OTIMIZAÇÃO POR MÁXIMA VEROSSIMILHANÇA DA DISTRIBUIÇÃO WEIBULL VIA SIMULAÇÃO

THOMÁS FREUD DE MORAIS GONÇALVES

Resumo. O objetivo desse trabalho é estimar por máxima verossimilhança os parâmetros da distribuição de probabilidade Weibull(b, c) usando simulação de Monte Carlo. Como o sistema de equações a ser resolvido não é linear, será elaborado um programa para otimização da função objetivo utilizando as linguagens `C` e `R`. Adicionalmente, empregar-se-á o método da transformada inversa para gerar sequências pseudoaleatórias com distribuição Weibull e um gerador congruencial misto para a obtenção de sequências de números pseudoaleatórios com distribuição uniforme, os quais serão necessários para gerar dados simulados a serem empregados no processo de otimização.

1. Introdução

A distribuição de probabilidade Weibull é obtida a partir de uma transformação aplicada à variáveis aleatórias exponenciais. É uma modelo bastante explorado na literatura e possui muitas aplicações em diversas áreas da ciência, tais como na previsão de terremotos [1], na resistência de materiais [2], na análise de sobrevivência [3] e em seguros [4]. Na referência [5, capitulo 7] consta um rico elenco de trabalhos sobre o assunto, separados por áreas de aplicação. O método de estimação por verossimilhança dos parâmetros da distribuição Weibull é um tema bastante estudado [6, 7].

Define-se uma variável aleatória $x \geq 0$ com distribuição Weibull, com parâmetro de forma c e de escala b, tal que $(b, c) \in \mathbb{R}^2_+$, com as respectivas funções de densidade e de probabilidade acumulada dadas, respectivamente, por

$$f(x; b, c) = \frac{cx^{c-1}}{b^c} \exp\left[-\left(\frac{x}{b}\right)^c\right], \tag{1}$$

$$F(x; b, c) = 1 - \exp\left(-\frac{x}{b}\right)^c. \tag{2}$$

Considerando um vetor de variáveis aleatórias $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$, a função de log-verossimilhança ℓ relativa a (1), onde $\ell : \mathbb{R}^2_+ \to \mathbb{R}$, é dada por

$$\ell(b, c) = n \ln c - nc \ln b + (c-1) \sum_{i=1}^{n} \ln x_i - \sum_{i=1}^{n} \left(\frac{x_i}{b}\right)^c. \tag{3}$$

Calculando as derivadas parciais de (3), em relação aos parâmetros (b, c), obtém-se as coordenadas do gradiente da função.

$$\frac{\partial \ell(b, c)}{\partial c} = \frac{n}{c} - n \ln b + \sum_{i=1}^{n} \ln x_i - \sum_{i=1}^{n} \left(\frac{x_i}{b}\right)^c \ln\left(\frac{x_i}{b}\right), \tag{4}$$

$$\frac{\partial \ell(b, c)}{\partial b} = \frac{-nc}{b} + \sum_{i=1}^{n} \frac{cx_i^c}{b^{c+1}}. \tag{5}$$

Date: 15 de Março de 2020.

Os estimadores $\hat{c}$ e $\hat{b}$ são calculados encontrando os valores de c e b para os quais (3) apresenta valor funcional máximo. Para tanto, basta igualar (4) e (5) à zero e resolver o sistema não linear que se forma. Como não existe solução fechada para o sistema de equações apresentado, recorre-se a um método numérico de otimização. Nesse trabalho é empregado o método BFGS, sobre o assunto recomenda-se [8, capitulo 6].

2. Metodologia

A função quantil $y : \mathbb{R} \to \mathbb{R}$, obtida a partir de (2), dada por (6), será usada para gerar as coordenadas do vetor $\mathbf{x}$ de amostras aleatórias. Os valores $p \in (0, 1]$ serão gerados por um gerador congruencial misto dado por (7).

$$y(p; b, c) = c(-\ln(1-p))^{\frac{1}{b}}, \tag{6}$$

$$z_{n+1} = (az_n + c) \mod M. \tag{7}$$

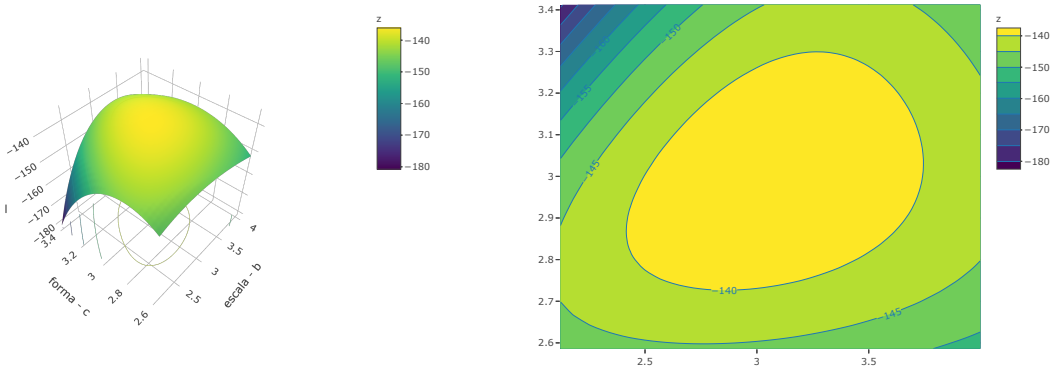
De posse dessas equações, a abordagem consiste em gerar, usando (7), um vetor $\mathbf{p} = [p_1, \ldots, p_n]^T$ com $p_i \sim U(0, 1)$ para $i = 1, \ldots, n$. Com isso, faz-se gerar um vetor $\mathbf{x}$ com as coordenadas $x_i \sim \text{Weibull}(b, c)$ para $i = 1, \ldots, n$. Nesse sentido, pode-se pensar em $\mathbf{x}$ como sendo a imagem do funcional de n coordenadas dado em (8).

$$\mathbf{x(p)} = \langle y(p_1; b, c), \ldots, y(p_n; b, c)\rangle. \tag{8}$$

Com isso, as equações (3), (4) e (5) serão definidas para o vetor $\mathbf{x}$, com n coordenadas, e a otimização será processada.

3. Simulação e Resultados

Considere um vetor aleatório $\mathbf{x}$ contendo n variáveis aleatórias Weibull com parâmetros $b = c = 3$ e $n = 100$. Gerou-se o gráfico da função em (3) considerando essas especificações, com a superfície e o mapa de contorno na Figura 1. Estamos diante de uma função côncava, *i.e.* duplamente diferenciável, indicando que a convergência no processo de otimização não será um problema. Nota-se, pela superfície e pelo platô que se forma no mapa de contorno, que os valores máximos de b, c estão próximos à 3, como esperado.



(A) Superfície da função de log-verossimilhança de uma distribuição Weibull.

(B) Mapa de contorno da função de log-verossimilhança de uma distribuição Weibull.

Figura 1. Superfície de log-verossimilhança e mapa de contorno de uma distribuição Weibull $c = b = 3$.

Como o objetivo é estimar os valores de b e c, para execução da otimização, é preciso definir um palpite inicial $\mathbf{g} = (b_0, c_0)$. Uma estratégia interessante seria escrever b e c como

funções da média e variância amostrais, depois disso, usar as amostras pra determinar um palpite inicial. Entretanto, no caso de uma distribuição Weibull, essa estratégia não seria viável, pois não existe forma fechada para escrever b ou c como funções da média e variância amostrais, o que significaria ter de fazer uma otimização não linear para isso, implicando, ainda assim, na necessidade de se definir um palpite inicial para um dos parâmetros. Mas como o domínio da função em análise é o $\mathbb{R}^2_+$, contorna-se questões tais fazendo-se uso do gráfico de contorno, a exemplo daquele que consta na Figura 1.

Para compor os resultados sumarizados na Tabela 1, foram executadas 10.000 réplicas de Monte Carlo para amostras fixadas em tamanhos de 10, 15, 25, 50 e 100. Além disso, foram calculadas a média, variância, assimetria e curtose de cada estimador, para cada tamanho de amostra, bem como o viés relativo em escala porcentual.

Estimador	Amostra	Média	Variância	Assimetria	Curtose	Viés %
$\hat{c}$	10	3.42	0.857	1.520	8.31	13.87
$\hat{b}$	10	2.98	0.092	0.021	3.06	-0.60
$\hat{c}$	15	3.28	0.507	1.150	5.71	9.20
$\hat{b}$	15	2.99	0.066	0.034	3.07	-0.38
$\hat{c}$	25	3.17	0.273	0.860	4.64	5.56
$\hat{b}$	25	2.99	0.042	-0.013	2.97	-0.24
$\hat{c}$	50	3.08	0.121	0.511	3.53	2.69
$\hat{b}$	50	3.00	0.022	-0.079	2.96	-0.15
$\hat{c}$	100	3.04	0.057	0.253	2.99	1.24
$\hat{b}$	100	3.00	0.011	-0.094	2.81	-0.09

TABELA 1. Resultados da simulação para cinco tamanhos diferentes de amostras.

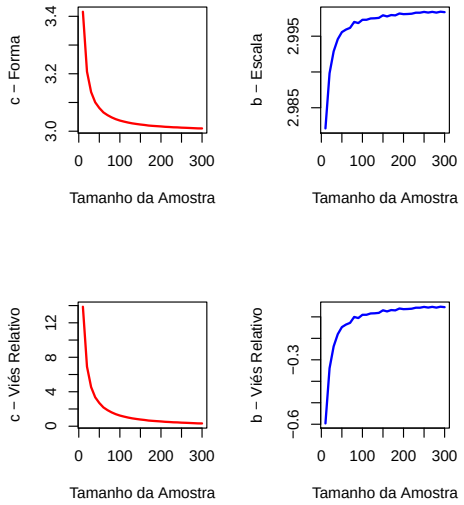


FIGURA 2. Estimadores e Viés relativo (%).

Observa-se que, conforme se aumenta o tamanho da amostra, a média dos valores dos estimadores tende para 3.00. Além disso, verifica-se que a variância dos estimadores é decrescente, e que o estimador $\hat{c}$, além de apresentar maior variância, a qual também decai

mais lentamente. O maior viés relativo pertence a $\hat{c}$, e assim como ocorre com a variância, aqui ele também diminui mais lentamente, o mesmo ocorre com a assimetria e a curtose.

De acordo com a Figura 2, conforme se aumenta o tamanho da amostra, os valores dos estimadores convergem, com o viés relativo tendendo à zero. Nessa figura, as amostras foram geradas em múltiplos de 10, partindo de 10 até 300. No mais, visualiza-se o comportamento do viés relativo do estimador $\hat{c}$, observando-se valores consideravelmente distantes de zero (de 2% à 13%) para amostras com tamanho inferior à 60, comparado ao viés relativo de $\hat{b}$ (ver Tabela 1).

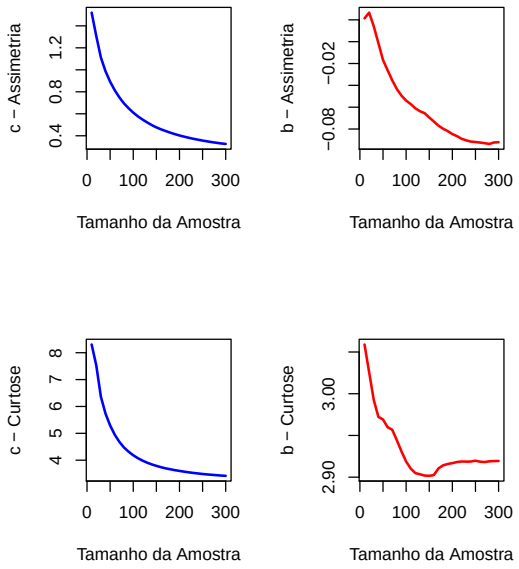


FIGURA 3. Assimetria e curtose dos estimadores.

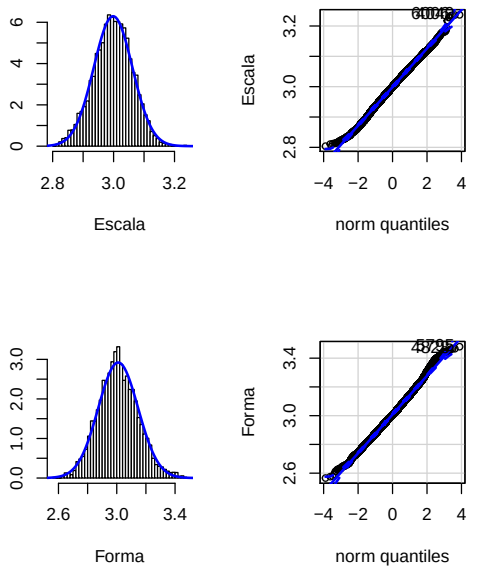


FIGURA 4. Histogramas e q-q plot para amostra de tamanho 300.

Na Figura 3 constam as medidas de assimetria e curtose de cada estimador. Observa-se, conforme aumenta o tamanho da amostra, que a assimetria tende à zero e a curtose tende à 3, mas no caso do estimador $\hat{c}$, essa convergência é mais lenta, *i.e* requer amostras maiores. Considerando esses resultados e aqueles que constam na Figura 4, verifica-se que a distribuição assintótica dos estimadores é normal.

4. Considerações sobre os programas implementados

Os programas foram escritos em linguagem `C`[1] e `R`[2]. Na linguagem `C`, foi adotada a biblioteca de computação numérica *gsl - gnu scientific library* para realizar as otimizações necessárias. O método de otimização adotado foi o *BFGS - Broyden-Fletcher-Goldfarb-Shanno*, pertencente à classe dos algoritmos do tipo quasi-Newton, implementado na *gsl*. Entretanto, optou-se por implementar a função quantil da distribuição Weibull e o gerador congruencial misto para geração de sequências pseudoaleatórias, conforme apresentado anteriormente. Em ambos os programas, o vetor de variáveis aleatórias uniformes $\mathbf{p}$, necessárias para gerar o vetor de amostras $\mathbf{x}$, conforme o funcional em (8), é gerado fora do laço de Monte Carlo.

Seja n o número de réplicas de Monte Carlo e seja j o tamanho da amostra a ser gerada em cada uma das n repetições, com $j \leq n$ e $|\mathbf{x}| = j$. Como as coordenadas de $\mathbf{p}$ serão usadas para gerar todas as amostras $\mathbf{x}$ ao longo das n repetições, e como cada amostra tem tamanho j, então serão necessários nj números uniformes, ou seja, $|\mathbf{p}| = nj$ com coordenadas $p_i \sim U(0,1)$, para gerar todas as variáveis Weibull a serem utilizadas nas n otimizações.

Com isso, a cada repetição de Monte Carlo, são selecionados j elementos de $\mathbf{p}$ e, a partir da equação (8), é gerado um vetor de amostras $\mathbf{x} = [x_1, \ldots, x_j]^T$, onde $x_i \sim Weibull(c, b)$, usado no processo de otimização não linear.

Para garantir que cada elemento de $\mathbf{p}$ seja selecionado uma única vez, um índice $\text{id} = s(j-1)$ com $s \in \{0, \ldots, n-1\}$ foi adotado. O que esse índice faz é mapear subconjuntos de $\mathbf{p}$, cada um com tamanho j, garantindo que, em cada repetição, o conjunto subsequente de tamanho j seja selecionado (em programação isso é denominado *offset*). Além disso, em cada repetição, a semente correspondente à última posição do subconjunto dos j elementos é armazenada e, sempre que há falha de convergência, o vetor $\mathbf{p}$ é destruído e, a partir da semente que fora armazenada, é gerado um novo $\mathbf{p}$ com $|\mathbf{p}| = (n-i)j$ onde i corresponde a repetição de Monte Carlo naquele instante. No caso de falha de convergência, s é reconfigurando para zero e o processo de incremento reinicia, mantendo a consistência do programa.

Em ambos os programas essa abordagem foi mantida, com algumas diferenças devidas às particularidades de cada linguagem. Como esperado, o programa executado em `C` apresenta melhor performance de execução em relação ao programa em `R` e essa diferença se amplia conforme aumenta o tamanho das amostras. Entretanto, a função `optim()` utilizada para otimizações em `R` apresenta excelente desempenho e não exige muito trabalho por parte do usuário. Na *gsl*, por outro lado, algumas cautelas devem ser tomadas para garantir o melhor desempenho. Ao configurar a função `gsl_multimin_fdfminimizer_set()`, os parâmetros `step_size`, o tamanho do incremento que vai definir a primeira estimativa dos parâmetros da função objetivo, e `tol`, a tolerância, têm bastante impacto sobre o número de falhas de convergência e, portanto, sobre desempenho da otimização. Verificou-se que para diferentes valores de c e b, mantendo os mesmos `step_size` e `tol`, a performance da otimização varia bastante, em algumas circunstâncias perdendo bastante

[1]Disponivel em: https://github.com/thfreud/NumericalStatistics

[2]Disponível em: https://github.com/thfreud/NumericalStatisticsR

desempenho, aumentando as falhas de convergência, em alguns casos, em mais de 70% e fazendo com que o programa em `C` entregue um desempenho inferior a versão em `R`.

Depois de alguns testes, identificou-se que, para os valores de $c = b = 3$, definindo `step_size = 1e-10` e `tol = 1e-1`, a otimização apresenta máxima performance, com zero falhas de convergência para tamanhos de amostras inferiores à 1000, superando a versão em `R`. É importante notar que uma tolerância alta pode fazer com que as estimativas obtidas pela otimização fiquem distantes do ótimo, mas uma tolerância muito baixa pode comprometer a convergência do algoritmo. Já o tamanho do salto inicial pode implicar em falhas de convergência sobretudo se o palpite inicial for bem próximo aos valores reais dos parâmetros da função ou se as derivadas da função a ser otimizada tendem à zero muito lentamente em uma região próxima aos extremos da função.

Amostra	Tempo de Execução R	Tempo de Execução C
10	3.77	2.98
30	5.36	3.88
50	7.04	5.30
70	8.7	6.91
100	10.88	9.19
1000	90.10	80.73

TABELA 2. Comparativo entre tempos (em segundos) de execução dos programas em R e C.

Na Tabela 2 pode-se comparar o desempenho dos programas em cada uma das versões. Como foi dito anteriormente, a versão em `C` entregou uma performance superior, sobretudo para o maior tamanho de amostra, já que a diferença entre os tempos de execução aumentou conforme a amostra foi aumentada. Outro fator de grande influência para a velocidade de convergência da otimização é o chute inicial $\mathbf{g} = (c_0, b_0)$. Naturalmente, quanto mais próximo o chute inicial for dos valores reais dos parâmetros da função, mais rápido ocorre a convergência, mas no caso da *gsl*, em algumas circunstâncias, chutes iniciais mais distantes dos pontos ótimos não aumentaram as falhas de convergência, enquanto alguns valores mais próximos implicaram em aumento nas falhas de convergência. Todavia, nessas situações, ajustes pertinentes em `step_size` e `tol` contornaram o problema.

5. Considerações Finais

Usar simulação de Monte Carlo para fazer otimização por máxima verossimilhança da distribuição Weibull não é demasiadamente custoso, sob o ponto de vista computacional. No entanto, conforme se aumenta o número de repetições de Monte Carlo e o tamanho da amostra, o tempo de execução se estende consideravelmente. Nesse sentido, o ideal é explorar oportunidades de otimizar o código, maximizando a performance, quando possível. No caso desse trabalho, gerar todos os números aleatórios necessários fora do laço de Monte Carlo melhorou consideravelmente o desempenho da aplicação em `R`. Na aplicação em `C`, essa estratégia foi adotada, em parte. Todavia, as falhas de convergência comprometem o desempenho do programa, dado que novas amostras devem ser geradas dentro do laço e como, a depender da configuração do programa de otimização, a taxa de falhas pode ser elevada, isso causa um desempenho precário, se negligenciado. Não houve esse problema em `R`, já que em todos os cenários analisados, houve zero falhas de convergência.

Uma maneira de atingir maiores velocidades na solução do problema abordado seria paralelizar o algoritmo, implementando-o em uma GPU. Como a aplicação consiste em

uma simulação, não haveria grandes problemas relacionados à latência da comunicação entre CPU e GPU, pois não seriam necessárias grandes transferências de dados entre as memórias dos dispositivos. Uma sugestão viável seria adotar as interfaces CUDA NVIDIA, a qual implementa uma biblioteca para lidar com álgebra linear `CUBLAS` e geração de sequências aleatórias `CURAND`. Usando essas bibliotecas, seria possível implementar um programa de otimização para ser executado em GPU. Para uma trabalho sobre a implementação do método *L-BFGS-B* em arquitetura paralela com GPU ver [9].

Por fim, reitera-se a importância de se verificar as possíveis variações de desempenho no processo de otimização quando se altera os valores de `step_size` e `tol`. Recordando que, a depender dos valores dos parâmetros da função objetivo, c e b no caso da distribuição Weibull, essas configurações devem ser reajustadas, além do palpite inicial, para garantir melhor taxa de sucesso em convergência.

Referências

[1] Yukio Hagiwara. Probability of earthquake occurrence as obtained from a weibull distribution analysis of crustal strain. *Tectonophysics*, 23(3):313–318, 1974.

[2] Kyriaki Corinna Datsiou and Mauro Overend. Weibull parameter estimation and goodness-of-fit for glass strength data. *Structural Safety*, 73:29–41, 2018.

[3] Kevin J Carroll. On the use and utility of the weibull model in the analysis of survival data. *Controlled clinical trials*, 24(6):682–701, 2003.

[4] Yiu-Kuen Tse. *Nonlife actuarial models: theory, methods and evaluation*. Cambridge University Press, 2009.

[5] Horst Rinne. *The Weibull distribution: a handbook*. CRC press, 2008.

[6] A. Clifford Cohen. Maximum likelihood estimation in the weibull distribution based on complete and on censored samples. *Technometrics*, 7(4):579–588, 1965.

[7] Richard L Smith and JC Naylor. A comparison of maximum likelihood and bayesian estimators for the three-parameter weibull distribution. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 36(3):358–369, 1987.

[8] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[9] Yun Fei, Guodong Rong, Bin Wang, and Wenping Wang. Parallel l-bfgs-b algorithm on gpu. *Computers & graphics*, 40:1–9, 2014.

Departamento de Estatística, Universidade Federal de Pernambuco, Pernambuco, Brasil