Complementos de Estatística
Instituto Superior Técnico
Abril 2022

Projeto Computacional

João Martinho (98855)
António Figueiras (99402)
Clara Pereira (99405)
Marta Sereno (99432)
Samuel Pearson (99441)
Tomás Juhos (99446)

Conteúdo

Exercício 1

1. a)

O objetivo deste exercício é, primeiramente, gerar números x, tal que $X \sim Multinomial(n, p_1, p_2, p_3)$, recorrendo a números binomiais. Para gerar os números binomiais da v.a. $X_i \sim Bin(n, p_i)$ vamos utilizar 3 algoritmos distintos. O primeiro utiliza uma soma de v.a's i.i.d. com $W \sim Ber(p)$ e está presente na função `binAlg1`. Este decorre da seguinte maneira:

1. Gerar n números independentes $u_1, ..., u_n$ da $U(0, 1)$
2. Efetuar a atribuição:

$$y_i = \begin{cases} 0, \text{ se } u_i > p \\ 1, \text{ se } u_i \leq p \end{cases} \quad i = 1, ..., n$$

3. $x = \sum_{i=1}^{n} y_i$

O segundo algoritmo utiliza duas variáveis aleatórias $U|(U \leq p)$ e $U|(U > p)$, onde $U \sim U(0, 1)$, e está presente na função `binAlg2`. Este decorre da seguinte forma:

1. Gerar um número u da $U(0, 1)$
2. Iniciar $k = 0$
3. $k = k + 1$
4. Se $u \leq p \implies \begin{cases} y_k = 1 \\ \text{Substituir u por } \frac{u}{p} \end{cases}$
5. Se $u > p \implies \begin{cases} y_k = 0 \\ \text{Substituir u por } \frac{u-p}{1-p} \end{cases}$
6. se $k = n$ retornar $x = \sum_{i=1}^{n} y_k$ e parar
7. Ir para o passo 3.

O terceiro algoritmo utiliza o facto de que se Y_i é uma sucessão de v.a's i.i.d. com $Y \sim Geo(p)$ e X é o menor inteiro tal que $\sum_{i=1}^{X+1} Y_i > n$ então $X \sim Bin(n, p)$. Este algoritmo está presente na função `binAlg3` e decorre da seguinte forma:

1. $i = 1$
2. $somayi = 0$
3. Gerar u_i da $U(0, 1)$
4. Obter y_i da $Geo(p)$ fazendo $y_i = \left[\frac{\log(u_i)}{\log(1-p)}\right] + 1$
5. $somayi = somayi + y_i$
6. $i = i + 1$
7. Se $somayi > n$ retornar $x = i - 2$ e parar

8. Ir para o passo 3.

Agora que já temos os algoritmos para gerar os números binomiais, resta-nos gerar os x's multinomiais. Para isso fizemos uma outra função, `multinomAlg`, que recebe não só n e p mas também uma função, `binfunc`, que vai servir para gerar os números binomiais necessários para a simulação de valores multinomiais. Isto evita que tenhamos de fazer outras 3 funções distintas para cada algoritmo presente no enunciado.
A simulação de x ocorre da seguinte forma:

1. Gerar x_1 de $X_1 \sim Bin(n, p_1)$
2. Gerar x_2 de $X_2 \sim Bin(n - x_1, \frac{p_2}{1-p_1})$
3. Gerar x_3 de $X_3 \sim Bin(n - x_1 - x_2, \frac{p_3}{1-p_1-p_2})$
4. Retornar $x = (x_1, x_2, x_3)$ e parar.

Para além dos 3 algoritmos anteriores utilizámos também o gerador da multinomial disponível no R, vamos chamar-lhe algoritmo 4.
Queremos agora gerar x_i para $i = 1, ..., m = 10000$, com $n = 10$ e $p = (0.2, 0.3, 0.5)$ e comparar os tempos das gerações. Os tempos de geração obtidos para os respetivos algoritmos foram:

(i) 0.1730399 segundos

(ii) 0.1641450 segundos

(iii) 0.2105019 segundos

(iv) 0.0279350 segundos

Assim podemos concluir que entre os 4 algoritmos, a geração incorporada no R é a mais rápida. Dos algoritmos por nós implementados 2^{o} parece ser ligeiramente mais rápido que os outros dois.

Vamos agora estimar, para as quatro alternativas de geração, a média e a matriz de covariâncias de X, assim como $P(X_1 = 0, X_2 = 4, X_3 = 6)$ e compará-las com os resultados teóricos.

Médias:

(i) (2.0142, 2.9928, 4.9930)

(ii) (1.9818, 3.0113, 5.0069)

(iii) (1.9956, 2.9987, 5.0057)

(iv) (1.9894, 2.9860, 5.0246)

Valor teórico: $E[X_j] = np_j$, logo $E[\mathbf{X}] = (2, 3, 5)$. Podemos então observar que o algoritmo (iv) e o algoritmo (ii) parecem afastar-se mais do valor teórico do que o algoritmo (i) e o algoritmo (iii), embora a diferença continue a ser relativamente pequena.

Matrizes de Covariância:

$$\hat{\mathbf{\Sigma}}_i = \begin{bmatrix} 1.6093593 & -0.612659 & -0.9967003 \\ -0.6126590 & 2.054354 & -1.4416946 \\ -0.9967003 & -1.441695 & 2.4383948 \end{bmatrix} \quad \hat{\mathbf{\Sigma}}_{ii} = \begin{bmatrix} 1.5982286 & -0.5700513 & -1.028177 \\ -0.5700513 & 2.0219745 & -1.451923 \\ -1.0281772 & -1.4519232 & 2.480100 \end{bmatrix}$$

$$\hat{\mathbf{\Sigma}}_{iii} = \begin{bmatrix} 1.5831390 & -0.6097667 & -0.9733723 \\ -0.6097667 & 2.0747058 & -1.4649391 \\ -0.9733723 & -1.4649391 & 2.438311 \end{bmatrix} \quad \hat{\mathbf{\Sigma}}_{iv} = \begin{bmatrix} 1.5776454 & -0.5649049 & -1.012741 \\ -0.5649049 & 2.0892129 & -1.524308 \\ -1.0127405 & -1.5243080 & 2.537049 \end{bmatrix}$$

Valores teóricos:

$$\mathbf{\Sigma} = \begin{bmatrix} 1.6 & -0.6 & -1.0 \\ -0.6 & 2.1 & -1.5 \\ -1.0 & -1.5 & 2.5 \end{bmatrix}$$

Podemos então concluir que o algoritmo (iii) é o que possui menores valores de covariância, menores que até o valor teórico, enquanto que o (i) e o (ii) parecem ser os que possuem maiores valores.

Vamos agora comparar, para os diferentes algoritmos, a probabilidade $P(X_1 = 0, X_2 = 4, X_3 = 6)$. O valor teórico é $\frac{10!}{0!4!6!}0.2^0 0.3^4 0.5^6 = 0.02657813$.
As estimativas obtidas foram:

(i) 0.02650000

(ii) 0.02690000

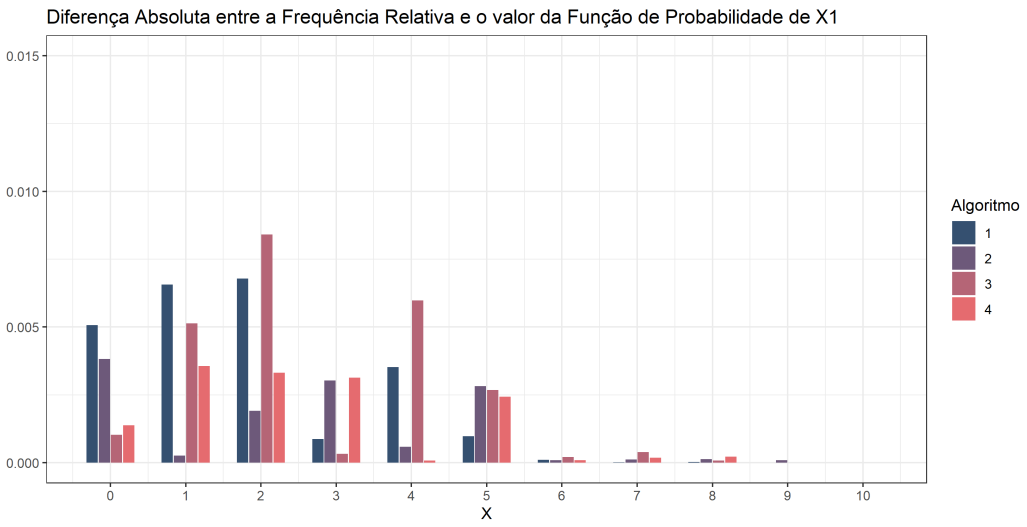
(iii) 0.02580000

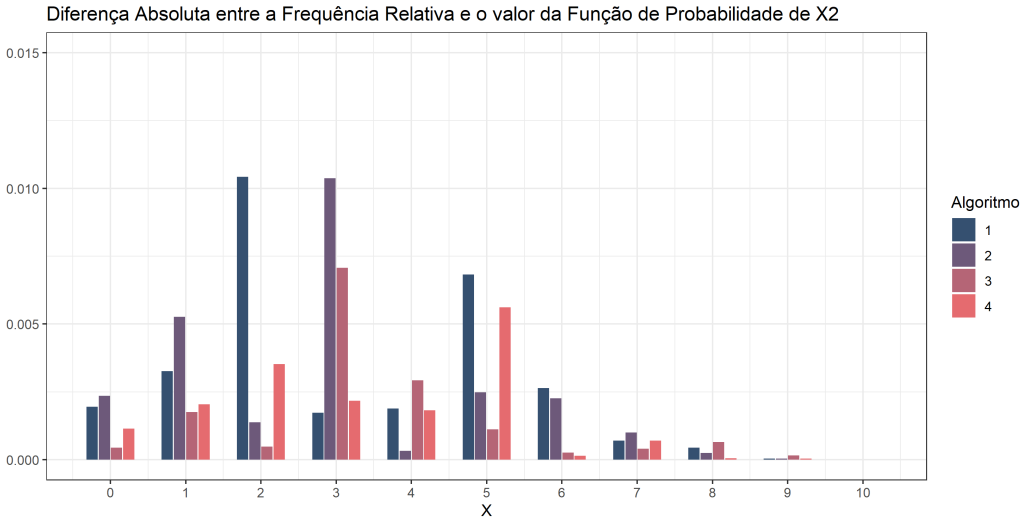
(iv) 0.02650000

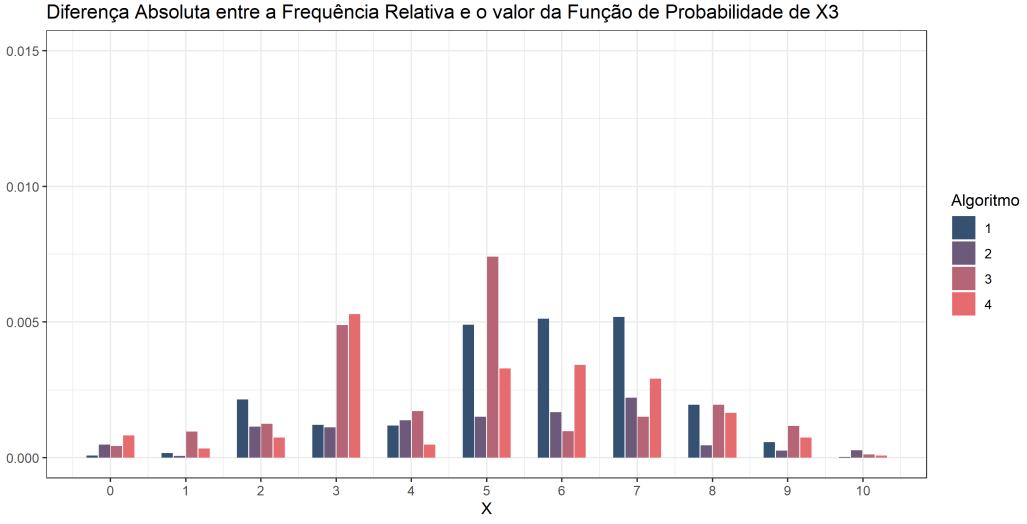
Assim, podemos concluir que o algoritmo (i) e o algoritmo (iv) são os que aproximam melhor a probabilidade ao valor teórico. É de notar que o algoritmo (iii) se afasta bastante deste mesmo valor teórico.

Queremos ainda, através de representações gráficas, avaliar qual dos quatro mecanismos de geração melhor se ajusta a esta distribuição multinomial. Para isso fizemos 3 gráficos de barras, um para cada binomial presente na distribuição multinomial. Neles estão representadas as diferenças absolutas entre as frequências relativas e os valores teóricos da função de probabilidade da respetiva binomial, para cada $x \in \{1, 2., ..., 10\}$.

Esta diferença permito-nos observar o quão desviada está a frequência relativa do pressuposto valor teórico, e por isso, quanto mais próxima a barra estiver do 0 melhor é. Se a barra estiver muito próxima de 0, para um certo x, então quer dizer que, para esse x, o mecanismo de geração se ajusta muito bem.
Segue-se então os três gráficos de barras, de X_1, X_2 e X_3, respetivamente (É de notar que a escala do eixo dos y's se manteve constante para uma melhor comparação):


Diferença Absoluta entre a Frequência Relativa e o valor da Função de Probabilidade de X1
0.015
0.010
0.005
0.000
0
1
2
3
4
5
6
7
8
9
10
X
Algoritmo
1
2
3
4


Diferença Absoluta entre a Frequência Relativa e o valor da Função de Probabilidade de X2
0.015
0.010
0.005
0.000
0
1
2
3
4
5
6
7
8
9
10
X
Algoritmo
1
2
3
4


Diferença Absoluta entre a Frequência Relativa e o valor da Função de Probabilidade de X3
0.015
0.010
0.005
0.000
0
1
2
3
4
5
6
7
8
9
10
X
Algoritmo
1
2
3
4

Após a visualização dos gráficos de barras podemos facilmente concluir que o algoritmo (i) parece ser o que se ajusta pior à distribuição binomial. Tanto o algoritmo (ii) como o (iv) se parecem ajustar bem à distribução binomial. O algoritmo (iii) parece ajustar-se um pouco melhor que o (i) mas pior do que o algoritmo (ii) e (iv). Com tudo em conta o algoritmo (ii) parece ser o que melhor se ajusta à multinomial.
Para fundamentar ainda mais as conclusões fizemos a soma das diferenças absolutas, ou seja, o comprimento total das barras e obtivemos os seguintes resultados:

(i) 0.07637011

(ii) 0.04913971

(iii) 0.06186382

(iv) 0.05141732

Assim tal como verificámos pela visualização dos gráficos, o algoritmo (ii) parece ajustar-se melhor à distribuição multinomial, seguindo-se do (iv) e por fim do (iii) e (i)

Por fim queremos concluir sobre qual foi o algoritmo mais eficiente para gerar x. Para concluirmos sobre a eficiência temos de ter em conta não só o custo computacional, mas também a precisão e o quão bem o algoritmo se ajusta à distribuição multinomial.
Assim, tendo em conta os resultados de tempos computacionais e as conclusões tiradas dos gráficos, podemos concluir que o algoritmo (ii) tende a ser o mais eficiente entre os 3 algoritmos por nós criados, já que é não só o algoritmo com menor tempo de geração, mas também o que se parece ajustar melhor à multinomial. É de notar no entanto que o algoritmo (iv) possui um tempo de geração muito menor a qualquer um dos outros 3 algoritmos sendo por isso uma melhor escolha para uma geração de uma grande quantidade de números da distribuição multinomial.
Assim, entre o algoritmo (ii) e o algoritmo (iv), este último tende a ser mais eficiente, já que possui um tempo de geração muito menor, o que é uma mais valia, mesmo sendo necessário abdicar de alguma precisão.

1. b)

Nesta alínea, temos como objetivo fazer uma estimativa pontual e uma intervalar, a aproximadamente 95%, de $\theta = P(\bar{X}_1 > 2)$, usando o estudo de uma simulação de Monte Carlo, sem redução de variância e com a técnica de redução de variância que utiliza a variável de controlo $U \sim U(0,1)$. As amostras da variável aleatória $X_1 \sim Bin(10, 0.2)$ foram todas geradas utilizando o **algoritmo (i)** da alínea anterior, pois concluiu-se que este era o que possuía menor variância na geração de X_1, dos três à nossa disposição. Para este estudo foram geradas $N = 10000$ amostras de dimensão $m = 100$.

(i) Sem redução de variância (Monte Carlo usual):

1. Geram-se as N amostras de dimensão m e é guardada a média de cada uma delas, originando $\bar{X}_1$;
2. É calculado Y, definido à custa de $\bar{X}_1$ através da função:

$$Y = I_{(\bar{X}_1 > 2)} = \begin{cases} 1, \text{ se } \bar{x}_{1i} > 2 \\ 0, \text{ c.c.} \end{cases} \quad i = 1, ..., N$$

3. É por fim feita a estimativa de θ:
   (a) Pontual: $\hat{\theta} = E(Y)$, sendo $E(Y)$ a média de Y.
   (b) Intervalar: É feita a aproximação de Y à distribuição $N \sim N(0,1)$, através do T.L.C., e é construído o intervalo de confiança com $1 - \alpha = 95\%$, para o valor de θ.

(ii) com a técnica de redução de variância que utiliza a variável de controlo $U \sim U(0,1)$:

1. Gera-se a Y da mesma maneira que no ponto anterior e gera-se também $U \sim U(0,1)$;
2. Calcula-se $\hat{c}^* = -\frac{\hat{cov}(Y,U)}{var(U)}$;
3. Geram-se novamente Y e U;
4. É definido T_c tal que $t_i = y_i + \hat{c}^*(u_i - E(U))$;
5. É por fim feita a estimativa de θ:
   (a) Pontual: $\hat{\theta} = E(T_c)$, sendo $E(T_c)$ a média de T_c.
   (b) Intervalar: É feita a aproximação de T_c à distribuição $N \sim N(0,1)$, através do T.L.C., e é construído o intervalo de confiança com $1 - \alpha = 95\%$, para o valor de θ.

Podemos então adiantar que seria de esperar, neste caso, que as estimativas pontuais de ambos os métodos fossem idênticas mas no caso das intervalares seria de esperar que a amplitude do intervalo de confiança para o valor de θ fosse menor no segundo método, por causa da redução de variância. Temos então os resultados:

(i) Sem redução de variância:

   (a) Estimativa pontual de θ: $\hat{\theta} = 0.4857$
   (b) Estimativa intervalar de θ: $(0.4759037, 0.4954963)$

(ii) Técnica de redução de variância aplicada:

   (a) Estimativa pontual de θ: $\hat{\theta} = 0.4840403$
   (b) Estimativa intervalar de θ: $(0.4742458, 0.4938347)$

Olhando para os resultados podemos verificar que a primeira hipótese se cumpre, sendo a estimativa pontual de θ idêntica nos dois casos. Mas a hipótese de que a amplitude diminui quando este método de redução de variância é aplicado não se cumpre, sendo os intervalos idênticos (a diferença das amplitudes é apenas 3.7×10^{-6}). Podemos então afirmar que o efeito deste método nas estimativas foi praticamente nulo. Se compararmos os valores da variância de T_c e de Y conseguimos perceber melhor o porquê destes resultados insatisfatórios, pois $\frac{\hat{var}(T_c)}{\hat{var}(Y)} = 0.9998249$, isto é, o valor da variância de Y praticamente não é afetado quando lhe é aplicado o método de redução deste caso. Isto, por fim, permite-nos apenas concluir que $U \sim U(0,1)$ não foi uma escolha acertada para variável de controlo.

Exercício 2

Tendo gerado um valor y de uma distribuição $Y \sim Gama(r, \frac{1-p}{p})$, sabemos que $(X|Y = y) \sim Poisson(y)$ é tal que $X \sim BinNeg(r, 1-p)$. Começamos então por implementar um algoritmo que, dado o parâmetro λ (neste caso $\lambda = y$), devolva x da distribuição $X \sim Poisson(\lambda)$. Criou-se então a função *gerapoisson*, que utiliza o seguinte método:

1. Gerar u da distribuição $Uniforme(0, 1)$
2. Fazer $i = 0$, $p = e^{-\lambda}$, $f = p$
3. Se $u < f$, fazer $x = i$ e parar
4. Se $u \geq f$, fazer $p = \frac{\lambda p}{i+1}$, $f = f + p$, $i = i + 1$
5. Ir para o passo 3

Sabendo que, dadas $Z_1, ..., Z_r$, variáveis tais que $Z_i \perp\!\!\!\perp Z_j, \forall i \neq j$ e $Z_i \overset{i.i.d}{\sim} Exp(\lambda)$, se tem $Y = \sum_{i=1}^{r} Z_i \sim Gama(r, \lambda)$ (neste caso, $\lambda = \frac{1-p}{p}$), concluimos que, para gerar y da distribuição Y, é necessário gerar $z_1, ..., z_r$ das distribuições $Z_1, ..., Z_r$, e calcular a respetiva soma: $y = \sum_{j=1}^{r} z_j$. Para gerar as distribuições Exponenciais, recorremos ao Método da Transformação Inversa:
Dado u da distribuição $F_Z(z) \sim Uniforme(0, 1)$,

$$F_Z(z) = 1 - e^{-\lambda z} \Leftrightarrow 1 - e^{\lambda z} = u \Leftrightarrow z = -\frac{ln(u)}{\lambda} \tag{1}$$

O algoritmo para gerar valores de uma Exponencial(λ) é muito simples:

1. Gerar u da distribuição $Uniforme(0, 1)$
2. Fazer $z = -\frac{ln(u)}{\lambda}$

Criou-se então uma função $BinNeg$ que recebe como argumentos a tripla (m, r, p), e que devolve um vetor com m valores aleatórios x de uma distribuição $X \sim BinNeg(r, 1-p)$.
Para gerar a amostra pelo algoritmo implementado avaliou-se a função nos parâmetros pedidos: $BinNeg(10000, 6, 0.7)$. A amostra pelo algoritmo do R obteve-se através da função $rnbinom(10000, 6, 0.3)$. Note-se que esta função recebe o parâmetro $q = 1 - p = 0.3$ em vez de $p = 0.7$ O algoritmo implementado é, no entanto, bastante mais lento do que a função *rnbinom* do R - o tempo de execução do primeiro é cerca de 100 vezes superior ao do segundo.
Para comparar as duas amostras, calcularam-se a média, mediana e a variância de ambas:

	Algoritmo Implementado	Algoritmo do R
Média	14.0489	13.9787
Mediana	13	13
Variância	46.13032	47.5406

Analisando os resultados, podemos concluir que o método implementado coincide aproximadamente com o algoritmo do R, que era o pretendido. Podemos também comparar os resultados "experimentais" com os valores teóricos para o valor esperado e variância.
É necessário, no entanto, deduzir tais valores, uma vez que a distribuição Binomial Negativa aceita 2 definições:

1. $Y \sim BinNeg(r,p)$ em que Y ="Número de provas de Bernoulli, com probabilidade de sucesso p, até se registarem r sucessos- esta é a definição que consta do formulário, e tem-se que $E(Y) = \frac{r}{p}$, $Var(Y) = \frac{r(1-p)}{p^2}$.

2. $X \sim BinNeg(r,p)$ em que X ="Número de insucessos em provas de Bernoulli, com probabilidade de sucesso p, até se registarem r sucessos- esta definição é utilizada pelo R e é a resultante do algoritmo implementado. Note-se que $X = Y - r$. Logo,

$$E(X) = E(Y - r) = E(Y) - r = \frac{r}{p} - r$$

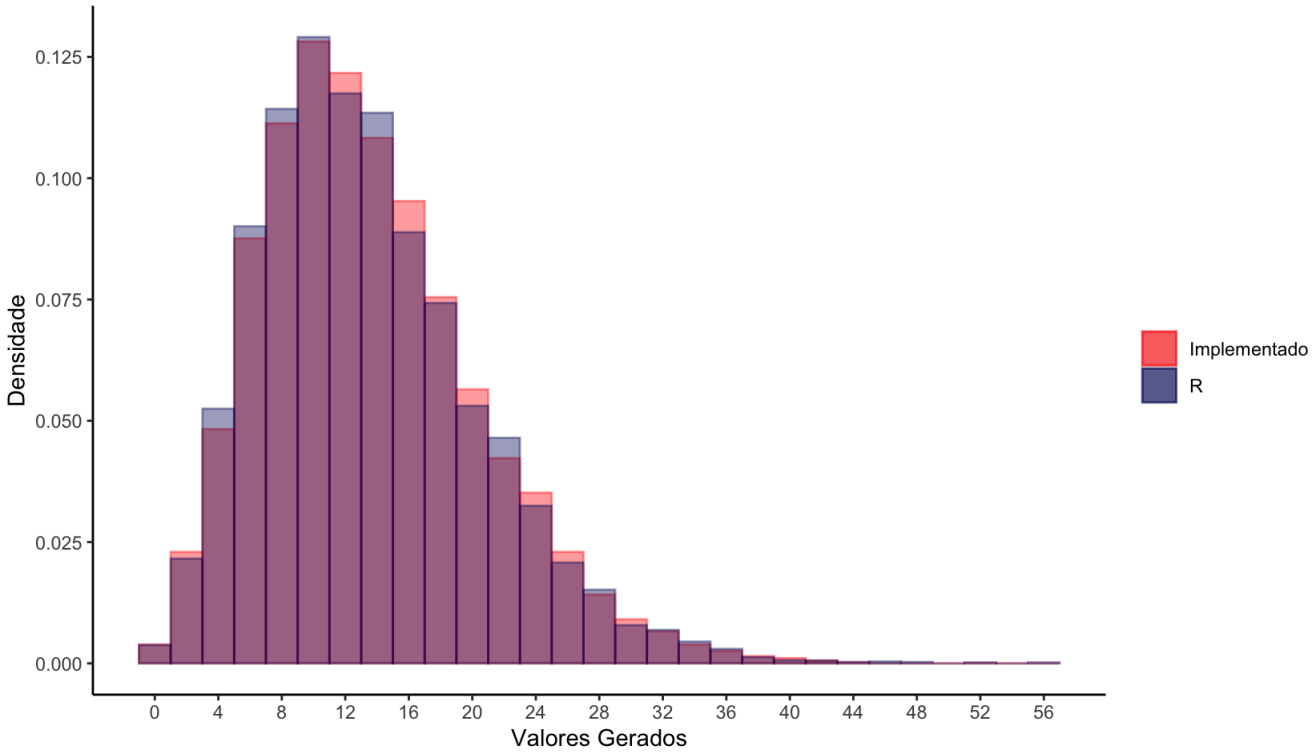
e

$$Var(X) = Var(Y - r) = Var(Y) = \frac{r(1-p)}{p^2}$$

.

Assim, com $X \sim BinNeg(6, 0.3)$, temos $E(X) = \frac{6(1-0.3)}{0.3^2} = 14$ e $Var(X) = \frac{6(0.7)}{(0.3)^2} = 46,67$. Conclui-se então que os valores gerados são concordantes com as previsões teóricas, como seria de esperar.

De seguida, analisaram-se os dados através da elaboração de um histograma. Tendo em conta os valores obtidos anteriormente, podemos esperar que os gráficos sejam semelhantes - não exatamente iguais, uma vez que não deixam de ser amostras aleatórias -, ou seja, que representem a mesma distribuição. Recorrendo ao comando *geom_histogram*, sobrepuseram-se os histogramas de frequência relativa das amostras, tendo-se obtido a seguinte representação:



Temos então a verificação gráfica que os métodos originam amostras com a mesma distribuição. Verifica-se também que a o valor esperado será ≈ 14, uma vez que os valores com maior densidade se concentram perto de 14.

Exercício 3

Queremos usar o Método da Transformação Inversa (MTI) para simular pares (x,y), ao invés do processo usual de simular apenas uma variável.
Para tal, efetuaram-se em primeiro lugar alguns cálculos:

$f_X(x) = \int_{-\infty}^{x} f_{X,Y}(x,y)dy = e^{-x} \cdot e^{-e^{-x}}$

$f_Y(y) = \int_{y}^{+\infty} f_{X,Y}(x,y)dx = e^{-2y} \cdot e^{-e^{-y}}$

$F_X(x) = \int_{-\infty}^{x} f_X(u)du = e^{-e^{-x}}$

$F_{Y|X=x}(y) = \int_{-\infty}^{y} f_{Y|X=x}(v)dv = \int_{-\infty}^{y} \frac{f_{X,Y}(x,v)}{f_X(x)}dv = e^{e^{-x}} \cdot e^{-e^{-y}}$

Notando que $f_X(x) \cdot f_Y(y) \neq f_{X,Y}(x,y)$, temos que as variáveis aleatórias X e Y não são independentes, pelo que a simulação de valores de x e y também não o deverá ser.

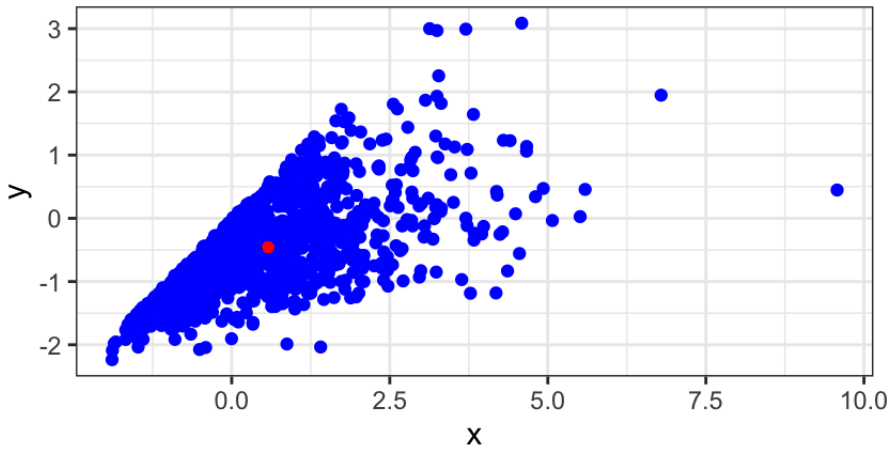
Recorrendo ao MTI, podemos simular um valor de x da seguinte forma:

1. Gerar um valor u da distribuição uniforme $U(0,1)$;
2. Fazer $F_X(x) = u \Leftrightarrow x = \log\left(\frac{1}{\log\left(\frac{1}{u}\right)}\right)$

Feito isto, e tendo em conta que $F_{Y|X=x}(y) = e^{e^{-x}} \cdot e^{-e^{-y}}$, podemos agora simular um valor de y para completar o par (x,y):

1. Gerar um valor u da distribuição uniforme $U(0,1)$;
2. Fazer $F_{Y|X=x}(y) = u \Leftrightarrow y = \log\left(\frac{1}{\log\left(\frac{e^{e^x}}{u}\right)}\right)$

Para simular (x_i, y_i) com $i = 1, ..., 1000$ basta repetir o processo 1000 vezes usando um ciclo for no R. Representação gráfica dos resultados da simulação:



Nota: o ponto a vermelho tem coordenadas correspondentes à média dos valores de x e de y simulados.

Os resultados estão em concordância com o domínio considerado, uma vez que os pontos obtidos se encontram abaixo da reta y=x. Era também de esperar que a média dos valores simulados estivesse próxima da origem, uma vez que $f_{X,Y}$ toma valores muito baixos quando x, y se afastam da origem.

Obtiveram-se então algumas medidas sumárias:
Média dos valores de, respetivamente, x e y: 0.5771973 e -0.4553313
Mediana de X e Y: $Me(X) = 0.3261034$ e $Me(Y) = -0.5325242$
Desvio padrão de X e Y: $DP(X) = 1.327354$ e $DP(Y) = 0.7820391$
Covariância: $Cov(X, Y) = 0.6504106$

Exercício 4

É-nos pedido para implementar um algoritmo com o método da aceitação-rejeição para gerar 10000 valores da variável aleatória $X \sim N(3, 2)$, tomando como candidatos valores de $Y \sim Exp(1)$. Para isso, dentro de um ciclo que corre até a amostra ter a dimensão desejada, começamos por gerar duas variáveis aleatórias uniformemente distribuídas entre 0 e 1, sendo que uma servirá para gerar $b \in Y$, outra para decidir se esta é aceite como valor de uma amostra da outra distribuição. Para realizar o método, as funções de densidade de probabilidade têm de ter o mesmo suporte, pelo que optámos

por um critério de aceitação que gera valores segundo uma distribuição normal (0,2), mas que são todos positivos, sendo que antes de os aceitar, alternamos o seu sinal entre ciclos, e somamos 3 para centrar a amostra no valor desejado. Ora, segundo o método da aceitação rejeição, para gerar valores que sigam uma distribuição G (neste caso N(0,2)) devemos gerar valores x segundo uma distribuição F (neste caso, Exp(1)) e $u \sim U(0,1)$, e aceitá-los se

$$u < \frac{F(x)}{G(x) * c}; c = \max\{\frac{F(y)}{G(y)}, y \in supp(F) = supp(G)\}$$

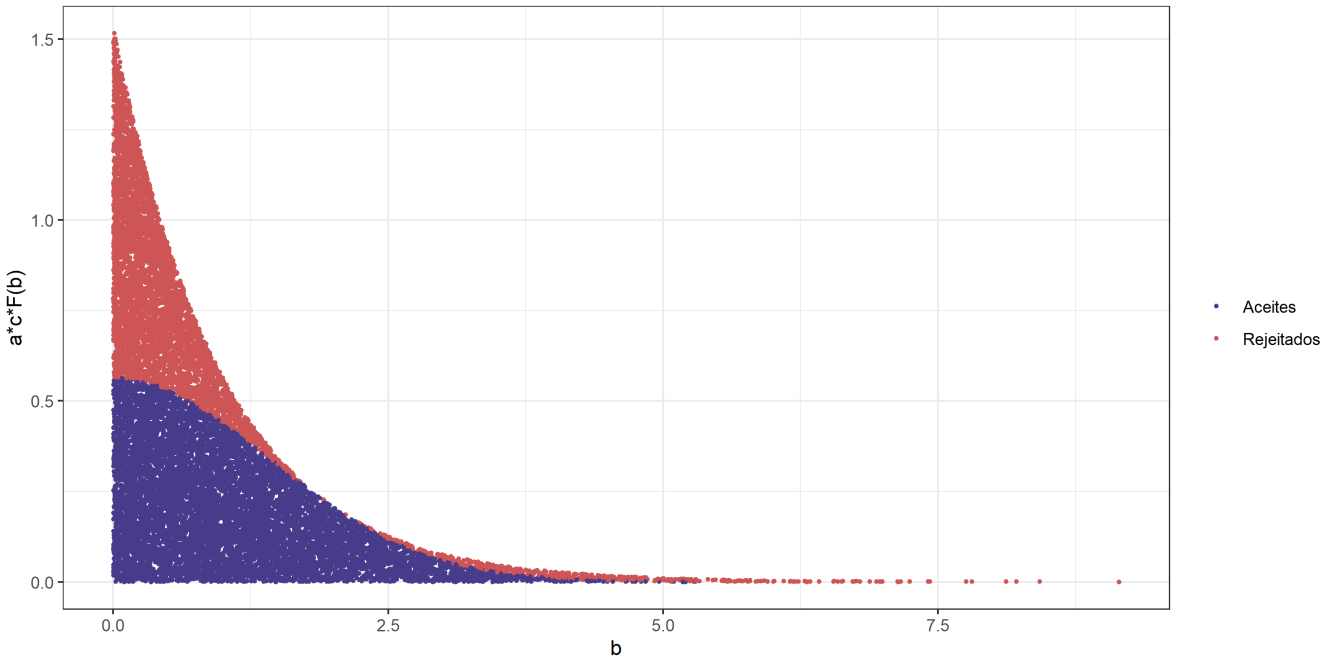
.

Neste caso,

$$c = \max\{\frac{\frac{2}{\sqrt{2 \cdot \pi \cdot 2}} e^{\frac{-x^2}{4}}}{e^{-x}}\} = \max\{\frac{2}{\sqrt{2 \cdot \pi \cdot 2}} e^{\frac{-x^2}{4}+x}\}$$

O máximo do expoente é atingido em $x = 2$, logo, $c = \frac{e}{\sqrt{\pi}} \approx 1.5336263$

Assim sendo, geramos $b \sim Exp(1)$, $a \sim U(0,1)$ e aceitamos b se $a < \frac{1}{1.5336263\sqrt{\pi}} e^{\frac{-b^2}{4}+b}$. Em caso afirmativo, adicionamos à amostra 3+b ou 3-b, alternando de iteração para iteração.
A probabilidade de um valor de b arbitrário ser aceite é $\frac{1}{c}$, logo, para gerar uma amostra de dimensão 10000, o ciclo será repetido, em média, 15336 vezes. No nosso programa, o ciclo corre 15266 vezes, ou seja, um erro de 0.456 %.
Através da biblioteca ggplot2, desenhámos num gráfico os pontos $(b, a \cdot F(b) \cdot c)$, a azul caso tenham sido aceites e a vermelho caso contrário. Obviamente que o bordo da curva a azul é a função de densidade de G(b) (se $a \cdot F(b) \cdot c < G(b)$ o valor é aceite) e o bordo da curva a vermelho é a função de densidade F(b) multiplicada por c ($a \cdot F(b) \cdot c < F(b) \cdot c$).



Em jeito de confirmação de resultados, calculámos a média e a variância da amostra obtida, sendo o valor da média muito próximo de 3 e o valor da variância muito próximo de 2, tal como esperávamos.

Exercício 5

Neste exercício, vamos tentar estimar $\mu = E(X)$ para uma variável aleatória $X \sim Exponencial(1)$ por uma simulação Monte Carlo usual, e em seguida com recurso a variáveis antitéticas, de modo a reduzir a variância da amostra.

Começamos pelo Método sem redução de variância:
Em cada amostra de dimensão n, $\mathbf{x} = (x_1, ..., x_n)$ temos que o estimador para o valor esperado é apenas $\hat{\mu}_n = \frac{1}{n}\sum_{i=1}^{n} x_i$. Assim, para cada n geramos $N = 10000$ amostras com n valores de x da distribuição $Exponencial(1)$, através do método da Transformação Inversa, cujo algoritmo foi deduzido em (1), e calculamos as respetivas médias.

Com uso de variáveis antitéticas:
O processo é bastante parecido ao anterior na medida em que utilizamos o método de Monte Carlo para estimar $\mu = E(X)$. Mas há uma diferença crucial: sempre que é gerada uma amostra $\mathbf{x}$ é gerada também uma outra amostra $\mathbf{y}$, identicamente distribuída e dependente de $\mathbf{x}$. As duas têm distribuição $Exponencial(1)$ e foram geradas através do método da Transformação Inversa. São então definidas as amostras $\mathbf{z}$ à custa das médias entre os valores de cada amostra $\mathbf{x}$ e $\mathbf{y}$.
Temos então o seguinte algoritmo:

Para i=1,..,.n:

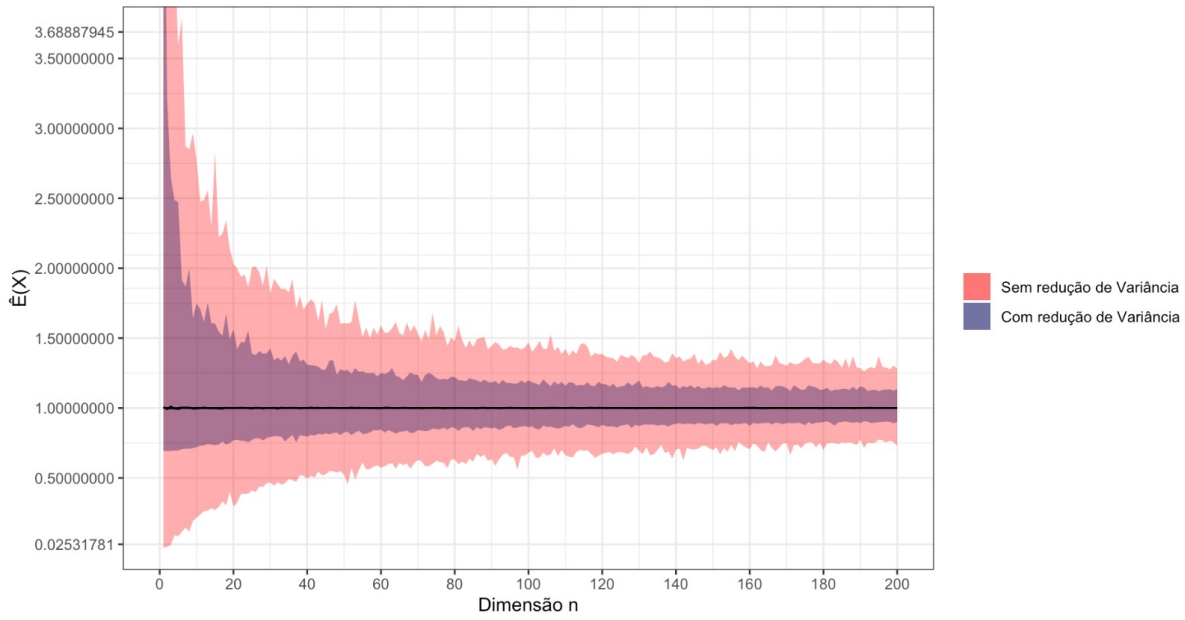
1. Gerar u da distribuição $Uniforme(0, 1)$
2. Fazer $x_i = -\frac{log(u)}{\lambda}$ e $y_i = -\frac{log(1-u)}{\lambda}$
3. Fazer $z_i = \frac{x+y}{2}$

Por fim, fazendo a média de cada amostra $\mathbf{z}$, temos então os nossos estimadores de $\mu = E(X)$: $\hat{\mu}_n = \frac{1}{n}\sum_{i=1}^{n} z_i$.

No gráfico abaixo temos então os resultados das nossas simulações, aprensentando no eixo das abcissas a dimensão n das amostras geradas e no eixo das ordenadas os valores esperados das amostras geradas. As duas faixas representam a distância entre os quantis 0.025 (fundo da faixa) e 0.975 (topo da faixa) da distribuição com os valores estimados de $\mu = E(X)$ das $N = 10000$ amostras geradas em ambos os casos. Desta forma não são representados valores muito discrepantes que estariam nas caudas dessa distribuição e poderiam levar a resultados indesejados.

Observando o gráfico podemos facilmente concluir que, à medida que a dimensão n da amostra aumenta, a amplitude de valores para o estimador $\hat{\mu}_n$ diminui, aproximando-se do valor esperado $\mu = 1$.
Podemos também concluir que este método de redução de variância está a ter o resultado desejado pois, apesar de ambas as faixas se aproximarem do valor teórico desejado $\mu = 1$ à medida que se aumenta a dimensão das amostras geradas, a faixa mais escura (que representa os casos onde este foi utilizado) está sempre bastante mais próximo desse valor, isto é, as amostras geradas com variáveis

antitéticas apresentam-se menos dispersas, e consequentemente a variância é menor. E é por isso mesmo que tal como referimos atrás, podemos concluir que a redução de variância com variáveis antitéticas tem o resultado desejado, neste caso.



Referências

[1] S. M. Ross (2009), *Introduction to Probability and Statistics for Engineers and Scientists,* Elsevier, Academic Press, 4th ed

[2] J. E. Gentle, (2003), *Random Number Generation and Monte Carlo Methods,* Springer, 2nd ed