Modelação Estocástica

Pedro Serrasqueiro
ISCTE-IUL

1º Semestre 2022/2023

Conteúdos

1 Introdução

Se é para amanhã
Deixa estar
Não faças hoje

- in *É p'ra amanhã...*
António Variações

O advento do computador, pelo menos na forma electrónica em que hoje o conhecemos, e que podemos situar na década de 1940, veio alterar o paradigma da ciência e potenciar um sem número de novas avenidas de investigação e descoberta científica.

Os humanos, de forma mais ou menos sofisticada, sempre se preocuparam com a incerteza associada aos acontecimentos futuros. Do século XVII em diante, a estatística, no sentido de campo do conhecimento dedicado à tentativa de quantificação dessa incerteza, sofreu uma grande evolução: foram teorizadas distribuições de probabilidade que reflectem aproximadamente muitos dos fenómenos que observamos, e criados métodos analíticos e geométricos que nos permitem, com um certo grau de confiança e/ou credibilidade, retirar conclusões a partir de amostras ou colecções de dados e generalizá-las para as populações em estudo. Citando alguns exemplos mais familiares, pensemos na formalização da distribuição normal, amplamente utilizada e atribuída a Galton e a Gauss; no teorema do limite central; no método dos mínimos quadrados ordinários, fundamental para a estimação de modelos de regressão linear (e nas suas derivações posteriores, como o método dos momentos generalizados, dos mínimos quadrados generalizados, etc.); na análise de variância (ANOVA), na análise de componentes principais e nos campos, mais abrangentes, da inteligência artificial, *machine learning*, etc.

A questão epistemológica de saber *o que podemos saber* a partir da observação da realidade, apesar de interessante, extravasa o âmbito deste texto. Independentemente das opiniões mais ou menos cépticas sobre a natureza do universo, precisamos, no dia-a-dia, de tomar decisões baseadas em dados e, em muitos casos, conseguimos fazê-lo de forma informada (sempre com algum risco de errar, mas que também se pode quantificar).

Os métodos de simulação, tornados em grande parte possíveis pelo extraordinário aumento da capacidade e velocidade de processamento dos computadores, permitem-nos ser, de certa forma, *preguiçosos*. Isto é, permitem-nos resolver alguns problemas utilizando *brute force*, em vez de procurar a solução analítica (*closed form*), ainda que ela possa existir. Por vezes, a solução analítica para um determinado problema pode ser muito, muito difícil ou, mesmo, não existir. Por isso, é necessário recorrer a métodos de simulação que nos permitam gerar um grande número de possibilidade/resultados, e, de um modo geral, avaliar o seu comportamento estatístico. Como exemplo da primeira situação, pensemos no Teorema do Limite Central (TLC), um resultado fundamental em estatística, já provado. O TLC, na sua forma mais simples, postula que, se retirarmos repetidamente amostras aleatórias de uma população, independentemente da forma como esta esteja distribuída, as médias amostrais (isto é, daquelas amostras) terão sempre um distribuição aproximadamente normal. Este resultado, se não tivesse sido formalizado (se não tivesse sido formulada uma "solução" analítica), poderia ser corroborado com um programa de computador relativamente simples (como **este**). Nesse sentido, estaríamos a ser *preguiçosos*, por não nos esforçarmos por encontrar uma solução analítica para o problema... Por outro lado, a evolução do valor de um portefólio de acções é um problema multidimensional muito complexo; está por inventar a bola de cristal que nos permita adivinhar o preço de uma acção no futuro, apesar das inúmeras tentativas. Neste caso, e muitas vezes para balizar o risco incorrido por determinado investidor, recorre-se à simulação de um grande número de possibilidades e à avaliação diferentes cenários futuros. Isto não resolve totalmente o problema, mas, em certa medida, ajuda. Aqui não estamos perante uma questão de preguiça, mas de impossibilidade (por enquanto).

Por tudo isto, neste texto abordaremos vários conceitos relacionados com a simulação em ciência de dados, que repartiremos em cinco blocos principais: no primeiro bloco, focar-nos-emos nos métodos de geração de números aleatórios, alicerces essenciais para qualquer simulação; no segundo bloco, introduziremos os conceitos de simulação de Monte Carlo e no terceiro, de Cadeias de Markov, particularizando alguns exemplos comummente utilizados em simulação; no quarto bloco analisamos os Métodos de Monte Carlo em Cadeias de Markov (MCMC), que fazem uso dos dois conceitos anteriores. Por fim, no quinto bloco, iremos abordar os chamados métodos de reamostragem, com particular ênfase no *bootstrap*, método muito útil para estimação de parâmetros quando estamos na presença de amostras de pequena dimensão, e/ou quando a distribuição da população não é conhecida.

Iremos recorrer sempre à linguagem de programação `R` para programação e execução dos exemplos apresentados.

Uma nota: os temas que iremos abordar enquadram-se no âmbito do que se chama *estatística computacional*. Isto significa que este corpo de conhecimento mistura conceitos de probabilidade e estatística com conceitos de programação de máquinas. Neste documento, equilibramos uma explicação intuitiva dos conceitos, sem prejuízo, onde aplicável, da sua formalização em notação aparentemente mais complexa, mas que tentamos descodificar. O texto não é um tratamento exaustivo dos temas a que alude, e pretende sobretudo servir como base para um estudo mais aprofundado.

2 Geração de números aleatórios

Comecemos com um exemplo: suponha que queremos observar 20 lançamentos da bola no jogo da roleta francesa cujos resultados, para o apostador, dependem do número saído (entre 0 e 36) após cada lançamento. Não é necessário ir ao casino, nem por capital em risco, para avaliar o resultado dos 20 lançamentos. É claro (infelizmente) que em conjuntos diferentes de 20 lançamentos obteremos sequências diferentes de números entre 0 e 36. Para gerar uma sequência hipotética (e aleatória) podemos recorrer ao `R`, utilizando a função `sample`:

```
sample(0:36, 20)
[1] 16 21  8 26 25  3 35 22 17  1 15 28 12 24 11 29 14 18 10  0
```

Uma grande parte da teoria estatística estabelecida assenta no conceito de *números aleatórios independentes e identicamente distribuídos* (ou i.i.d.). Aqui, *independente* significa que o próximo número saído não depende do anterior, e *identicamente distribuído* significa que provém de uma população cuja distribuição permanece fixa para todos os lançamentos. No exemplo da roleta, estamos perante a distribuição uniforme discreta: todos os números (o universo) têm a mesma probabilidade (daí o termo uniforme) de "sair", que, neste caso, é 1/37, a menos que a roleta esteja viciada.

Apesar do pressuposto de aleatoriadade ser fundamental em muitas aplicações, na realidade a geração de números aleatórios com computadores assenta em algoritmos determinísticos, isto é, em conjuntos de instruções definidas à partida que permitem "adivinhar" os próximos números. Ou seja, os números gerados não são *verdadeiramente aleatórios*, mas antes *pseudo-aleatórios*. Em seguida, aprofundamos um pouco mais sobre esta distinção, e sobre a razão pela qual a pseudo-aleatoriadade não invalida os resultados das nossas simulações.

2.1 Números verdadeiramente aleatórios

A obtenção de números verdadeiramente aleatórios assenta, em geral, na observação/medição de um processo físico específico, como sejam:

- O lançamento de um dado, e o registo do número na face voltada para cima após o lançamento
- O lançamento de uma moeda "ao ar", e o registo de "cara" (0) ou "coroa" (1)
- Medição do ruído atmosférico, utilizando um microfone e registando os decibéis assim detectados

Outro (interessante) exemplo é a utilização da posição da lava em *lava lamps* para efeitos de geração de chaves de encriptação (https://www.cloudflare.com/en-gb/learning/ssl/lava-lamp-encryption/).

Estes métodos, por assentarem em processos físicos, geram números totalmente imprevisíveis e são, por isso, úteis em aplicações onde essa imprevisibilidade é fundamental (como a encriptação de comunicações). No entanto, a obtenção de números aleatórios por estas vias apresenta algumas desvantagens para a maioria das aplicações em simulação, nomeadamente:

- São caros de obter (enviar um microfone para a atmosfera, pagar a alguém, ou comprar uma máquina, para lançar um dado o dia todo)
- A reprodutibilidade dos resultados não é garantida
- Frequentemente, a verdadeira aleatoriadade não é verificada

Sobretudo porque nos importa poder reproduzir as nossas experiências e simulações, nas mesmas condições, a utilização de números verdadeiramente aleatórios está de certo modo restringida a certos tipos de aplicações. Uma alternativa prática, como referimos anteriormente, é o recurso à geração de números pseudo-aleatórios, gerados por algoritmos que, ainda que determinísticos, nos dão uma 'ilusão" de aleatoriadade muito útil na prática.

2.2 Números pseudo-aleatórios

Os números pseudo-aleatórios são gerados a partir de "fórmulas" e de um valor inicial (a que chamamos *seed*). Isto significa que é gerado ou fornecido um valor inicial e que o próximo número é obtido a partir do(s) número(s) anteriores. Por isso, as instâncias geradas são previsíveis e, em dado momento, a sequência repetir-se-á. No entanto, é possível assegurar, na construção destas "fórmulas", que a repetição da sequência só ocorre após um número muito grande de iterações, o que nos dá a ilusão de aleatoriadade que, na prática, nos permite efectuar simulações tratando os números como se fossem verdadeiramente aleatórios.

Em seguida, iremos abordar a construção de diferentes tipos de geradores de números pseudo-aleatórios. Os métodos utilizados têm vantagens e desvantagens, consoante o problema de simulação em questão (por exemplo, se a distribuição da população é uma distribuição teórica conhecida ou não). De particular interesse é a geração de números pseudo-aleatórios obtidos de uma distribuição uniforme contínua com domínio em (0,1). O interesse desta distribuição é motivado por dois factores: por um lado, todos os números entre 0 e 1 são obtidos com igual probabilidade e, por outro, os números obtidos podem ser interpretados como probabilidades (no mínimo 0 - acontecimento impossível e, no máximo, 1 - acontecimento certo).

2.3 Números pseudo-aleatórios uniformes: o método congruencial linear

O Método Congruencial Linear (também conhecido por algoritmo de Lehmer) é um método de geração de números pseudo-aleatórios (inteiros) que assenta na ideia de congruência (como o nome indica) e numa função (linear) relativamente simples:

$$X_{n+1} = (a * X_n + c)\ mod\ m \tag{1}$$

A Equação 1 significa que o número seguinte da sequência (X_{n+1}) resulta:

1. Da multiplicação do número anterior (X_n) por uma constante a
2. Da soma de uma constante c ao resultado desta multiplicação (note-se que este passo e o anterior constituem uma função linear do tipo $y = mx + b$)
3. Do resto da divisão inteira (*mod*) de $(a * X_n + c)$ por m (a *congruência* resulta do facto de o próximo número resultar do resto desta divisão inteira)

Antes de elaborarmos um pouco mais sobre o método, vejamos um exemplo concreto , definindo $a = 8$, $c = 3$ e $m = 7$, e escolhendo arbitrariamente $X_0 = 0$ (o valor inicial é 0):

1. $X_0 = 0$
2. $X_1 = (8 * 0 + 3)\ mod\ 7 \equiv 3$
3. $X_2 = (8 * 3 + 3)\ mod\ 7 \equiv 6$ (o resto da divisão inteira de $(8 * 3 + 3) = 27$ por 7 é igual a 6, porque $7 * 3 = 21$ e $27 - 21 = 6$)
4. $X_3 = (8 * 6 + 3)\ mod\ 7 \equiv 2$
5. $X_4 = (8 * 2 + 3)\ mod\ 7 \equiv 5$
6. $X_5 = (8 * 5 + 3)\ mod\ 7 \equiv 1$
7. $X_6 = (8 * 1 + 3)\ mod\ 7 \equiv 4$
8. $X_7 = (8 * 4 + 3)\ mod\ 7 \equiv 0$

O que podemos concluir? O leitor mais atento terá reparado que o número de valores inteiros diferentes gerados (7) até a sequência se começar a repetir é igual ao divisor escolhido (7), e que os números gerados se situam no intervalo entre 0 e 6. A este número (de valores inteiros diferentes passíveis de serem gerados até à repetição da sequência) chamamos o *período máximo* (em inglês: *full period*) do gerador, e é igual ao

divisor escolhido. No nosso exemplo, só se produzem 7 valores (discretos) entre 0 e 6, inclusive. Deverá ser intuitivo pensar que o resto da divisão por 7 só poderá ser, no mínimo 0 (se o dividendo for um múltiplo de 7) e, no máximo 6, se o dividendo estiver à distância de -1 do 'próximo" múltiplo de 7.

O nosso exemplo é apenas ilustrativo, pois gera um número reduzido de valores diferentes e, por isso, seria pouco útil em qualquer aplicação prática. Interessa-nos, em geral, que a sequência contenha um grande número de instâncias diferentes até se repetir, e, por isso, existem algumas escolhas típicas para os parâmetros a, c, e m que asseguram essa necessidade. Deve notar-se, ainda, que para um gerador deste tipo ter *período máximo*, devem estar asseguradas as três condições seguintes, em simultâneo (caso contrário, o número de valores inteiros diferentes gerados será inferior ao divisor escolhido):

1. Os números a e m devem ser relativamente primos entre si (ou seja, o seu máximo divisor comum é 1)
2. $a - 1$ deverá ser divisível por todos os factores primos de m
3. $a - 1$ deverá ser múltiplo de 4 se m também for

No nosso exemplo:

1. $a = 8$ e $m = 7$ são relativamente primos entre si, pois não têm factores comuns à excepção do 1 (7 é primo, e 8 tem 4 e 2 como divisores)
2. $a - 1 = 7$ é divisível por todos os factores primos de $m = 7$
3. $m = 7$ não é múltiplo de 4, pelo que $a - 1 = 7$ não necessita de o ser

NOTAS:

1. Um exemplo semelhante, com um erro, está disponível no YouTube[1] - consegue detectar o erro?
2. Experimente construir um gerador com $a = 4$, $c = 3$ e $m = 8$, verifique as três condições acima e o resultado dos números gerados - o período deverá ser inferior ao período máximo
3. Neste link (Wikipedia: Linear Congruential Generator) poderá ver os parâmetros mais comuns utilizados nos diferentes LCG

Por fim, relembro-nos do que dissemos no fim da subsecção anterior: que é de particular interesse gerar números aleatórios entre 0 e 1, pois podem ser interpretados como probabilidades e usados como *input* para outros geradores de números não uniformes. Todavia, os números que gerámos com o LCG acima são exclusivamente inteiros. Para obter números (com igual probabilidade) no intervalo pretendido, através dos números inteiros gerados, basta-nos dividir o qualquer número resultante da Equação 1 pelo divisor escolhido (m), já que nenhum dos números gerados será maior que esse divisor:

$$u_n = \frac{x_n}{m} \tag{2}$$

Uma forma de fazer um LCG para números inteiros em `R` é a seguinte:

```
#### MÉTOGO CONGRUENCIAL LINEAR

lcm <- function(N, x0, a, c, m){

  x <- rep(0,N) # preenche o vector x com zeros

  x[1] <- x0 #coloca o valor inicial na primeira posição do vector

  for (i in 2:N) x[i] <- (a*x[i-1]+c) \%\% m # gera recursivamente os números seguintes
```

[1] https://www.youtube.com/watch?v=BAv3GKPZLTE

```
  x <- x/m

  return(x) # devolve o resultado

}
```

e os resultados podem ser obtidos correndo:

```
> lcm(N = 10, x0 = 0, a = 8, c = 3, m = 7) # chama a função, especificando os argumentos

[1] 0.0000000 0.4285714 0.8571429 0.2857143 0.7142857
0.1428571 0.5714286 0.0000000 0.4285714 0.8571429
```

2.4 Números pseudo-aleatórios não uniformes: o método da transformação inversa

Na secção anterior abordámos o método congruencial linear, com o objectivo de gerar números pseudo-aleatórios "retirados" de uma distribuição uniforme contínua, nomeadamente no intervalo (0,1). Nesta secção, iremos debruçar-nos sobre o método da **transformação inversa** como alternativa para gerar números não uniformes, i.e. obtidos a partir de distribuições de probabilidade em que cada número não tem igual probabilidade ocorrer.

Intuição

Antes de formalizarmos o método, vejamos a ideia que lhe subjaz, que é relativamente simples. O método da transformação inversa consiste em gerar aleatoriamente um valor de probabilidade acumulada (recorrendo à distribuição uniforme contínua entre 0 e 1) e "perguntar" à função de distribuição (CDF) respectiva qual é o valor, no eixo horizontal, que lhe corresponde, registando esse valor como o valor simulado. Repetindo esse processo n vezes, obtemos assim n valores simulados a partir da distribuição pretendida.

Suponha que determinada variável aleatória X segue uma distribuição normal, com média 0 e desvio-padrão 1, ou seja $X \sim N(0, 1)$. Suponha, também, que gerámos o conjunto A, de quatro valores aleatórios representando probabilidade acumulada em (0,1):

$$A = \{0.0668072, 0.6914625, 0.8413447, 0.9772499\} \tag{3}$$

Vamos "perguntar" à distribuição normal a quantos desvios-padrão acima (ou abaixo) da média corresponde a acumulação de cada uma das probabilidades aleatoriamente geradas. A estas probabilidades acumuladas (que são valores hipotéticos da função de distribuição) correspondem as abcissas -1.5, 0.5, 1 e 2, respectivamente, que seriam, neste caso, os nossos valores simulados (ver Figura 1).

Formalização

Para compreendermos melhor em que consiste o método da transformação inversa, vamos derivar o seu caso geral e ilustrar com um exemplo, recorrendo à distribuição exponencial. Antes da derivação, estabelecemos alguns conceitos de interesse:

- O objectivo deste método é gerar números aleatórios de uma qualquer distribuição genérica, seguida por uma determinada variável aleatória, i.e. $X \sim f_X(x)$
- Queremos fazer uso de números gerados aleatoriamente a partir da distribuição uniforme contínua (representando probabilidades acumuladas). Mais especificamente, queremos aplicar uma transformação à probabilidade acumulada que nos permita obter o quantil da distribuição objectivo (o número efectivamente simulado), i.e. $T(U) = X$

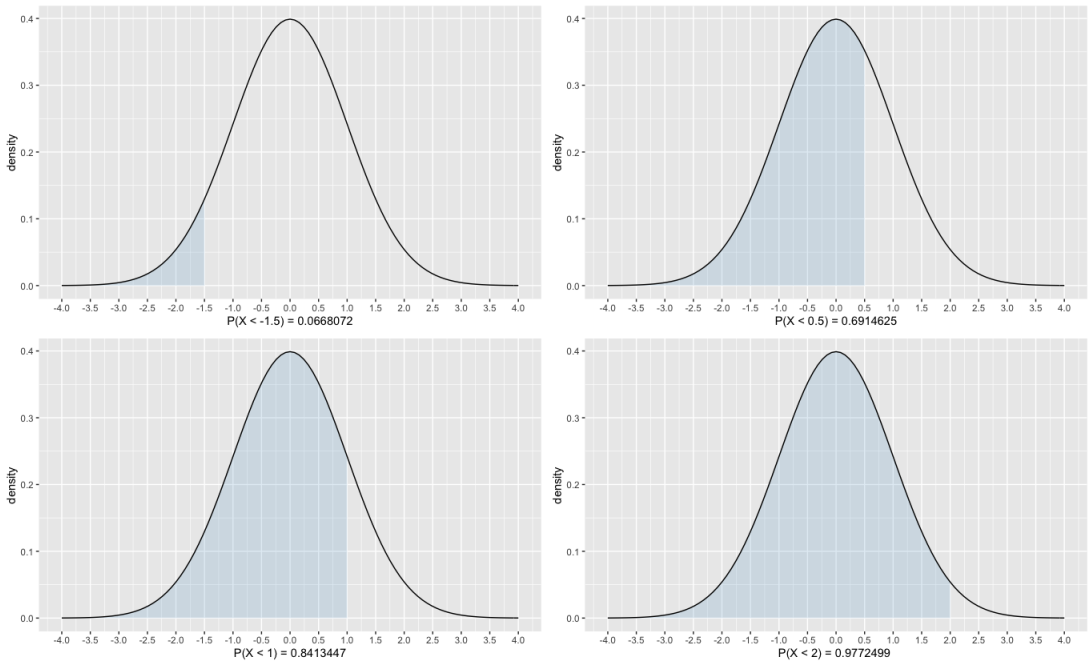

Figura 1: O método da transformação inversa: geramos aleatoriamente as áreas a azul, e procuramos no eixo horizontal o valor a que essa probabilidade acumulada corresponde. Assim, neste caso, teríamos simulado os números -1.5, 0.5, 1 e 2, a partir da distribuição normal $N \sim (0, 1)$

- Faremos uso do conceito de *função inversa*. A função inversa é uma função que, dado o output da função original, devolve o argumento (a "abcissa") da função. Por exemplo, se a função original for $y = 2x$, a sua função inversa será $x = y/2$. Em termos práticos, estamos a resolver a equação em ordem a x
- A função de distribuição (CDF) de uma variável aleatória X representa-se por $F_X(x)$ e denota a probabilidade acumulada até x, $F_X(x) = P[X \leq x]$, ou seja, a probabilidade de ocorrer x ou qualquer número inferior (no domínio de X)

Então, se $T(U) = X$:

$$F_X(x) = P[X \leq x] \iff F_X(x) = P[T(U) \leq x] = P[U \leq T^{-1}(x)] = T^{-1}(x) \iff T(x) = F_X^{-1}(x) \quad (4)$$

Em português, este resultado, que parece trivial, prova que a transformação que devemos aplicar ao número gerado aleatoriamente a partir da distribuição uniforme contínua é a função inversa da função de distribuição objectivo. Apesar de ser um resultado geral (aplica-se a qualquer distribuição, assumindo que é possível calcular a função inversa), vamos particularizar com um exemplo para entender melhor.

Suponhamos que determinada variável aleatória X segue uma distribuição exponencial, i.e. $X \sim Exp(\lambda)$. A distribuição exponencial tem funções densidade de probabilidade e de distribuição dadas por:

$$f(x) = e^{-\lambda x}, x \geq 0 \quad (5)$$

$$F_X(x) = 1 - e^{-\lambda x}, x \geq 0 \quad (6)$$

De acordo com o resultado da Equação 4, a transformação que devemos aplicar à probabilidade aleatoriamente gerada é a função inversa da CDF objectivo. Então:

$$F_X(x) = 1 - e^{-\lambda x} \iff 1 - F_X(x) = e^{-\lambda x} \iff ln(1 - F_X(x)) = -\lambda x \iff x = -\frac{ln(1 - F_X(x))}{\lambda} \quad (7)$$

Uma vez que $F_X(x)$ representa a acumulação de probabilidade (que geramos a partir da distribuição uniforme), podemos substituir esta expressão por U, de uniforme. Assim, temos finalmente:

$$F_X^{-1}(x) = -\frac{ln(1 - U)}{\lambda} \quad (8)$$

O que significa isto, na prática? Que se quisermos gerar um número aleatório proveniente de uma distribuição exponencial com parâmetro 5, i.e. $X \sim Exp(\lambda = 5)$, basta gerar uma aleatoriamente uma probabilidade acumulada, por exemplo $U = 0.5$, e proceder ao respectivo cálculo. Neste caso:

$$-ln(1 - 0.5)/5 = 0.138629436... \quad (9)$$

O método pode ser implementado para este exemplo correndo o código:

```
inv_exp <- function(lambda, n){

    sims <- numeric(n)  #vector com zeros para aceitar valores numéricos

    for (i in 1:n) sims[i] = -log(1-runif(1))/lambda #função inversa como no exemplo
    return(sims)  #devolver o resultado
}
```

E os resultados podem ser replicados para 3 simulações a partir de $Exp(5)$ (`set.seed(2022)`):

```
set.seed(2022)
> inv_poi(lambda = 5, n = 3)
[1] 0.33853962 0.20840443 0.02564137
```

No `R`, existem alternativas pré-programadas para a simulação a partir de várias distribuições teóricas conhecidas. No exemplo acima, as simulações podem ser obtidas utilizando a função `rexp`, onde `r` significa *random*. A sintaxe é semelhante para outras distribuições: `rnorm, rbinom, runif, rt, rchisq`, etc., sendo necessário parametrizar cada uma das funções com os respectivos argumentos. Note que, em muitas aplicações, a distribuição a partir da qual se pretende simular é específica ao problema em questão. O método da transformação inversa fornece um método simples para a obtenção de amostras caso a função de distribuição e a sua função inversa sejam matematicamente tratáveis.

Para finalizar, elencamos os passos do algoritmo:

1. Derivar a função inversa da função de distribuição objectivo (i.e. a partir da qual queremos simular), $F_X^{-1}(u)$
2. Programar um comando ou função para calcular $F_X^{-1}(u)$
3. Para cada número a simular
   (a) Gerar um número aleatório u proveniente da distribuição $Uniforme\,(0, 1)$
   (b) Produzir o resultado simulado $x = F_X^{-1}(u)$

2.5 Números pseudo-aleatórios não uniformes: o método da aceitação-rejeição

O método da transformação inversa, que abordámos na secção anterior, é muito útil para a geração de números aleatórios a partir de distribuições cuja função de distribuição é conhecida e facilmente invertível. No entanto, em algumas aplicações, deparamo-nos com distribuições cuja forma não se aproxima de qualquer distribuição teórica, ou com distribuições que, apesar de poderem ser bem descritas por uma qualquer função (e.g. polinomial), não são facilmente integráveis e/ou invertíveis. Por isso, a simulação pode não ser fácil ou possível através da transformação inversa. Nesses casos, poderá ser mais adequado utilizar o método da aceitação-rejeição (em inglês, *rejection sampling*), uma vez que este não exige o conhecimento, à priori, da CDF objectivo.

Suponhamos que a nossa variável de interesse, X, segue uma qualquer distribuição empírica $f(x)$, i.e. $X \sim f(x)$. O método da aceitação-rejeição consiste, em primeiro lugar, em escolher uma qualquer distribuição $g(x)$, que, quando multiplicada por uma constante c, é maior ou igual que a distribuição $f(x)$. Ou seja:

$$\frac{f(x)}{cg(x)} \leq 1 \tag{10}$$

A escolha de $g(x)$ é arbitrária (i.e. pode ser uma qualquer função densidade de probabilidade), mas deve ser tão semelhante quanto possível à função objectivo $f(x)$ para que a simulação seja eficiente. O passo seguinte é gerar um número aleatório da função proveniente da função $g(x)$, e computar o rácio na Equação 10. Como, por definição, $cg(x) \geq f(x)$, o resultado será sempre inferior a 1, permitindo que seja interpretado como uma frequência (i.e. probabilidade).

O último passo consiste em gerar aleatoriamente um número u proveniente da distribuição uniforme contínua, no intervalo [0,1], e compará-lo com o resultado da Equação 10. Se:

- $u < \frac{f(x)}{cg(x)}$, aceitamos (e guardamos) o valor simulado
- Caso contrário, rejeitamos a simulação e prosseguimos para a iteração seguinte

Em seguida, vamos analisar um exemplo concreto [2] e extrair daí a intuição que subjaz a este método.

Suponha que a população a partir da qual queremos gerar números aleatórios tem função de distribuição de probabilidade dada pelo polinómio $f(x)$, com domínio em (0,1), tal que:

$$f(x) = \frac{3}{2}x^3 + \frac{11}{8}x^2 + \frac{1}{6}x + \frac{1}{12}, x \in (0,1) \tag{11}$$

Suponha também que a função não é fácil de inverter. Assim sendo, iremos utilizar o método de aceitação-rejeição e escolhemos para $g(x)$, a distribuição uniforme contínua $U(0,1)$, uma vez que tem o mesmo domínio que $f(x)$ Analisemos, visualmente a nossa escolha:
A distribuição uniforme, escolhida para $g(x)$, para satisfazer a condição $cg(x) \geq f(x)$ deverá ser multiplicada pelo máximo de $f(x)$, ou seja, $f(1) = 3.125$, ou seja $c = 3.125$.
Que conclusão podemos tirar da Figura 3? Debrucemo-nos sobre duas instâncias específicas de x e comparêmo-las: $x = 0.7$ e $x = 0.9$.

- Quando $x = 0.7$, $f(x) = 1.38825$, e $f(x)/cg(x) = 0.44424$. Isto significa que, ao gerar aleatoriamente números u_i provenientes de $U(0,1)$, $x = 0.7$ será seleccionado cerca de 44% das vezes em que seja obtido.
- Quando $x = 0.9$, $f(x) = 2.440583$ e $f(x)/cg(x) = 0.7809867$. Isto significa que $x = 0.9$ será seleccionado cerca de 78% das vezes em que seja obtido.
- Em termos relativos, $x = 0.9$ será seleccionado com $0.7809867/0.44424 = 1.75$ vezes mais frequência do que $x = 0.7$.

[2] Adaptado de https://www.youtube.com/watch?v=kMb4JlvuGlw

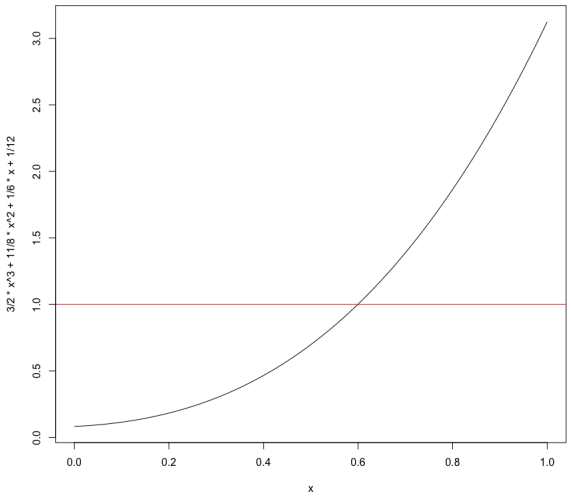

Figura 2: A função $f(x)$ é monótona crescente e tem o seu máximo absoluto em $f(1) = 3.125$

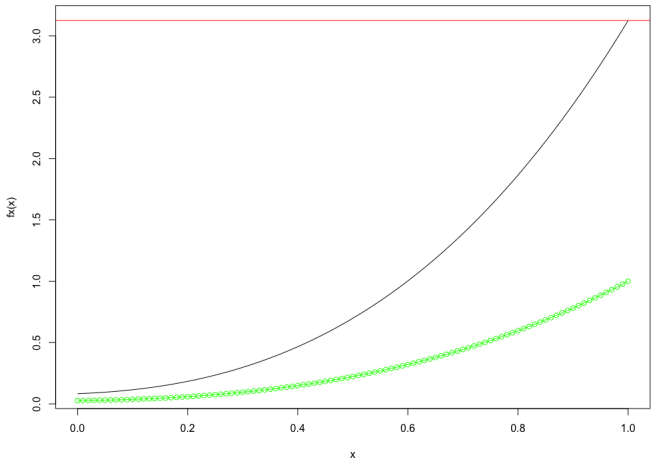

Figura 3: As funções $f(x)$ (a preto), $cg(x)$ (a vermelho), e $f(x)/cg(x)$ (a verde)

- Desta forma, parece razoável que este método, se repetido um grande número de vezes, para diferentes valores de u_i, gere a distribuição inicial $f(x)$, em que, por ser monótona crescente, os valores mais "baixos" ocorrem com menor probabilidade e vice-versa.

O método da aceitação-rejeição, para o nosso exemplo, pode ser implementado correndo o código:

```
#Método da Aceitação-rejeição

X <- runif(5000, 0, 1) #Gerar 5000 números da distribuição g(x) proposta
U <- runif(5000, 0, 1) #Gerar 5000 números da distribuição Uniforme(0,1)

#Nota: neste caso as duas distribuições coincidem

fx <- function(x){ #Programa a f.d.p f(x), a dist. da qual desejamos simular
  new_x <- 3/2*x^3 + 11/8*x^2 + 1/6*x + 1/12
  return(new_x)
}

count = 1 #Estabelece um contador para o número de iterações percorridas
accept = c() #Cria um vector vazio onde iremos guardar as simulações aceites

#Rodar o loop até 5000 iterações ou até obter 1000 números simulados
while(count <= 5000 & length(accept) < 1000){

  #Escolher o valor de u simulado acima em U, com posição correspondente ao número da iteração
  u_sim <- U[count]

  #Calcular o valor de cg(x)
  cg_x <- fx(X[count])/(3.125*dunif(X[count],0,1))

  #Comparar os dois valores
  if(u_sim <= cg_x){

    #Se o valor proveniente da U(0,1) for inferior a cg(x), então:

    #Aceitar a simulação acrescentando uma linha ao vector accept
    accept = rbind(accept, X[count])
    count = count + 1}

    #Caso contrário, descartar a simulação, aumentar o contador em 1, e seguir para
    a próxima iteração
    count = count + 1
  }}

  #Inspecção visual da amostra gerada
  require(ggplot2)
  ggplot(data.frame(accept), aes(x=accept)) +
  geom_histogram(aes(y=..density..), binwidth=0.19, color="black", fill="skyblue3", alpha = 0.3) +
  stat_function(fun = fx, col = "black")
```

A inspecção da figura 4 permite-nos concluir que é verosímil que as 1000 observações simuladas sejam provenientes da distribuição $f(x)$. Não devemos esperar que coincidam *exactamente*, uma vez que pode haver erro de amostragem; no entanto, quanto maior for o número de simulações, maior coincidência devemos esperar entre a distribuição teórica e a distribuição empírica. Devemos notar, também, que ao avaliar os resultados das simulações, não estamos limitados à inspecção visual: podemos aplicar testes estatísticos específicos (de

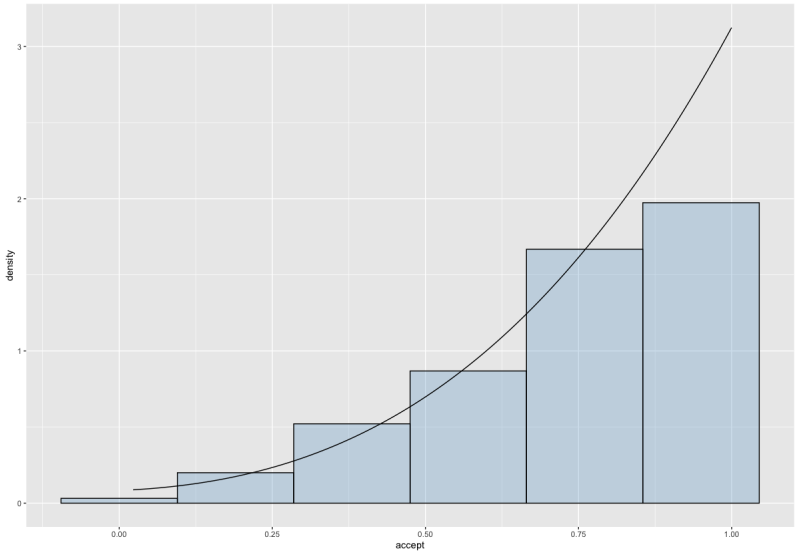

Figura 4: A azul, o histograma das 1000 observações simuladas (o vector `accept`) e, a preto, a função densidade $f(x)$

goodness-of-fit) que nos permitem concluir de forma mais credível acerca da proveniência da amostra.

Para concluir a nossa abordagem ao método da aceitação-rejeição, destacamos três pontos, mais formais, de interesse:

1. A probabilidade de aceitar um determinado número como válido para a simulação é igual a $f(X)/cg(X)$:

$$P(aceitar|X) = P(U < \frac{f(X)}{cg(X)}|X) = \frac{f(X)}{cg(X)} \tag{12}$$

   Para entender melhor, veja acima as conclusões que retirámos da Figura 3.

2. A probabilidade total de aceitação para qualquer iteração é:

$$\sum_x P(aceitar|x) * P(X = x) = \sum_x \frac{f(x)}{cg(x)} * g(x) = \frac{1}{c} \tag{13}$$

   A Equação 13 significa que, em média, para um determinado valor de x, são necessárias c iterações para que seja aceite/seleccionado. Daí decorre que o número de iterações até à aceitação de x tem distribuição geométrica (por memória, a distribuição geométrica dá-nos a probabilidade de um "sucesso" ocorrer após um determinado número de tentativas).

3. Recorremos ao Teorema de Bayes para provar que, ao recorrer ao método da aceitação-rejeição, estamos de facto a obter simulações provenientes da função pretendida. No caso discreto (o contínuo é análogo), para cada k tal que $f(k) > 0$,

$$P(k|aceite) = \frac{P(aceite|k) * g(k)}{P(aceite)} = \frac{[f(k)/cg(k)] * g(k)}{1/c} = f(k) \tag{14}$$

2.6 Números pseudo-aleatórios não uniformes: outras transformações

Em alguns casos, é possível aplicar transformações meramente aritméticas a variáveis aleatórias, que seguem uma determinada distribuição, de forma a simular outras variáveis que seguem uma distribuição diferente. Os casos clássicos são as distribuições derivadas da distribuição normal (qui-quadrado, t de Student e F de Snedecor). Pegando no primeiro caso, a título de exemplo, é um resultado amplamente utilizado e estabelecido que o quadrado de uma variável normal padrão (isto é, com média 0 e desvio-padrão 1) segue a distribuição do qui-quadrado com um grau de liberdade. Abaixo listam-se alguns exemplos de transformações possíveis e, em seguida, ilustram-se os casos 2 e 4:

1. Se $Z \sim N(0,1)$, então $V = Z^2 \sim \chi^2(1)$
2. Se $U \sim \chi^2(m)$ e $V \sim \chi^2(n)$ forem independentes, então $F = \frac{U/m}{V/n}$ tem distribuição F de Snedecor com (m, n) graus de liberdade
3. Se $Z \sim N(0,1)$ e $V \sim \chi^2(n)$ forem independentes, então $T = \frac{Z}{\sqrt{V/n}}$ tem distribuição t de Student com n graus de liberdade
4. Se $U \sim \text{Gamma}(r, \lambda)$ e $V \sim \text{Gamma}(s, \lambda)$ forem independentes, então $X = \frac{U}{U+V}$ tem distribuição $\text{Beta}(r, s)$

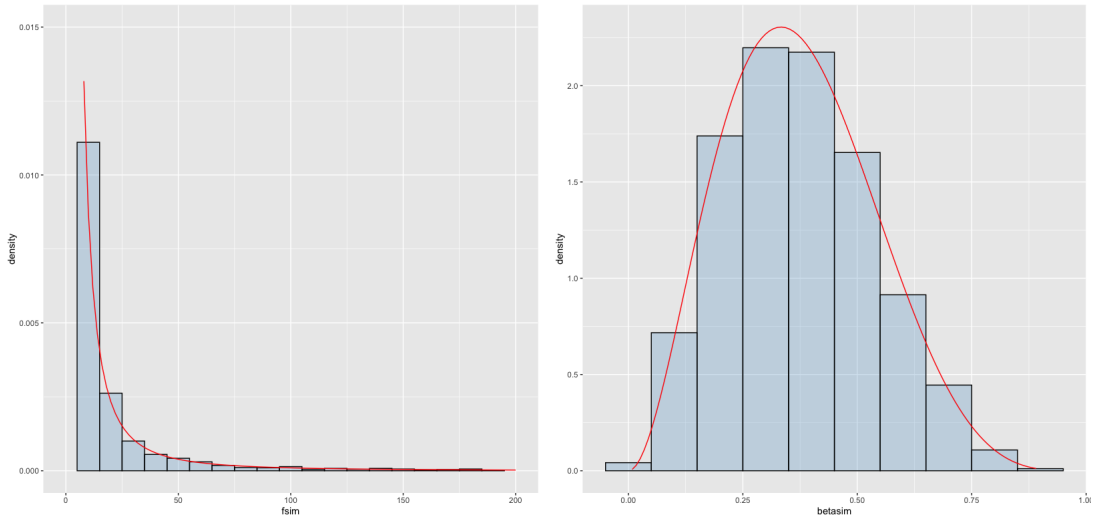

Figura 5: À esquerda, a distribuição dos valores simulados a partir de $F(5,2)$ e a respectiva distribuição teórica. À direita, o mesmo para a distribuição $Beta(3,5)$

A os gráficos exibidos na Figura 5 ilustram os casos 2 e 4 elencados acima. À esquerda, podemos observar a distribuição empírica do resultado da geração de 10.000 números aleatórios de duas distribuições do qui-quadrado, com 5 e 2 graus de liberdade, respectivamente. Aplicando a transformação descrita no caso 2, obtemos valores simulados a partir da distribuição $F(5,2)$. A linha a vermelho representa a distribuição teórica. Como na secção anterior, esta inspecção visual indica que é plausível que os números aleatórios tenham sido obtidos a partir da distribuição pretendida. O mesmo se aplica ao gráfico à direita, que ilustra o caso 4. Foram gerados 10.000 números aleatórios de cada uma das distribuições $\Gamma(3,2)$ e $\Gamma(5,2)$ e, aplicando a transformação respectiva, foram obtidos 10.000 números provenientes da distribuição $Beta(3,5)$.

As simulações (e os respectivos gráficos) podem ser obtidas correndo o código que apresentamos em seguida:

```
library(ggplot2)
```

```
#Geração 10.000 instâncias das distribuições do Qui-quadrado
a = rchisq(10000,5)
b = rchisq(10000,2)

#Aplicação da transformação pretendida, para obter a distribuição F
fsim <- (a/5)/(b/2)

#Geração do histograma das simulações e densidade teórica
ggplot(data.frame(fsim), aes(x=fsim)) +
  geom_histogram(aes(y=..density..),binwidth=10, color="black", fill="skyblue3", alpha = 0.3) +
  stat_function(fun = df, args = c(5,2), col = "red") +
  xlim(0,200) +
  ylim(0,0.015)

#Geração 10.000 instâncias das distribuições Gamma
c = rgamma(10000,3,2)
d = rgamma(10000,5,2)

#Aplicação da transformação pretendida, para obter a distribuição Beta
betasim <- c/(c+d)

#Geração do histograma das simulações e densidade teórica
ggplot(data.frame(betasim), aes(x=betasim)) +
  geom_histogram(aes(y=..density..),binwidth=0.1, color="black", fill="skyblue3", alpha = 0.3) +
  stat_function(fun = dbeta, args = c(3,5), col = "red")
```

Concluímos esta secção com duas notas relativas a transformações de variáveis aleatórias:

1. A geração de números aleatórios utilizando este tipo de transformações pode ser útil em casos em que não esteja disponível um gerador para a distribuição a partir da qual pretendemos simular, e é vantajosa quando as variáveis aleatórias que servem de base à transformação podem ser simuladas a partir de geradores já existentes. Nesse sentido, os casos 2 e 4, que usámos para ilustração do conceito, são também eles redundantes uma vez que o `R` dispõe de geradores já programados para as distribuições F e $Beta$.

2. Os quatro casos apresentados não são uma lista exaustiva das relações existentes entre distribuições de probabilidade (veja a Figura 6). A análise e descoberta destas relações é, em si, um campo fértil de investigação académica.

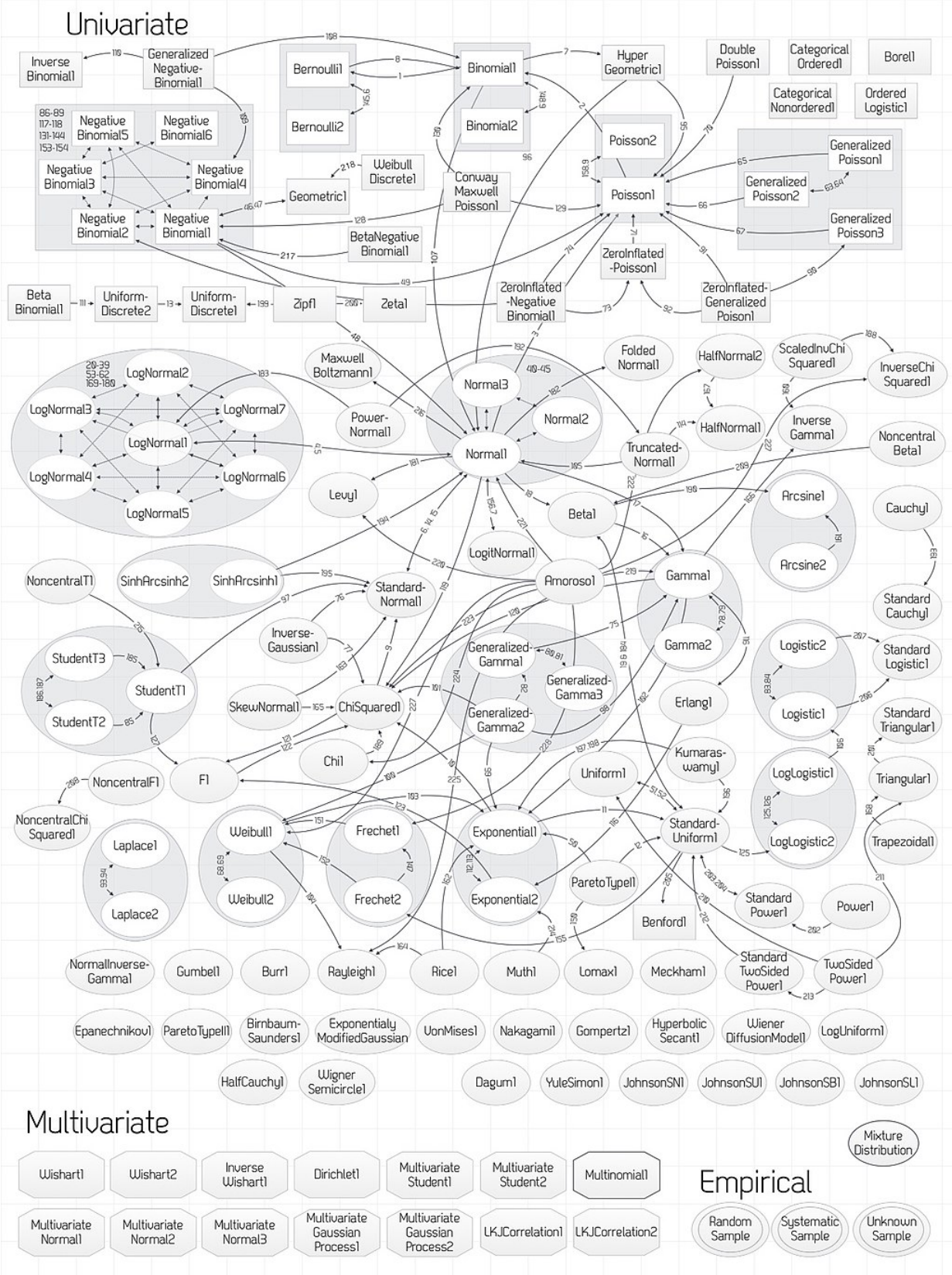

Figura 6: Relações entre distribuições de probabilidade. Obtido de Wikipedia: Relationships among probability distributions

2.7 Números pseudo-aleatórios não uniformes: somas e misturas de variáveis aleatórias

Nesta secção, iremos abordar os conceitos de somas e misturas de variáveis aleatórias, as suas distribuições de probabilidade e, por último, exemplificar a geração de números aleatórios a partir dessas distribuições. Para entendermos a utilidade destes conceitos, pensemos na razão (ou numa das razões) pela qual as distribuições de probabilidade nos são úteis no contexto da modelação estocástica.

Quando observamos um determinado fenómeno (por exemplo, o número de clientes que chegam a uma determinada loja, por hora), é razoável assumir que não saberemos quantos clientes chegarão na próxima hora. No entanto, se tivermos conhecimento (na maior parte das vezes, empírico) da distribuição de probabilidade inerente, podemos quantificar esta *incerteza*. Simplificando, pensemos que, se soubermos que, em média, chegam 3 clientes por hora à nossa loja, será improvável que, na próxima hora, vejamos 150 clientes entrar pela porta. Podemos, muitas vezes, calcular a probabilidade da chegada de 0, 1, 2, 3... clientes, o que poderá ser útil para, por exemplo, dimensionar a quantidade de recursos humanos afectos a determinado turno de trabalho. Generalizando, as distribuições de probabilidade são *modelos*, isto é, simplificações da realidade, que se esperam descrever aproximadamente essa realidade, com um grau de fiabilidade que nos permita tomar decisões úteis do ponto de vista prático.

Na literatura académica, existem centenas de distribuições teóricas conhecidas, que se aplicam a um número muito grande de contextos. O exemplo acima (a loja e a afluência de clientes), desde que assumidos alguns pressupostos, poderá ser bem descrito pela distribuição de Poisson; a altura de um ser humano escolhido ao acaso da população mundial poderá ser bem descrita pela distribuição normal, etc. No entanto, apesar de podermos dispor de um leque alargado de distribuições, é frequente depararmo-nos com situações que requerem uma maior flexibilidade do ponto de vista da sua modelação. É aqui que as somas e/ou misturas de variáveis aleatórias nos são particularmente úteis: em geral, permitem-nos conceptualizar distribuições alternativas que descrevem um determinado aspecto da realidade cuja descrição estaria, de outra forma, limitada pela "rigidez" das distribuições teóricas conhecidas.

2.7.1 Somas de variáveis aleatórias

Suponha que a empresa *Books Much?, Lda.* opera no sector livreiro (na venda de livros a retalho) e dispõe de três lojas na cidade de Lisboa. Suponha também que, como as lojas se situam em localizações diferentes, o número de clientes que entram, por hora, difere entre as lojas, mas é adequadamente descrita pela distribuição de $Poisson(\lambda_i)$, em que o parâmetro λ_i representa o número médio de clientes que entram, por hora, na loja i:

- Loja 1: $X_1 \sim Poisson(\lambda_1 = 3)$
- Loja 2: $X_2 \sim Poisson(\lambda_2 = 5)$
- Loja 3: $X_3 \sim Poisson(\lambda_3 = 7)$

Portanto, na Loja 1 entram, em média, 3 clientes por hora, de acordo com a distribuição de Poisson, sendo o mesmo valor 5 e 7 para as Lojas 2 e 3, respectivamente. Assim, modelar a afluência dos clientes permite-nos responder a questões como "*Qual é a probabilidade de, em determinada hora, entrarem 8 clientes na Loja 2?*". Em `R`, esta probabilidade é facilmente obtida recorrendo ao comando `dpois(x = 8, lambda = 5)` (0.065, aproximadamente).

Coloquemos, agora, uma questão distinta: "*Qual é a probabilidade de, em determinada hora, entrarem 13 clientes, no total, em todas as lojas?*". Repare que, para responder a esta questão, a variável aleatória de interesse é agora o número *total* de clientes por hora, ou seja, a soma do número de clientes que entram nas Lojas 1, 2 e 3, que podemos descrever como uma *soma de variáveis aleatórias*, $S = X_1 + X_2 + X_3$. A variável S tem a "sua" distribuição de probabilidade, que difere das três distribuições individuais. Neste caso, como se trata da soma de três variáveis que seguem distribuição de Poisson, a variável S seguirá também a distribuição de Poisson, com parâmetro λ resultante da soma dos λ_i individuais, ou seja $S \sim Poisson(\lambda = 15)$.

Deverá ser claro que, podendo dispor deste resultado teórico, a nossa questão é facilmente respondida calculando "directamente" a probabilidade pretendida ($P(S = 13) = 0.0956...$) utilizando o comando `dpois(x = 13, lambda = 15)`. Da mesma forma, podemos recorrer ao gerador do `R` para gerar aleatoriamente, por exemplo, dez instâncias da nossa variável S, que representam o número total de clientes chegados (às três lojas em conjunto) em dez horas distintas:

```
> rpois(10,15)
 [1]  7 19 13 10 12 17 16 14 18 14
```

Em seguida, ilustramos, no entanto, a geração de números aleatórios através da soma das variáveis aleatórias individualmente, i.e. como se não dispuséssemos do resultado teórico que supramencionado:

```
#Somas: exemplo Lojas

n <- 10000 #Para gerar 1000 números aleatórios

#Para gerar 1000 instâncias de cada loja, individualmente
set.seed(123)
X_1 <- rpois(n, 3)
X_2 <- rpois(n, 5)
X_3 <- rpois(n, 7)
X <- cbind(X_1, X_2, X_3)

#Para gerar a variável S, resultante da soma das 3 v.a.

S <- rowSums(X)
meanvar <- data.frame(mean(S))
colnames(meanvar) = "Mean"

#Gerar o histograma dos valores contidos no vector S
require(ggplot2)
ggplot(data.frame(S), aes(x = S)) +
  geom_histogram(aes(y = stat(count)/length(S)), binwidth=5, color="black", fill="skyblue3", alpha = 0.3
```

Para avaliar a plausibilidade de os valores gerados serem provenientes da distribuição pretendida, comparamos os quantis empíricos com os teóricos, assim como a média (que no caso da distribuição de Poisson é dada pelo parâmetro λ). Os resultados indicam que os valores provêm, de facto, de uma distribuição de Poisson com média 15:

```
> x <- c(0.25,0.5,0.75)

> summary(S)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   3.00   12.00   15.00   14.96   18.00   34.00

> qpois(x, 15)
[1] 12 15 18 #Os primeiro, segundo e terceiro quartis teóricos coincidem com os empíricos

> mean(S)
[1] 14.9567 #A média de S é aproximadamente 15 (a diferença deve-se ao erro de amostragem)
```

Antes de passarmos ao conceito de *mistura* de variáveis aleatórias, formalizamos em seguida o conceito de soma de variáveis aleatórias i.i.d.:

Definição 2.1 (Soma de v.a. aleatórias i.i.d.) *Se $X_1, X_2, ..., X_n$ forem n variáveis aleatórias i.i.d., então $S = X_1 + X_2 + ... + X_n$ chama-se a convolução de ordem n e tem função de distribuição $F_X^{*(n)}$.*

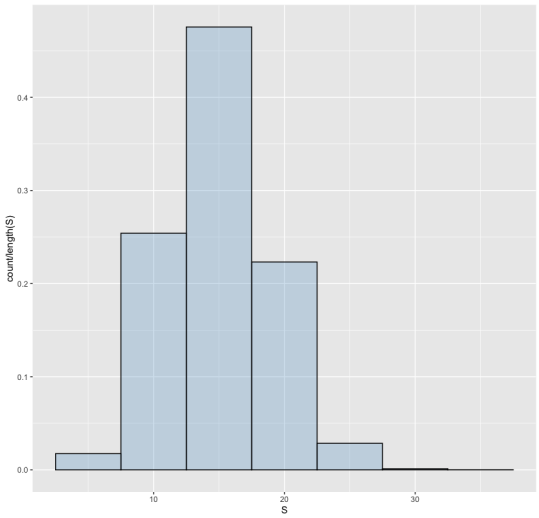

Figura 7: Distribuição empírica dos 10.000 valores gerados aleatoriamente através da soma $S = X_1 + X_2 + X_3$.

Notamos, ainda, que várias distribuições estão relacionadas por convolução, como no exemplo acima, em que a soma de variáveis que seguem distribuição de Poisson tem ela própria distribuição de Poisson. O mesmo se aplica à soma de variáveis i.i.d. que sigam distribuição Normal ou distribuição do qui-quadrado. Outro exemplo é a soma de variáveis aleatórias que sigam a distribuição exponencial. A convolução de n variáveis $Exp(\lambda)$ independentes segue a distribuição $Gamma(n, \lambda)$. Para finalizar, ressalvamos ainda que, em geral, quaisquer variáveis aleatórias podem ser somadas, independentemente da sua distribuição, e que essa soma não tem necessariamente de seguir uma distribuição teórica conhecida.

2.7.2 Misturas de variáveis aleatórias

Passamos, agora, ao conceito de **mistura** de variáveis aleatórias e às distribuições de probabilidade que decorrem dessas misturas. Como o nome indica, uma mistura implica, pelo menos, a existência de duas variáveis aleatórias (a que chamamos as *componentes*) que podem ou não seguir uma distribuição teórica conhecida. As misturas são particularmente úteis em contextos que envolvem a descrição de uma quantidade (e.g. peso, altura ou volume) para uma população geral, a qual é passível de ser desagregada em subgrupos para os quais a quantidade de interesse pode ser calculada de forma separada, por ser "diferente" nesses subgrupos. De uma forma simples, a geração de números aleatórios a partir da distribuição resultante da mistura pode ser descrita como uma "situação" em que, às vezes, retiramos elementos de um subgrupo e, outras vezes, retiramos elementos do outro subgrupo. Vejamos um exemplo (com três subgrupos), antes de formalizar o conceito:

Suponha que a observação dos veículos que circulam numa determinada autoestrada permitiu à concessionária (a empresa *Caminho das Pedras, S.A.*) concluir que 1/3 dos veículos são ligeiros de passageiros, 1/3 dos veículos são ligeiros de mercadorias e 1/3 do veículos são motociclos. Adicionalmente, suponha que se sabe que o *peso*, em kg, de cada um dos tipos de veículo segue as seguintes distribuições:

- $X_1 \sim N(\mu = 1500, \sigma = 150)$ [Passageiros]
- $X_2 \sim N(\mu = 2500, \sigma = 200)$ [Mercadorias]
- $X_3 \sim N(\mu = 350, \sigma = 100)$ [Motociclos]

Suponha, agora, que irá escolher 1.000 veículos (a circular na autoestrada) ao acaso: é natural que a amostra contenha, aproximadamente, 333 veículos de cada tipo. A distribuição do peso desses veículos, por tipo e

que circulam na autoestrada é dada pela *mistura* das distribuições de X_1, X_2 e X_3. O resultado da simulação é ilustrado no gráfico à esquerda na Figura 8.

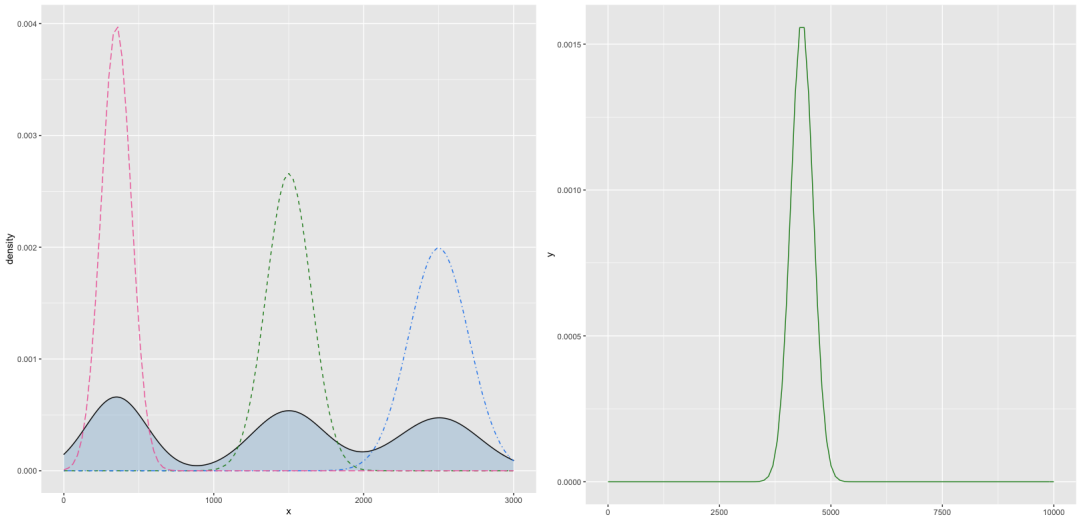

Figura 8: À esquerda: Distribuição do peso dos veículos que circulam na autoestrada (a mistura de X_1, X_2 e X_3, a preto) e das componentes individuais: X_1 (verde), X_2 (azul) e X_3 (rosa). À direita: Distribuição do peso total dos veículos que circulam na autoestrada, ou seja, da convolução $S = X_1 + X_2 + X_3$

Antes de formalizarmos o conceito de mistura de variáveis aleatórias, importa distingui-lo do conceito de soma. No nosso exemplo, a densidade resultante da mistura (que é a média ponderada das densidades das diferentes componentes) permite-nos calcular a probabilidade de, escolhendo um veículo ao acaso, o peso deste se situar, entre, por exemplo, 1000 e 2000kg. A densidade resultante da convolução (que resulta da soma das três variáveis aleatórias), por outro lado, permite-nos calcular a probabilidade de, seleccionando ao acaso um ligeiro de passageiros, um ligeiro de mercadorias e um motociclo, o seu peso conjunto se situar, por exemplo, naquele mesmo intervalo (esta distribuição é ilustrada pelo gráfico à direita na Figura 8).

A mistura de variáveis aleatórias pode ser expressa em termos da sua função de probabilidade (massa ou densidade, consoante as v.a. sejam discretas ou contínuas), $f(x)$ ou em termos da sua função de distribuição $F_X(x)$:

$$f(x) = \sum_{j=1}^{n} \theta_j f_j(x), x > 0 \tag{15}$$

ou

$$F_X = \sum_{j=1}^{n} \theta_j F_{X_j} \tag{16}$$

em que $f_j(x)$ é a função densidade da componente j e θ_j é o peso atribuído à componente j (no nosso exemplo, $\theta_j = \frac{1}{3}, j = 1, 2, 3$).

Sem perda de generalidade, formalizamos em seguida o algoritmo para geração de números aleatórios a partir de uma mistura de três componentes:

1. Gerar um número inteiro $k \in 1, 2, 3$, onde $\theta_j = P(K = k), j = 1, 2, 3$
   Se $K = 1$, gerar um número aleatório a partir de $f_1(x)$
   Se $K = 2$, gerar um número aleatório a partir de $f_2(x)$
   Se $K = 3$, gerar um número aleatório a partir de $f_3(x)$

2. Repetir os passos anteriores n vezes, para geração de uma amostra de tamanho n, guardando o resultado de cada iteração

Em R, a mistura utilizada no exemplo ilustrativo pode ser gerada correndo o seguinte código:

```
##Misturas: exemplo dos veículos

set.seed(123)
n <- 1000 #Dimensão da amostra

#Componentes

  #Pesos
  p <- c(1/3, 1/3, 1/3)

  #Média e desvio-padrão
  means  <- c(1500, 2500, 350)
  stdevs <- c(150, 200, 25)

#Geração da amostra
k <- sample(1:3, size = n, replace = TRUE, prob = p)
mean <-  means[k]
stdev <- stdevs[k]
x <- rnorm(n, mean = mean, sd = stdev)

#Output gráfico

ggplot(data.frame(x), aes(x=x)) +
  geom_density(aes(y=..density..), fill = "skyblue3", alpha = 0.3) +
  stat_function(fun = dnorm, args = c(1500,150), col = "forestgreen", linetype = "dashed") +
  stat_function(fun = dnorm, args = c(2500,200), col = "dodgerblue2", linetype = "dotdash") +
  stat_function(fun = dnorm, args = c(350, 100), col = "hotpink2", linetype = "longdash") +
xlim(0,3000)
```

3 Simulação de Monte Carlo

No capítulo anterior, abordámos vários métodos de geração de números pseudo-aleatórios. A geração destes números, a partir de diferentes distribuições de probabilidade (conhecidas ou não) é a pedra basilar de muitos dos problemas que enfrentamos em Ciência de Dados. Nomeadamente, a resolução de problemas determinísticos - para os quais exista uma solução analítica que não seja conhecida ou seja difícil de obter - pode ser tentada recorrendo a métodos a que chamamos *numéricos* ou, tipicamente, de *Simulação de Monte Carlo*.

Para entendermos em que consiste esta *classe* de métodos, vejamos antes um pouco de história:

Apesar de terem existido formulações anteriores do método de Monte Carlo, a sua invenção é atribuída ao cientista americano (de ascendência polaca) Stanislaw Ulam, no final dos anos 1940, enquanto investigava a difusão de neutrões no núcleo de uma arma nuclear (Ulam também fez parte do Projecto Manhattan, o famoso projecto desenvolvido durante a Segunda Guerra Mundial, dedicado à criação da bomba atómica). Curiosamente, a sua inspiração parece não ter advindo da física nuclear, mas antes de um conhecido jogo de cartas (muitas vezes associado à procrastinação...): o *Solitário*. Quem já tenha procrastinado um pouco com este jogo, sabe que este nem sempre tem solução (nem sempre se conseguem "arrumar" as cartas todas). O problema que Ulam se colocou foi o de calcular a probabilidade (portanto, *a priori*) de um jogo de Solitário, com um baralho de 52 cartas, poder ser concluído. Após as suas tentativas de encontrar uma solução analítica, infrutíferas segundo o próprio, terá ocorrido a Ulam que, simulando um grande número de jogos e, desse total, contando aqueles que se houvessem concluído com sucesso, poder-se-ia calcular aproximadamente a probabilidade pretendida. Rapidamente lhe terá ocorrido que este método poderia ser aplicado no trabalho que estava a desenvolver em *Los Alamos*, e transmitiu a sua ideia a John von Neumann (cientista americano de ascendência húngara). Como o trabalho de ambos era secreto, e, como quase todos os projectos secretos, precisava de um nome mais ou menos humorístico, decidiram chamar-lhe *Monte Carlo*, uma vez que o tio de Ulam, um jogador prolífico, tinha o hábito de pedir dinheiro emprestado a familiares para jogar no casino homónimo, situado no Mónaco.

A história da origem dos métodos de simulação de Monte Carlo ilustra bem o conceito que lhe subjaz: em vez de procurarmos uma solução analítica para resolver um problema, podemos, se conhecermos (ou pudermos supor) os *inputs* do modelo, simular um grande número de amostras que nos permite fazer análises qualitativas, descritivas e/ou calcular quantidades amostrais de interesse (*estimativas*, como a média, por exemplo), que se esperam ser próximas dos valores populacionais. Estas estimativas permitem-nos retirar conclusões práticas num vasto número de aplicações a processos reais (e.g. simulação de eventos discretos, Cadeias de Markov).

Este capítulo encontra-se organizado da seguinte forma: começaremos por uma aplicação simples destes métodos de simulação, recorrendo a um exemplo geométrico e, em seguida, abordaremos os conceitos de *erro padrão* e *erro quadrático médio*, por serem particularmente úteis na avaliação do comportamento de estimadores.

3.1 Um exemplo geométrico

Considere um círculo de raio 1, contido num quadrado de lado 2, conforme exibido na Figura 9. Suponha que vai lançar dardos, de forma aleatória, e que é garantido que esses dardos aterram sempre dentro do quadrado. A pergunta a que tentaremos responder, de forma numérica (simulada), é a seguinte:

Qual é a probabilidade de, lançando um dardo aleatoriamente, este aterrar no interior do ***círculo?***

Deverá ser intuitivo para o leitor que, uma vez que todos os lançamentos estão contidos no quadrado, basta dividir a área do círculo pela área do quadrado para obter a probabilidade pretendida:

- Evento Z: Dardo lançado aleatoriamente aterra no interior do círculo
- Área do círculo: $A_c = \pi r^2 = \pi * 1 * 1 = \pi$
- Área do quadrado: $A_q = c * l = 2 * 2 = 4$

De onde resulta que:

$$P[Z] = \frac{A_c}{A_q} = \frac{\pi}{4} \tag{17}$$

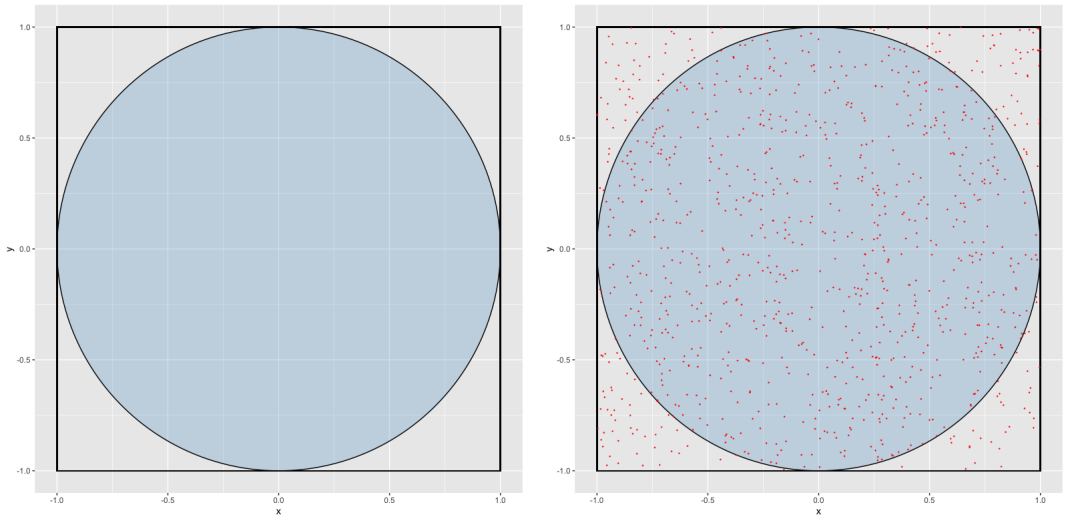

Figura 9: À esquerda: o "alvo", i.e. o quadrado de lado 2 contendo um círculo com o mesmo diâmetro. À direita, simulação de mil dardos lançados aleatoriamente. Espera-se que aproximadamente $\frac{1000\pi}{4} \sim 785$ dardos estejam contidos dentro do círculo.

Supondo que o resultado para $P[Z]$ não era conhecido, ou seja, a probabilidade de um dardo lançado aleatoriamente aterrar no interior do círculo, podemos recorrer ao método de Monte Carlo para tentar descobrir a solução:

1. Geramos aleatoriamente 1000 dardos lançados para o interior do quadrado (pretende-se um número grande de simulações)
2. Contamos o número de dados que aterraram dentro do círculo
3. Dividimos o número obtido no ponto anterior por 1000 (o total)

A simulação exibida na figura 9 conta com 812 dardos contidos no interior do círculo, o que equivale a uma proporção de 0.812. Este valor está relativamente próximo do que sabemos ser o verdadeiro valor: $\pi/4 \sim 0.7853982....$

Este exemplo simples permite-nos entender o conceito de simulação de Monte Carlo. Ao não conhecer o valor verdadeiro de uma quantidade, podemos obter uma "ideia" da sua dimensão se dispusermos do processo gerador dos dados. No nosso caso, este processo gerador é uma máquina que lança dardos aleatoriamente, mas que aterram sempre dentro de um quadrado com determinadas características.

Ao leitor mais atento, poderá ocorrer que se, em média, deveríamos ter obtido aproximadamente 785 ($\pi/4 * 1000$) dardos no interior do círculo, mas obtivemos 812, isto representa uma diferença de 27 dardos (ou 2.7% do total) que pode ser significativa. A clarificação desta questão prende-se com um aspecto importante da simulação pelo método de Monte Carlo: para que os resultados sejam válidos, importa abordar problemas que tenham uma solução *assintótica*, isto é, em que se possa observar uma convergência da quantidade estimada para o que seria o seu valor se o número de amostras (dardos, neste caso) gerado aleatoriamente fosse infinito. Daí decorre que, quanto maior for o número de amostras geradas, maior proximidade devemos esperar entre o valor estimado pela simulação e o valor verdadeiro.

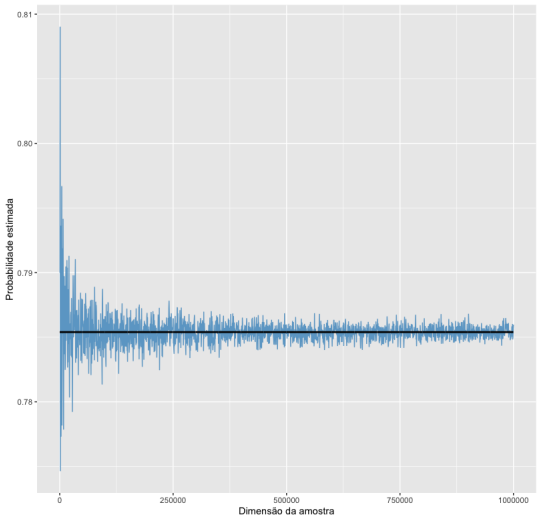

Figura 10: À medida que a dimensão da amostra aumenta, as estimativas de probabilidade variam cada vez menos em torno do valor verdadeiro, $\pi/4$, ilustrado pela linha horizontal a preto. As amostras foram simuladas para $n \in]0; 1 * 10^6]$ em incrementos de 500 observações.

A Figura 10 mostra a probabilidade estimada (com `set.seed(2022)`) em função da dimensão (crescente) da amostra, e ilustra o que referimos no parágrafo anterior: para amostras pequenas (menos dardos), as simulações resultam em proporções mais erráticas em torno do valor que sabemos ser o verdadeiro. À medida que a dimensão da amostra aumenta, a estimativa daquela proporção tende a aproximar-se, ou a variar cada vez menos em torno da "verdade" (o valor estimado para uma amostra $n = 1.000.000$ foi 0.785971, compare com $\pi/4$ e veja se coincidem); isto deverá ser intuitivo pensando no caso limite - se dispuséssemos de toda a população, a estimativa seria sempre igual ao valor verdadeiro. Repare que, na análise desta convergência, estão presentes duas ideias: a da diminuição da variabilidade "ao longo" de n, e a quantificação dessa mesma variabilidade.

O nosso exemplo pode ser replicado correndo o código:

```
set.seed(2022)
```

```
n <- 1000 #Número de simulações pretendidas
x <- runif(n, -1, 1) #Abcissas
y <- runif(n, -1, 1) #Ordenadas
dartSim <- data.frame(cbind(x,y))

#O comando matrix(runif(2*n, -1, 1), ncol = 2) produziria igualmente a matriz dartSim
#Fazemos em separado para ilustrar as coordenadas

radius = 1

countvec <- (x^2 + y^2 <= radius^2) #Condição geométrica do círculo com centro em (0,0)
prob <- (sum(countvec)/length(countvec)) #Proporção de TRUE no vector countvec
prob

#Output gráfico
require(ggplot2)

circleFun <- function(center = c(0,0),diameter = 2, npoints = 100){
  r = diameter / 2
  tt <- seq(0,2*pi,length.out = npoints)
  xx <- center[1] + r * cos(tt)
  yy <- center[2] + r * sin(tt)
  return(data.frame(x = xx, y = yy))
}

dat <- circleFun(c(0,0),2,npoints = 1000)

plot1 <- ggplot(dat,aes(x,y)) +
  geom_polygon(color = "black", fill = "skyblue3", alpha = 0.3) +
  geom_rect(aes(xmin = -1,
                xmax = 1,
                ymin = -1,
                ymax = 1), color = "black" , alpha = 0) +
  coord_equal()

plot2 <- ggplot(dat,aes(x,y)) +
  geom_polygon(color = "black", fill = "skyblue3", alpha = 0.3) +
  geom_rect(aes(xmin = -1,
                xmax = 1,
                ymin = -1,
                ymax = 1), color = "black" , alpha = 0) +
  coord_equal() +
  geom_point(data = dartSim, aes(x=x, y=y), shape = 8, size = 0.1, alpha = 0.9, color = "red")

require(gridExtra)
grid.arrange(plot1, plot2, ncol = 2)
```

3.2 Erro Padrão de um estimador de Monte Carlo

Prossigamos com o nosso exemplo geométrico: a nossa simulação resultou na probabilidade estimada $P\hat{[}Z] = 0.812$. No entanto, esta é apenas uma estimativa (pontual) da probabilidade. De facto, nada nos garante que, numa outra simulação com 1000 dardos, o número de dardos no interior do círculo se venha a repetir. Esta ideia de alteração de "amostra para amostra" remete para a *variabilidade* das estimativas que realizamos

recorrendo à simulação de Monte Carlo. O código abaixo permite-nos obter 20 amostras de 1000 dardos cada, que resultam em 20 probabilidades estimadas:

```
> set.seed(2022)
> means <- replicate(20, dartsim2(n)) #Geração aleatória de 20 médias amostrais

[1] 0.812 0.793 0.765 0.791 0.815 0.813 0.783 0.787
0.773 0.786 0.773 0.790 0.799 0.794 0.802 0.812 0.767
[18] 0.786 0.793 0.791
```

De facto, os valores variam mas nunca se afastam demasiado de $\pi/4 \sim 0.785$. Note que, se estipularmos uma variável aleatória X que assume o valor 1 para dardos no interior do círculo e 0 no caso contrário, o estimador que utilizámos pode ser escrito como:

$$\bar{X} = \hat{p} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{18}$$

ou seja, a proporção de dardos no interior do círculo é a média amostral de uma variável dicotómica, que só assume os valores 0 e 1.

Como $\hat{p}$ varia, e não podemos saber qual será o seu valor na "próxima" amostra, podemos tratar a média amostral como uma variável aleatória. Admitir este tratamento permite-nos calcular a sua média, variância, desvio-padrão, assimetria, curtose, etc. Assim, estabelecemos o conceito de **erro padrão**, que não é mais que o desvio-padrão de uma estatística amostral. Como gerámos acima 20 amostras e calculámos as respectivas médias amostrais, podemos calcular a média dessas 20 médias correndo o código:

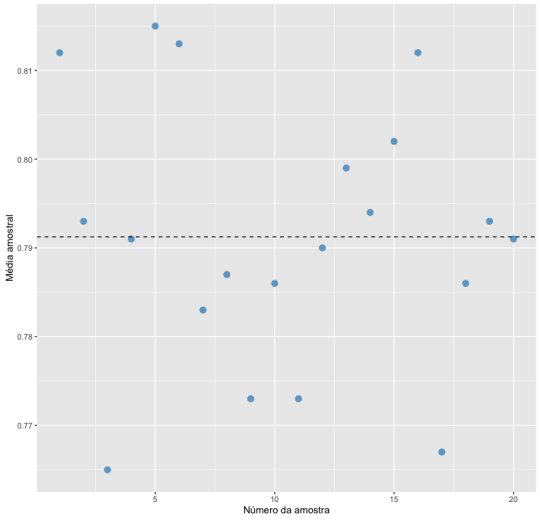

Figura 11: A dispersão das 20 médias amostrais geradas em torno da sua média aritmética

```
> set.seed(2022)
> n = 1000
> means <- replicate(20, dartsim2(n)) #Ver função dartsim2 acima
> mean(means)
[1] 0.79125
> sd(means)
[1] 0.0148213
```

Na prática, o que este resultado nos diz é que, tendo em conta que as 20 proporções estimadas se encontram perto de 0.78, uma variação típica de 0.0148213 não deverá causar grande volatilidade entre amostras. Com

o erro padrão calculado, é muito improvável que a próxima amostra só veja, por exemplo, 5 dardos (em 1000) dentro do círculo.

Para concluir, formalizando, o estimador do erro padrão de uma estimativa pontual de Monte Carlo é dado por:

$$\hat{SE}_{MC} = \frac{1}{\sqrt{N}}\sqrt{\frac{1}{N-1}\sum_{j=1}^{N}(\hat{\theta}^{(j)} - \hat{\theta}_{MC})^2} \tag{19}$$

em que N é o número de estimativas obtidas. O termo no interior da raiz é a variância amostral corrigida, resultante da soma dos desvios (ao quadrado) de cada uma das estimativas ($\hat{\theta}^{(j)}$) para a sua média amostral ($\hat{\theta}_{MC}$), e da divisão dessa soma por $N-1$. Como a nossa estimativa pontual de Monte Carlo é uma média, o multiplicando $1/\sqrt{N}$ decorre das Equações 20 e 21 que, por clareza e sem perda de generalidade, mostramos para o caso particular estimador da média aritmética:

$$VAR[\bar{X}] = VAR\left(\frac{\sum_{i=1}^{n} x_i}{n}\right) = \frac{1}{n^2}VAR\left(\sum_{i=1}^{n} x_i\right) = n\sigma^2/n^2 \iff VAR[\bar{X}] = \sigma^2/n \tag{20}$$

$$SE_{\bar{X}} = \sqrt{VAR[\bar{X}]} = \sqrt{\sigma^2/n} = \sigma/\sqrt{n} \tag{21}$$

3.3 Erro Quadrático Médio de um estimador de Monte Carlo

O Erro Quadrático Médio (EQM) de um estimador de Monte Carlo é uma medida de variabilidade do estimador em torno do valor populacional que pretende estimar. Prosseguindo com o exemplo dos dardos, suponha que se propuseram dois estimadores T_1 e T_2 para p, a verdadeira proporção de dardos que aterra no interior do círculo:

$$T_1 = \frac{\sum_{i=1}^{n} X_i}{n} \tag{22}$$

$$T_2 = \frac{X_1 + X_n}{2} \tag{23}$$

ou seja, a sua estimativa para p é dada pela média do primeiro e último elementos da amostra (note que no caso de uma variável dicotómica, T só assume os valores 0, 0.5 e 1). O nosso objectivo é avaliar o comportamento de $\bar{X}$ e de T quanto à proximidade entre as suas estimativas e o valor populacional que no nosso caso é $\pi/4$. Para esse efeito, geramos 20 amostras de 1000 observações e calculamos t_1^* e t_2^* (Equações 22 e 23) para cada uma daquelas amostras, adaptando a função que criámos anteriormente:

```
> set.seed(2022)
> n <- 1000 #Número de simulações pretendidas
> dartsim3 <- function(n){
+
+   x <- runif(n, -1, 1) #Abcissas
+   y <- runif(n, -1, 1) #Ordenadas
+   dartSim <- data.frame(cbind(x,y))
+
+   countvec <- (x^2 + y^2 <= 1) #Condição geométrica para que um ponto esteja contido no círculo
+   T1 <- (sum(countvec)/length(countvec)) #Estimador T1
+   T2 <- (countvec[1] + countvec[n])/2    #Estimador T2
+   return(c(T1, T2))
+ }
> replicate(20, dartsim3(1000))

      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10]
```

```
[1,] 0.812 0.793 0.765 0.791 0.815 0.813 0.783 0.787 0.773 0.786
[2,] 0.500 1.000 0.500 1.000 0.500 0.500 0.500 1.000 1.000 0.500

  [,11] [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20]
[1,] 0.773  0.79 0.799 0.794 0.802 0.812 0.767 0.786 0.793 0.791
[2,] 1.000  1.00 0.500 0.500 1.000 1.000 0.500 1.000 1.000 1.000
```

O EQM de um estimador $\hat{\theta}$ para um parâmetro θ é dado por:

$$E[(\hat{\theta} - \theta)^2] = \frac{\sum_{i=1}^{n}(\hat{\theta_i} - \theta)^2}{n} \tag{24}$$

No nosso exemplo, $\theta = \pi/4$ e $\hat{\theta_i}$ representa cada uma das estimativas que gerámos com o código acima, para cada um dos estimadores T_1 e T_2.

Para computar a Equação 24 para cada um dos estimadores, podemos correr o código:

```
> set.seed(2022)
> darts <- replicate(20, dartsim3(10000))
> eqm_t1 <- sum((darts[,1] - pi/4)^2)/n
> eqm_t2 <- sum((darts[,2] - pi/4)^2)/n
> print(c(eqm_t1, eqm_t2))
[1] 6.168551e-04 4.605459e-05
```

A partir dos resultados obtidos, podemos inferir que ambos os estimadores *erram* pouco, em média, ao apresentar valores para $p = \pi/4$ (`eqm_t1` e `eqm_t2` apresentam valores baixos). Por outro lado, o EQM do estimador T_2 é inferior ao do seu competidor T_1. Se estabelecermos como único critério preferir o estimador que produza menos variabilidade em torno do parâmetro, o estimador T_2 (proporção amostral) parece ser o melhor estimador entre os dois (será?). Note que, para além disso, ambos os estimadores são não enviesados para p. Deixa-se como exercício a replicação da simulação com outras *seeds* e a observação da relação entre o EQM de T_1 e T_2.

Muitas vezes, este tipo de estimador é designado por *loss* ou *cost function*, remetendo bem para o que o EQM pretende medir. A sua aplicação é útil para além da comparação de estimadores. Nomeadamente, a conhecida relação

$$EQM(\hat{\theta}) = VAR(\hat{\theta}) + env^2(\hat{\theta}) \tag{25}$$

permite a decomposição do EQM em duas fontes e por consequência uma maior granularidade na avaliação do comportamento do estimador.

Com esta breve exposição do conceito de EQM de um estimador de Monte Carlo, concluímos este capítulo, onde vimos como a simulação repetida a partir de um processo de gerador de dados nos permite quantificar alguns parâmetros de interesse. No capítulo seguinte iremos abordar o conceito de Cadeia de Markov. No Capítulo 5 iremos abordar uma classe de métodos (MCMC) que combina estes dois conceitos.

4 Cadeias de Markov

O conceito de Cadeia de Markov foi desenvolvido pelo matemático russo Andrey Markov no início do séc. XX. As Cadeias de Markov são definidas como processos estocásticos (sequências de números aleatórios) e têm um vasto número de aplicações em ciência. São representadas, essencialmente, por um conjunto de estados em que a cadeia se pode encontrar e pelas probabilidades de transição entre esses estados. Por conveniência, os estados de uma cadeia são frequentemente representados por instantes no tempo $t, t+1, ...t+n$, pelo que o t-ésimo elemento da cadeia se representa por X_t.

A sua característica fundamental é chamada propriedade de **dependência de Markov** e consiste na interligação, por construção, entre as observações ao longo da sequência, com uma particularidade: o próximo elemento da sequência (X_{t+1}) depende *apenas* do elemento anterior (X_t).

Desde a sua formalização por Markov, as Cadeias evoluíram para um campo de conhecimento, por oposição a um mero conceito, tendo sido produzida muita literatura não só quanto às suas propriedades e possíveis adaptações, mas também de aplicações práticas a diversas áreas do conhecimento que recorrem à inferência estatística.

Nesse sentido, e porque aqui nos focamos sobretudo em simulação de números e eventos aleatórios a partir de distribuições-alvo, o conceito de Cadeia de Markov será apresentado de forma abreviada. Nesta secção, depois de abordarmos um pouco de história, iremos ilustrar o conceito através de uma cadeia simples, recorrendo a um exemplo clássico. Em seguida, abordaremos alguns algoritmos de simulação baseados em Cadeias de Markov e, onde necessário, faremos referência a algumas propriedades das cadeias, para aprofundar um pouco mais o nosso conhecimento sobre estas.

4.1 Origem

Para entendermos a origem das Cadeias de Markov, importa recuar um pouco mais atrás no tempo, para perceber o que motivou o matemático russo para o seu desenvolvimento. No último quarto do séc. XVI, o matemático suíço Jakob Bernoulli escreveu um livro - *Ars Conjectandi*. Neste livro, Bernoulli abordou vários conceitos relacionados com análise combinatória e probabilidades, entre os quais a Lei dos Grandes Números, tipicamente ilustrada pelo exemplo de uma urna contendo bolas de duas cores.

Imagine que uma urna contém 1000 bolas, que sabe terem duas cores (branco ou preto), mas não sabe qual é a proporção de bolas de cada cor lá contidas. De forma simples, o que Bernoulli estipulou foi que caso pretendamos descobrir essa proporção, basta-nos retirar, com reposição, um grande número de bolas da urna e registar a cor de cada bola saída. No final, contar o número de bolas pretas e dividi-lo pelo total de repetições permite-nos calcular, pelo menos de forma aproximada, a verdadeira proporção daquelas bolas na urna. Se, ao fim de 1.000.000 de repetições, 300.000 tiverem resultado em bolas pretas, podemos afirmar com algum grau de certeza que a urna contém 30% de bolas daquela cor. Apesar de, agora, parecer trivial, esta questão teve (e tem) ramificações de índole mais teológica, senão filosófica: significa então que, se pudermos observar repetidamente qualquer processo, chegamos necessariamente à "lei" que governa esse processo? Ou não existem leis naturais, e alguns acontecimentos futuros são totalmente imprevisíveis?

Para acrescentar à confusão gerada, Bernoulli (e outros autores, posteriormente) observaram também que a média amostral de uma qualquer variável aleatória seguia com elevada frequência a distribuição binomial, da qual a distribuição Normal é um caso limite. Nekrasov, um teólogo que se tornou matemático, advogava fortemente a existência de livre arbítrio, tendo-se manifestado veementemente contra a possibilidade de termos uma espécie de destino *estatístico* governado pela distribuição normal. Isto tê-lo-á levado a fazer uma afirmação forte, a de que a independência é uma condição necessária à validade da Lei dos Grandes Números. Por outras palavras, o que Nekrasov quis dizer foi que, em jogos de azar, como a urna ou a roleta, a ocorrência de um determinado evento (e.g. Sair o número 7) não depende do número anterior, já que a sua probabilidade de ocorrência se mantém constante de tentativa para tentativa. No entanto, na vida real, os

processos que observamos registam algum tipo de auto-correlação, ou seja, o que acontece a seguir depende frequentemente do que aconteceu antes. Neste caso, ao contrário dos jogos de azar, defendeu Nekrasov, a Lei dos Grandes Números não se aplicaria, uma vez que não seria possível obter, por repetição continuada do processo, um estado dito estacionário para o qual as probabilidades associadas à ocorrência dos eventos convergissem.

Markov, um forte opositor das ideias de Nekrasov, provou, através das cadeias suas homónimas, que, mesmo no caso de dependência entre observações e estando verificadas algumas condições de regularidade, é possível "chegar" ao tal estado estacionário, ou seja, obter a distribuição de probabilidade inerente ao processo de geração dos dados observados.

Hoje em dia, as aplicações possíveis das Cadeias de Markov são virtualmente infinitas, e estão por trás de coisas tão distintas como sejam a avaliação do risco de um portefólio de activos financeiros ou a ferramenta de *autocomplete* que encontramos na maioria dos nossos telemóveis. Muitos dos problemas com que hoje nos deparamos (científicos, comerciais, etc.) são extremamente complexos pela quantidade de dimensões que envolvem. A combinação das Cadeias de Markov com os métodos de Monte Carlo, como veremos mais à frente, permite-nos tentar resolver esses problemas.

4.2 Um exemplo metereológico

Suponha que num determinado planeta existem dois eventos mutuamente exclusivos: num determinado dia ou faz Sol, ou faz Chuva. Na terminologia das Cadeias de Markov, estes dois eventos chamam-se *estados*, no sentido de a cadeia se poder encontrar em qualquer dos dois estados num determinado momento do tempo.

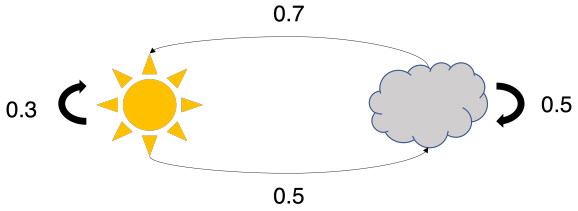

Figura 12: Um diagrama representando uma Cadeia de Markov, com dois estados possíveis e as respectivas probabilidades de transição entre eles

Suponha também que as probabilidades de transição entre os dois estados da cadeia são os exibidos na Figura 12. Por exemplo, a probabilidade de fazer Sol no dia $t+1$, sabendo que que fez Sol no dia t é igual 0.3; de forma análoga, a probabilidade de fazer Chuva no dia $t+1$, sabendo que fez Sol no dia t é igual a 0.5.

No contexto das Cadeias de Markov, estas probabilidades são denominadas *probabilidades de transição*, e, por conveniência, são em geral "registadas" numa matriz chamada a *matriz de transição* (*transition matrix*, em inglês), esquematicamente representada na Tabela 1.

Tabela 1: Matriz de transição entre estados da Cadeia de Markov

		X_{t+1}	
		Hoje vai estar	
		Sol	**Chuva**
X_t	**Sol**	0.3	0.7
Ontem estava	**Chuva**	0.5	0.5

Está implícita a propriedade de dependência de Markov: o estado do dia seguinte só depende do estado do dia anterior, e em particular através da estrutura correlacional estabelecida pela matriz de transição. Vejamos como interpretá-la:

- As *linhas* da matriz são as distribuições de probabilidade condicionada (ao dia anterior). A primeira linha da nossa matriz contém as probabilidades de hoje fazer Sol ou Chuva, *dado que ontem fez sol*.
- A soma dos elementos de cada linha da matriz é igual a 1, porque são probabilidades e representam tudo o que pode acontecer - Sol ou Chuva - dado que já aconteceu outra coisa (fez Sol ou Chuva).
- Seleccionando o elemento (2,1) da nossa matriz: a probabilidade de fazer Sol dado que a *um* passo atrás fez Chuva é igual a 0.5

A matriz de transição é um elemento fundamental da cadeia, pois é ela o "motor" de geração do estado seguinte. Por outro lado, a sua multiplicação permite obter as probabilidades de um determinado estado ocorrer após n passos da cadeia. Vejamos as matrizes abaixo,

$$\mathsf{P}^1 = \begin{bmatrix} 0.3 & 0.7 \\ 0.5 & 0.5 \end{bmatrix}$$

$$\mathsf{P}^2 = \begin{bmatrix} 0.3 & 0.7 \\ 0.5 & 0.5 \end{bmatrix} * \begin{bmatrix} 0.3 & 0.7 \\ 0.5 & 0.5 \end{bmatrix} = \begin{bmatrix} 0.44 & 0.56 \\ 0.40 & 0.60 \end{bmatrix}$$

$$\mathsf{P}^9 = \begin{bmatrix} 0.416667 & 0.583333 \\ 0.416667 & 0.583333 \end{bmatrix}$$

$$\mathsf{P}^{10} = \begin{bmatrix} 0.416667 & 0.583333 \\ 0.416667 & 0.583333 \end{bmatrix}$$

onde P^n representa a matriz de transição entre estados, a um passo, após n passos da cadeia e resulta da multiplicação sucessiva de P^1. Assim, ao elemento (1,1) da matriz $\mathsf{P}^2 = 0.44$ corresponde a probabilidade de fazer Sol ao segundo passo da cadeia, sabendo que fez sol no primeiro passo. Por outras palavras, se fez Sol na segunda-feira, a probabilidade de fazer Sol na terça-feira é igual a 0.44.

O leitor atento terá reparado que as matrizes P^9 e P^{10} são idênticas. Essa "igualdade" reflecte a característica fundamental das Cadeias de Markov: após um determinado número de passos, a cadeia converge em distribuição para uma distribuição estacionária, e por isso as probabilidades não se alteram com o avançar da cadeia. No nosso exemplo, a probabilidade marginal (i.e. não condicional) de ter estado Sol num dia escolhido ao acaso é igual a 0.416667; a partir da matriz de transição, que quantifica as relações de dependência entre os estados, é possível obter, observadas algumas condições de regularidade, a distribuição de probabilidade marginal da ocorrência daqueles estados. A probabilidade de fazer Chuva *sabendo* que fez

Sol é diferente da probabilidade de fazer Chuva *independentemente* do que tenha acontecido antes.

Na realidade, a matriz de transição P é a representação em forma matricial da *função de transição* da cadeia, que a especifica completamente. Por conveniência de notação, passaremos a referir-nos à função de transição $p(i,j)$ para significar a(s) probabilidade(s) de transição, em um passo, o estado i para o estado j. Para nos referirmos ao espaço de estados, isto é, ao conjunto de todos os estados passíveis de ocorrer, utilizaremos a letra U.

Como já referimos, a tarefa de interesse, para já, é encontrar a distribuição de probabilidade marginal dos estados. No nosso exemplo, isto significa que estamos interessados em encontrar a probabilidade de fazer Sol ou Chuva num dia ao acaso, $P[Sol]$ e $P[Chuva]$. Esta distribuição, se existir, chama-se *estacionária* e denota-se por π.

Uma distribuição $\pi(i), i \in U$ diz-se **estacionária** com função de transição $p(.,.)$ se $\pi = \pi p(.,.)$. No nosso contexto, isto significa que, a partir de n iterações da cadeia, esperamos que as probabilidades contidas em $p^n(i,j)$ se mantenham constates com o produto sucessivo de p.

Quando a distribuição estacionária é encontrada, diz-se que a cadeia está em *equilíbrio*. Este equilíbrio pode ser descrito pela expressão

$$\pi(i) = \sum_j \pi(j)p(j,i) \tag{26}$$

em que $\pi(i)$ é a probabilidade marginal da cadeia se encontrar no estado i, e $p(i,j)$ é a probabilidade de o estado i se seguir ao estado j em apenas um passo. A Equação 26 é conhecida como **condição global de equilíbrio** e representa, do ponto de vista de um estado individual i, uma aplicação do conhecido Teorema da Probabilidade Total, no contexto das Cadeias de Markov. Por outras palavras, a probabilidade de a cadeia se encontrar no estado i é dada pela soma das plausibilidades de transitar do estado j para o estado i, percorrido todo o espaço de estados através de j.

No nosso exemplo, esta distribuição pode ser obtida de forma analítica (nem sempre é o caso), resolvendo o seguinte sistema de equações:

$$\begin{cases} P[Sol] = \pi(1) = 0.3\pi(1) + 0.5\pi(2) \\ \pi(1) + \pi(2) = 1 \end{cases} \leftrightarrow \begin{cases} \pi(1) = \frac{5}{12} = 0.41666(6) \\ \pi(2) = \frac{7}{12} = 0.58333(3) \end{cases} \tag{27}$$

Repare que a primeira equação do sistema corresponde à Equação 26 aplicada ao nosso exemplo, e que os valores obtidos correspondem aos contidos nas matrizes P^9 e P^{10} acima.

4.3 Definições

Como já dissemos, uma das características das Cadeia de Markov é a estrutura da dependência entre os estados. Especificamente, o estado "seguinte" depende apenas do estado anterior:

$$P[X_{t+1}|X_t, X_{t-1}, ..., X_0] = P[X_{t+1}|X_t], \forall t \geq 0 \tag{28}$$

Sendo tentadora a ideia da convergência para uma distribuição estacionária, a sua concretização depende da verificação de algumas condições de regularidade. Ou seja, para que possamos, por exemplo, simular números ou amostras a partir de uma variável aleatória cuja distribuição não conhecemos de forma analítica, devemos observar alguns cuidados ao construir a cadeia pretendida. Especificamente, pretendemos garantir a existência e unicidade da distribuição $\pi(.)$. As definições que veremos em seguida prendem-se exactamente com essas condições de regularidade.

Definição 4.1 (Irredutibilidade) *Uma Cadeia de Markov diz-se* ***irredutível*** *se todos os estados da cadeia comunicarem entre si.*

Por outras palavras, é possível atingir qualquer estado j partindo de qualquer estado i num número finito de passos. A Figura 13 representa uma cadeia que *não* respeita esta condição, uma vez que há duas *classes* de comunicação; se o estado inicial do sistema pertencer a 1, 2, 3, não é possível atingir os estados 4, 5, 6.

Para o âmbito do nosso estudo, é útil compreender o espaço de estados U como o domínio de uma variável aleatória. Suponha que se extraem números consecutivos de uma qualquer distribuição normal, com μ e σ constantes. O domínio $]-\infty, +\infty[$ é o espaço de estados (contínuo) da nossa variável, e é possível obter, na próxima iteração da cadeia, qualquer número naquele domínio, com probabilidade variável mas positiva. Nesse sentido, uma cadeia assim configurada dir-se-ia irredutível, por ser possível alcançar qualquer estado a partir de outro. A existência de estados que ocorrem apenas uma vez, para nunca mais ocorrerem, é um exemplo de violação da irredutibilidade.

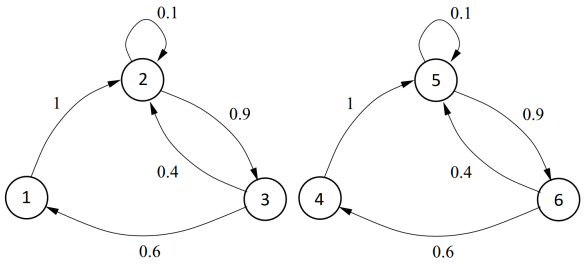

Figura 13: Esta Cadeia de Markov *não é irredutível*. Nem todos os estados comunicam entre si.

Definição 4.2 (Recorrência) *Uma Cadeia de markov diz-se* ***recorrente*** *se regressa infinitas vezes ao estado de onde parte com probabilidade 1.*

Em particular, diz-se **recorrente positiva** se o tempo médio de retorno ao estado i, quando parte dele, for finito.

Repare que, se o espaço de estados U for discreto, a irredutibilidade implica a recorrência positiva da cadeia: se qualquer estado é acessível a partir de qualquer outro, nenhuma das probabilidades de transição entre estados é nula. Ou seja, podemos assegurar recorrência positiva da cadeia se pudermos garantir $p(i,j) > 0\ \forall (i,j)$.

Pode ser demonstrado que qualquer Cadeia de Markov, irredutível, e recorrente positiva, não só tem uma distribuição estacionária π única como essa distribuição estacionária é única. Este resultado tem elevado interesse prático para os algoritmos que veremos adiante. No entanto, ainda que a estacionariedade e unicidade da distribuição possam ser garantidas, o mesmo não se pode dizer da convergência da cadeia para π. Para tal, é necessário, cumulativamente, assegurar que a cadeia é *aperiódica*.

Definição 4.3 (Periodicidade) *Um estado i tem período k se qualquer retorno ao estado i ocorrer em exactamente k iterações.*

Na Figura 14 encontra-se ilustrada uma cadeia com 4 estados (0, 1, 2, 3), em que o estado 0 é visitado a cada três passos, se iniciarmos o sistema em 0. Se definirmos π_n como o vector de probabilidade de transição, iniciar o sistema em 0 escreve-se $\pi_0 = (1,0,0,0)$ (a probabilidade de o sistema se encontrar no estado 0 no passo 0 é 1, porque escolhemos iniciá-lo aí). Da mesma forma, facilmente vemos que $\pi_1 = (0, 1/3, 2/3, 0)$: tendo começado em 0, no passo 1 ou progredimos para 1 ou para 2 (com probabilidades 1/3 e 2/3, respectivamente). No segundo passo, $\pi_2 = (0,0,0,1)$: estando em 1 ou 2 progredimos para 3 com probabilidade 1. Finalmente, no terceiro passo, $\pi_3 = (1,0,0,0)$: estando em 3 progredimos para 0 com probabilidade 1. A partir daqui, o sistema repete-se e regressa ao estado 0 a cada três passos. Diz-se, então que o estado 0 tem

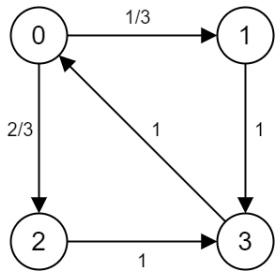

Figura 14: Esta Cadeia de Markov *não é aperiódica*. O estado 0 é sempre visitado a cada três passos (se o sistema for inicializado em 0).

período $k = 3$.

Formalmente, o período do estado i é dado por:

$$k = min\{n \geq 1 : p^n(i, i) > 0\} \tag{29}$$

e uma cadeia diz-se **aperiódica** se todos os seus estados tiverem período $k = 1$. Por outras palavras, se for sempre possível que o próximo estado da cadeia seja igual ao estado actual.

Um corolário importante para compreender os métodos de simulação que veremos mais à frente, é que numa cadeia *irredutível* todos os estados têm o mesmo *período*. Pode ser demonstrado que uma Cadeia de Markov com espaço de estados U discreto, *irredutível* e *aperiódica*, que se diz **ergódica**, converge para uma distribuição *estacionária*. Na prática, os métodos que aqui nos interessam asseguram a ergodicidade através de uma condição suficiente mas não necessária, a que chamamos *reversibilidade*.

Definição 4.4 (Reversibilidade) *Uma Cadeia de Markov diz-se* ***reversível*** *se*

$$P(X_{t+1}|X_t = i) = P(X_{t+1}|X_{t+2} = i) \tag{30}$$

O termo "reversível", dito de uma cadeia, significa que esta se comporta de forma igual, em termos de distribuição, independentemente da direcção em que seja percorrida. Assegurar a reversibilidade equivale a verificar que

$$\pi(i)p(i, j) = \pi(j)p(j, i), \forall(i, j) \tag{31}$$

conhecida como **condição detalhada de equilíbrio** (Equação 31). Informalmente, tudo se passa como se todos os estados estivessem a trocar à mesma frequência entre si, para cada par de estados da cadeia. Esta condição unifica as condições de regularidade que enunciámos: uma cadeia reversível é por definição ergódica, convergindo para π. A utilização de cadeias *reversíveis* está na essência dos métodos que abordamos em seguida.

5 Métodos de Monte Carlo em Cadeias de Markov (MCMC)

Revisitemos a seguinte ideia: gerar números aleatórios a partir de uma distribuição implica conhecê-la de alguma forma.

Os métodos de simulação que abordámos no Capítulo 2 permitem-nos gerar esses números obtendo um número suficientemente grande de amostras i.i.d. e posteriormente calcular quantidades de interesse através da colecção obtida. Em muitas situações, nomeadamente quando pretendemos fazer inferência a partir de distribuições multivariadas com relações de dependência de elevada complexidade, recorre-se aos Métodos de Monte Carlo em Cadeias de Markov (MCMC, na sua sigla em inglês) que, apesar de envolverem um maior número de tentativas para a "descoberta" da distribuição-alvo, permitem a resolução de problemas mais difíceis e inacessíveis pelos métodos clássicos.

Embora os seus fundamentos estejam fora do âmbito deste texto, não é possível mencionar os métodos MCMC sem mencionar a inferência bayesiana, e em particular a aplicabilidade daqueles métodos a problemas desta natureza. Neste tipo de problemas, pretende-se frequentemente obter a distribuição *a posteriori* $f(\theta|x)$ fazendo uso da relação $f(\theta|x) = \frac{f(x|\theta)f(\theta)}{f(x)}$, e procurando por exemplo a moda de $f(\theta|x)$ como o valor mais provável para o(s) parâmetro(s) pretendidos. Em particular, os algoritmos que iremos abordar para este efeito assentam numa lógica de aceitação ou rejeição semelhante ao método que vimos no Capítulo 2. Uma característica útil comum a estes métodos consiste em não ser necessário o conhecimento da constante de normalização de $f(\theta|x)$ para se poder simular a partir dela. A constante de normalização permite tratar $f(\theta|x)$ como uma função densidade de probabilidade (no caso da distribuição normal, a constante é $\frac{1}{\sqrt{2\pi}}$).

5.1 O algoritmo de Metropolis-Hastings

O algoritmo de Metropolis-Hastings (MH) é, na realidade, uma classe de métodos MCMC que permite a convergência para uma distribuição estacionária de determinada Cadeia de Markov. De forma sucinta, o algoritmo consiste na proposta sucessiva de valores para o próximo estado da cadeia, provenientes de uma distribuição dita **proponente**, dependente do estado anterior e conhecida. Os valores propostos são sujeitos a uma regra de decisão que consiste na avaliação da *condição detalhada de equilíbrio* (Equação 31), entre a distribuição-alvo $f(.)$ e a distribuição proponente $g(.|X_t)$, aceitando com maior probabilidade propostas onde o *desequilíbrio* é maior.

Em seguida vamos elencar e descrever cada um dos passos do algoritmo, para depois o implementar com recurso a `R`. Para uma melhor compreensão e consistência com a notação do capítulo anterior, faz-se notar que ao numerador da Equação 32 equivale, com os devidos ajustes, qualquer um dos termos da igualdade da Equação 31. Os estados da cadeia, neste contexto, são todos os valores pertencentes ao domínio da distribuição-alvo a partir da qual se pretende simular. Na descrição que se segue, pode entender-se, de forma lata, X_t como i e Y como j, letras que utilizámos anteriormente para descrever estados arbitrários de uma cadeia.

Para ilustrar o funcionamento do algoritmo, iremos definir como objectivo a geração de números aleatórios a partir da distribuição Beta(1,6). Importa fazer notar que existem formas mais eficientes de fazer esta simulação (o gerador de base do `R` é `rbeta`), e que a dificuldade na construção deste tipo de métodos pode muitas vezes começar logo pela dificuldade de aproximar convenientemente a distribuição-alvo. Portanto, aqui tudo se passa como se conhecêssemos a expressão da distribuição-alvo Beta(1,6), mas não houvesse qualquer gerador (ou método) alternativo. O procedimento consiste nos passos seguintes:

1. Definir a função objectivo $f(.)$
2. Escolher a distribuição proponente $g(.|X_t)$
3. Simular Y a partir de $g(.|X_t)$
4. Simular a partir de $U(0,1)$

5. Calcular a probabilidade de aceitação

$$\alpha(X_t, Y) = \frac{f(Y)g(X_t|Y)}{f(X_t)g(Y|X_t)} \tag{32}$$

6. Aceitar o valor proposto se

$$U(0,1) \leq \frac{f(Y)g(X_t|Y)}{f(X_t)g(Y|X_t)} \tag{33}$$

e então $X_{t+1} = Y$. Caso contrário, $X_{t+1} = X_t$.

Vejamos os passos do algoritmo, um a um:

Passo 1: Definir a distribuição-alvo $f(.)$

Como dissemos anteriormente, pretendemos gerar números aleatórios a partir da distribuição Beta. A distribuição Beta(α, β) tem f.d.p.,

$$f(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)}, \alpha, \beta > 0 \tag{34}$$

que por sua vez tem o mesmo significado que $\pi(.)$ na Equação 31. Ou seja, a nossa distribuição-alvo, a partir da qual queremos simular, é $f(x)$, que assume a sua forma consoante os parâmetros α e β. Para o nosso exemplo, escolhemos arbitrariamente $\alpha = 1$ e $\beta = 6$. Assim, o nosso objectivo concreto passa a ser simular valores a partir de Beta(1,6). A Figura 15 exibe a densidade da distribuição.

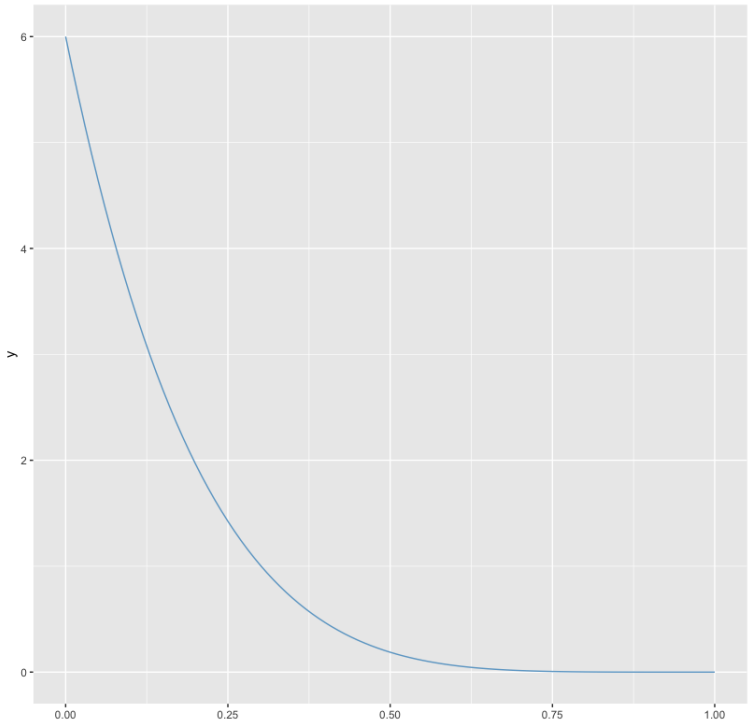

Figura 15: A função densidade de probabilidade da distribuição Beta(1,6), a partir da qual queremos simular. Note o domínio em (0,1).

Passo 2: Escolher a distribuição proponente $g(.|X_t)$

Em geral, podemos usar qualquer distribuição como proponente para gerar valores para X_{t+1}, desde que tenha o mesmo domínio que a função objectivo. No entanto, a escolha desta distribuição tem influência na velocidade de convergência da cadeia, e a sua optimização é objecto de estudo, fora do âmbito deste texto. A utilização de distribuições simétricas reveste-se de especial interesse, como veremos adiante.

Para o nosso exemplo, vamos propor a distribuição triangular (ver Figura 16). Em geral, esta distribuição pode ser parametrizada por a, b e c, que são o limite inferior, o limite superior e o vértice superior (centro) do triângulo, respectivamente.

Por memória, a função densidade é dada por ramos (dada a existência de um vértice):

$$g(x;a,b,c) = \begin{cases} 0 & x < a \\ \frac{2(x-a)}{(b-a)(c-a)} & a \leq x < c \\ \frac{2}{b-a} & x = c \\ \frac{2(b-x)}{(b-a)(b-c)} & c < x \leq b \\ 0 & x > b \end{cases} \tag{35}$$

Passo 3: Simular Y a partir de $g(.|X_t)$

É a partir de $g(.|X_t)$ que vamos construir a Cadeia de Markov. Para que a simulação seja eficiente, vamos definir o seu domínio em (0,1), o mesmo domínio da distribuição Beta. Como, numa cadeia de Markov, o estado seguinte só pode depender do anterior, vamos estipular que o valor proposto será obtido a partir de uma distribuição triangular em que o centro é dado pelo estado anterior (ver Figura 16). A função `rtri` do pacote `EnvStats` permite-nos simular a partir de uma distribuição triangular com quaisquer limites e centro. No nosso caso, vamos inicializar o vector de amostras com a instância X_0 e produzir a cadeia de valores *propostos*:

```
########### METROPOLIS-HASTINGS ###########
require(EnvStats)

prop <- function(nprop){

  #FUNÇÃO para gerar números correlacionados
  #a partir da distribuição candidata

  #Vector com 0's para guardar as nprop simulações
  X <- numeric(nprop)

  #Inicializar o vector com um elemento arbitrário
  X[1] = rtri(1, min = 0, max = 1, mode = 0.5)

  #Gerar observações correlacionadas
  #A moda (c) da distribuição da proposta seguinte
  #é igual ao estado anterior da cadeia

  for (i in 2:nprop){
    X[i] <- rtri(1, min = 0, max = 1, mode = X[i-1])
  }
```

```
  #Ver os resultados
  return(X)
}
```

A Figura 16 mostra 10.000 valores propostos pela distribuição triangular, respeitando a dependência do estado anterior. A questão que poremos, em seguida, é a de aceitar ou rejeitar cada um destes valores, de modo a obter a nossa função objectivo Beta(1,6).

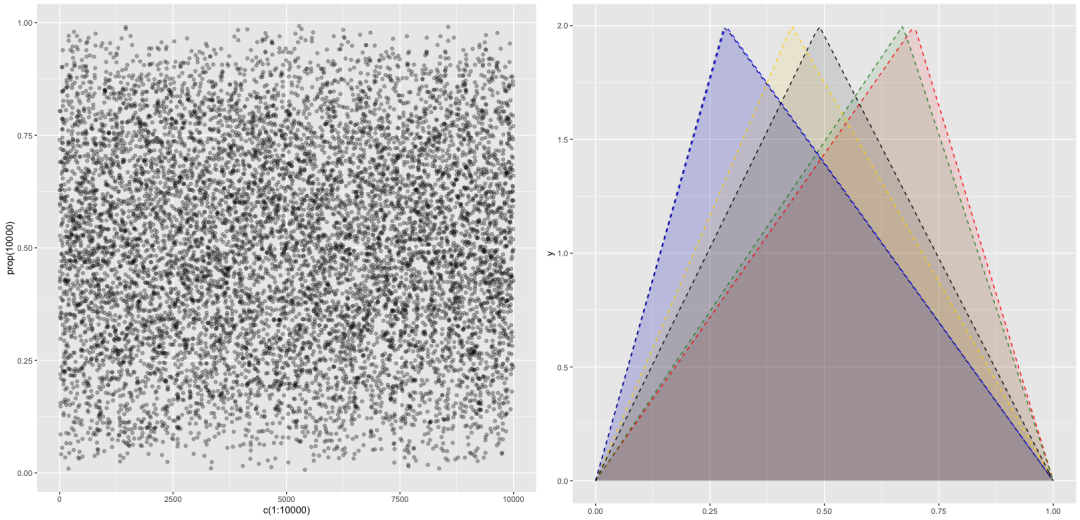

Figura 16: 10.000 valores propostos pela distribuição proponente (à esquerda) e as distribuições a partir dos quais os primeiros 6 valores de Y foram propostos

Passo 4: Simular a partir de $U(0,1)$

Os valores simulados a partir da distribuição uniforme servirão de crivo para a nossa regra de aceitação dos valores propostos, como veremos mais à frente. Para já, vamos simular 10.000 valores a partir de $U(0,1)$ e guardá-los num vector.

```
set.seed(2022) #Para replicabilidade dos resultados
nprop = 10000
unis <- runif(nprop)
```

Passo 5: Calcular a probabilidade de aceitação $\alpha(X_t, Y)$

O rácio $\alpha(X_t, Y)$ é o amâgo do algoritmo de Metropolis-Hastings, e decorre do equilíbrio que descrevemos na Equação 31. Para entendermos melhor, vamos definir r_f e r_g tal que $r_f * r_g = \alpha(X_t, Y)$, ou seja

$$r_f = \frac{f(Y)}{f(X_t)}, r_g = \frac{g(X_t|Y)}{g(Y|X_t)} \tag{36}$$

e

$$\alpha(X_t, Y) = \frac{f(Y)g(X_t|Y)}{f(X_t)g(Y|X_t)} = r_f * r_g \tag{37}$$

Relembre, também, que a função $f(.)$ é a nossa função objectivo, Beta(1,6), a partir da qual queremos simular. O rácio r_f assumirá valores elevados quando seja muito mais "provável" a ocorrência de y (valor proposto) em Beta(1,6) do que a ocorrência de x_t (valor actual da cadeia). Em geral, a nossa regra consistirá em aceitar valores mais altos de r_f: se o valor proposto ocorre com mais frequência na função objectivo do que o valor anterior, então tendemos a aceitar esse valor. Caso contrário, a cadeia mantém o valor anterior

$(X_{t+1} = X_t)$.

O rácio r_g, no nosso exemplo, é o quociente entre a densidade de X_t na distribuição triangular com moda $c = Y$ e a densidade de Y na distribuição triangular com moda $c = X_t$, e mede a "relação de troca" entre os estados. Note que, para distribuições simétricas, $g(X_t|Y) = g(Y|X_t)$ (porquê?) e, então

$$\alpha(X_t, Y) = r_f = \frac{f(Y)}{f(X_t)} \tag{38}$$

que é o caso particular, ainda que precedente no tempo, do **algoritmo de Metropolis** (sem o Hastings). Em geral, o uso de uma distribuição proponente simétrica permite a convergência do algoritmo. No nosso exemplo, a distribuição não é necessariamente simétrica, pelo que r_g terá de ser avaliado a cada iteração da cadeia.

Passo 6: Aceitação do valor proposto

A aceitação de cada valor proposto é avaliada consoante a regra:

$$U(0,1) \leq \frac{f(Y)g(X_t|Y)}{f(X_t)g(Y|X_t)} \tag{39}$$

Veja que, ao utilizar $U(0,1)$, estamos a aceitar iterativamente amostras que tendem a reflectir, à medida que a cadeia converge, a distribuição-alvo. Por vezes, o rácio $\alpha(X_t, Y)$ excede o valor 1, o que não acontece com $U(0,1)$. Por outras palavras, valores de $\alpha \geq 1$ são sempre aceites, o que pode gerar atrasos na convergência da cadeia, que pode assim permanecer "muito tempo" em regiões de elevada densidade, demorando a percorrer todo ou a maior parte do espaço de estados. Note que a regra envolvendo $U(0,1)$ resulta em que a probabilidade de aceitação seja

$$min\left\{1, \frac{f(Y)g(X_t|Y)}{f(X_t)g(Y|X_t)}\right\} \tag{40}$$

Note ainda que se $U(0,1) \leq \alpha(X_t, Y)$, então $X_{t+1} = Y$. Caso contrário, $X_{t+1} = X_t$.

Agora que analisámos os seus passos, vamos implementar o algoritmo para simular números aleatórios a partir da distribuição Beta(1,6):

```
#ALGORITMO DE METROPOLIS-HASTINGS

require(EnvStats)
mh_beta <- function(nprop){

  k = 0 #Contador de rejeições
  unis <- runif(nprop) #Geração de números uniformes para regra de aceitação

  #Vector com 0's para guardar as simulações aceites
  X <- numeric(nprop)

  #Inicializar o vector com um elemento arbitrário
  X[1] <- rtri(1, min = 0, max = 1, mode = 0.5)

  #Gerar observações correlacionadas
  #A moda (c) da distribuição da proposta seguinte
  #é igual ao estado anterior da cadeia

  for (n in 2:nprop){
    xt <- X[n-1]

    y <- rtri(1, min = 0, max = 1, mode = xt)
```

```
    #Regra de aceitação
    num <- dbeta(y, 1, 6)  * dtri(xt, 0, 1, mode = y)  # O segundo termo é g(xt|y)
    den <- dbeta(xt, 1, 6) * dtri(y, 0, 1, mode = xt)   # O segundo termo é g(y|xt)

    if(unis[n] <= num/den) {X[n] <- y} else {
      X[n] <- xt  #Rejeitar
      k = k+1  #Aumentar o contador de rejeições
    }
  }

  #Exportar os resultados
  print(k) #Ver o número de rejeições
  return(X)
}
```

A Figura 17 indica que a cadeia convergiu para a distribuição estacionária pretendida, ou seja, a colecção seleccionada a partir dos 10.000 números gerados segue uma distribuição muito próxima de Beta(1,6)

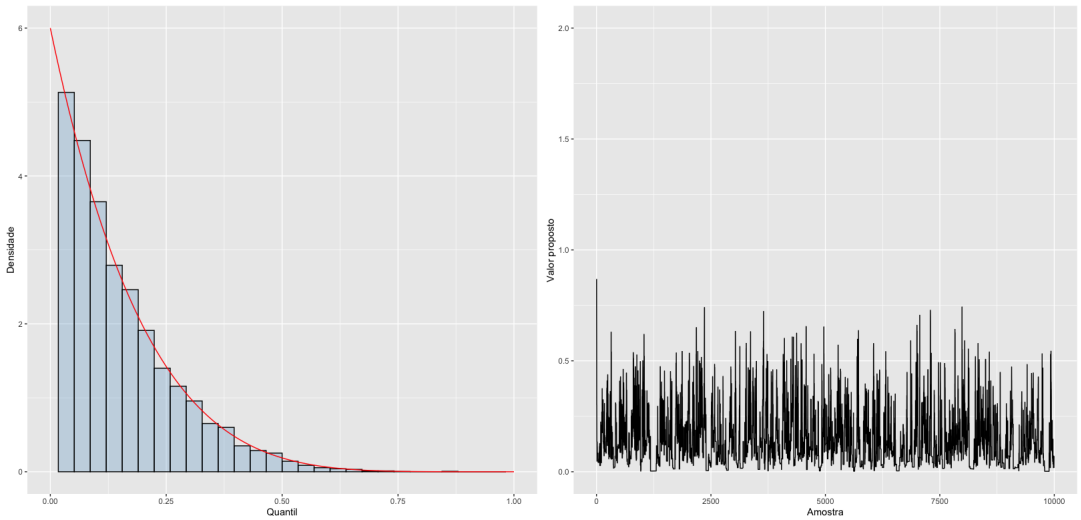

Figura 17: À esquerda: histograma de valores aceites vs. densidade objectivo. À direita: série de valores propostos

Em seguida, iremos abordar três algoritmos: Passeio Aleatório, Amostrador Independente e Amostrador de Gibbs. Como veremos, estes algoritmos são casos particulares do algoritmo de Metropolis-Hastings. Para os ilustrar, iremos escolher distribuições-alvo conhecidas para as quais existem alternativas mais eficientes para simulação, sem perda de generalidade.

5.1.1 O Passeio Aleatório

O Passeio Aleatório (em inglês, *Random Walk*) é um caso particular do algoritmo de Metropolis-Hastings. A sua particularidade advém de duas condições: por um lado, a utilização de distribuições proponentes simétricas, e, por outro, da modelação da dependência entre passos dada por $Y = X_t + Z$. Esta relação de dependência consiste em propor um número aleatório Y resultante da soma do passo anterior (X_t) a um número gerado aleatoriamente a partir de uma distribuição simétrica (e.g. $g(Y|X_t) \sim N(\mu = X_t, \sigma^2)$).

Note que a utilização de uma distribuição proponente simétrica reduz o cálculo da probabilidade de aceitação à Equação 38.

A sua implementação é relativamente simples e semelhante ao caso geral. A função cujo código se segue permite simular a partir de ${\chi^2}_{df=2}$, ou seja, essa é distribuição-alvo para a qual se pretende que a cadeia convirja. Adicionalmente, escolheu-se $N(\mu = X_t, \sigma^2)$ para distribuição proponente. Isto significa que o número proposto para X_{t+1} é obtido a partir de uma distribuição normal com o centro dado pelo estado anterior X_t.

Os argumentos da função são `x0`, o estado inicial (arbitrário) da cadeia, `n`, o número de iterações pretendido, `sd`, o desvio-padrão da distribuição proponente, e `df`, os graus de liberdade para parametrizar ${\chi^2}_{df=2}$.

```
#####PASSEIO ALEATÓRIO

rw <- function(x0, n, sd, df){

  X <- numeric(n)                   #Vector para guardar as simulações, i.e. a cadeia
  unis <- runif(n)                  #Vector de números uniforme para a regra de aceitação
  X[1] <- x0                        #Estado inicial da cadeia
  k = 0

  for (i in 2:n){

    xt <- X[i-1]
    y <- rnorm(1, xt, sd)           #Distribuição proponente dependente do estado anterior xt

    num <- dchisq(y, df)            #Numerador da prob. de aceitação
    den <- dchisq(xt, df)           #Denominador da prob. de aceitação

    if(unis[i] <= (num/den)){       #Aceitar ou rejeitar
      X[i] <- y} else {
      X[i] = xt
      k = k+1}                      #Contador de rejeições
  }
  return(list(X=X, k=k))            #Devolver os resultados
}
```

Em seguida, vamos gerar quatro cadeias distintas (i.e. com desvios-padrão diferentes para cada uma), para avaliar o comportamento do algoritmo, com os seguintes parâmetros:

```
set.seed(2022)
rw.1 <- rw(x0 = 50, n = 2000, sd = 0.05, df = 2)
rw.2 <- rw(x0 = 50, n = 2000, sd = 0.50, df = 2)
rw.3 <- rw(x0 = 50, n = 2000, sd = 2.00, df = 2)
rw.4 <- rw(x0 = 50, n = 2000, sd = 16.0, df = 2)
```

Apesar de implementação relativamente simples, a convergência do algoritmo é particularmente sensível à variância da distribuição proponente. A Figura 18 evidencia essa sensibilidade. Os gráficos são apresentados dois a dois, para cada valor de σ (em linha) e mostram, à esquerda, a distribuição (densidade) da cadeia após simulação de 2000 valores. À direita, os gráficos representam a cadeia propriamente dita, ou seja, cada um dos 2000 passos da cadeia e o respectivo estado. As linhas horizontais representam o domínio de ${\chi^2}_{df=2}$, ou quase, uma vez que a linha superior é dada por `qchisq(0.99, 2)`. O objectivo é avaliar a convergência da cadeia para o interior destas bandas, onde se encontram os estados da distribuição-alvo.

Escolhemos `x0 = 50` por ser um *outlier* de ${\chi^2}_{df=2}$. Assim, podemos ver claramente a convergência da cadeia. Note que, no primeiro caso ($\sigma = 0.05$) a variância é "reduzida", pelo que o rácio r_f resulta consecutivamente

em valores altos de $\alpha(X_t, Y)$, e por isso quase todos os pontos são aceites. Por outro lado, após 2000 iterações o passeio aleatório continua a encontrar-se "muito longe" da distribuição-alvo. Isto é evidente pela densidade das 2000 iterações, muitíssimo diferente da de ${\chi^2}_{df=2}$, e pela distância da cadeia às bandas horizontais. Em sentido contrário, no último caso ($\sigma = 16$), a distribuição converge "muito rápido", no sentido de se aproximar das bandas, mas é menos eficiente, no sentido em que os valores propostos são rejeitados com mais frequência. Isto é visível nos vários segmentos horizontais da cadeia, significando que esta permaneceu no estado anterior (rejeitou o valor proposto).

Assim, um problema de interesse é o da calibração da variância da distribuição proponente. Não existindo uma regra exacta que nos permita optimizar essa calibração, a heurística prevalente na literatura indica que a variância deverá ser tal que a taxa de rejeição do algoritmo se situe entre 15% e 50%. No nosso caso, apenas a terceira cadeia apresenta uma taxa de rejeição neste intervalo, e é por isso a que melhor equilibra o *trade off* entre rapidez de convergência e eficiência, entre as quatro alternativas apresentadas.

```
> cbind(rw.1$k, rw.2$k, rw.3$k, rw.4$k)/2000
       [,1]   [,2]   [,3] [,4]
[1,] 0.0115 0.1115 0.5035 0.89
```

Neste tipo de algoritmos, é frequente descartar-se um conjunto das primeiras simulações, definindo um *burn in period*. Informalmente, como as cadeias levam algum tempo a convergir, as instâncias iniciais da cadeia tendem a distorcer a distribuição que se pretende estacionária. No nosso exemplo, a terceira cadeia apresenta uma densidade muito mais próxima de ${\chi^2}_{df=2}$ após a remoção das primeiras 500 iterações. Veja o gráfico à esquerda na terceira linha da Figura 18 e compare com a Figura 19.

```
require(ggplot2)

vec <- rw.3$X[-(1:500)] #Remover as primeiras 500 iterações

#Gráfico para comparação com a densidade teórica

plot9 <- ggplot(data.frame(vec), aes(x=vec)) +
  geom_histogram(aes(y=..density..), color="black", fill="skyblue3", alpha = 0.3, bins = 30) +
  stat_function(fun = dchisq, args=c(2), col = "red") +
  xlab("sigma = 2")+
  ggtitle("Cadeia rw.3 | Sigma = 2")
```

5.1.2 O Amostrador Independente

O Amostrador Independente (AI) é outro caso particular do algoritmo de MH. A sua particularidade advém da utilização de distribuições proponentes cujo(s) parâmetro(s) não dependem do estado anterior da cadeia. Assim, a probabilidade de aceitação de qualquer valor proposto Y é dada por:

$$\alpha(X_t, Y) = \frac{f(Y)g(X_t)}{f(X_t)g(Y)} \tag{41}$$

onde a diferença para a Equação 37 reside na utilização de $g(X_t)$ em vez de $g(X_t|Y)$ e vice-versa. Note que a independência da distribuição proponente não significa a independência entre X_{t+1} e X_t, uma vez que Y é comparado com X_t a cada iteração. O Amostrador Independente é relativamente fácil de implementar e tende a ser tanto melhor quanto maior seja a semelhança entre $f(.)$ e $g(.)$. No entanto, têm sido amplamente demonstradas as suas fragilidades no que toca à velocidade de convergência em muitas aplicações práticas.

Para ilustrar o algoritmo, suponha que observou uma amostra de uma qualquer variável aleatória ($n = 500$), cuja distribuição é dada pela Figura 20.

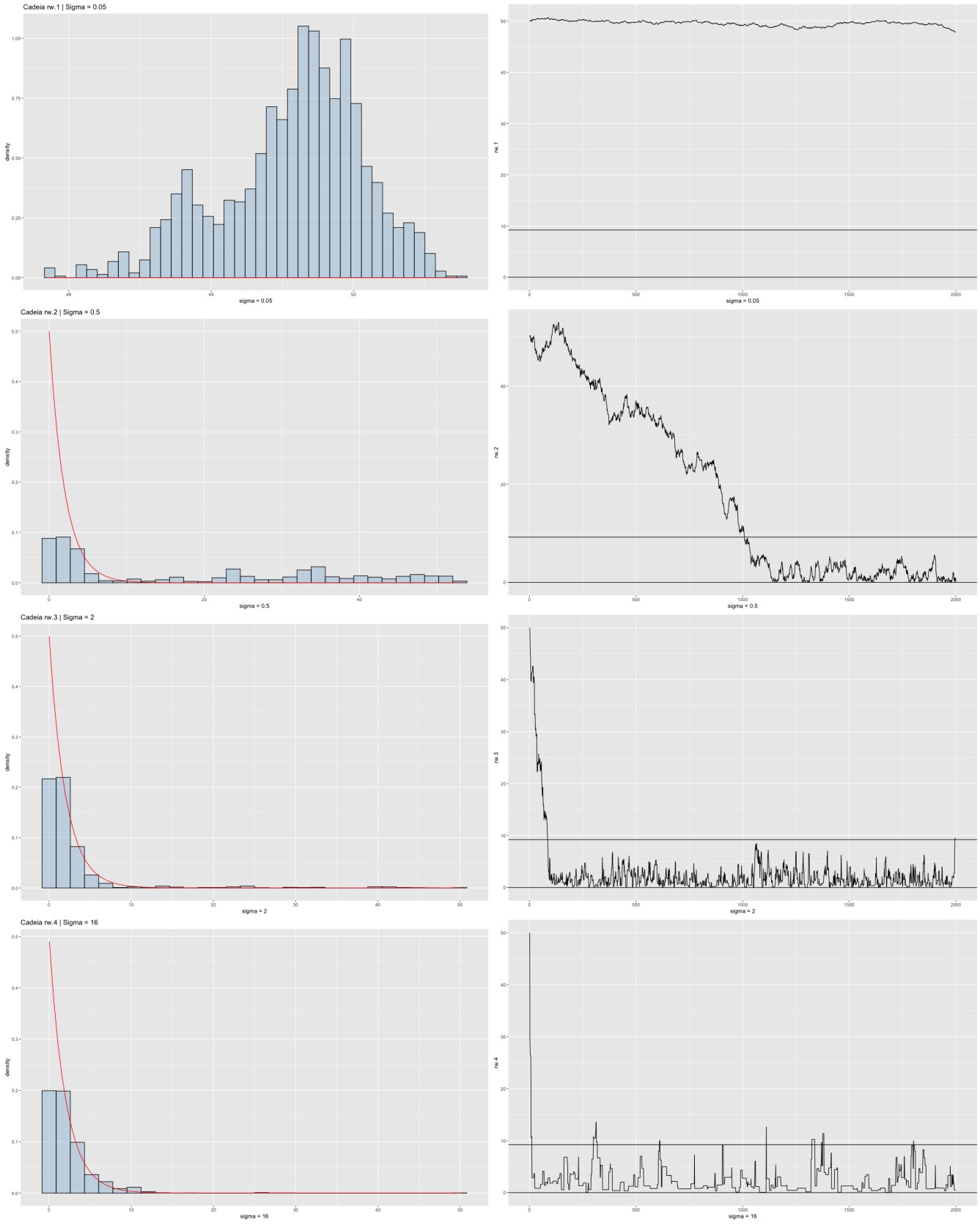

Figura 18: Densidade de quatro passeios aleatórios com diferentes desvios-padrão, crescentes de cima para baixo, após 2000 iterações. É visível o impacto de `sd` na velocidade de convergência.

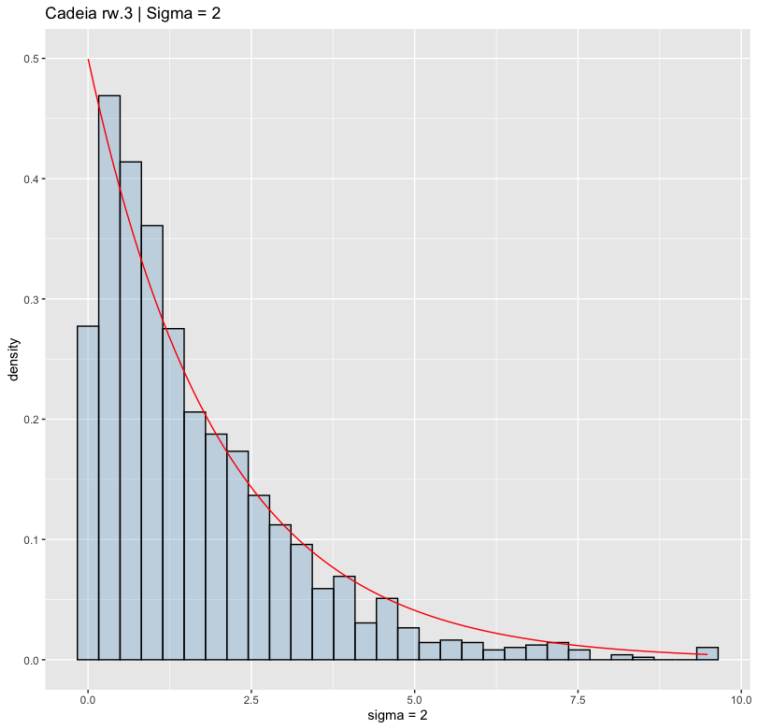

Figura 19: Densidade da terceira cadeia após desconsideração do *burn in period*. A melhor aproximação a ${\chi^2}_{df=2}$ parece indicar a convergência da cadeia.

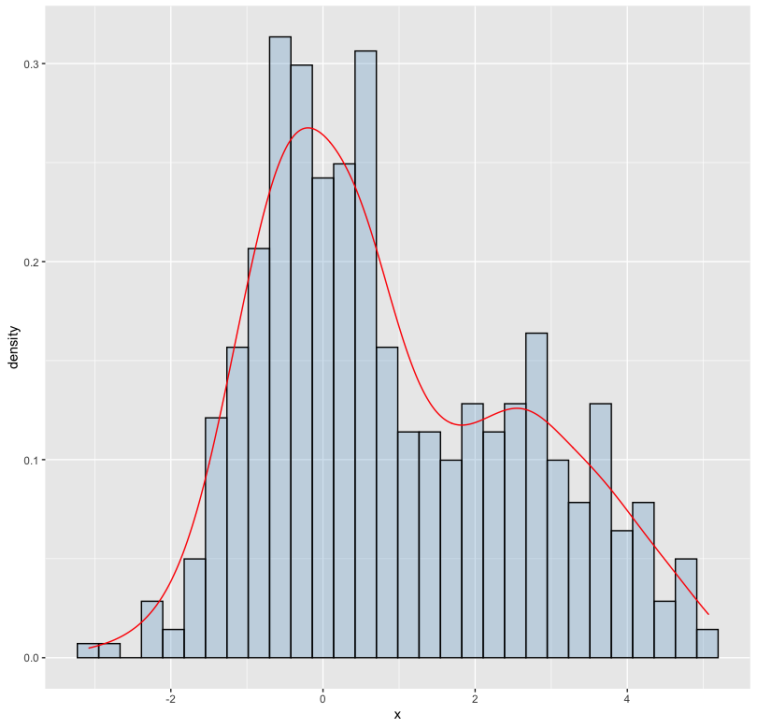

Figura 20: Densidade da amostra observada, que parece ser bem descrita por uma mistura de distribuições normais com médias 0 e 3.

Suponha também que, por inspecção dos dados, lhe parece razoável que a densidade da amostra seja descrita pela mistura de duas distribuições normais com médias $\mu_1 = 0$ e $\mu_2 = 3$, com variância $\sigma_1^2 = \sigma_2^2 = 1$. Com as componentes da mistura completamente especificadas, i.e. conhecendo os seus parâmetros, pode interessar-lhe encontrar os pesos θ e $(1 - \theta)$, que supomos desconhecer, atribuídos a cada componente. Relembre que uma mistura de distribuições pode ser expressa pela Equação 15, e que, no caso da mistura de duas distribuições normais pode ser descrita por

$$\theta N(\mu_1, \sigma_1^2) + (1 - \theta)N(\mu_2, \sigma_2^2), \tag{42}$$

em que θ é o peso da primeira componente, e a densidade da mistura é dada por

$$f^*(x) = \theta f_1(x) + (1 - \theta)f_2(x), \tag{43}$$

em que f_1 e f_2 são as densidades das duas distribuições normais, e em que $f^*(x)$ será a distribuição-alvo do Amostrador Independente. A mistura do nosso exemplo é então,

$$\theta N(0, 1) + (1 - \theta)N(3, 1) \tag{44}$$

Vamos utilizar o AI para propor uma estimativa para θ, com base na informação de que dispomos, utilizando a distribuição $U(0, 1)$ como distribuição proponente. Note que o domínio em (0,1) assume particular importância, uma vez que θ é uma probabilidade.

```
####### AMOSTRADOR INDEPENDENTE

#Geração da amostra observada
set.seed(2022)
mu <- c(0, 3)
sigma <- c(1,1)

i <- sample(1:2, size = 500, replace=TRUE, prob=(c(0.7, 0.3)))
x <- rnorm(500, mu[i], sigma[i])

#Densidade da amostra
plot1 <- ggplot(data.frame(x), aes(x=x)) +
  geom_histogram(aes(y=..density..), color="black", fill="skyblue3", alpha = 0.3, bins = 30) +
  geom_density(color = "red")

#Amostrador independente

ai <- function(data, l, mu, sigma){

#data é a amostra
#l é o número de iterações pretendidas
#mu e sigma são os parâmetros das componentes da mistura-alvo

X <- numeric(l)   #A cadeia
unis <- runif(l)  #Vector uniforme para a regra de aceitação
y <- runif(l)     #Distribuição proponente
X[1] = 0.5        #Inicializar o vector com p = 0.5

for(i in 2:l){

  #Densidades avaliadas em cada observação da amostra

      #Com o valor proposto
        fy <- y[i]       * dnorm(data, mu[1], sigma[1]) +
```

```
              (1-y[i])   * dnorm(data, mu[2], sigma[2])

      #Com o valor anterior
        fx <- X[i-1]     * dnorm(data, mu[1], sigma[1]) +
              (1-X[i-1]) * dnorm(data, mu[2], sigma[2])

      #Probabilidade de aceitação
      alpha <- prod(fy / fx) * (dunif(X[i-1]) / dunif(y[i]))

  #Geração da cadeia
  if (unis[i] <= alpha){
  X[i] <- y[i]}
  else
  {X[i] <- X[i-1]}
}
  #Devolução dos resultados - mean_theta é a proposta para theta
  return(list(x = X, mean_theta = mean(X)))
}
```

A função `ai` permite-nos estimar os pesos das componentes de uma mistura de duas distribuições normais, utilizando a distribuição $U(0,1)$ como distribuição proponente. Em particular, permite-nos obter uma estimativa para o peso "verdadeiro" (θ) da primeira componente, $N(0,1)$, que especifica o peso da segunda, $N(3,1)$. A Figura 21 mostra, à esquerda, a densidade dos valores propostos para θ. A distribuição é aproximadamente simétrica, com o seu centro em torno de 0.7. De facto, a amostra gerada com `set.seed(2022)` tem média 0.6933623, um valor muito próximo do valor que utilizámos para a nossa amostra supostamente observada (veja o início do código do algoritmo acima). A Cadeia de Markov e a Figura 21 podem ser obtidas correndo a função `ai` para $n = 10000$:

```
set.seed(2022)
ai_chain <- ai(x, 10000, mu = c(0,3), sigma = c(1,1))
ai_chain <- ai_chain$x[-(1:2000)]

plot2 <- ggplot(as.data.frame(ai_chain), aes(x=c(1:8000))) +
         geom_line(aes(y=ai_chain))

plot3 <- ggplot(as.data.frame(ai_chain), aes(x=ai_chain)) +
         geom_histogram(bins = 10, color = "black", fill = "skyblue3", alpha = 0.3)

require(gridExtra)
grid.arrange(plot3, plot2, ncol = 2)
```

Em resumo, a construção do AI permite-nos concluir acerca do peso a atribuir à primeira componente da mistura que queremos aproximar. Especificamente, escolhendo a média de $f(\theta|x)$ como valor para θ, a nossa mistura estimada seria dada, aproximadamente, por $0.69 * N(0,1) + 0.31 * N(3,1)$, muito semelhante à distribuição utilizada para gerar a amostra. Como exercício, sugere-se o uso de diferentes distribuições proponentes e monitorização do seu tempo de convergência.

5.1.3 O Amostrador de Gibbs

O Amostrador de Gibbs (AG) é o último caso particular do algoritmo MH que iremos abordar. Até agora, temos abordado a convergência de Cadeias de Markov numa perspectiva univariada (e.g. para a distribuição de um parâmetro ou de *uma* variável aleatória). A particularidade do AG advém da sua adequação a problemas de estimação que envolvem duas ou mais dimensões, e que por isso envolvem distribuições multivariadas. O algoritmo adapta-se particularmente bem a situações em que é difícil obter amostras da função densidade (ou massa) conjunta das k dimensões, mas em que dispomos das distribuições condicionais completamente

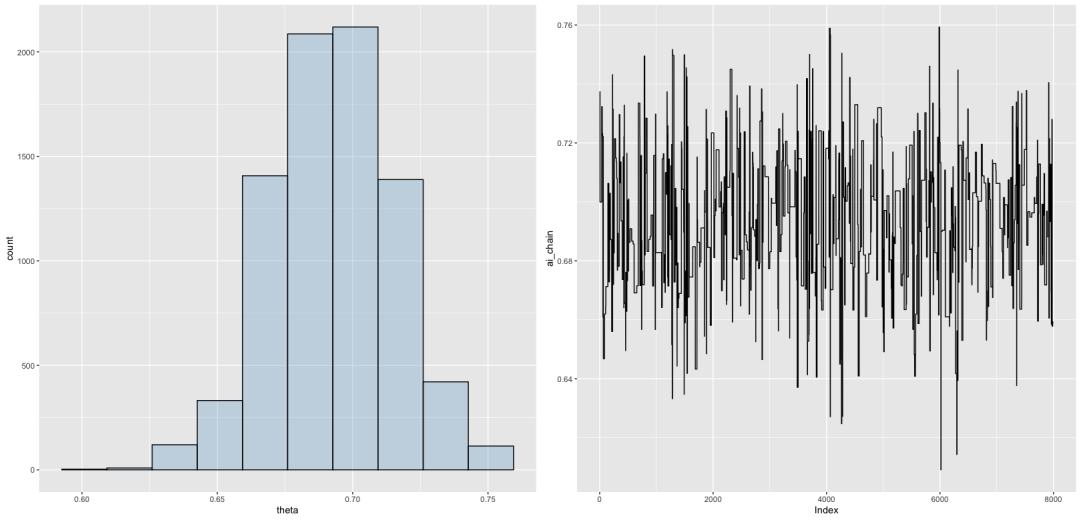

Figura 21: Densidade estimada de θ ($n = 8000$) e a cadeia gerada pelo Amostrador Independente, que aparenta convergir para $\theta = 0.7$.

especificadas.

Outro aspecto de interesse prende-se com a aceitação dos valores obtidos da distribuição proponente, que no AG são sempre aceites, por oposição ao caso geral de MH em que alguns valores propostos são rejeitados. Dada a grande prevalência de problemas de estimação multivariada, o AG é amplamente utilizado, e existem diferentes adaptações do algoritmo a tipos específicos de problema. Aqui, abordaremos apenas o Amostrador de Gibbs básico, e ilustraremos o seu funcionamento através de um exemplo simples, com uma solução analítica trivial, mas a que recorreremos por clareza.

Suponha que duas variáveis X e Y têm função de massa de probabilidade conjunta dada por:

$f(X,Y)$	**1**	**2**	**3**	**4**
1	0.05	0.03	0.12	0.10
2	0.30	0.15	0.02	0.03
3	0.01	0.02	0.03	0.04
4	0.04	0.04	0.01	0.01

Tabela 2: A distribuição de probabilidade conjunta $f(X,Y)$ a partir da qual queremos simular.

Na presença de uma variável aleatória bivariada, as observações são na realidades pares de valores (x, y). O elemento (1,1) da tabela significa que a probabilidade de obter o par $(X = 1, Y = 1)$ é igual a 0.05. No nosso caso, o espaço de estados é discreto, porque ambas as variáveis assumem valores, discretos, entre 1 e 4, inclusive. As distribuições condicionais estão completamente especificadas, porque se podem obter a partir da distribuição conjunta. Por exemplo, $f(Y|X = 1)$ é dada por:

$f(Y\|X = x)$	**1**	**2**	**3**	**4**
$X = 1$	0.17	0.10	0.40	0.33

Tabela 3: A distribuição condicional de Y dado que $X = 1$. Note que a soma das probabilidades é igual a 1.

notando que $P[Y = 1|X = 1] = 0.05/(0.05 + 0.03 + 0.12 + 0.1) = 0.17$. O AG permite-nos obter amostras sucessivas retiradas de cada distribuição condicional até à convergência para a distribuição estacionária

bivariada (se existir). Em seguida elencam-se os passos para implementação do AG para o caso bivariado, por clareza e sem perda de generalidade:

1. Inicializar a cadeia com um vector arbitrário em $t = 0 : (X_0 = x, Y_0 = y)$
2. A cada iteração $t = 1, 2, ...$
   - Simular X_t a partir de $P[X_t = x_t|Y = y_{t-1}]$
   - Simular Y_t a partir de $P[Y_t = y_t|X = x_t]$
   - Guardar os valores propostos x_t e y_t como estado da cadeia no momento t
   - Repetir até ao número de iterações desejadas

Quando o problema envolve a modelação de mais de duas variáveis, os passos acima estendem-se às variáveis seguintes, no sentido em que o AG retira amostras sucessivas de cada distribuição condicional. No `R`, o algoritmo para o nosso exemplo pode ser programado correndo o código

```
#Amostrador de Gibbs para duas v.a. 1:4
gibbs <- function(data, nreps, x0, y0){

  #A Cadeia
  XY <- matrix(0, nrow = nreps, ncol = 2)

  #Estado inicial
  XY[1,1] <- x0
  XY[1,2] <- y0

  #Geração da cadeia
  for (i in 2:nreps){

    y <- XY[i-1, 2]

    #Cálculo de f(x|y)
    cond_y    <- dist[,y]/sum(dist[,y])
    #Simulação de f(x|y)
    XY[i,1]   <- sample(1:4, 1, prob = cond_y)

    #Cálculo de f(y|x)
    cond_x    <- dist[XY[i,1],]/sum(XY[i,1])
    #Simulação de f(y|x)
    XY[i, 2]  <- sample(1:4, 1, prob = cond_x)
    }
    return(XY)
}
```

Implementada a função `gibbs`, podemos utilizá-la para gerar a Cadeia de Markov com 5000 iterações, depois de introduzir a função de probabilidade conjunta. Note que nas linhas finais se descartam as primeiras 1500 observações (*burn in*), e se controla a convergência, de forma indicativa, calculando os valores amostrais de $E[XY]$ e $COV(X, Y)$, próximos dos verdadeiros (a comparação e os cálculos dos parâmetros deixam-se como exercício):

```
###### AMOSTRADOR DE GIBBS

#Função de probabilidade conjunta (dada)
data <- c(0.05,  0.03, 0.12, 0.1,
          0.30,  0.15, 0.02, 0.03,
```

```
          0.01,  0.02, 0.03, 0.04,
          0.035, 0.04, 0.01, 0.015)

#Formatação dos dados
dist <- as.data.frame(t(matrix(data, nrow = 4)))
colnames(dist)  <- c(1:4)
row.names(dist) <- c(1:4)


> set.seed(2022)
> ag_chain <- gibbs(dist, 5000, 1, 1)
> ag_chain <- as.data.frame(ag_chain[-(1:1500),])
> colnames(ag_chain) <- c("X", "Y")
>
> mean(ag_chain[,1]*ag_chain[,2])
[1] 4.211714
> cov(ag_chain[,1], ag_chain[,2])
[1] -0.1585145
```

Concluída a abordagem aos três casos particulares do algoritmo de Metropolis-Hastings, encerramos o capítulo dedicado aos métodos MCMC. No capítulo seguinte, abordamos os métodos de *reamostragem*: uma classe de métodos de estimação que recorre a métodos de Monte Carlo, com várias aplicações, que incluem a estimação de parâmetros e a selecção de modelos.

6 Métodos de reamostragem

Os métodos de reamostragem (em inglês, *resampling*) datam do final dos anos 1970, e são uma classe de procedimentos para simular números aleatórios a partir de uma distribuição desconhecida. Por outras palavras, quando a única informação disponível é uma qualquer amostra de tamanho n, podemos utilizar estes métodos para aproximar a distribuição da variável aleatória e fazer inferência sobre os seus parâmetros. Neste capítulo, iremos abordar o conhecido método *bootstrap* e, em seguida, técnicas de validação cruzada para selecção de modelos.

6.1 Bootstrap

O procedimento geral envolve a simulação repetida (com reposição) de valores a partir da amostra obtida, o que equivale a utilizar a função de distribuição da amostra como estimador da função de distribuição da variável aleatória. Por isso, é importante assegurar que a amostra seja o mais representativa possível da população que a originou. Em muitas aplicações, este método é útil para a estimação de parâmetros e estatísticas associadas às distribuições amostrais dos seus estimadores. No entanto, e sobretudo se os custos associados à omissão de valores não incluídos na amostra forem elevados (e.g. valores extremos), a estimação via *bootstrap* poderá não produzir resultados que sirvam adequadamente a tomada de decisão. De forma ilustrada, o *bootstrap* consiste em gerar um histograma da distribuição da amostra e simular outras amostras a partir desse histograma.

Suponha que obteve a seguinte amostra ($n = 10$), já ordenada, e que está interessado em conhecer a distribuição da variável aleatória X que a gerou:

```
> set.seed(2022)
>smpl <- sort(rbinom(n, 20, prob = 0.7))

>[1] 12 13 13 13 14 15 16 16 17 17
```

É fácil verificar que seleccionando repetidamente e de forma uniforme elementos da amostra, obteremos o valor 12 em aproximadamente 10% das iterações, o valor 13 em 30% das iterações, etc. Assim, 5 amostras possíveis com $n = 20$ via *bootstrap* são

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]   13   13   13   17   17   13   14   17   17    15
[2,]   16   17   13   16   13   13   15   17   15    14
[3,]   13   13   14   16   17   17   16   16   16    13
[4,]   15   15   15   14   17   13   17   14   16    16
[5,]   13   13   17   16   16   13   13   12   17    14

     [,11] [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20]
[1,]   15    17    16    13    13    12    17    12    16    14
[2,]   17    16    17    17    15    13    15    17    16    15
[3,]   17    17    14    15    13    15    16    16    13    17
[4,]   16    16    14    14    13    16    13    13    17    15
[5,]   12    13    15    15    17    13    15    13    17    12
```

e podem ser obtidas correndo o código

```
#########BOOTSTRAP

set.seed(2022)

#Dimensão da amostra obtida
n <- 10
```

```
#Amostra n=10 obtida a partir da dist. bin. (supostamente desconhecida)
smpl <- sort(rbinom(n, 20, prob = 0.7))

#Cálculo das frequências relativas para amostragem
probs <- matrix(table(smpl)/n, nrow = 1)

#Dimensão das amostras a gerar
size <- 20

#Gerar N amostras
N <- 5
S <- matrix(nrow = 1, ncol = size)

for (j in 1:N){

#Criação de vector para guardar as simulações
X <- matrix(ncol = size)

#Geração da amostra
for (i in 1:size){
  X[i] = sample(unique(smpl), 1, prob = probs)
}
S <- rbind(S,X)
}
S <- S[-1,]
```

Suponha, agora, que está interessado na média populacional $E[X] = \mu$. Para estimar μ, podemos obter a média amostral de cada uma das 5 amostras geradas e propor a média dessas médias para o valor de μ (de acordo com o Teorema do Limite Central). Note que a amostra foi gerada a partir de uma distribuição binomial com $n = 20$ e $p = 0.7$, pelo que $E[X] = np = 14$. Ou seja, a nossa proposta de $\bar{X}_G = 14.9$ está perto do valor de μ.

```
> rowMeans(S)
[1] 14.70 15.35 15.20 14.95 14.30

> mean(rowMeans(S))
[1] 14.9
```

Gerando 1000 amostras de $n = 20$, em vez de 5 como fizemos anteriormente, podemos obter as respectivas 1000 médias amostrais, e inspeccionar a sua densidade, conforme se mostra na Figura 22. As simulações foram obtidas ajustando o procedimento acima, pelo que se omite o código. Note que, apesar de a forma da distribuição parecer adequadamente normal, o centro da distribuição não convergiu para o valor verdadeiro, sendo natural que isso ocorra, uma vez que a amostra obtida inicialmente não contém todos os valores [0-20] que podem ser assumidos por X. Assim, o exemplo remete bem para os cuidados a ter quando se toma a distribuição da amostra como aproximação da distribuição real. A média $\bar{X}_G$ das 1000 médias amostrais simuladas é igual a 14.6033. O conjunto de passos que seguimos até aqui é uma aplicação típica dos métodos *bootstrap*.

Assim, o algoritmo *bootstrap* clássico (há outros) para obtenção de uma amostra de dimensão n pode implementar-se com os passos seguintes:

1. Observar uma amostra de dimensão k, $X^* = (x_1, ..., x_k)$
2. Para cada iteração $i = 1, 2, ..., n$

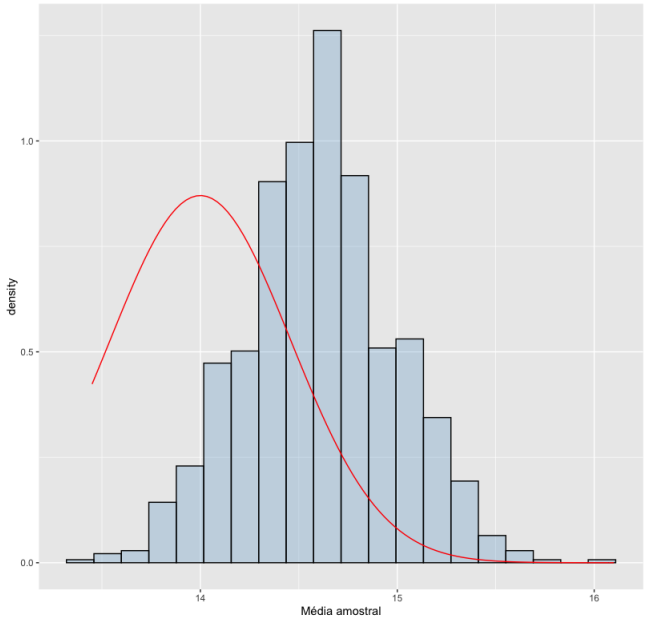

Figura 22: Distribuição estimada de $\bar{X}$, com base em 1000 amostras ($n = 20$). Note que o centro da distribuição não convergiu para $E[X] = np = 14$ por insuficiência da informação na amostra. A linha a vermelho representa a "densidade" real de $\bar{X}$.

- Seleccionar, de maneira uniforme e com reposição, um elemento da amostra ao acaso
- Guardar o elemento seleccionado
- Incrementar i

Em seguida, iremos abordar duas estatísticas típicas no contexto da estimação de parâmetros utilizando métodos *bootstrap*: o viés de um estimador e o seu erro-padrão. Para esse efeito, continuaremos a recorrer aos das 1000 amostras ($n = 20$) que simulámos acima.

6.1.1 Viés de um estimador via *bootstrap*

Relembre que o viés (em inglês, *bias*) de um estimador $\hat{\theta}$ para um parâmetro θ é dado pelo desvio médio das estimativas produzidas por $\hat{\theta}$ para o valor verdadeiro θ. Ou seja,

$$bias(\hat{\theta}) = E[\hat{\theta} - \theta] = E[\hat{\theta}] - \theta \tag{45}$$

Em geral, o não enviesamento (em inglês, *unbiasedness*) é uma característica desejável de um estimador, que se procura por construção. Na prática, isto significa, no nosso exemplo, que seria desejável poder assegurar que a média das estimativas produzidas por $\bar{X}_i$ a cada amostra i é na realidade igual a 14. No caso particular da média amostral para observações i.i.d., é fácil a verificação do seu não enviesamento para a média populacional μ:

$$bias(\bar{X}) = E[\bar{X}] - \mu = \frac{1}{n} E\left[\sum_{i}^{n} x_i\right] - \mu = (n/n)\mu - \mu = 0 \tag{46}$$

O leitor atento terá reparado que o cálculo do viés de um estimador envolve o conhecimento do verdadeiro valor do parâmetro que se propõe estimar. Na realidade, o problema de estimação coloca-se precisamente por desconhecimento do valor do parâmetro, pelo que o viés obtido via *bootstrap* é um viés também ele estimado, e dado por

$$\hat{bias}(\hat{\theta}) = \bar{\hat{\theta}}^* - \hat{\theta} \tag{47}$$

em que $\bar{\hat{\theta}}$ é a média das N estimativas obtidas para θ e $\hat{\theta}$ é o valor da estatística amostral com base na amostra observada. No nosso exemplo de 1000 amostras, se denotarmos a média da amostra inicial por $\bar{X}_I$, o viés estimado (Equação 47) é dado por

```
> mean(rowMeans(S)) - mean(smpl)
[1] 0.0033
```

Note que ao calcular o viés do estimador em relação a $\bar{X}_I$ (`mean(smpl)`) em vez de em relação a μ estamos de certa forma a aceitar que, apesar de desconhecermos μ, o comportamento do estimador será o mesmo em termos do seu valor esperado, independentemente do centro da distribuição amostral.

No nosso exemplo, $\bar{X}_G$ corresponde a `mean(rowMeans(S))`, e a estimativa para o enviesamento é 0.0033, o que, dada a escala aparente da variável aleatória, sugere o não enviesamento do estimador. Note que um viés positivo implica que o estimador tende a sobrestimar o valor verdadeiro, e vice-versa.

6.1.2 Erro Padrão de um estimador via *bootstrap*

A estimativa *bootstrap* para o erro-padrão de um estimador (em inglês, *standard error*) é dada simplesmente pelo desvio-padrão das N réplicas das estatísticas amostrais obtidas. No nosso exemplo, pode ser facilmente obtido para a média amostral através de

```
> sd(rowMeans(S))
[1] 0.3794576
```

em que a matriz `S` contém cada uma das 1000 amostras obtidas, `rowMeans(S)` são as 1000 médias amostrais e `sd(rowMeans(S))` é o erro-padrão estimado para a média amostral da variável de interesse X.

O erro-padrão é uma medida da variabilidade das estimativas produzidas por um qualquer estimador e, por isso, preferem-se estimadores cujo erro-padrão seja inferior. Formalmente, é dado por

$$\hat{se}(\hat{\theta}) = \sqrt{\frac{1}{(N-1)} \sum_{i=1}^{N} (\hat{\theta}_i - \bar{\hat{\theta}})^2} \tag{48}$$

em que $\hat{\theta}_i$ são as estimativas individuais da estatística amostral e $\bar{\hat{\theta}}$ é a média dessas mesmas estimativas.

No nosso exemplo, a estimativa $\hat{se}(\bar{X}) = 0.3794576$ não dista excessivamente do valor verdadeiro dado por $\sqrt{kpq/n} = \sqrt{20 * (0.7 * 0.3)/20} = 0.4582576$, tendo em conta a escala aparente da variável com média em torno de 14.

O erro-padrão é uma quantidade de especial interesse pela sua utilidade, uma vez que não só permite parametrizar a distribuição de uma estatística amostral, mas também recorrer a todos os métodos daí decorrentes, como sejam a estimação por intervalos de confiança ou a realização de testes de hipóteses paramétricos.

6.2 Métodos de validação cruzada: *n-fold* e *K-fold*

Os métodos de validação cruzada que abordaremos são técnicas de reamostragem tipicamente utilizadas para a quantificação de erros de estimação. Em geral, estas técnicas consistem na partição de uma amostra num conjunto de *treino*, com base no qual se procede à estimação da estatística ou modelo de interesse, e um conjunto de *teste* com base no qual se afere, quantificando, o erro de previsão da estatística ou modelo estimado.

A validação cruzada tem várias aplicações: a técnica conhecida como *jackknife*, por exemplo, é útil para o cálculo do viés e do erro-padrão de estimadores, e é uma alternativa ao *bootstrap*. No contexto de estimação de modelos (e.g. regressão) os resultados produzidos por estes métodos podem ser utilizados como critério de escolha do modelo mais adequado para descrever o fenómeno de interesse, seleccionando-o de um conjunto de modelos competidores. É neste último contexto que em seguida ilustramos os conceitos de validação cruzada *n-fold* e *K-fold*.

6.2.1 Validação cruzada *n-fold* (leave-one-out)

Suponha que se observaram os dados da Figura 23, relativamente às variáveis aleatórias X e Y, e que se pretende estimar um modelo de regressão que descreva adequadamente o processo de geração de Y com base em X. Para esse efeito, propõem-se dois modelos (linear e quadrático), com o objectivo de escolher aquele que, de entre os dois, produza um erro de previsão menor:

1. Linear: $Y = \beta_0 + \beta_1 X + \epsilon$
2. Quadrático: $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon$

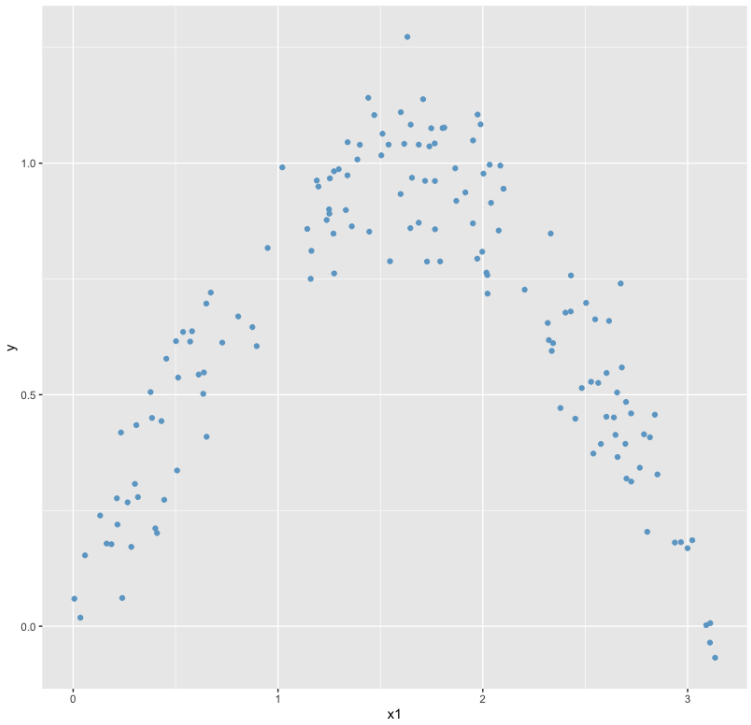

Figura 23: Amostra observada ($n = 150$) do par (X, Y). A relação entre as variáveis não aparenta ser linear.

Através da inspecção visual da Figura 24 é evidente que uma forma quadrática parece ajustar-se melhor à natureza dos dados do que uma forma linear. No entanto, para quantificar a bondade do ajustamento de cada modelo, podemos recorrer ao método de validação cruzada *n-fold*, cujos passos se descrevem em seguida:

1. Para $k = 1, ..., n$, definir (x_k, y_k) como ponto de teste, onde n é a dimensão da amostra observada
   - Estimar o modelo utilizando apenas as $n - 1$ observações, ou seja (x_i, y_i), com $i \neq k$

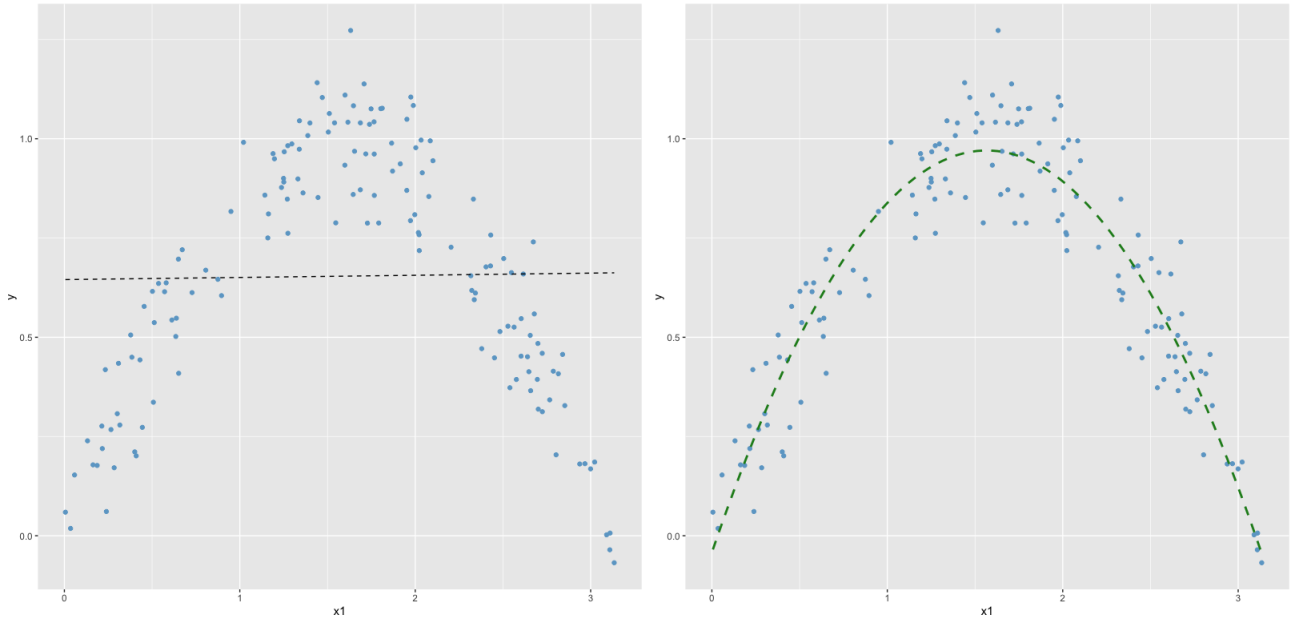

Figura 24: Modelos candidatos para modelar $Y \sim X$: Linear (à esquerda) e Quadrático (à direita).

- Calcular a previsão do modelo para o ponto de teste (x_k, y_k)
- Calcular o erro de previsão para o ponto de teste, $e_k = y_k - \hat{y}_k$

2. Estimar a média aritmética dos quadrados dos erros $\hat{\sigma}^2_\epsilon = \frac{1}{n}\sum_{k=1}^{n} e_k^2$

Na prática, trata-se de obter n amostras em que cada amostra corresponde à amostra original excluindo as observações à vez (daí a designação alternativa *leave-one-out* para este método). Para cada uma das n amostras, estima-se o modelo sob avaliação, e utiliza-se o ponto excluído para "perguntar" ao modelo qual seria a sua previsão para y_k dado que $X = x_k$. Ao cálculo da distância entre a estimativa e o ponto de teste $(e_k = y_k - \hat{y}_k)$ chamamos *erro de previsão*. Note que, ao estimar n vezes o modelo, obtém os correspondentes n erros de previsão. O cálculo da média dos quadrados destes erros é comummente utilizado como critério para comparação com modelos competidores, escolhendo-se, de um conjunto, o modelo com menor erro quadrático médio de previsão como o melhor ajustado. A Figura 24 exibe os modelos linear e quadrático ajustados aos dados observados. O código abaixo permite ajustar os modelos e quantificar o erro de previsão utilizando validação cruzada do tipo *leave-one-out*:

```
#################### VALIDAÇÃO CRUZADA

###LEAVE-ONE-OUT


#Simulação dos dados observados
set.seed(2022)
n <- 150
x1 <- runif(n, 0, pi)
s <- data.frame(x1 = x1, y = rnorm(n, 0, 0.1) + sin(x1))

#Vector para guardar os erros de previsão
errors <- matrix(nrow = n, ncol = 2)
```

```
#Validação cruzada leave-one-out
for (k in 1:n){

lin <- lm(y[-k] ~ x1[-k], s)                #Estimação do modelo linear sem k
qua <- lm(y[-k] ~ x1[-k] + I(x1[-k]^2), s)  #Estimação do modelo quadrático sem k

#Previsão linear para o ponto de teste
y_l <- lin$coefficients[1] + lin$coefficients[2] * s[k,1]

#Previsão quadrática para o ponto de teste
y_q <- qua$coefficients[1] + qua$coefficients[2] * s[k,1] + qua$coefficients[3] * s[k,1]^2

errors[k,1] = (s[k,2] - y_l)^2 #Quadrado do erro linear
errors[k,2] = (s[k,2] - y_q)^2 #Quadrado do erro quadrático
}
```

A estimativa $\hat{\sigma}^2_\epsilon = \frac{1}{n}\sum_{k=1}^{n} e_k^2$ para cada um dos modelos é:

```
> #Resultados
> print(c(paste("Linear:", mean(errors[,1])), paste("Quadrático:", mean(errors[,2]))))
[1] "Linear: 0.102424108598359"     "Quadrático: 0.012281204884293"
```

Como era expectável através da inspecção visual, o modelo linear apresenta um erro de previsão maior, dado que a recta proposta se ajusta particularmente mal aos dados observados. O modelo quadrático, por sua vez, apresenta um erro de previsão cerca de 9 vezes inferior ao do linear.

Assim, a utilização deste critério recomendaria a escolha do modelo quadrático como o melhor ajustado à amostra e, espera-se, à descrição da relação entre X e Y.

6.2.2 Validação cruzada K-*fold*

A validação cruzada K-*fold* pode ser vista como um caso geral da validação n-*fold*. O método consiste em dividir aleatoriamente a amostra observada em $K = 1, ..., k$ grupos, estimando K vezes o modelo sob avaliação excluindo os dados do grupo k. O dados excluídos são utilizados à vez para testar a bondade de ajustamento do modelo. Repare que a dimensão da amostra no nosso exemplo é $n = 150$ e, portanto, validação cruzada K-*fold* onde $K = 150$ equivale neste caso à validação cruzada n-*fold*.

O pacote `cvTools` permite-nos fazer a validação cruzada sem necessidade de programar o algoritmo de raiz (os dados utilizados no código abaixo são os mesmos do exemplo anterior cuja geração se repete por clareza):

```
###K-FOLD

library(cvTools)

#Simulação dos dados observados
set.seed(2022)
n <- 150
x1 <- runif(n, 0, pi)
s <- data.frame(x1 = x1, y = rnorm(n, 0, 0.1) + sin(x1))

#Partição aleatória da amostra em 5 grupos
grupos <- cvFolds(nrow(s), K = 5, R = 1)

#Estimação do modelo linear
lin <- lm(y ~ x1, data = s)
```

```
#Estimação do modelo quadrático
qua <- lm(y ~ x1 + I(x1^2), data = s)

#Cálculo do erro quadrático médio de previsão para cada modelo
cv_lin <- cvLm(lin, cost = mspe, folds = grupos, seed = 2022)
cv_qua <- cvLm(qua, cost = mspe, folds = grupos, seed = 2022)
```

Aqui, optou-se por dividir os dados em 5 grupos. Os resultados podem obter-se correndo

```
> #Resultados
> print(c(cv_lin$cv, cv_qua$cv))
        CV         CV
0.10118665 0.01244026
```

Repare na semelhança entre estes resultados e aqueles que obtivemos utilizando a validação *n-fold*.

7 Notas Finais

Os métodos que abordámos neste texto têm um objectivo geral comum: a estimação da distribuição de variáveis aleatórias. A implementação dos algoritmos resulta na simulação mais ou menos rápida de amostras a partir da distribuição pretendida, e é nesse acto de simulação que se dá a convergência para o resultado pretendido.

A escolha da ferramenta de estimação a utilizar depende da natureza do problema, e por isso a nenhum dos conceitos que abordámos se aplica a expressão *one size fits all*. Os resultados produzidos pelos modelos devem ser sempre sujeitos a técnicas de validação que controlem a sua estabilidade.

O que é válido para as técnicas de modelação em geral, também é válido aqui. Os resultados dos modelos dependem por um lado da qualidade dos dados em que se baseiam, e por outro na fiabilidade da técnica de estimação escolhida. Como tal, a compreensão adequada da natureza do problema deve ser encarada como um passo fulcral no uso destes métodos.

A modelação estocástica (e a Ciência de Dados em geral), é um campo fértil de investigação científica. O acesso aos conteúdos para a uma aprendizagem introdutória é fácil, sendo tão úteis as comunidades tipo `stackexchange` como as explicações disponíveis no YouTube, Wikipedia e outros *websites*. Nestes casos, é muitas vezes necessário confrontar informação de fontes distintas, por diferenças de notação ou resultados contraditórios.

Para um estudo mais guiado e profundo, referem-se os livros *Estatística Bayesiana*, 2ª ed., de Paulino et. al [ed. Fundação Calouste Gulbenkian, 2018], *Statistical Computing with R*, 2ª ed., de Rizzo [ed. CRC Press, 2019] e *Simulation for Data Science with R*, 1ª ed., de Templ [ed. PACKT Publishing, 2016]. Todos estes livros disponiblizam, por sua vez, um conjunto vasto de referências e de aplicações reais das técnicas demonstradas.