniciada.

Figura 14: Executando `usethis::use_git()` para a criação do Controle de Versão *Git*.

```
> usethis::use_git()
There are 5 uncommitted files:
* '.gitignore'
* '.Rbuildignore'
* 'DESCRIPTION'
* 'MeuPrimeiroPacoteR.Rproj'
* 'NAMESPACE'
Is it ok to commit them?

1: Absolutely
2: Negative
3: Nope

Selection: 1
√ Adding files
√ Making a commit with message 'Initial commit'
```

Fonte: Elaboração Própria

Para a criação do repositório, executou-se `usethis::use_github()` que gerou o repositório da *Figura 15*.

Figura 15: Executando `usethis::use_github()` para a criação do repositório *GitHub*.

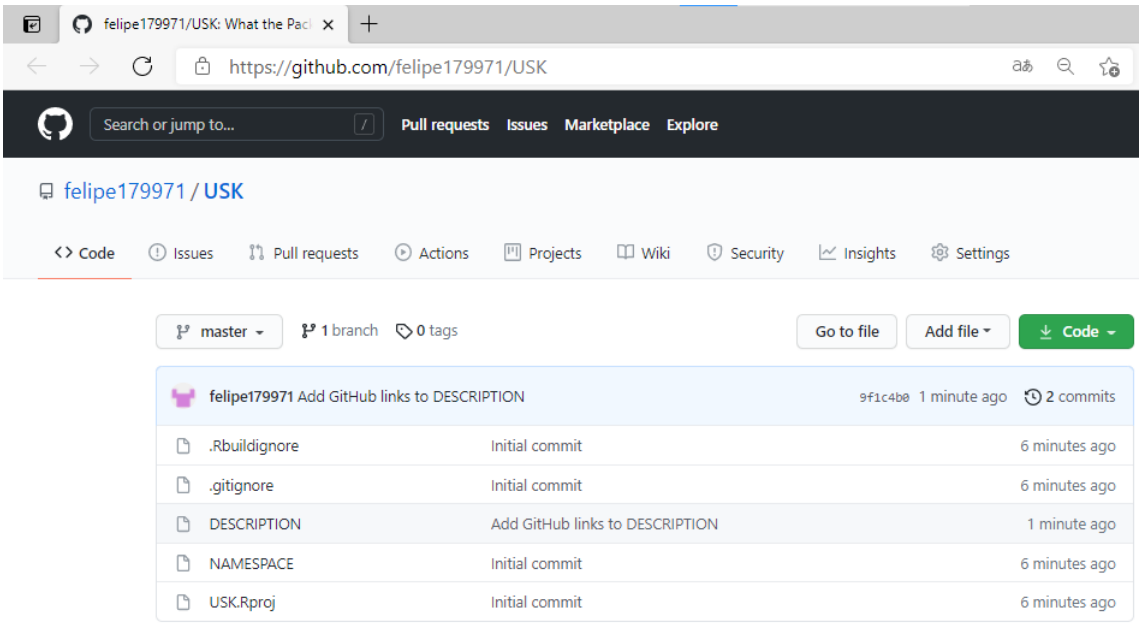
```
> usethis::use_github()
i Defaulting to 'https' Git protocol
√ Setting active project to 'C:/Users/felip/Documents/TCC/USK'
√ Checking that current branch is default branch ('master')
√ Creating GitHub repository 'felipe179971/USK'
√ Setting remote 'origin' to 'https://github.com/felipe179971/USK.git'
√ Setting URL field in DESCRIPTION to 'https://github.com/felipe179971/USK'
√ Setting BugReports field in DESCRIPTION to 'https://github.com/felipe179971/USK/issues'
There is 1 uncommitted file:
* 'DESCRIPTION'
Is it ok to commit it?

1: For sure
2: Nope
3: Negative

Selection: 1
√ Adding files
√ Making a commit with message 'Add GitHub links to DESCRIPTION'
√ Pushing 'master' branch to GitHub and setting 'origin/master' as upstream branch
√ Opening URL 'https://github.com/felipe179971/USK'
```

Fonte: Elaboração Própria

Figura 16: Repositório criado por meio da função `usethis::use_github()`.



Fonte: Elaboração Própria

3.3.3 Inserindo as Funções na pasta R

Neste trabalho optou-se por carregar todas as funções (Figura 17a) e posteriormente salva-las na pasta **R** utilizando `use_package()`, que foi inserida em um `for()`, no

Código 5. Com esse procedimento, todas as funções vistas em **3.1 Algoritmo** (Subseção 3.1) foram inseridas na pasta (Figura 17b), mas sem sua devida configuração.

Código 5: Criando os arquivos *.R* com as funções.

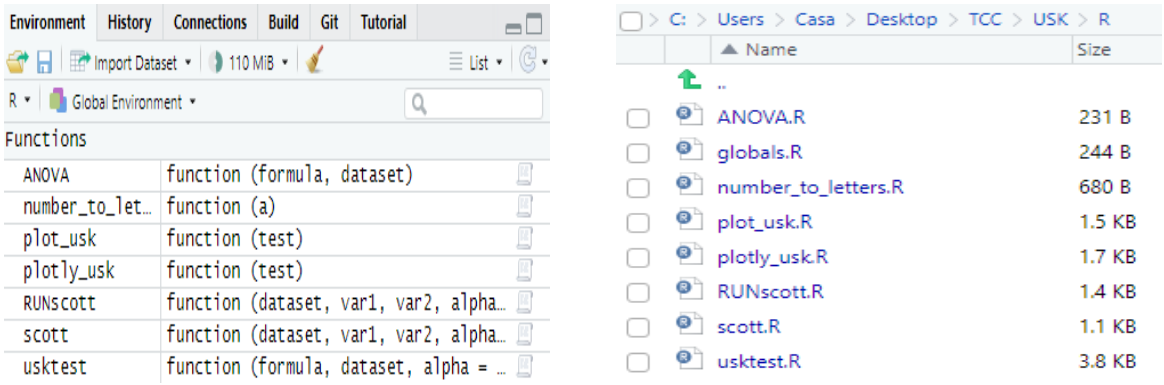
```
#Vetor contendo os nomes das funcoes
    x<-c("ANOVA","scott","number_to_letters","RUNscott","usktest",
         "plot_usk","plotly_usk")
#Adicionando a x_{i-esima} funcao ao projeto
    for(i in 1:length(x)){
        dump(x[i], file = paste0("~/TCC/USK/R/",x[i],".R"))
    }
```

Além desses arquivos **.R**, tornou-se necessária a adição de **globals.R** (Código 6) com intuito de definir algumas funções e variáveis.

Código 6: Criando o arquivo *globals.R*.

```
utils::globalVariables(c("aes","Treatment","Ordem","Mean","Group","geom_
    point","scale_y_continuous","geom_errorbar","theme_bw","labs","theme"
    ,"element_text","Treatment","y","Mean","dados"))
```

Figura 17: Funções carregadas e inseridas na pasta **R** por meio do *Código 5*.



(a) Carregando as funções; (b) Funções salvas na pasta.

Fonte: Elaboração Própria.

3.3.3.1 Explicando as Funções Implementadas

Como parte da documentação, todas as funções receberam comentários através das *tags* do pacote `roxygen2`, onde apenas as exportadas aos usuários foram totalmente detalhadas em língua inglesa.

As *tags* utilizadas foram:

1. **@title**: título da função;
2. **@name**: nome da função;

3. **@description**: descrição sobre o que a função faz;
4. **@param**: descrição dos parâmetros;
5. **@return**: descrição sobre a saída da função;
6. **@author**: autor(es) do pacote;
7. **@references**: referencial bibliográfico;
8. **@seealso**: funções ou pacotes indicados pelo autor acerca do assunto;
9. **@examples**: exemplo prático de uso da função;
10. **@import**: dependências (pacotes necessários para a execução da função);
11. **@importFrom**: para chamar apenas funções específicas de determinados pacotes;
12. **@encoding**: esquema de codificação;
13. **@export**: marcação para explicitar que a função estará disponível publicamente;

3.3.3.2 Testando as Funções Inseridas na Pasta R

Para testar se as funções foram inseridas e estavam rodando corretamente, utilizou-se o *Código 7* para roda-las sem que as mesmas estivessem carregadas no *Rstudio*. Ou seja, utilizando-as diretamente da pasta **R**, como pode ser visto na *Figura 18*.

Código 7: Executando as funções diretamente da pasta *R*.

```

#Carregando
    devtools::load_all()
#Criando a base
    taus=c(4,4,-4,-4,9,-9)
    Tratamento<-as.factor(rep(c(paste("trat",
                            seq(1:length(taus)))),3))
    erro<-rnorm(3*length(taus),0,1)
    y<-2+taus+erro
    y[round(runif(1,min=1,max=length(y)),0)]<-NA
    dados<-data.frame(y,Tratamento)
#Scott-Knott
    test<-usktest(y~Tratamento,dados)
#plotly
    plotly_usk(test)
#ggplot2
    plot_usk(test)
#Verificando que 'nao' 'estao' no 'diretorio'
    x<-c("usktest","plotly_usk","plot_usk")
```

```
20     for(i in 1:3){
21       print(exists(x[i],where=globalenv(),inherits=FALSE))
22     }
```

Figura 18: Carregando, usando e verificando, por meio do *Código 7*, que as funções não estão no diretório.

```
6 trat 6     d     -6.65 -6.92 -6.23
> #plotly
> plotly_usk(test)
> #ggplot2
> plot_usk(test)
>
> x<-c("usktest","plotly_usk","plot_usk")
> for (i in 1:3){
+  print(exists(x[i],where=globalenv(),inherits=FALSE))
+ }
[1] FALSE
[1] FALSE
[1] FALSE
```

Fonte: Elaboração Própria.

3.3.4 Descrição do Pacote Implementado

Um dos arquivos gerados pelo `devtools` foi o **DESCRIPTION**, de preenchimento bastante intuitivo: Deve-se descrever o objetivo do pacote, autores (Felipe Ferreira, como autor e criador - *aut* e *cre*; e Eduardo Gomes, como orientador de dissertação - *ths* [5]), e-mail para contato, etc.

A parte que requereu maior atenção foi na escolha da licença, um requisito fundamental ao se compartilhar o pacote. Dentre as disponíveis, utilizou-se a GPL-3, umas das mais permissivas licenças disponíveis (GITHUB INC. b., 2021).

Após o preenchimento das informações e escolha da licença, o arquivo ficou no formato do *Código 8*:

Código 8: PARTE I - Configuração do *DESCRIPTION*.

```
1
2 Package: USK
3 Title: Unbalanced Scott-Knott (USK)
4 version: 1.0.0.0
5 Authors@R:
6     c(person("Felipe","Ferreira",email="felipe179971@hotmail.com",role =
      c("aut", "cre")),
7     person("Eduardo ","Gomes",role = c("ths")))
8 Description: This package are used to do the Scott-Knott cluster
    analysis (1974) for unbalanced designs proposed at 2017 (CONRADO, T.
```

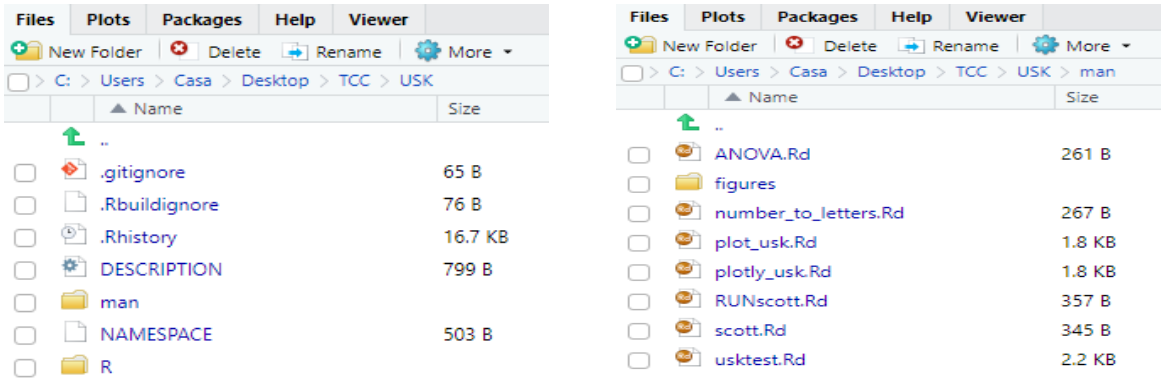
[5] Do Original, *Thesis advisor [ths]: A person under whose supervision a degree candidate develops and presents a thesis, mémoire, or text of a dissertation* (**Code List for Relators**, 2021).

```
    V; FERREIRA, D. F.; SCAPIM, C. A.; MALUF, W. R.).
License: GPL-3
Encoding: UTF-8
LazyData: true
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.1.2
URL: https://github.com/felipe179971/USK
BugReports: https://github.com/felipe179971/USK/issues
Imports:
    dplyr,
    ggplot2,
    pkgcond,
    plotly,
    purrr
Depends:
    R (>= 2.10)
Suggests:
    rmarkdown,
    knitr,
    testthat (>= 3.0.0)
Config/testthat/edition: 3
VignetteBuilder: knitr
```

3.3.5 Documentação do Pacote Implementado

Nesta etapa criou-se a pasta **man** (Figura 19), que é obrigatória para a implementação de um pacote *R*, contendo arquivos **Rd** (todos no formato da *Figura 20a*). Para isto, utilizou-se a função `devtools::document()`, que também configurou o **NAMESPACE** (Figura 20b) com as informações sobre as funções que deveriam ser exportadas e importadas (dependências).

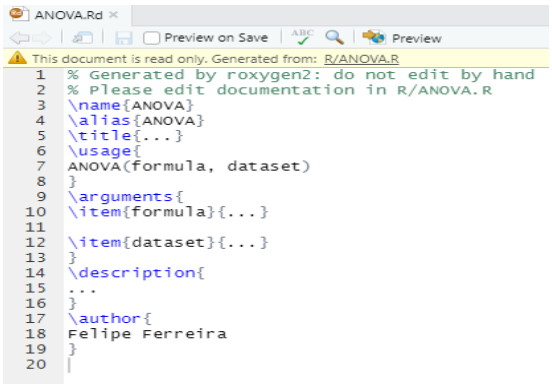
Figura 19: Pasta **man** gerada por meio da função `devtools::document()`.



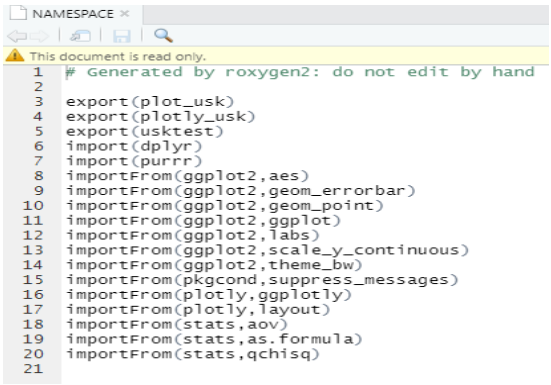
(a) Pasta *man* criada; (b) Arquivos *.RD* criados.

Fonte: Elaboração Própria.

Figura 20: Configuração dos arquivos gerados por `devtools::document()`.



(a) *ANOVA.Rd*;



(b) *NAMESPACE*

Fonte: Elaboração Própria.

3.3.5.1 Checando a Documentação

Para checar a documentação, utilizou-se da função `devtools::check_man()` para verificar a ausência de erros (Figura 21). Além desta verificação, pode-se visualizar a documentação por meio da função `help(...)`.

Figura 21: Checando a documentação por meio da função `devtools::check_man()`.

```
> devtools::check_man()
i Updating USK documentation
i Loading USK
Writing NAMESPACE
Writing NAMESPACE
i Checking documentation...
v No issues detected
```

Fonte: Elaboração Própria.

3.3.6 Testes Unitários

Diversas modificações foram realizadas ao longo da construção deste pacote e, além do controle de versão *Git*, que auxiliou na visualização de possíveis mudanças equivocadas, tornou-se de suma importância verificar, constantemente, se as mudanças não geraram erros ou *outputs* diferentes do esperado.

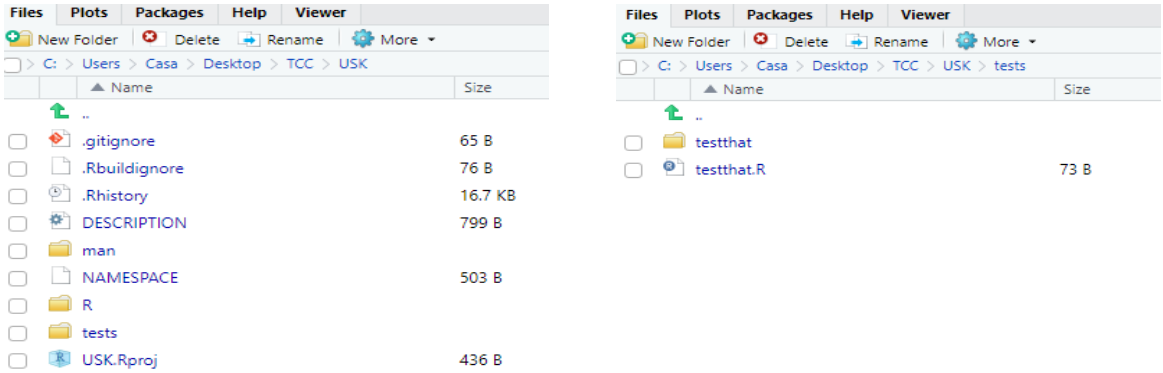
Para isso, utilizou-se da função `usethis::use_testthat()` (que cria a pasta **tests** contendo a pasta **testthat** e um arquivo **.R** - Figura 22), para a realização de testes unitários. Dentre as diversas funções disponíveis, utilizou-se apenas duas:

1. `expect_equal(...)`: A execução deve retornar um valor específico (com ou sem

margem de erro);

2. `expect_error(...)`: a execução deve retornar a mensagem de erro especificada.

Figura 22: Criação da pasta *testthat* por meio da função `usethis::use_testthat()`.



(a) Pasta *tests*; (b) Arquivos da pasta *tests*

Fonte: Elaboração Própria.

Ao todo, foram criadas 36 funções de checagem. Todas inseridas no arquivo **test.R** na pasta **testthat**. A execução foi feita através de `devtools::test()`, retornando a *Figura 23*, indicando que as modificações não mudaram os valores esperados.

Figura 23: Resultado da execução de `devtools::test()`.

```
√ |        36 | test [12.0s]

== Results =============================
Duration: 12.0 s

[ FAIL 0 | WARN 0 | SKIP 0 | PASS 36 ]
```

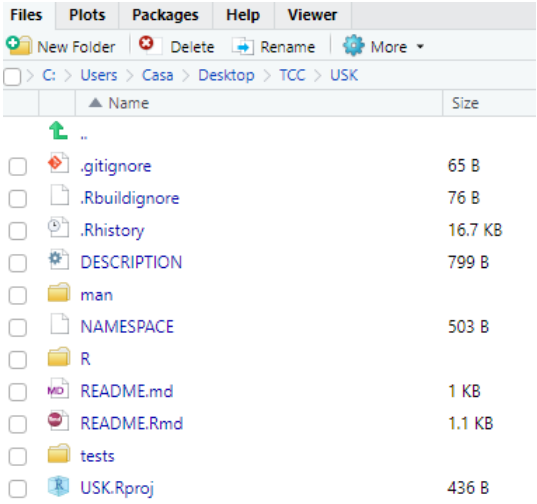
Fonte: Elaboração Própria.

3.3.7 Criando o Site Repositório do Pacote

Além de servir como repositório, o *GitHub* permite que o desenvolvedor disponibilize um site personalizado ao usuário final. Ou seja, permite que a página contenha todas as pastas, códigos e uma documentação feita via **Rmarkdown**.

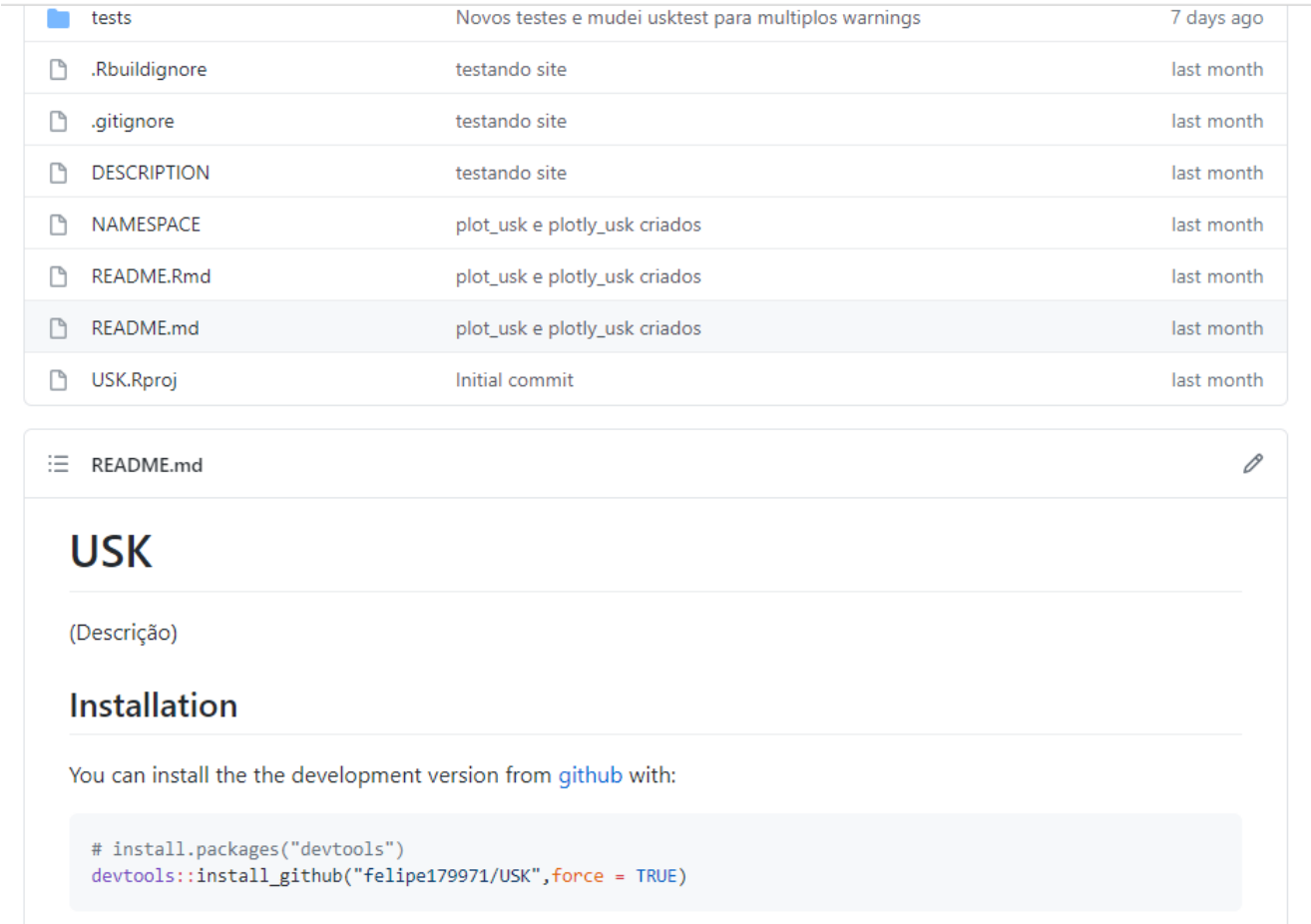
Para criar os arquivos necessários, executou-se a função `usethis::use_readme_rmd()`, criando os arquivos **README.md** e **README.Rmd** (Figura 24), onde editou-se, seguindo os padrões do **Rmarkdown**, a página *GitHub* (Figura 25).

Figura 24: Criação dos arquivos *README.md* e *README.Rmd* através da função `usethis::use_readme_rmd()`.



Fonte: Elaboração Própria.

Figura 25: Site *GitHub* com os arquivos e documentação criada através da função `usethis::use_readme_rmd()`.



Fonte: Elaboração Própria.

3.3.8 Chegando se o Pacote Satisfaz os Requisitos para Submissão `CRAN`

Para simular o processo de construção, anexação e instalação do pacote, utilizou-se da função `devtools::check()`. A saída é extensa e deve-se ficar muito atento as mensagens de erros ou avisos, que informam desde que o horário do *Windows* está dessincronizado, a não instalação de algum pacote necessário, como o `Rtools` ou a existência de erros estruturais.

Apesar de as notas (*notes*) não inviabilizarem a criação do pacote via *GitHub*, optou-se por resolver todos os *warnings*, *errors* e *notes*, indicando o cumprimento de vários requisitos para submissão `CRAN`.

Figura 26: Verificação feita através da função `devtools::check()`.

```
-- R CMD check results ------------------------------------------------- USK 1.0.0.0 ----
Duration: 1m 29.4s

0 errors √ | 0 warnings √ | 0 notes √
> devtools::load_all()
i Loading USK
> devtools::document()
i Updating USK documentation
i Loading USK
Writing NAMESPACE
Writing NAMESPACE
```

Fonte: Elaboração Própria.

3.3.9 Disponibilizando o Pacote ao Público via *GitHub*

Para disponibilizar a implementação, foram utilizados 3 comandos em `Terminal` (Código 9), mas antes, devido à criação da página personalizada via **Rmarkdown**, a cada atualização do *GitHub*, fora necessário abrir o arquivo **README.Rmd** para tricota-lo (*Knit*).

Código 9: Disponibilizando a implementação para download via *GitHub*.

```
1 git add .
2 git commit -m "comentario"
3 git push -u origin master
```

Com a página gerada após executar *Knit* aberta e executando o *Código 9*, (Figura 27) disponibilizou-se o pacote para *download*.

Figura 27: Executando o *Código 9* para disponibilizar o pacote para *download*.

```
Felipe@DESKTOP-5TLJTE3 MINGW64 ~/Desktop/TCC/USK (master)
$ git add .
warning: LF will be replaced by CRLF in NAMESPACE.
The file will have its original line endings in your working directory

Felipe@DESKTOP-5TLJTE3 MINGW64 ~/Desktop/TCC/USK (master)
$ git commit -m "27.06.2021"
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

Felipe@DESKTOP-5TLJTE3 MINGW64 ~/Desktop/TCC/USK (master)
$ git push -u origin master
Enumerating objects: 25, done.
Counting objects: 100% (25/25), done.
Delta compression using up to 8 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (13/13), 3.92 KiB | 1.96 MiB/s, done.
Total 13 (delta 11), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (11/11), completed with 6 local objects.
To https://github.com/felipe179971/USK.git
   e78c132..4329ff3  master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

Fonte: Elaboração Própria.

3.4 Simulação de Monte Carlo

Ao definir os efeitos dos tratamentos do modelo representado na Equação 2.1.1, buscou-se criar determinado número de grupos distintos η para cada um dos cenários: para verificar o Erro Tipo I, $\eta = 1$ ($\tau_1 = \tau_2 = ... = \tau_k$); e para verificar o Poder do Teste, $\eta = k$ ($\tau_1 \neq \tau_2 \neq ... \neq \tau_k$).

Dessa forma, sendo X_i o número de grupos da i-ésimo iteração, ou seja, o número de grupos resultantes da execução de `uskteste(...)` para o i-ésimo banco de dados, tem-se que,

$$\begin{cases} X_i = 1, \text{ se a } i - \text{ésima iteração for classificada com } \eta \text{ grupos;} \\ X_i = 0, \text{ se a } i - \text{ésima iteração não for classificada com } \eta \text{ grupos.} \end{cases} \tag{3.4.1}$$

Ou seja,

$$a_M = \frac{1}{M} \sum_{i=1}^{M} X_i = \text{proporção de acertos.} \tag{3.4.2}$$

Logo, a_M, o estimador de a, representa nosso resultado empírico do Erro Tipo I ou do Poder do Teste, a depender do cenário.

3.4.1 Determinando o Número de Iterações

Ao se executar os passos descritos por Paula (2014) para a obtenção, com 95% de confiança, de uma diferença entre a_M e a de no máximo 0,05, obteu-se $d = 0,0255102$.

Diferentemente do proposto, optou-se por iniciar com 1.000 iterações, para evitar uma sobrecarga computacional ao se refazer as Simulações de Monte Carlo até que se encontrasse um número que satisfizesse as condições propostas.

Após a execução, observou-se que todos os cenários obtiveram $\frac{S}{\sqrt{M}} < d$ (para mais detalhes, consultar **Apêncide C - $\frac{S}{\sqrt{M}}$ para as mil Iterações Realizadas nos Diferentes Cenários para verificar o Erro Tipo I**: Subseção 6.3 ; e **Apêncide D - $\frac{S}{\sqrt{M}}$ para as mil Iterações Realizadas nos Diferentes Cenários para verificar o Poder do Teste**: Subseção 6.4). Concluindo-se, assim, que esse número de iteração satifaz as condições e entrega, com 95% de confinaça, um resultado com diferença máxima de 0,05 entre o estimado e seu estimador.

3.4.2 Implementação Computacional para Realização das Simulações

Com o pacote `USK` completamente implementado, criou-se diversas funções que viabilizassem a execução da verificação do Erro Tipo I e do Poder do Teste de forma empírica, que podem ser consultadas na página: <https://github.com/felipe179971/USK_Simulation/blob/master/simulation/simulation.R>.

Devido a utilização dessa implementação na verificação e apresentação dos resultados de `uskteste(...)`, `plotly_usk(...)` e `plot_usk(...)`, mais detalhes acerca da metodologia e principais pontos que interferiram na construção dos cenários serão tratados em **4 Resultados** (Seção 4).

4 Resultados e Discussões

Dentre os diversos métodos utilizados para validação e apresentação dos resultados de uma implementação computacional, como por exemplo a aplicação em situações práticas ou teóricas, optou-se por fazer uma verificação empírica de todos os parâmetros de `uskteste(...)`, ao mesmo tempo em que se valida as funções desenvolvidas para as Simulações de Monte Carlo e as saídas gráficas `plotly_usk(...)` e `plot_usk(...)`.

4.1 Análise Empírica da `uskteste(...)` e da Implementação Computacional Destinada às Simulações de Monte Carlo

Apesar do intuito principal consistir na execução de Simulações Monte Carlo, as funções também foram úteis e testadas visualmente para verificar se as funções do pacote `USK` e as criadas para as simulações se comportavam de forma esperada em diferentes aspectos.

4.1.1 Analisando a Construção de Banco de Dados com Porcentagem de Dados Faltantes

Para calcular a porcentagem de valores perdidos, utilizou-se da função CEILING(...), que utiliza o maior número inteiro seguinte, por exemplo, `ceiling(2.1)` a `ceiling(2.9)` retornam o valor 3. Optou-se por essa função para que, independentemente do número de tratamentos, 1% de dados faltantes retorne, no mínimo, um tratamento perdido. Além disso, a escolha das parcelas a serem perdidas foi realizada de forma totalmente aleatória, mesmo que implicasse em possíveis perdas em um único tratamento, mas com uma única restrição: nenhum tratamento poderia ficar com todas as suas observações ausentes.

Os bancos foram construídos considerando o modelo representado na Equação 2.1.1, inicialmente com 2 colunas (*Treatment*, contendo a indicação do tratamento ao qual a observação y_{ij} pertence; e *Original*, com os valores y_{ij} gerados sem nenhuma alteração), posteriormente criou-se a coluna *Missing*, compondo-se por *Original* após a aplicação da perda dos dados e, por fim, a coluna *Imputed* que utiliza *Missing* para fazer a imputação dos dados faltantes.

Exemplificando, criou-se 2 bancos de dados pequenos com porcentagem de dados faltantes de 5 e 30 porcento (ver Tabela 5)[6].

[6]Para replicação, consultar <https://github.com/felipe179971/USK_Simulation/blob/master/simulation/simulation.R>, linha 783 (*Banco de Dados e Porcentagem de Dados Faltantes:*).

Tabela 5: Criando dois bancos de dados idênticos, com alteração, apenas, nas porcentagens de dados perdidos.

Treatment	**Porcentagem de Dados Perdidos**					
	5%			**30%**		
	Original	**Missing**	**Imputed**	**Original**	**Missing**	**Imputed**
1	15,813426	15,813426	15,81343	15,81343	15,81343	15,81343
2	27,070378	27,070378	27,07038	27,07038	27,07038	27,07038
1	18,927325	18,927325	18,92733	18,92733	NA	9,753839
2	23,337031	23,337031	23,33703	23,33703	23,33703	23,33703
1	7,767166	7,767166	7,767166	7,767166	7,767166	7,767166
2	32,848031	NA	36,18911	32,84803	NA	24,0515
1	-4,448588	-4,448588	-4,44859	-4,44859	NA	6,857693
2	25,654908	25,654908	25,65491	25,65491	25,65491	25,65491

Fonte: Elaboração própria.

Importante observar que, dependendo do número de observações e de tratamentos, pode-se incorrer em números iguais de dados faltantes. Além disso, nas Simulações de Monte Carlo, devido à utilização do `set.seed(...)`, a cada iteração, o `seed` foi alterado de forma a garantir a replicabilidade da simulação sem que se gerasse bancos de dados iguais.

4.1.2 Verificando `plotly_usk(...)`, `plot_usk(...)` e o parâmetro `alpha` da `usktest(...)`

Lembrando que α representa o Erro Tipo I teórico ao qual o pesquisador está disposto a aceitar, esse parâmetro da função dimensiona a sensibilidade do teste: um α muito grande representa que o pesquisador quer que o teste identifique e separe pequenas variações em grupos distintos, mesmo que o H_0 ($\tau_1 = \tau_2 = ... = \tau_k$) seja verdadeiro.

Para verificar o parâmetro `alpha` da `usktest(...)` por meio desse comportamento, executou-se o teste para um banco de dados (Tabela 6)[7] com $y_{ij} \sim N(\mu + \tau_i = 1 + \tau_i;\ \sigma^2 = 0,5)$, 4 observações em cada tratamento, 1 dado perdido no tratamento 2 e com efeitos significativos, teoricamente, formando dois grupos: o primeiro composto pelos $\tau_1 = 1$ e $\tau_2 = 1$ e o segundo pelos $\tau_3 = -1$ e $\tau_4 = -1$.

[7]Para replicar: consultar <https://github.com/felipe179971/USK_Simulation/blob/master/simulation/simulation.R>, linha 795 (*Gráficos e Alpha:*).

Tabela 6: Banco de dados criado para verificação dos gráficos e do α.

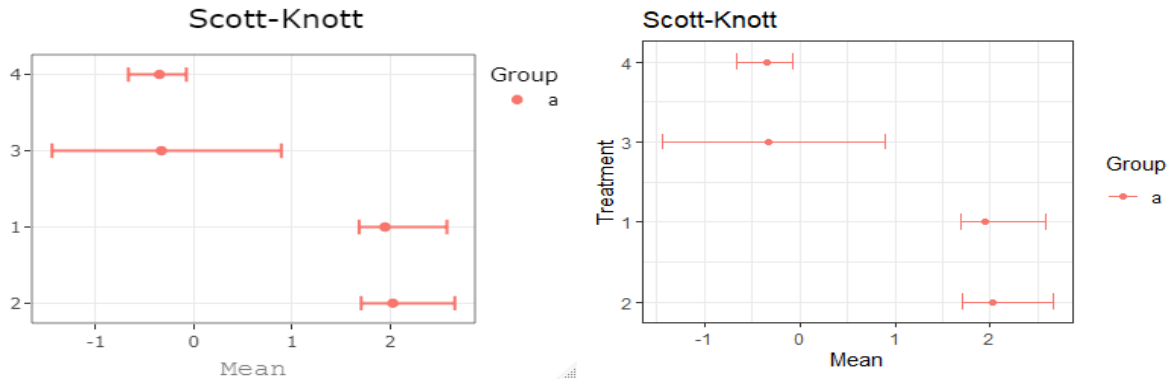
Observação	**Tratamento**			
	1	**2**	**3**	**4**
1	2,581343	1,707038	0,892733	-0,66629686
2	1,776717	NA	-1,44486	-0,43450917
3	1,769149	1,722230	-0,01007	-0,07519111
4	1,685937	2,661610	-0,76068	-0,21871393

Fonte: Elaboração própria.

Com o objetivo de verificar a visualização gráfica, os testes foram rodados nas duas funções: `plotly_usk(...)` e `plot_usk(...)`.

Conforme observado na Figura 28 [8], para um $\alpha = 0$, mesmo os tratamentos sendo distintos, o algoritmo classificou todos os dados em um grupo só, para não correr o risco de cometer o Erro Tipo I.

Figura 28: Execução de `usktest(...)`, com $\alpha = 0$, para realizar o teste de Scott-Knott ajustado para dados desbalanceados com base na Tabela 6 .



(a) `plotly_usk(...)`; (b) `plot_usk(...)`.

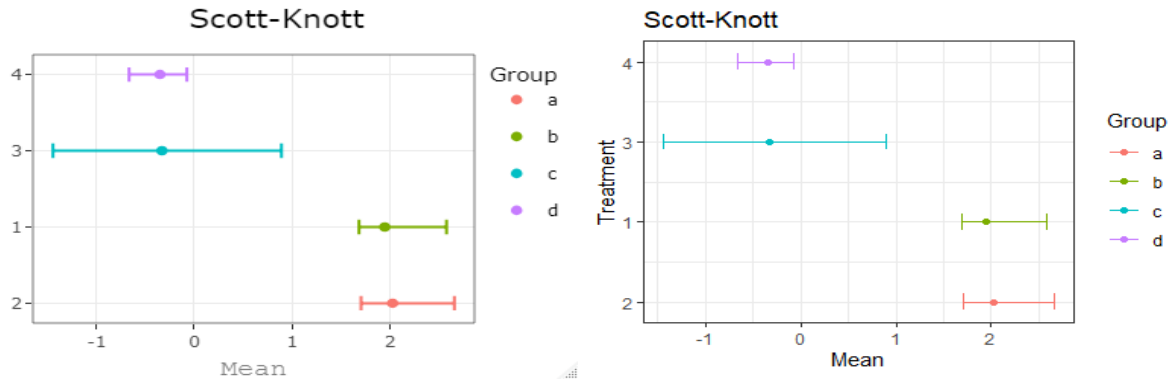
Fonte: Elaboração Própria.

Já na Figura 29 [9], com um $\alpha = 1$, todos foram separados em grupos distintos, obedecendo ao comando do pesquisador em aceitar, totalmente, a rejeição de H_0 quando H_0 verdadeiro.

[8]Para replicar: consultar <https://github.com/felipe179971/USK_Simulation/blob/master/simulation/simulation.R>, linhas 809 e 810.

[9]Para replicar: consultar <https://github.com/felipe179971/USK_Simulation/blob/master/simulation/simulation.R>, linhas 812 e 813.

Figura 29: Execução de `usktest(...)`, com $\alpha = 1$, para realizar o teste de Scott-Knott ajustado para dados desbalanceados com base na Tabela 6.

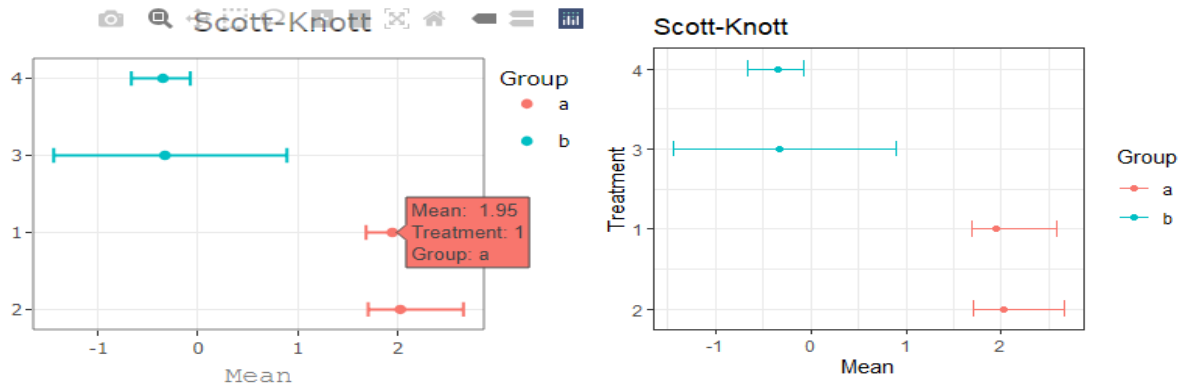


(a) `plotly_usk(...)`; (b) `plot_usk(...)`.

Fonte: Elaboração Própria.

Levando em consideração um $\alpha = 0,05$, muito utilizado pelos pesquisadores, o teste se comportou conforme o esperado, classificando os tratamentos de forma bastante intuitiva, como pode ser observado na Figura 30 [10]. No gráfico da Figura 30a, optou-se por visualizar detalhes acerca do tratamento 1 ao passar o mouse sobre seu respectivo ponto no gráfico, utilizando e testando, dessa forma, umas das funcionalidades *plotly*.

Figura 30: Execução de `usktest(...)`, com $\alpha = 0,05$, para realizar o teste de Scott-Knott ajustado para dados desbalanceados com base na Tabela 6.



(a) `plotly_usk(...)`; (b) `plot_usk(...)`.

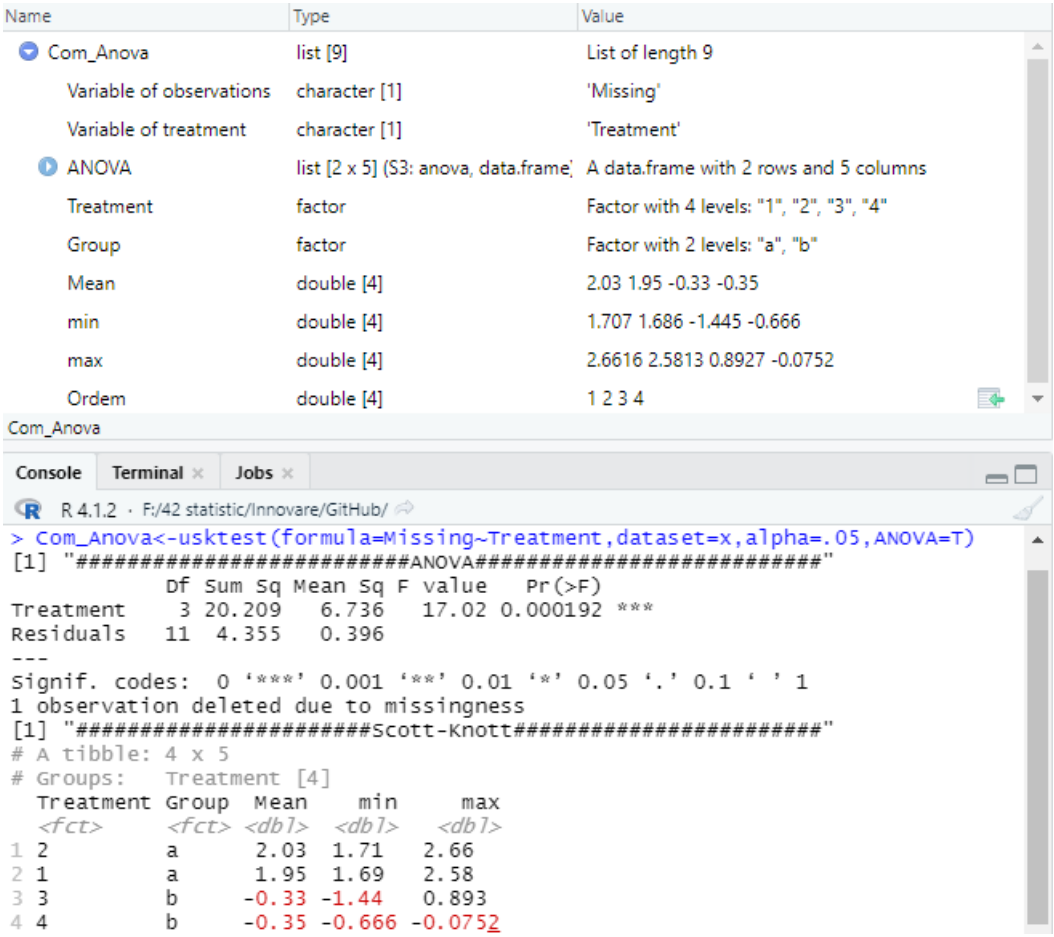
Fonte: Elaboração Própria.

[10] Para replicar: consultar <https://github.com/felipe179971/USK_Simulation/blob/master/simulation/simulation.R>, linhas 815 e 816.

4.1.3 Verificando o Parâmetro `ANOVA` da `usktest(...)` e a saída no `Console`

Utilizando a Tabela 6, verificou-se, também, o parâmetro ANOVA da função `usktest(...)` e as saídas no `Console` do *RStudio* [11]. Na Figura 31 vemos que, com `ANOVA=T`, é retornado uma lista de 9 e, caso contrário, 8 elementos (Figura 32). Observe que, independentemente de o usuário pedir ou não, o resultado é exibido no `Console` e seu armazenamento é opcional e no formato de lista.

Figura 31: Execução de `usktest(...)`, com `ANOVA=T`, para realizar o teste de Scott-Knott ajustado para dados desbalanceados com base na Tabela 6.



Name	Type	Value
Com_Anova	list [9]	List of length 9
Variable of observations	character [1]	'Missing'
Variable of treatment	character [1]	'Treatment'
ANOVA	list [2 x 5] (S3: anova, data.frame)	A data.frame with 2 rows and 5 columns
Treatment	factor	Factor with 4 levels: "1", "2", "3", "4"
Group	factor	Factor with 2 levels: "a", "b"
Mean	double [4]	2.03 1.95 -0.33 -0.35
min	double [4]	1.707 1.686 -1.445 -0.666
max	double [4]	2.6616 2.5813 0.8927 -0.0752
Ordem	double [4]	1 2 3 4

Com_Anova

Console Terminal Jobs

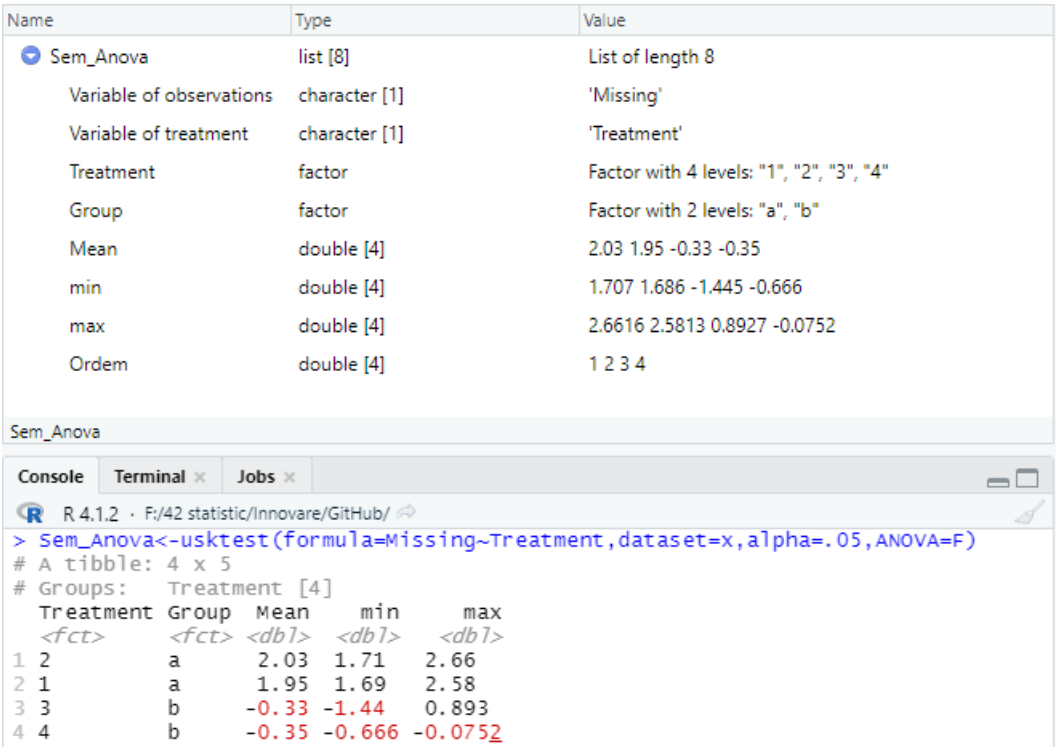
R 4.1.2 · F:/42 statistic/Innovare/GitHub/

```
> Com_Anova<-usktest(formula=Missing~Treatment,dataset=x,alpha=.05,ANOVA=T)
[1] "##########################ANOVA##########################"
            Df Sum Sq Mean Sq F value   Pr(>F)
Treatment    3 20.209   6.736   17.02 0.000192 ***
Residuals   11  4.355   0.396
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
1 observation deleted due to missingness
[1] "#######################Scott-Knott#######################"
# A tibble: 4 x 5
# Groups:   Treatment [4]
  Treatment Group  Mean    min     max
  <fct>     <fct> <dbl>  <dbl>   <dbl>
1 2         a      2.03  1.71   2.66
2 1         a      1.95  1.69   2.58
3 3         b     -0.33 -1.44   0.893
4 4         b     -0.35 -0.666 -0.0752
```

Fonte: Elaboração Própria.

[11] Para replicar: consultar <https://github.com/felipe179971/USK_Simulation/blob/master/simulation/simulation.R>, linha 817 (*ANOVA e saída no Console:*).

Figura 32: Execução de `usktest(...)`, com `ANOVA=F`, para realizar o teste de Scott-Knott ajustado para dados desbalanceados com base na Tabela 6.



Name	Type	Value
Sem_Anova	list [8]	List of length 8
Variable of observations	character [1]	'Missing'
Variable of treatment	character [1]	'Treatment'
Treatment	factor	Factor with 4 levels: "1", "2", "3", "4"
Group	factor	Factor with 2 levels: "a", "b"
Mean	double [4]	2.03 1.95 -0.33 -0.35
min	double [4]	1.707 1.686 -1.445 -0.666
max	double [4]	2.6616 2.5813 0.8927 -0.0752
Ordem	double [4]	1 2 3 4

Sem_Anova

Console Terminal Jobs

R 4.1.2 · F:/42 statistic/Innovare/GitHub/

```
> Sem_Anova<-usktest(formula=Missing~Treatment,dataset=x,alpha=.05,ANOVA=F)
# A tibble: 4 x 5
# Groups:   Treatment [4]
  Treatment Group  Mean    min     max
  <fct>     <fct> <dbl>  <dbl>   <dbl>
1 2         a      2.03  1.71   2.66
2 1         a      1.95  1.69   2.58
3 3         b     -0.33 -1.44   0.893
4 4         b     -0.35 -0.666 -0.0752
```

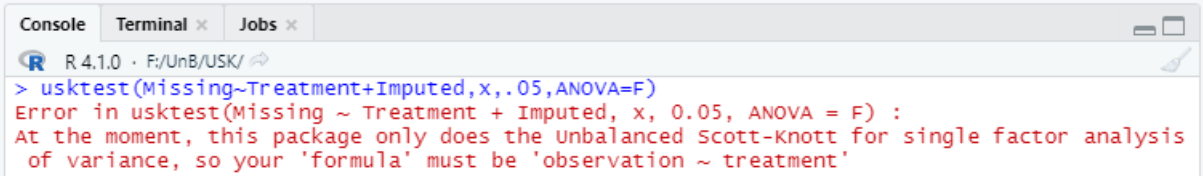
Fonte: Elaboração Própria.

4.1.4 Verificando as Mensagens de Erro da `usktest(...)`

Sendo uma etapa importante para auxiliar o pesquisador acerca de possíveis equívocos e orienta-lo à execução adequada, foram construídas 5 mensagens de erros que serão visualmente verificadas [12].

Lembrando que a função trabalha com a sintaxe *'observação ∼ tratamento'*, realizou-se o teste da seção anterior com uma alteração no parâmetro `formula`, retornando o erro esperado (ver Figura 36).

Figura 33: Execução de `usktest(...)`, com `formula='Missing~Treatment+Imputed'`.



Console Terminal Jobs

R 4.1.0 · F:/UnB/USK/

```
> usktest(Missing~Treatment+Imputed,x,.05,ANOVA=F)
Error in usktest(Missing ~ Treatment + Imputed, x, 0.05, ANOVA = F) :
At the moment, this package only does the Unbalanced Scott-Knott for single factor analysis
 of variance, so your 'formula' must be 'observation ~ treatment'
```

Fonte: Elaboração Própria.

[12] Para replicar: consultar <https://github.com/felipe179971/USK_Simulation/blob/master/simulation/simulation.R>, linha 820 (*Mensagens de Erro*).

Para verificar os erros nos casos em que as variáveis estejam na classe errada, criou-se 2 novas variáveis com as seguintes classes: *Treatment_ Character* que é a variável *Treatment* transformada em caracteres; e *Missing_factor*, sendo *Missing* transformada em fator.

Observe na Figura 37 que, com a presença exclusiva do erro representado pela variável *Treatment_ Character* (variável responsável por indicar os tratamentos não ser um fator), a função executa o teste avisando ao pesquisador que efetuou a devida conversão da classe.

Figura 34: Execução de `usktest(...)`, utilizando variáveis com classes não compatíveis.

```
Console   Terminal ×   Jobs ×
R 4.1.2 · F:/TCC/USK_Simulation/
> usktest(Missing_factor~Treatment_Character,x,.05,ANOVA=F)
Error in usktest(Missing_factor ~ Treatment_Character, x, 0.05, ANOVA = F) :

The variable 'Missing_factor' must be numeric
In addition: Warning message:
In usktest(Missing_factor ~ Treatment_Character, x, 0.05, ANOVA = F) :
  The variable 'Treatment_Character' has been changed to format 'factor'
> usktest(Missing~Treatment_Character,x,.05,ANOVA=F)
# A tibble: 4 x 5
# Groups:   Treatment_Character [4]
  Treatment_Character Group  Mean    min     max
  <fct>               <fct> <dbl>  <dbl>   <dbl>
1 2                   a      2.03  1.71   2.66
2 1                   a      1.95  1.69   2.58
3 3                   b     -0.33 -1.44   0.893
4 4                   b     -0.35 -0.666 -0.0752
Warning message:
In usktest(Missing ~ Treatment_Character, x, 0.05, ANOVA = F) :
  The variable 'Treatment_Character' has been changed to format 'factor'
```

Fonte: Elaboração Própria.

Por fim, analisou-se a composição das variáveis com a construção da variável *y_ 1Missing_ NA*, representando um experimento contendo um dos tratamentos composto, completamente, por dados faltantes (todas as observações do *Tratamento 1* da Tabela 6 foram atribuídas com *NA*) e *Treatment_ unico*, para um experimento contendo apenas um tratamento (Todos os tratamentos da Tabela 6 foram classificados como *Tratamento 1*) (ver Figura 38).

Figura 35: Execução de `usktest(...)` para testar apenas um tratamento ou experimento contendo um dos tratamentos composto apenas por *NA*.

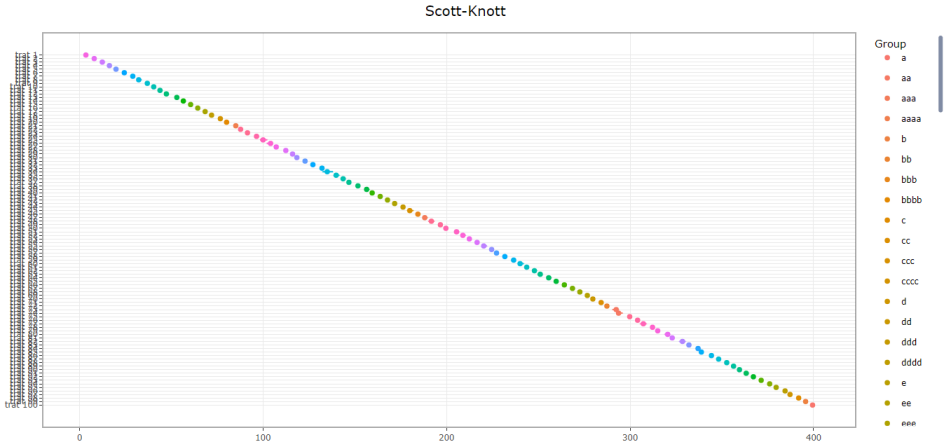
```
Console   Terminal ×   Jobs ×
R 4.1.0 · F:/UnB/USK/
> usktest(y_1Missing_NA~Treatment,x,.05,ANOVA=F)
Error in usktest(y_1Missing_NA ~ Treatment, x, 0.05, ANOVA = F) :
All 'Treatment' must have more than 1 observations
> usktest(Missing~Treatment_unico,x,.05,ANOVA=F)
Error in usktest(Missing ~ Treatment_unico, x, 0.05, ANOVA = F) :
The variable 'Treatment_unico' must have more than 1 type of treatment
```

Fonte: Elaboração Própria.

4.1.5 Verificando o Comportamento da `usktest(...)`, `plot_usk(...)` e `plotly_usk(...)` em Experimentos com Muitos Tratamentos Significativos

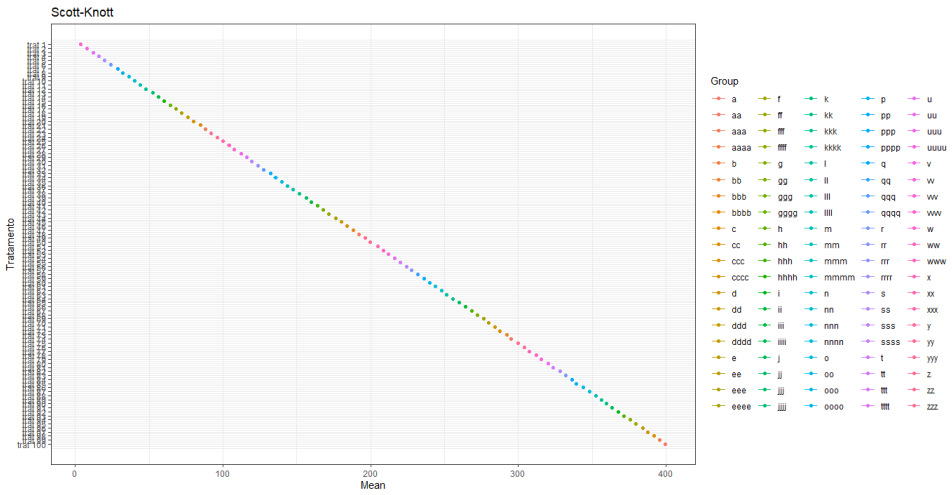
O pacote `USK` foi construído de modo a classificar os tratamentos em grupos distintos independentemente do número de tratamentos significativos. Mas, como pode ser observado na Figura 36 e na Figura 37 [13], sua visualização gráfica acaba por repetir as cores e apresentar sobrescrita dos nomes dos tratamentos.

Figura 36: Visualização gráfica, via `plotly_usk(...)`, de Experimento com muitos grupos distintos.



Fonte: Elaboração Própria.

Figura 37: Visualização gráfica, via `plot_usk(...)`, de Experimento com muitos grupos distintos.



Fonte: Elaboração Própria.

[13] Para replicar: consultar <https://github.com/felipe179971/USK_Simulation/blob/master/simulation/simulation.R>, linha 830 (*Muitos tratamentos*).

Além disso, a apresentação no `console` é feita de forma suprimida, como pode ser visto na Figura 38.

Figura 38: Visualização do `console` ao se executar o teste em Experimento com muitos grupos distintos.

```
Console   Terminal ×   Jobs ×
R 4.1.0 · F:/UnB/USK/
> usktest(y~Tratamento,dados,ANOVA=T)
[1] "##########################ANOVA##########################"
             Df  Sum Sq Mean Sq F value Pr(>F)
Tratamento   99 3955670   39956  148424 <2e-16 ***
Residuals   199      54       0
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
1 observation deleted due to missingness
[1] "#######################Scott-Knott#######################"
# A tibble: 100 x 5
# Groups:   Tratamento [100]
   Tratamento Group  Mean   min   max
   <fct>      <fct> <dbl> <dbl> <dbl>
 1 trat 100   a      400.  399.  400.
 2 trat 99    b      396.  395.  396.
 3 trat 98    c      392.  392.  392.
 4 trat 97    d      388.  387.  388.
 5 trat 96    e      384.  384.  384.
 6 trat 95    f      380.  380.  380.
 7 trat 94    g      376.  376.  377.
 8 trat 93    h      372.  371.  372.
 9 trat 92    i      368.  367.  368.
10 trat 91    j      364.  363.  364.
# ... with 90 more rows
> |
```

Fonte: Elaboração Própria.

4.2 Executando as Simulações de Monte Carlo

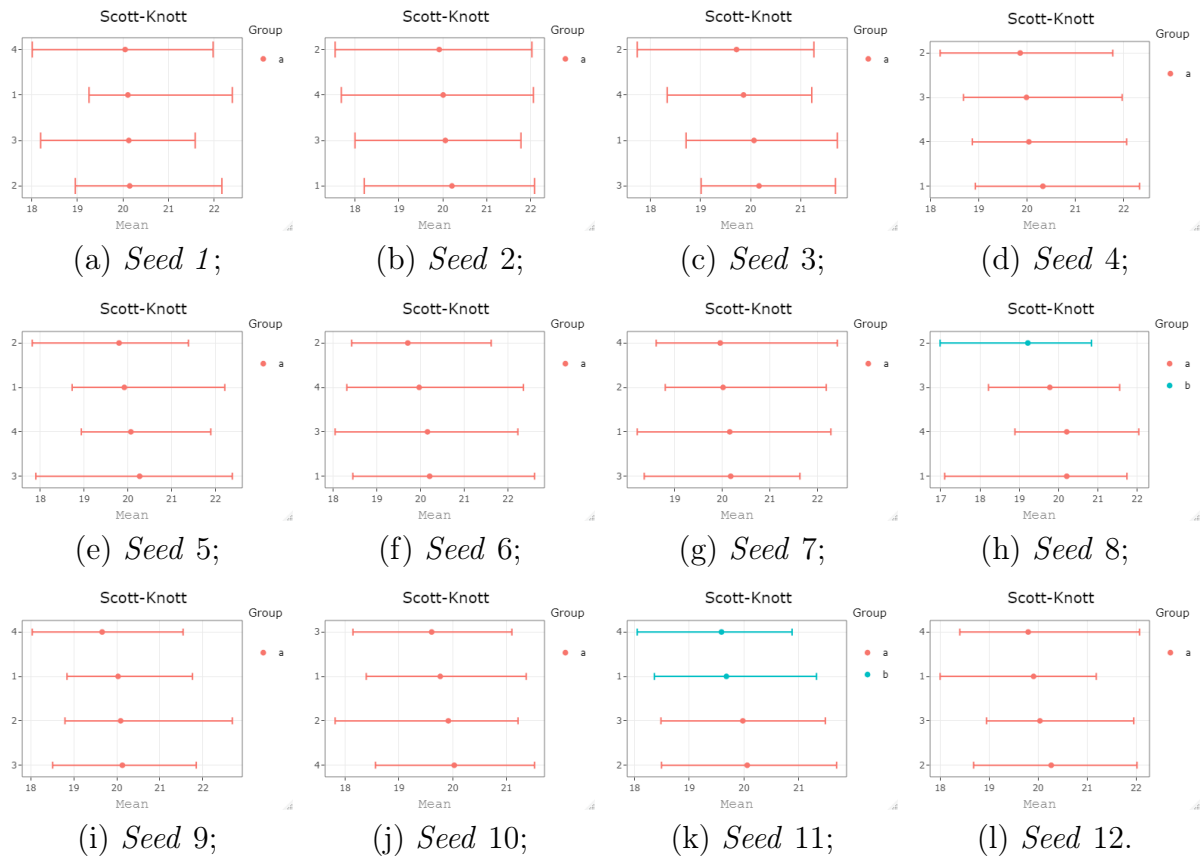
Ao todo, foram realizadas trezentas e sessenta mil (360.000) simulações que buscaram verificar o comportamento do Skott-Knott Modificado e do Skott-Knott Tradicional após Imputação Múltipla em 4 aspectos: poucos tratamentos e poucas observações; poucos tratamentos e muitas observações; muitos tratamentos e poucas observações; e muitos tratamentos e muitas observações.

Em todos os bancos de dados o efeito comum (μ) foi definido, de forma arbitrária, como 20 e os resultados apresentados em forma gráfica, com a tabela podendo ser consultada no **Apêncide A - Resultados das Simulações para Verificar o Erro Tipo I** (Subseção 6.1) e **Apêncide B - Resultados das Simulações para Verificar o Poder do Teste** (Subseção 6.2).

Para exemplificar o processo de simulação a ser realizado, considerando-se o cenário *poucos tratamentos e muitas observações* para verificação do Erro Tipo I (mais detalhes em **Simulação III: Poucas Observações e Muitos Tratamentos para Ve-**

rificar o Erro Tipo I- parágrafo 4.2.1.3), gerou-se 12 bancos de dados (Figura 39) [14]: nesse experimento inicial e observacional, realizando o Skott-Knott Modificado para as doze primeiras observações, com 20 dados faltantes e considerando o α teórico de 0,20, observou-se 2 resultados diferentes do esperado (Figura 39h e Figura 39k), que era de 1 grupo.

Figura 39: Visualização gráfica das doze primeiras iterações utilizando o modelo da Equação 4.2.3 com 20% de dados perdidos e $\alpha = 0,20$.



(a) *Seed 1*; (b) *Seed* 2; (c) *Seed* 3; (d) *Seed* 4;

(e) *Seed* 5; (f) *Seed* 6; (g) *Seed* 7; (h) *Seed* 8;

(i) *Seed* 9; (j) *Seed* 10; (k) *Seed* 11; (l) *Seed* 12.

Fonte: Elaboração Própria.

Nesse exemplo, foram executadas 12 iterações com proporção de acerto de $0,83$, onde a esperada, devido ao $\alpha = 20$, era de $0,80$. Esse processo, com poucas iterações e apenas para o Erro Tipo I do Skott-Knott Modificado, será realizado com 1 mil iterações em 4 cenários distintos para o Erro Tipo I e para o Poder do Teste, com obtenção da proporção de acertos para os três testes: o referencial, Skott-Knott Tradicional com a base antes da perda dos dados; Skott-Knott Modificado; e o Skott-Knott Tradicional após Imputação Múltipla .

[14]Para replicar: consultar <https://github.com/felipe179971/USK_Simulation/blob/master/simulation/simulation.R>, linha 855 (*12 Bancos de dados para o Erro Tipo I*).

4.2.1 Erro Tipo I

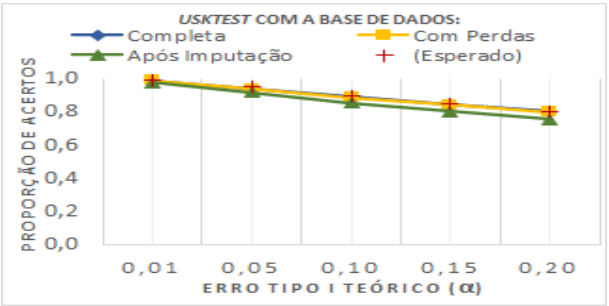
Dado que o Erro Tipo I representa a rejeição de H_0 quando H_0 verdadeira, os mil banco de dados foram criados com tratamentos não significativos e observou-se a proporção de simulações que resultaram em mais de um grupo, levando-se em consideração a perda de 5, 10, 15 e 20 porcento dos dados, $\alpha = 0,01;\ 0,05;\ 0,10;\ 0,15;$ e $0,20$ e $\sigma = 1$.

4.2.1.1 Simulação I: Poucas Observações e Poucos Tratamentos para Verificar o Erro Tipo I

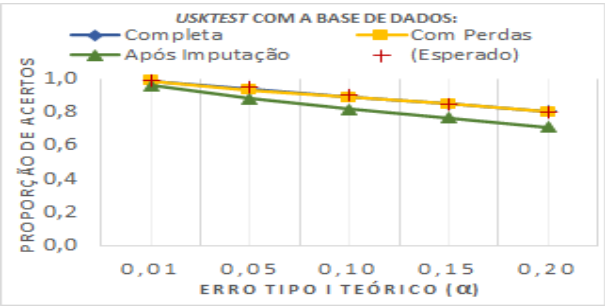
O cenário demonstrado na Equação 4.2.1 consiste em 4 tratamentos não significativos contendo, cada um, 5 observações, gerando banco de dados com 20 unidades experimentais. Os resultados dos 45.000 experimentos feitos nesse cenário estão apresentados graficamente na Figura 40 [15].

$$y_{ij} = 20 + \tau_i + \varepsilon_{ij} \begin{cases} \tau_1 = \tau_2 = \tau_3 = \tau_4 = 0; \\ \varepsilon_{ij} \sim N(0,1); \\ j = 1, 2, ..., 5. \end{cases} \tag{4.2.1}$$

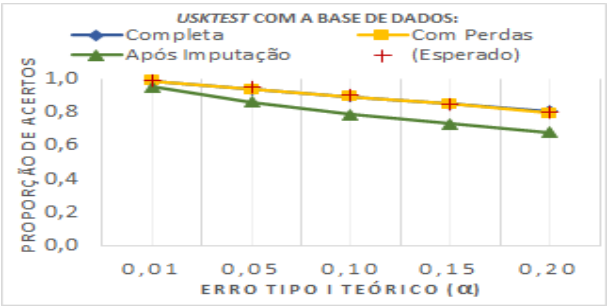
Figura 40: Simulação I - Verificando o Erro Tipo I, empiricamente, por meio do cenário da Equação 4.2.1.



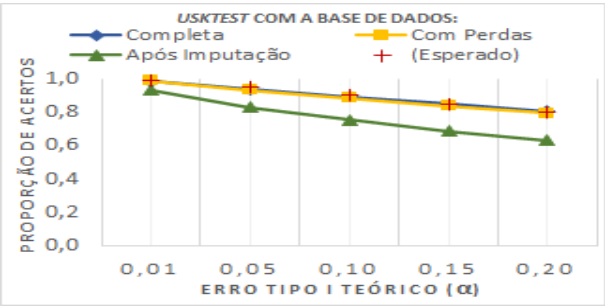
(a) 5% de dados perdidos;



(b) 10% de dados perdidos;



(c) 15% de dados perdidos;



(d) 20% de dados perdidos.

Fonte: Elaboração Própria.

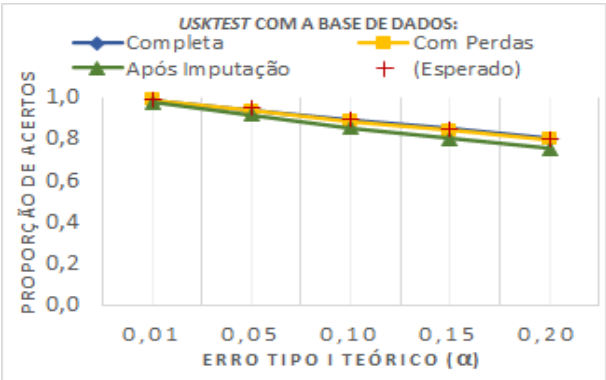
[15] Para replicar: consultar <https://github.com/felipe179971/USK_Simulation/blob/master/simulation/simulation.R>, linha 101 (*Simulação I*).

4.2.1.2 Simulação II: Muitas Observações e Poucos Tratamentos para Verificar o Erro Tipo I

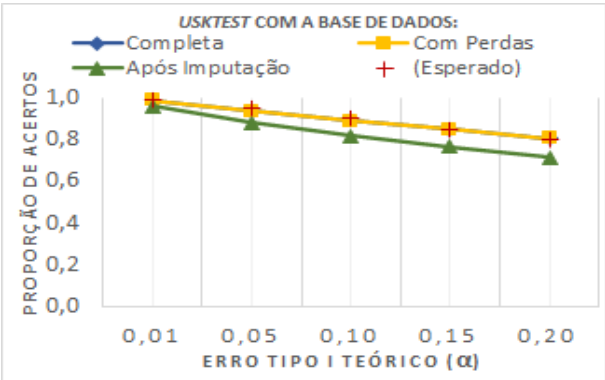
O cenário demonstrado na Equação 4.2.2 consiste em 4 tratamentos não significativos contendo, cada um, 25 observações, gerando banco de dados com 100 unidades experimentais. Os resultados dos 45.000 experimentos feitos nesse cenário estão apresentados graficamente na Figura 41 [16].

$$y_{ij} = 20 + \tau_i + \varepsilon_{ij} \begin{cases} \tau_1 = \tau_2 = \tau_3 = \tau_4 = 0; \\ \varepsilon_{ij} \sim N(0,1); \\ j = 1, 2, ..., 25. \end{cases} \tag{4.2.2}$$

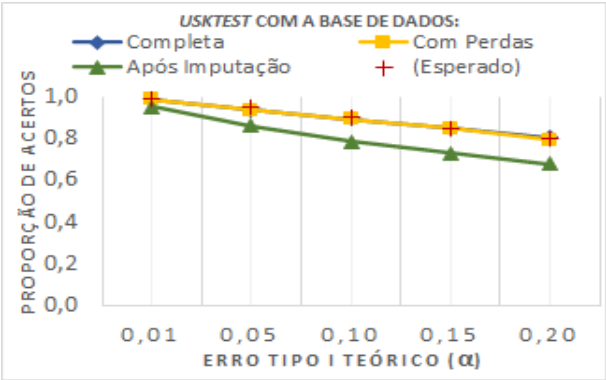
Figura 41: Simulação II - Verificando o Erro Tipo I, empiricamente, por meio do cenário da Equação 4.2.2.



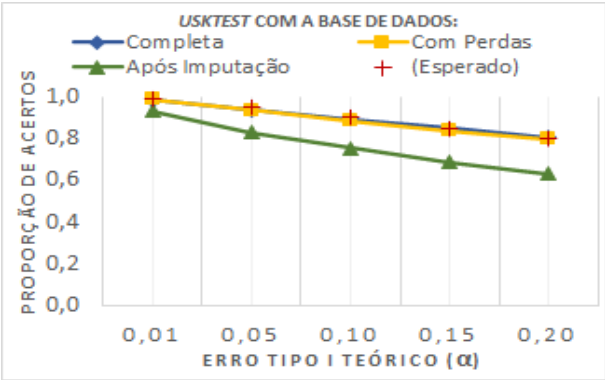
(a) 5% de dados perdidos;



(b) 10% de dados perdidos;



(c) 15% de dados perdidos;



(d) 20% de dados perdidos.

Fonte: Elaboração Própria.

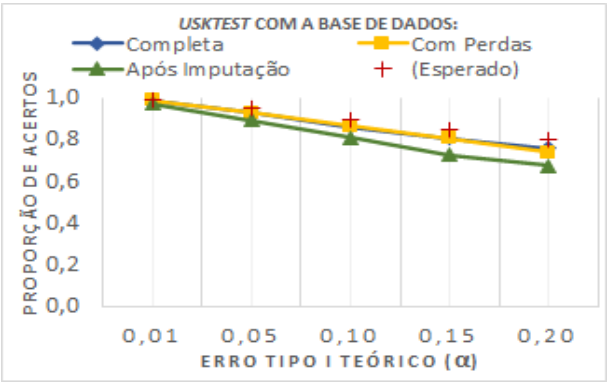
[16] Para replicar: consultar <https://github.com/felipe179971/USK_Simulation/blob/master/simulation/simulation.R>, linha 182 (*Simulação II*).

4.2.1.3 Simulação III: Poucas Observações e Muitos Tratamentos para Verificar o Erro Tipo I

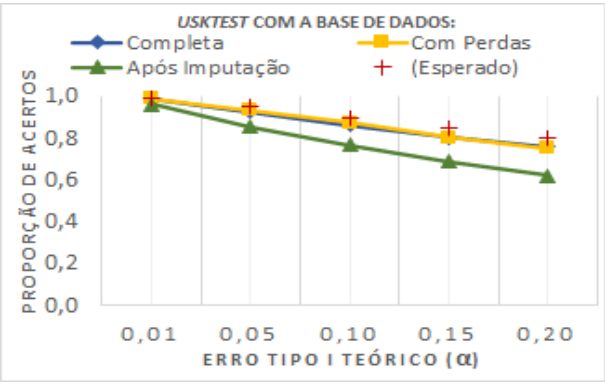
O cenário demonstrado na Equação 4.2.3 consiste em 10 tratamentos não significativos contendo, cada um, 10 observações, gerando banco de dados com 100 unidades experimentais. Os resultados dos 45.000 experimentos feitos nesse cenário estão apresentados graficamente na Figura 42 [17].

$$y_{ij} = 20 + \tau_i + \varepsilon_{ij} \begin{cases} \tau_1 = \tau_2 = ... = \tau_{10} = 0; \\ \varepsilon_{ij} \sim N(0,1); \\ j = 1, 2, ..., 10. \end{cases} \tag{4.2.3}$$

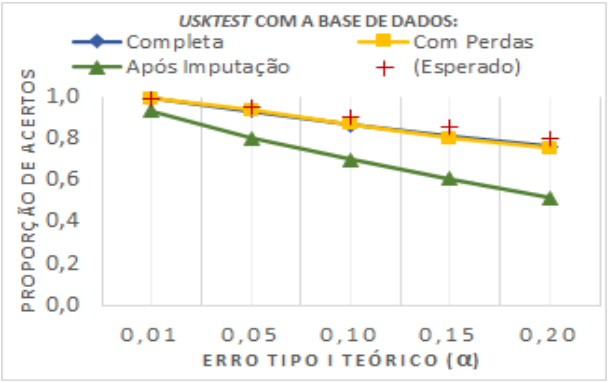
Figura 42: Simulação III - Verificando o Erro Tipo I, empiricamente, por meio do cenário da Equação 4.2.3.



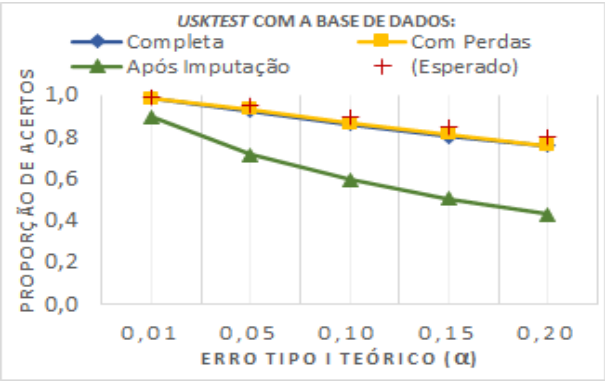
(a) 5% de dados perdidos;



(b) 10% de dados perdidos;



(c) 15% de dados perdidos;



(d) 20% de dados perdidos.

Fonte: Elaboração Própria.

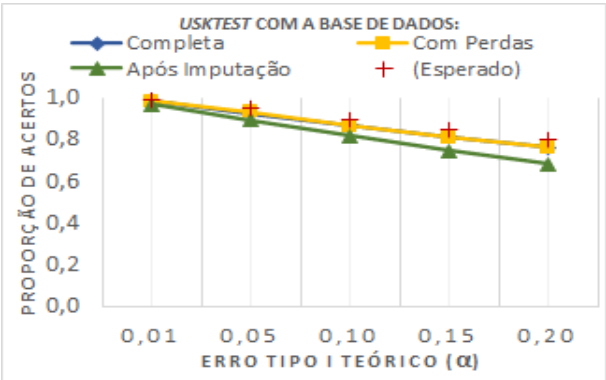
[17] Para replicar: consultar <https://github.com/felipe179971/USK_Simulation/blob/master/simulation/simulation.R>, linha 262 (*Simulação III*).

4.2.1.4 Simulação IV: Muitas Observações e Muitos Tratamentos para Verificar o Erro Tipo I

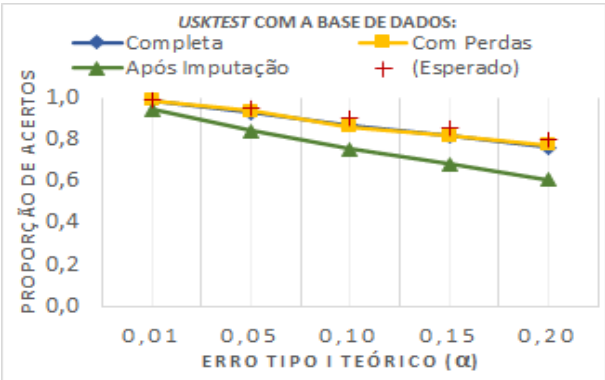
O cenário demonstrado na Equação 4.2.4 consiste em 10 tratamentos não significativos contendo, cada um, 50 observações, gerando banco de dados com 500 unidades experimentais. Os resultados dos 45.000 experimentos feitos nesse cenário estão apresentados graficamente na Figura 43 [18].

$$y_{ij} = 20 + \tau_i + \varepsilon_{ij} \begin{cases} \tau_1 = \tau_2 = ... = \tau_{10} = 0; \\ \varepsilon_{ij} \sim N(0,1); \\ j = 1, 2, ..., 50. \end{cases} \tag{4.2.4}$$

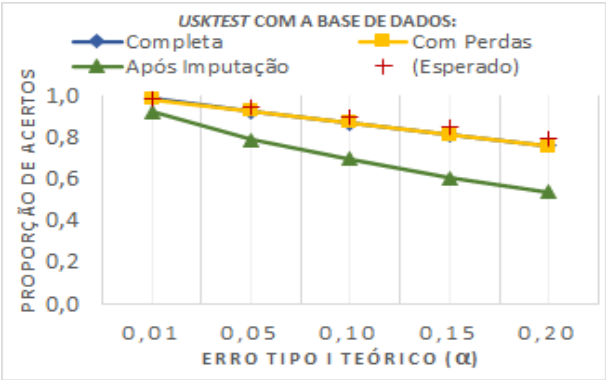
Figura 43: Simulação IV - Verificando o Erro Tipo I, empiricamente, por meio do cenário da Equação 4.2.4.



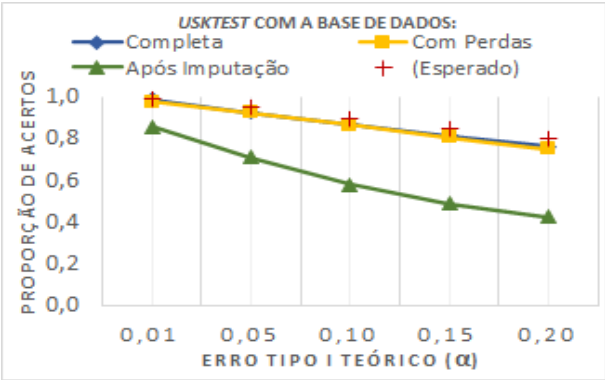
(a) 5% de dados perdidos;



(b) 10% de dados perdidos;



(c) 15% de dados perdidos;



(d) 20% de dados perdidos.

Fonte: Elaboração Própria.

[18] Para replicar: consultar <https://github.com/felipe179971/USK_Simulation/blob/master/simulation/simulation.R>, linha 344 (*Simulação IV*).

4.2.1.5 Considerações Acerca do Erro Tipo I

Conforme observado nas simulações, a execução do Skott-Knott Modificado apresentou resultados similares ao Tradicional, ambos próximos ao α teórico esperado, sendo que o aumento das parcelas perdidas não implicou em grandes desvios do empírico em relação ao teórico. Por outro lado, a aplicação do Skott-Knott Tradicional após realização da técnica de imputação múltipla apresentou-se de difícil controle em relação ao Erro Tipo I, principalmente à medida em que o número de parcelas perdidas aumenta.

Portanto, considerando o Erro Tipo I, a aplicação da modificação proposta por Conrado apresentou-se com desempenho satisfatório, inclusive em banco de dados com poucas observações e muitas parcelas perdidas. Desse modo, sua aplicação é mais indicada à aplicação da Imputação Múltipla para posterior execução do Scott-Knott Tradicional no tocante ao controle desse tipo de Erro.

4.2.2 Poder do Teste

Diferentemente da simulação anterior, em que a rejeição era um fator negativo devido aos bancos terem sido construídos sem efeitos dos tratamentos, o foco dessa análise é a de verificar o poder que o teste tem em identificar e classificar os tratamentos significativos em grupos distintos.

Observando que a distância entre os efeitos dos tratamentos é altamente impactada pelo sigma, ou seja, a depender do número de observações e do α teórico, os tratamentos $\tau_1 = 1$ e $\tau_2 = -1$ (distância 2) podem ser distintos se tiverem pouca variabilidade, mas, com muita, essa diferença torna-se insignificante.

Portanto, optou-se por não fixar o σ, afim de não interferir de forma positiva ou negativa nos resultados, sendo este verificado para 5 valores distintos: 1, 2, 3, 4 e 5; com um α fixado em 0,05 (quanto maior o α, maior a chance de cometer o Erro Tipo I, o que poderia beneficiar e interferir na análise do poder); e por fim, a distância mínima entre τ_a e τ_b $(a \neq b)$ é de 5.

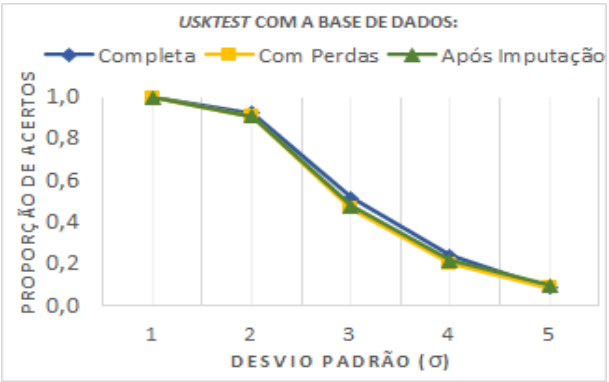
4.2.2.1 Simulação V: Poucas Observações e Poucos Tratamentos para Verificar o Poder do Teste

O cenário demonstrado na Equação 4.2.5 consiste em 4 tratamentos contendo, cada um, 5 observações, gerando banco de dados com 20 unidades experimentais. Os resultados dos 45.000 experimentos feitos nesse cenário estão apresentados graficamente na Figura 44 [19].

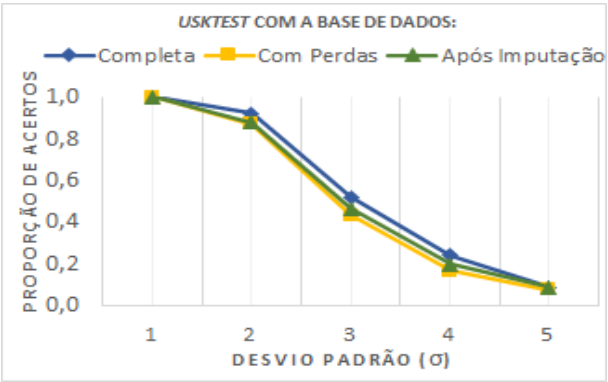
[19] Para replicar: consultar <https://github.com/felipe179971/USK_Simulation/blob/master/simulation/simulation.R>, linha 429 (*Simulação V*).

$$y_{ij} = 20 + \tau_i + \varepsilon_{ij} \begin{cases} \tau_1 = 10,\ \tau_2 = 5,\ \tau_3 = -5 \text{ e } \tau_4 = -10; \\ \varepsilon_{ij} \sim N(0, \sigma); \\ j = 1, 2, ..., 5. \end{cases} \tag{4.2.5}$$

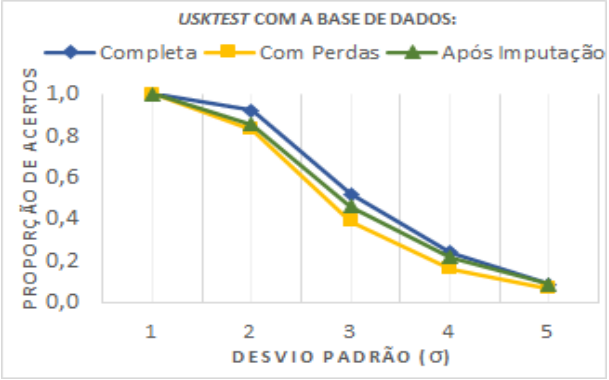
Figura 44: Simulação V - Verificando o Poder do Teste, empiricamente, por meio do cenário da Equação 4.2.5.



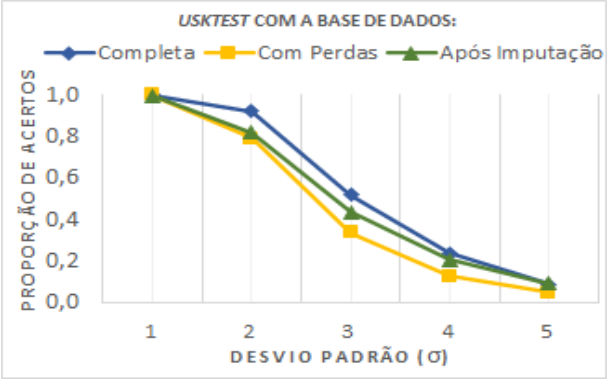
(a) 5% de dados perdidos;



(b) 10% de dados perdidos;



(c) 15% de dados perdidos;



(d) 20% de dados perdidos.

Fonte: Elaboração Própria.

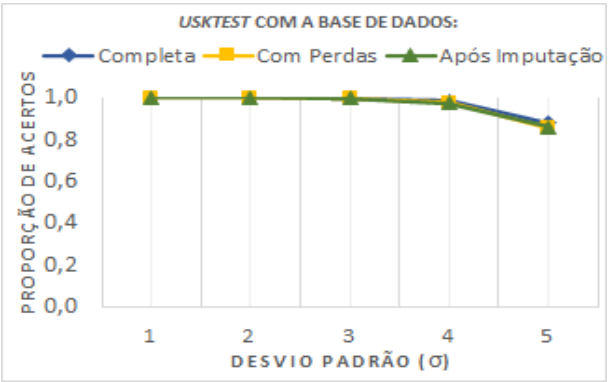
4.2.2.2 Simulação VI: Muitas Observações e Poucos Tratamentos para Verificar o Poder do Teste

O cenário demonstrado na Equação 4.2.6 consiste em 4 tratamentos contendo, cada um, 25 observações, gerando banco de dados com 100 unidades experimentais. Os resultados dos 45.000 experimentos feitos nesse cenário estão apresentados graficamente na Figura 45 [20].

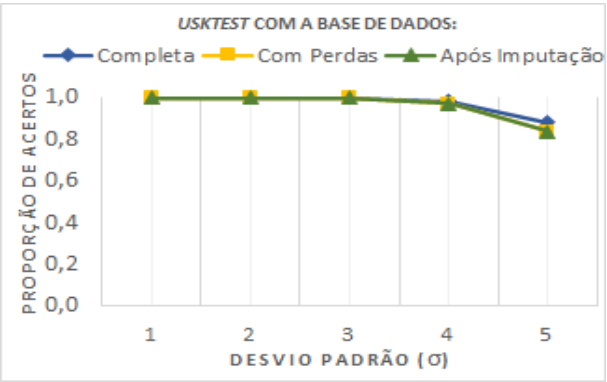
[20] Para replicar: consultar <https://github.com/felipe179971/USK_Simulation/blob/master/simulation/simulation.R>, linha 514 (*Simulação VI*).

$$y_{ij} = 20 + \tau_i + \varepsilon_{ij} \begin{cases} \tau_1 = 10,\ \tau_2 = 5,\ \tau_3 = -5 \text{ e } \tau_4 = -10; \\ \varepsilon_{ij} \sim N(0, \sigma); \\ j = 1, 2, ..., 25. \end{cases} \tag{4.2.6}$$

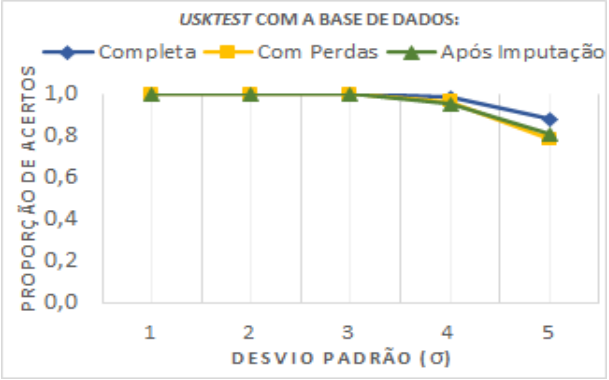
Figura 45: Simulação VI - Verificando o Poder do Teste, empiricamente, por meio do cenário da Equação 4.2.6.



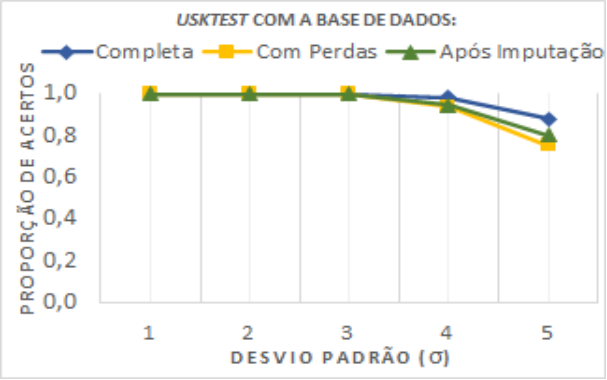
(a) 5% de dados perdidos;



(b) 10% de dados perdidos;



(c) 15% de dados perdidos;



(d) 20% de dados perdidos.

Fonte: Elaboração Própria.

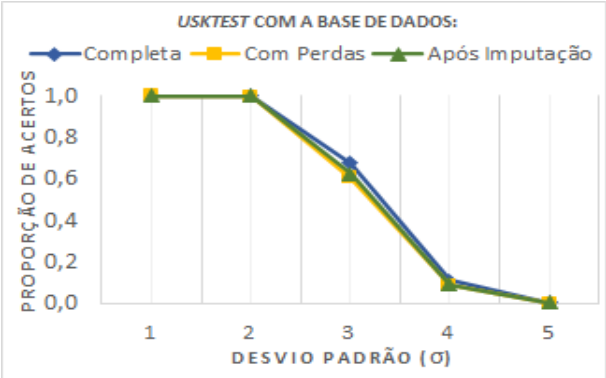
4.2.2.3 Simulação VII: Poucas Observações e Muitos Tratamentos para Verificar o Poder do Teste

O cenário demonstrado na Equação 4.2.7 consiste em 10 tratamentos contendo, cada um, 10 observações, gerando banco de dados com 100 unidades experimentais. Os resultados dos 45.000 experimentos feitos nesse cenário estão apresentados graficamente na Figura 46 [21].

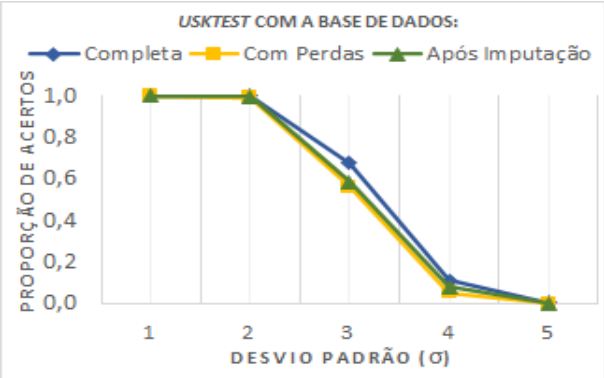
[21] Para replicar: consultar <https://github.com/felipe179971/USK_Simulation/blob/master/simulation/simulation.R>, linha 600 (*Simulação VII*).

$$y_{ij} = 20 + \tau_i + \varepsilon_{ij} \begin{cases} \tau_1 = 25,\ \tau_2 = 20,\ \tau_3 = 15,\ \tau_4 = 10,\ \tau_5 = 5, \\ \tau_6 = -5,\ \tau_7 = -10,\ \tau_8 = -15,\ \tau_9 = -20 \text{ e } \tau_{10} = -25; \\ \varepsilon_{ij} \sim N(0, \sigma); \\ j = 1, 2, ..., 10. \end{cases} \tag{4.2.7}$$

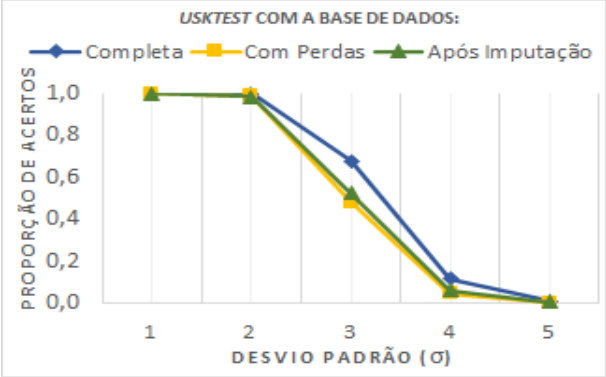
Figura 46: Simulação VII - Verificando o Poder do Teste, empiricamente, por meio do cenário da Equação 4.2.7.



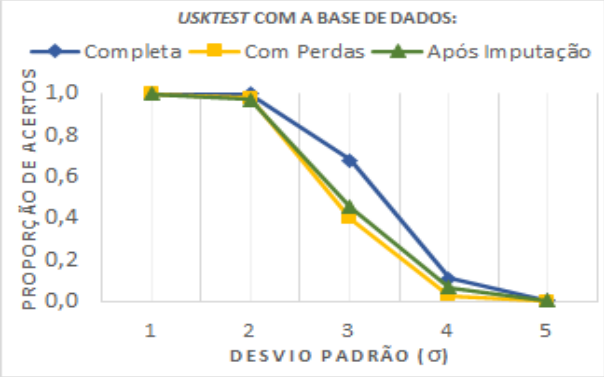
(a) 5% de dados perdidos;



(b) 10% de dados perdidos;



(c) 15% de dados perdidos;



(d) 20% de dados perdidos.

Fonte: Elaboração Própria.

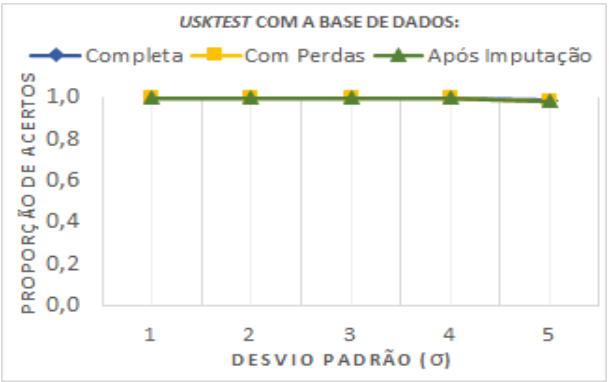
4.2.2.4 Simulação VIII: Muitas Observações e Muitos Tratamentos para Verificar o Poder do Teste

O cenário demonstrado na Equação 4.2.8 consiste em 10 tratamentos contendo, cada um, 50 observações, gerando banco de dados com 500 unidades experimentais. Os resultados dos 45.000 experimentos feitos nesse cenário estão apresentados graficamente na Figura 47 [22].

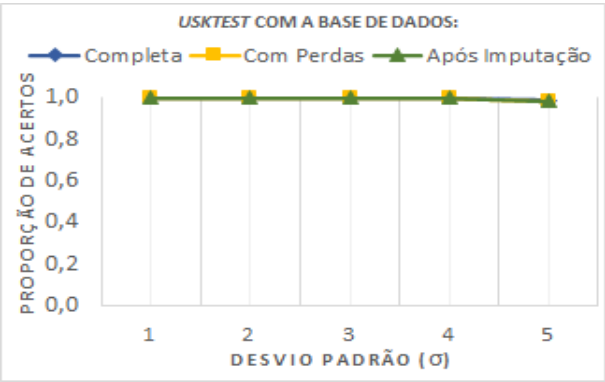
[22]Para replicar: consultar <https://github.com/felipe179971/USK_Simulation/blob/master/simulation/simulation.R>, linha 681 (*Simulação VIII*).

$$y_{ij} = 20 + \tau_i + \varepsilon_{ij} \begin{cases} \tau_1 = 25,\ \tau_2 = 20,\ \tau_3 = 15,\ \tau_4 = 10,\ \tau_5 = 5, \\ \tau_6 = -5,\ \tau_7 = -10,\ \tau_8 = -15,\ \tau_9 = -20 \text{ e } \tau_{10} = -25; \\ \varepsilon_{ij} \sim N(0, \sigma); \\ j = 1, 2, ..., 50. \end{cases} \tag{4.2.8}$$

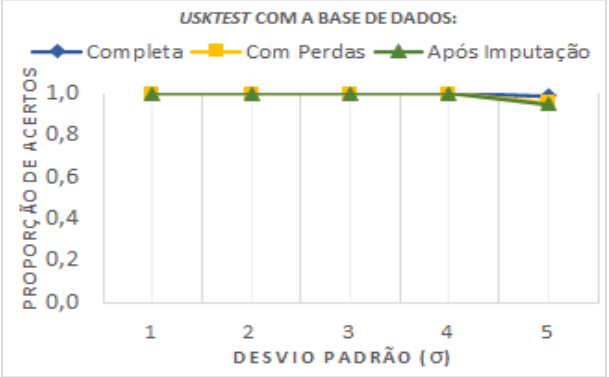
Figura 47: Simulação VIII - Verificando o Poder do Teste, empiricamente, por meio do cenário da Equação 4.2.8.



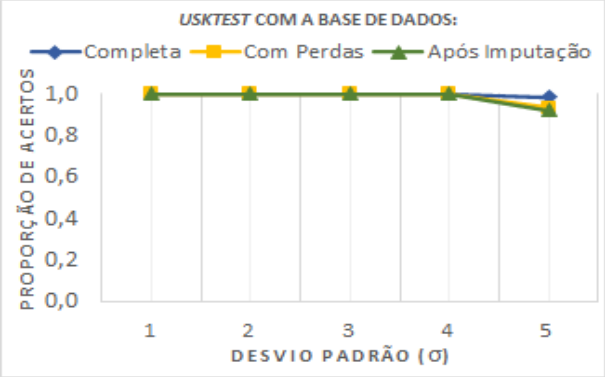
(a) 5% de dados perdidos;



(b) 10% de dados perdidos;



(c) 15% de dados perdidos;



(d) 20% de dados perdidos.

Fonte: Elaboração Própria.

4.2.2.5 Considerações Acerca do Poder do Teste

Conforme observado nas simulações, os testes apresentaram alto Poder do Teste, com convergência à 100% de acertos, para todos as porcentagens de parcelas perdidas testadas, à medida em que o número de observações cresce. Importante atentar-se ao fato de que o Poder está diretamente relacionado ao delineamento do experimento: a realização de experimentos com poucas observações necessita, obrigatoriamente, de pouca variabilidade dos dados para que se tenha desempenho satisfatório.

Ademais, o aumento das porcentagens de parcelas perdidas demonstrou pouco

impacto, sendo esse mais perceptível nas *Figura 44* e *Figura 46*, ou seja, em poucas observações e, principalmente, com muita variabilidade.

Portanto, em relação ao Poder do Teste, realizar a Imputação Múltipla para posterior aplicação do Skott-Knott Tradicional ou realizar a modificação proposta por Conrado demonstraram-se igualmente satisfatórios, com alto Poder e diferindo pouco do Poder do referencial (Skott-Knott Tradicional com a base antes da perda dos dados).

4.3 Conclusões

Esse trabalho, como previsto, apresentou todos os passos para a criação do pacote `USK`, que foi disponibilizado para *download* aberto ao publico via *GitHub*, com implementação computacional de consulta aberta no site do pacote: <https://github.com/felipe179971/USK>. Além disso, contou com trezentos e sessenta mil (360.000) experimentos na realização de Simulações de Monte Carlo que demostraram a robustez e adequação ao que dele, teoricamente, se esperava.

Por fim, por meio desse trabalho, concluiu-se que, independentemente do cenário, realizar o Skott-Knott após Imputação Múltiplas não diferiu, no que diz respeito ao Poder do Teste, de forma visualmente significativa do Skott-Knott Tradicional com dados antes da perda de parcelas; entretanto, apresenta grande dificuldade no controle do Erro Tipo I. Dessa forma, a modificação proposta por Conrado e implementadas no pacote `USK` é a melhor alternativa por também apresentar desempenhos de Poder próximos ao método tradicional aplicado em bases de dados completas, mas mantendo um alto controle do Erro Tipo I.

5 Referências

CONRADO, Thiago Vincenzi et al. Adjusting the Scott-Knott cluster analyses for unbalanced designs. **Crop Breeding and Applied Biotechnology**, Viçosa, v. 17, n. 1, p. 1-9, Mar. 2017. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1984-70332017000100001&lng=en&nrm=iso>. Acesso em: 05 Jan. 2021.

MARC Code List for Relators. **Code List for Relators**,26 Abr. 2021. Disponível em: . Acesso em:<https://www.loc.gov/marc/relators/relaterm.html>.

METROPOLIS, Nicholas; ULAM, S. The Monte Carlo Method. **Journal of the American Statistical Association**, vol. 44, no. 247, Set. 1949, p. 335–341. JSTOR, Disponível em: <www.jstor.org/stable/2280232>. Acesso em: 5 Jan. 2021.

MONTGOMERY, Douglas C.. **Design and analysis of experiments**, Edição 9, Arizona State University: Wiley, 2017.

NUNES, Luciana; KLüCK, Mariza; FACHEL, Jandyra. Uso da imputação múltipla de dados faltantes: uma simulação utilizando dados epidemiológicos. **Cad. Saúde Pública**, Rio de Janeiro, volume 51, p. 268-278, fev. 2009. Disponível em: <https://www.scielo.br/j/csp/a/XW3NwV7T5VL77WN7d7TJ3ZR/?format=pdf>. Acesso em: 5 fev. 2021

PALCZEWSKI, Jan; PALCZEWSKI, Andrzej. **Monte Carlo Simulation**. [21−−?]. 54 slides. Disponível em: <https://www.mimuw.edu.pl/~apalczew/CFP_lecture4.pdf>. Acesso em: 26 out. 2021.

PAULA, Renato Ricardo de. **Método de Monte Carlo e Aplicações**. 2014. Dissertação (Graduação em Matemática com Ênfase em Matemática Computacional) - Instituto de Ciências Exatas - ICEx, Universidade Federal Fluminense, Rio de Janeiro, 2014.

SCOTT, A. J.; KNOTT, M. A Cluster Analysis Method for Grouping Means in the Analysis of Variance. **Biometrics**, vol. 30, no. 3, Set. 1974, p. 507–512. JSTOR, Disponível em: <www.jstor.org/stable/2529204>. Acesso em: 5 Jan. 2021.

SILVA, Eloise Cury da; FERREIRA, Daniel Furtado; BEARZOTI, Eduardo. **Avaliação do poder e taxas de erro tipo I do Teste de Scott-Knott por meio do método de Monte Carlo**. 1999, 9 f. Dissertação (Mestrado em Ciênc. agrotec.) - Universidade Federal de Lavras, Minas Gerais, p. 687-69.

SILVA, Cristiane Mariana Rodrigues da. **Uso do teste de Scott-Knott e da análise de agrupamentos, na obtenção de grupos de locais para experimentos com cana-de-açúcar**. 2007. Dissertação (Mestrado em Estatística e Experimentação Agronômica) - Escola Superior de Agricultura Luiz de Queiroz, Universidade de São

Paulo, Piracicaba, 2008. Disponível em: <https://www.teses.usp.br/teses/disponiveis/11/11134/tde-12032008-151057/pt-br.php>. Acesso em: 10 Jan. 2021.

GITHUB INC. a. **GNU General Public License v3.0**: GNU GPLv3. c2021. Disponível em: <https://choosealicense.com/licenses/gpl-3.0/>. Acesso em: 13 fev. 2021.

GITHUB INC. b. **Where the world builds software**: Millions of developers and companies build, ship, and maintain their software on GitHub—the largest and most advanced development platform in the world. c2021. Disponível em: <https://github.com/about>. Acesso em: 13 fev. 2021.

VAN BUUREN, Stef. **The book Flexible Imputation of Missing Data**. 2th ed. Boca Raton, FL.: Chapman Hall/CRC Press, 2018. Disponível em: <https://stefvanbuuren.name/fimd/>.

WICKHAM, Hadley; GROLEMUND, Garrett. **R for Data Science**: Import, Tidy, Transform, Visualize, and Model Data. United States: R4DS, 17 jan. 2017. E-book. Disponível em: <https://r4ds.had.co.nz/>. Acesso em: 25 Jan. 2021.

WICKHAM, Hadley; BRYAN, Jenny. **R packages**: Organize, test, document and share your code. United States: O'Reilly Media, 28 Abr. 2015. E-book. Disponível em: <https://r-pkgs.org/>. Acesso em: 1 fev. 2021.

6 Apêndices

6.1 Apêncide A - Resultados das Simulações para Verificar o Erro Tipo I

Simulação I - Poucas Observações e Poucos Tratamentos para Verificaro Erro Tipo I

α	Porcentagem de Parcelas Perdidas								
	0%	**5%**		**10%**		**15%**		**20%**	
	Completa	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**
0,01	0,989	0,994	0,985	0,993	0,983	0,995	0,97	0,992	0,954
0,05	0,936	0,947	0,919	0,938	0,894	0,94	0,859	0,931	0,835
0,10	0,871	0,882	0,855	0,885	0,805	0,873	0,771	0,865	0,728
0,15	0,814	0,826	0,786	0,817	0,739	0,827	0,708	0,805	0,644
0,20	0,767	0,76	0,723	0,762	0,673	0,766	0,637	0,755	0,579

Simulação II - Muitas Observações e Poucos Tratamentos para Verificar o Erro Tipo I

α	Porcentagem de Parcelas Perdidas								
	0%	**5%**		**10%**		**15%**		**20%**	
	Completa	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**
0,01	0,986	0,988	0,976	0,988	0,959	0,99	0,952	0,987	0,933
0,05	0,938	0,937	0,918	0,935	0,881	0,936	0,86	0,935	0,83
0,10	0,892	0,887	0,855	0,888	0,818	0,893	0,788	0,885	0,755
0,15	0,849	0,843	0,804	0,848	0,768	0,848	0,731	0,837	0,686
0,20	0,805	0,801	0,757	0,804	0,713	0,793	0,679	0,796	0,633

Simulação III - Poucas Observações e Muitos Tratamentos para Verificar o Erro Tipo I

α	Porcentagem de Parcelas Perdidas								
	0%	**5%**		**10%**		**15%**		**20%**	
	Completa	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**
0,01	0,991	0,989	0,974	0,99	0,962	0,992	0,932	0,988	0,899
0,05	0,929	0,93	0,895	0,936	0,856	0,935	0,799	0,937	0,72
0,10	0,865	0,866	0,81	0,877	0,77	0,868	0,696	0,868	0,599
0,15	0,807	0,809	0,728	0,806	0,689	0,799	0,606	0,816	0,507
0,20	0,761	0,739	0,675	0,753	0,623	0,751	0,513	0,76	0,431

Simulação IV - Muitas Observações e Muitos Tratamentos para Verificar o Erro Tipo I

α	Porcentagem de Parcelas Perdidas								
	0%	**5%**		**10%**		**15%**		**20%**	
	Completa	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**
0,01	0,988	0,984	0,969	0,986	0,942	0,984	0,927	0,978	0,86
0,05	0,926	0,935	0,894	0,935	0,841	0,929	0,791	0,927	0,71
0,10	0,869	0,867	0,817	0,86	0,753	0,874	0,699	0,868	0,581
0,15	0,815	0,812	0,746	0,818	0,683	0,816	0,607	0,806	0,488
0,20	0,762	0,763	0,685	0,772	0,608	0,76	0,542	0,753	0,427

Fonte: Elaboração própria.

6.2 Apêncide B - Resultados das Simulações para Verificar o Poder do Teste

Simulação V - Poucas Observações e Poucos Tratamentos para Verificar o Poder do Teste

σ	Porcentagem de Parcelas Perdidas								
	0%	**5%**		**10%**		**15%**		**20%**	
	Completa	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**
1	1	1	1	1	1	1	1	1	0,998
2	0,923	0,909	0,91	0,872	0,881	0,833	0,856	0,791	0,821
3	0,519	0,469	0,48	0,435	0,464	0,389	0,461	0,34	0,438
4	0,241	0,202	0,217	0,167	0,199	0,161	0,219	0,128	0,208
5	0,089	0,088	0,099	0,074	0,086	0,066	0,088	0,049	0,092

Simulação VI - Muitas Observações e Poucos Tratamentos para Verificar o Poder do Teste

σ	Porcentagem de Parcelas Perdidas								
	0%	**5%**		**10%**		**15%**		**20%**	
	Completa	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**
1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1
3	0,999	0,999	0,999	0,999	0,999	0,999	0,999	0,999	0,998
4	0,985	0,976	0,975	0,973	0,971	0,963	0,954	0,94	0,945
5	0,88	0,856	0,859	0,837	0,838	0,785	0,809	0,754	0,802

Simulação VII - Poucas Observações e Muitos Tratamentos para Verificar o Poder do Teste

σ	Porcentagem de Parcelas Perdidas								
	0%	**5%**		**10%**		**15%**		**20%**	
	Completa	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**
1	1	1	1	1	1	1	1	1	1
2	0,999	0,997	0,999	0,992	0,995	0,99	0,986	0,981	0,972
3	0,678	0,607	0,625	0,564	0,586	0,482	0,522	0,399	0,453
4	0,113	0,089	0,093	0,055	0,081	0,044	0,058	0,031	0,068
5	0,007	0,005	0,006	0,002	0,003	0	0,004	0	0,004

Simulação VIII - Muitas Observações e Muitos Tratamentos para Verificar o Poder do Teste

σ	Porcentagem de Parcelas Perdidas								
	0%	**5%**		**10%**		**15%**		**20%**	
	Completa	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**
1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1	1
5	0,987	0,982	0,982	0,973	0,972	0,956	0,95	0,933	0,922

Fonte: Elaboração própria.

6.3 Apêncide C - $\frac{S}{\sqrt{M}}$ para as mil Iterações Realizadas nos Diferentes Cenários para verificar o Erro Tipo I

Simulação I ($\frac{S}{\sqrt{M}}$) - Poucas Observações e Poucos Tratamentos para Verificaro Erro Tipo I

α	Porcentagem de Parcelas Perdidas								
	0%	**5%**		**10%**		**15%**		**20%**	
	Completa	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**
0,01	0,0033	0,0024	0,0038	0,0026	0,0041	0,0022	0,0054	0,0028	0,0066
0,05	0,0077	0,0071	0,0086	0,0076	0,0097	0,0075	0,0110	0,0080	0,0117
0,10	0,0106	0,0102	0,0111	0,0101	0,0125	0,0105	0,0133	0,0108	0,0141
0,15	0,0123	0,0120	0,0130	0,0122	0,0139	0,0120	0,0144	0,0125	0,0151
0,20	0,0134	0,0135	0,0142	0,0135	0,0148	0,0134	0,0152	0,0136	0,0156

Simulação II ($\frac{S}{\sqrt{M}}$) - Muitas Observações e Poucos Tratamentos para Verificar o Erro Tipo I

α	Porcentagem de Parcelas Perdidas								
	0%	**5%**		**10%**		**15%**		**20%**	
	Completa	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**
0,01	0,0037	0,0034	0,0048	0,0034	0,0063	0,0031	0,0068	0,0036	0,0079
0,05	0,0076	0,0077	0,0087	0,0078	0,0102	0,0077	0,0110	0,0078	0,0119
0,10	0,0098	0,0100	0,0111	0,0100	0,0122	0,0098	0,0129	0,0101	0,0136
0,15	0,0113	0,0115	0,0126	0,0114	0,0134	0,0114	0,0140	0,0117	0,0147
0,20	0,0125	0,0126	0,0136	0,0126	0,0143	0,0128	0,0148	0,0127	0,0152

Simulação III ($\frac{S}{\sqrt{M}}$) - Poucas Observações e Muitos Tratamentos para Verificar o Erro Tipo I

α	Porcentagem de Parcelas Perdidas								
	0%	**5%**		**10%**		**15%**		**20%**	
	Completa	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**
0,01	0,0030	0,0033	0,0050	0,0031	0,0060	0,0028	0,0080	0,0034	0,0095
0,05	0,0081	0,0081	0,0097	0,0077	0,0111	0,0078	0,0127	0,0077	0,0142
0,10	0,0108	0,0108	0,0124	0,0104	0,0133	0,0107	0,0146	0,0107	0,0155
0,15	0,0125	0,0124	0,0141	0,0125	0,0146	0,0127	0,0155	0,0123	0,0158
0,20	0,0135	0,0139	0,0148	0,0136	0,0153	0,0137	0,0158	0,0135	0,0157

Simulação IV ($\frac{S}{\sqrt{M}}$) - Muitas Observações e Muitos Tratamentos para Verificar o Erro Tipo I

α	Porcentagem de Parcelas Perdidas								
	0%	**5%**		**10%**		**15%**		**20%**	
	Completa	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**	**Perdas**	**Imputação**
0,01	0,0034	0,0040	0,0055	0,0037	0,0074	0,0040	0,0082	0,0046	0,0110
0,05	0,0083	0,0078	0,0097	0,0078	0,0116	0,0081	0,0129	0,0082	0,0144
0,10	0,0107	0,0107	0,0122	0,0110	0,0136	0,0105	0,0145	0,0107	0,0156
0,15	0,0123	0,0124	0,0138	0,0122	0,0147	0,0123	0,0155	0,0125	0,0158
0,20	0,0135	0,0135	0,0147	0,0133	0,0154	0,0135	0,0158	0,0136	0,0156

Fonte: Elaboração própria.

6.4 Apêncide D - $\frac{S}{\sqrt{M}}$ para as mil Iterações Realizadas nos Diferentes Cenários para verificar o Poder do Teste I

Simulação V ($\frac{S}{\sqrt{M}}$) - Poucas Observações e Poucos Tratamentos para Verificar o Poder do Teste

σ	Porcentagem de Parcelas Perdidas								
	0%	5%		10%		15%		20%	
	Completa	Perdas	Imputação	Perdas	Imputação	Perdas	Imputação	Perdas	Imputação
1	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,001
2	0,008	0,009	0,009	0,011	0,010	0,012	0,011	0,013	0,012
3	0,016	0,016	0,016	0,016	0,016	0,015	0,016	0,015	0,016
4	0,014	0,013	0,013	0,012	0,013	0,012	0,013	0,011	0,013
5	0,009	0,009	0,009	0,008	0,009	0,008	0,009	0,007	0,009

Simulação VI ($\frac{S}{\sqrt{M}}$) - Muitas Observações e Poucos Tratamentos para Verificar o Poder do Teste

σ	Porcentagem de Parcelas Perdidas								
	0%	5%		10%		15%		20%	
	Completa	Perdas	Imputação	Perdas	Imputação	Perdas	Imputação	Perdas	Imputação
1	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
2	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
3	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001
4	0,004	0,005	0,005	0,005	0,005	0,006	0,007	0,008	0,007
5	0,010	0,011	0,011	0,012	0,012	0,013	0,012	0,014	0,013

Simulação VII ($\frac{S}{\sqrt{M}}$) - Poucas Observações e Muitos Tratamentos para Verificar o Poder do Teste

σ	Porcentagem de Parcelas Perdidas								
	0%	5%		10%		15%		20%	
	Completa	Perdas	Imputação	Perdas	Imputação	Perdas	Imputação	Perdas	Imputação
1	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
2	0,001	0,002	0,001	0,003	0,002	0,003	0,004	0,004	0,005
3	0,015	0,015	0,015	0,016	0,016	0,016	0,016	0,015	0,016
4	0,010	0,009	0,009	0,007	0,009	0,006	0,007	0,005	0,008
5	0,003	0,002	0,002	0,001	0,002	0,000	0,002	0,000	0,002

Simulação VIII ($\frac{S}{\sqrt{M}}$) - Muitas Observações e Muitos Tratamentos para Verificar o Poder do Teste

σ	Porcentagem de Parcelas Perdidas								
	0%	5%		10%		15%		20%	
	Completa	Perdas	Imputação	Perdas	Imputação	Perdas	Imputação	Perdas	Imputação
1	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
2	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
3	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
4	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
5	0,004	0,004	0,004	0,005	0,005	0,006	0,007	0,008	0,008

Fonte: Elaboração própria.