Propriedades e estimação de uma nova distribuição monoparamétrica.

Rafael Souza e Silva: 13696370, Lucca Baptista Silva Ferraz: 13688134 e Arthur Hiratsuka: 13687108

Inferência Bayesiana - Adriano Suzuki

Resumo

Este estudo analisa uma nova distribuição uniparamétrica, comparando-a com modelos tradicionais em análises de confiabilidade e sobrevivência. Utilizando abordagens estatísticas clássicas e Bayesianas, incluindo simulações MCMC e o algoritmo Metropolis-Hasting, avalia-se diferentes prioris e suas implicações. O artigo destaca a eficácia e a aplicabilidade desta nova distribuição em contextos estatísticos variados.

1 Introdução

Este trabalho propõe o estudo que inclui a expansão e aprimoramento do modelo inverso de Gompertz -ja estabelecido na literatura estatística- por meio de sua simplificação uniparamétrica. Essa distribuição é especialmente adequada para estudos de confiabilidade e sobrevivência, uma vez que modela o tempo de vida útil dos fenômenos observados.

Diante de propriedades desejáveis, tais como simplicidade e eficiência, distribuições monoparamétricas se revelam amplamente úteis quando em contraste com modelos mais complexos: a abordagem uniparamétrica reflete na flexibilização do modelo, pois promove análises com menos suposições e facilita a adequação a uma variedade de dados sem a necessidade de estimar múltiplos parâmetros.

Faz-se, por consequência, a análise da distribuição referida através dos métodos clássicos e bayesianos, sob efeito de incorporar ao estudo o paradigma de conhecimento prévio e incertezas, contrastando-os à estatística frequentista.

2 A nova distribuição monoparamétrica: construção do modelo e propriedades

Sob o pretexto de iniciar o estudo, faz-se uma breve introdução acerca do modelo em questão: a formulação da distribuição "A"se dá a partir da elaboração de uma função de distribuição cumulativa que deriva de um caso especial do modelo inverso de Gompertz (modelo multiparamétrico frequentemente utilizado em estudos demográficos) para uma única variável aletória.

Considera-se, portanto, que uma variável aleatória X segue a distribuição A, caracterizada pelo parâmetro único β quando sua função densidade de probabilidade é dada por:

$$F_X(x;\beta) = \exp\left(\frac{1}{\beta}\left(1 - \exp\left(\frac{x}{\beta}\right)\right)\right)$$

2.1 Gráfico da FDA

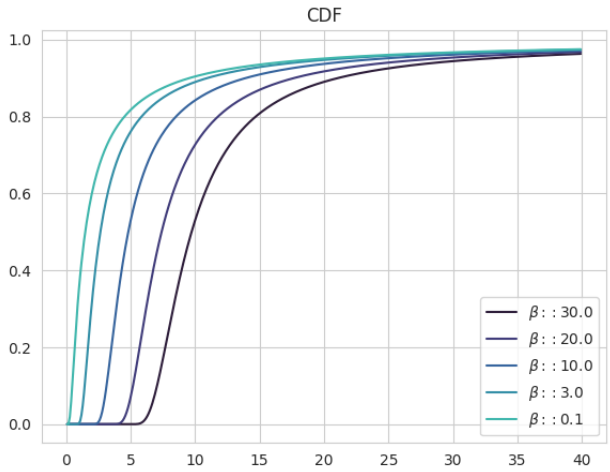

Figura 1: Plot da função densidade de probabilidade acumulada feito a partir da biblioteca Seaborn. Foram utilizados múltiplos valores de beta. Todos os códigos referenciados nesse artigo estão presentes no apêndice.

$$Com\ x > 0\ e\ \beta > 0.$$

Sua FDP pode, por conseguinte, ser encontrada a partir da derivada de $Fx(x;\beta)$ em relação a x:

$$f_X(x;\beta) = \frac{1}{x^2}\exp\left(\frac{1}{\beta}\left(1-\exp\left(\frac{x}{\beta}\right)\right)\right) + \frac{\exp\left(\frac{x}{\beta}\right)}{\beta x}$$

Com $x > 0$.

2.2 Gráfico da FDP

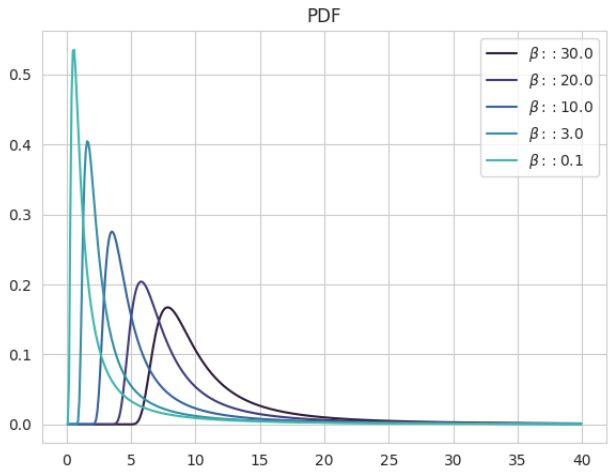

Figura 2: Plot da função densidade de probabilidade.

2.3 Propriedades

2.3.1 Unimodalidade

A partir da análise gráfica das funções de distribuição (ver figura 1), evidencia-se o comportamento unimodal da FDP, o qual é explorado nesta subseção.

Percebe-se, portanto, que os dados se concentram em um único pico, facilitanto a percepção de sua distribuição central. Calcula-se, pois, o valor da moda simplesmente através da derivada da FDP, isto é, a partir da seguinte equação:

$$\frac{d}{dx} f_X(x;\beta) = 0$$

$$\exp\left(\frac{x}{\beta}\right) - \frac{2x}{\beta} - 1 = 0$$

2.3.2 Confiabilidade

Define-se a confiabilidade como a probabilidade de um sistema continuar operando sem falhas até determinado ponto no tempo. A função de confiabilidade pode ser obtida a partir do complemento da FDA, dessa forma, expressa-se a função de confiabilidade Rx como:

$$R_X(x;\beta) = 1 - \exp\left(\frac{1}{\beta}\left(1-\exp\left(\frac{x}{\beta}\right)\right)\right)$$

O gráfico de R_x, por consequência, é dado por:

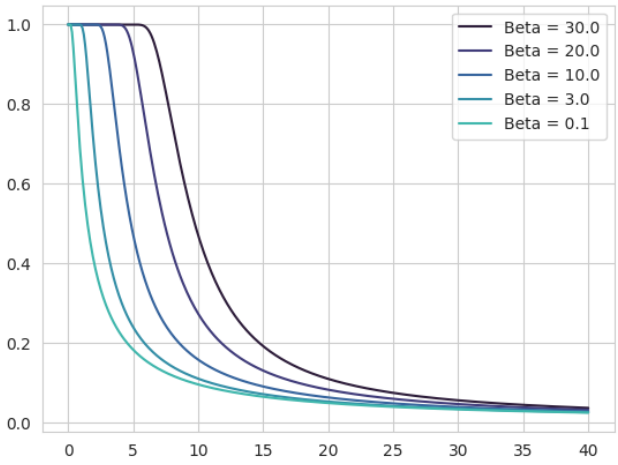

Figura 3: Plot da função confiabilidade.

2.3.3 Função de risco

Define-se a taxa de risco de dado sistema como a probabilidade de "falha"instantânea, sob a condição que ele tenha "sobrevivido"até dado momento. A função de risco pode ser obtida por meio do quociente da FDP pela complementar da FDA, ou seja:

$$h(t) = \frac{f(t)}{1-F(t)}$$

$$h_X(x;\beta) = \frac{\exp\left(\frac{x}{\beta}\right)}{x^2\exp\left(\frac{1}{\beta}\left(\exp\left(\frac{x}{\beta}\right)-1\right)\right)}$$

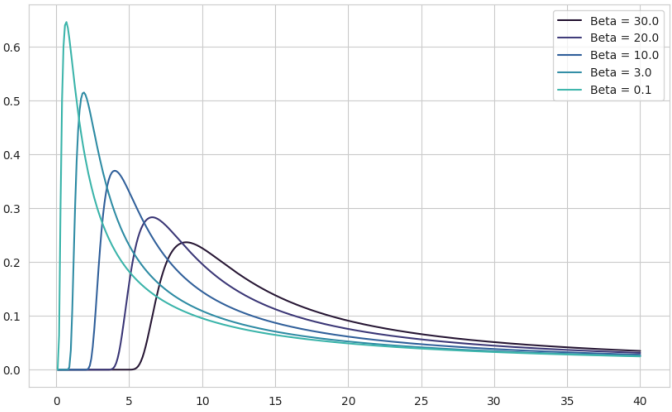

Figura 4: Plot da função de risco.

Destaca-se o formato pouco convencional da função de risco, cuja taxa inicial de falhas é elevada, seguida de um decréscimo que se estabiliza. Sugere*se, portanto, que para dado sistema, após superar o período crítico, ele se torna progressivamente mais confiável.

2.3.4 Função quantil

A função quantil da distribuição, utilizada futuramente na geração de amostras para os estudos de simulação, pode essencialmente ser obtida através da inversa da FDA. No presente estudo, a função quantil é essencialmente derivada a partir de métodos numéricos. Sua expressão analítica é dada por:

$$x_q = \frac{\ln(1-\beta\ln(q))}{\beta}$$

3 Estudo de simulação através do método clássico

Sob efeito de avaliar a eficácia do método de Estimação de Máxima Verossimilhança no parâmetro β, foi realizado um estudo de simulação sobre amostras geradas em múltiplos tamanhos. Descreve-se, nesta seção, cada passo do experimento.

3.1 Geração dos dados

A fim de padronizar os expetimentos, promovendo resultados mais fidedignos, gerou-se um arquivo contendo mil réplicas de dados uniformes para cada tamanho explorado na simulação.

Através do método da transformação inversa, isto é, por meio da função quantil da distribuição, é possível obter amostras aleatórias da distribuição em análise ao calcular $X_i = F^{-1}(Ui)$ para os dados gerados anteriormente.

3.2 Estimador de máxima verossimilhança

Diante das amostras obtidas anteriormente, a função de verossimilhança é o produto das funções de densidade para cada ponto de dados, dependendo do parâmetro da distribuição. Têm-se, portanto:

$$\mathcal{L}(\beta; x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} f_X(x_i; \beta)$$

Aplica-se, posteriormente, a função logaritmica na expressão obtida, sob efeito de simplificar os cálculos ao transformar os produtos em somas sem alterar o comportamento da função, isto é, o maximizador global da verossimilhança se mantém.

O objetivo é encontrar o valor de β que maximiza a log-verossimilhança, obtido a partir da derivada primeira da equação igualada a zero:

$$LL = \frac{1}{\beta} - \sum_{i=1}^{n} \frac{x_i}{\beta} \exp(\beta x_i) - \sum_{i=1}^{n} (1 - 2\ln(x_i))$$

A solução para a equação é aproximada através de métodos numéricos diante da impossibilidade de uma solução analítica. No caso, a função "fsolve"da biblioteca scipy se apropria de noções de otimização não linear sob efeito de aproximar a solução da equação sem sua derivada explícita.

O objetivo é estimar as métricas de viés e variância do estimador obtido através do método de máxima verossimilhança. Mais especificamente, são calculados para cada replicação, o RAB, a média, o MSE, o RMSE e a probabilidade de cobertura.

Tabela 1: Resultados da Simulação para $\beta = 0.125$

Size	Mean	Var	Bias	MSE	RMSE	PC
25	0.214629	0.044276	0.089629	0.052309	0.228712	0.933
50	0.173871	0.019995	0.048871	0.022384	0.149612	0.935
100	0.145898	0.007447	0.020898	0.007884	0.088791	0.945
200	0.136500	0.003540	0.011500	0.003672	0.060599	0.947
400	0.131363	0.001713	0.006363	0.001754	0.041880	0.948

Tabela 2: Resultados da Simulação para $\beta = 0.6$

Size	Mean	Var	Bias	MSE	RMSE	PC
25	0.698618	0.074206	0.098618	0.083932	0.289710	0.932
50	0.653746	0.035250	0.053746	0.038139	0.195292	0.940
100	0.622565	0.013778	0.022565	0.014287	0.119529	0.951
200	0.612607	0.006740	0.012607	0.006899	0.083057	0.949
400	0.607330	0.003317	0.007330	0.003370	0.058056	0.946

Tabela 3: Resultados da Simulação para $\beta = 1$

Size	Mean	Var	Bias	MSE	RMSE	PC
25	1.106558	0.100101	0.106558	0.111455	0.333849	0.932
50	1.058147	0.048430	0.058147	0.051811	0.227621	0.941
100	1.024350	0.019263	0.024350	0.019856	0.140910	0.954
200	1.013706	0.009520	0.013706	0.009708	0.098527	0.948
400	1.008151	0.004709	0.008151	0.004775	0.069105	0.946

4 Comparação com modelos já consolidados

Com o propósito de aprofundar a análise da distribuição, compara-se o modelo proposto com outros já condolidados na literatura estatística. Para tal finalidade, optou-se por modelos de complexidade similar -isto é, modelos também uniparamétricos- uma vez que eles oferecem uma base de comparação justa para as métricas de adequação e eficiência.

Foram utilizadas, pois, as distribuições exponencial, exponencial inversa, Lindley, Rayleigh e Rayleigh Inversa para validar o estudo.

4.1 Conjunto de dados

Para efeito de reprodutibilidade, foi utilizado o mesmo conjunto de dados do artigo original, isto é:

Tabela 4: The Strength of Glass of the Aircraft Window Data

18.83	20.80	21.657	23.03	23.23	24.05	24.321
25.50	25.52	25.80	26.69	26.77	26.78	27.05
27.67	29.90	31.11	33.20	33.73	33.76	33.89
34.76	35.75	35.91	36.98	37.08	37.09	39.58
44.045	45.29	45.381				

4.2 Critérios comparativos

4.2.1 Negative Log-Likelihood

Medida da probabilidade de que o modelo tenha gerado os dados observados. Um valor maior (menos negativo) indica um melhor ajuste do modelo aos dados.

4.2.2 Teste de Distância Kolmogorov-Smirnov (K-S)

Máxima distância entre as funções de distribuição acumulada dos dados observados e do modelo. É usado para testar a hipótese de que os dados seguem uma distribuição específica

4.2.3 Critério de Informação de Akaike (AIC)

Critério baseado na teoria da informação que avalia a qualidade do modelo, equilibrando o ajuste do modelo e a complexidade (número de parâmetros). O valor do AIC é comparativo, isto é, não fornece uma medida em termos absolutos da qualidade do modelo, uma vez que depende, por exemplo, da escala dos dados.Um valor menor do AIC é preferível

[0]Dados retirados de: *[Alshenawy, R. A new one parameter distribution: properties and estimation with application sto complete and type II censored data. JOURNAL OF TAIBAH UNIVERSITY FOR SCIENCE, v. 14, n. 1, p. 11-18, 2020. Disponível em: <https://www.tandfonline.com/doi/epdf/10.1080/16583655.2019.1698276>. Acesso em: 17/12/2023.].*

4.2.4 Critério de Informação Bayesiano (BIC)

Análogo ao AIC, utiliza de uma maior penalização para o número de parâmetros, favorecendo modelos mais simples.

4.2.5 Critério de Informação de Hannan-Quinn (HQIC)

Também análogo ao AIC, segue outra medida de penalização.

4.3 Resultados

Tabela 5: Comparação dos Modelos - Parte 1

Distribution	MLE	-LL	KS_stat	KS_p
Exponencial	0.032455	137.264447	0.458623	1.748867e-06
Inverse Exponencial	29.215334	137.261497	0.474696	6.155969e-07
Lindley	0.062988	126.994191	0.365453	3.219054e-04
Rayleigh	0.001000	118.222345	0.318876	2.651648e-03
Inverse Rayleigh	810.503208	118.200626	0.325347	2.015911e-03
A	125.662000	107.950308	0.161626	3.543295e-01

Tabela 6: Comparação dos Modelos - Parte 2

Distribution	AIC	CAIC	BIC	HQIC
Exponencial	276.528894	276.657926	277.962881	276.996338
Inverse Exponencial	276.522995	276.652027	277.956982	276.990439
Lindley	255.988382	256.117414	257.422369	256.455826
Rayleigh	238.444691	238.573723	239.878678	238.912135
Inverse Rayleigh	238.401253	238.530285	239.835240	238.868697
A	217.900616	218.029648	219.334603	218.368060

Conforme já analisado no artigo base, concluise, através da metodologia clássica, que a distribuição A possui um menor valor para todos os índices analisados, intuindo o melhor desempenho dentre os modelos testados.

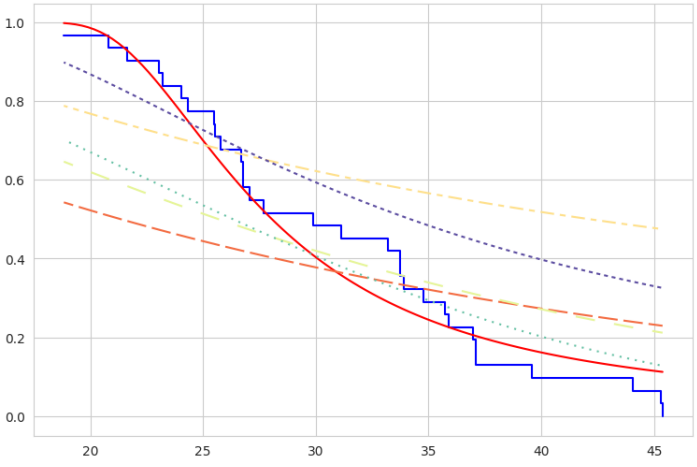

Figura 5: Plot da função de confiabilidade estimada para os índices utilizados na comparação. O código dos plots a segguir podem ser obtidos aqui.

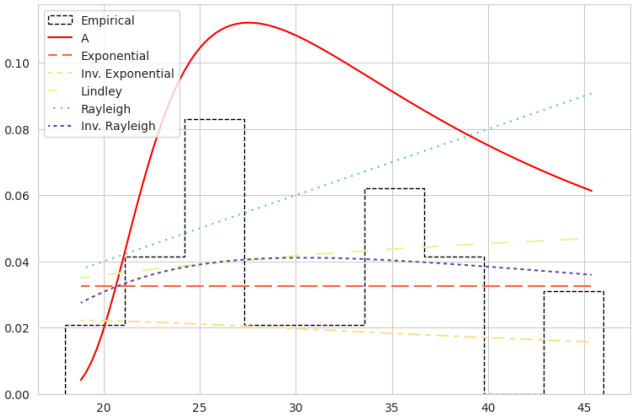

Figura 6: Plot da função de de risco estimada para os índices utilizados na comparação.

Verifica-se, ainda, a unimodalidade da função log-likelihood, que atinge seu valor máximo quando $\beta = 125,66$.

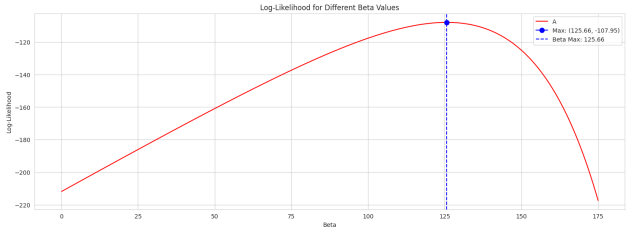

Figura 7: Plot da função log-likelihood.

5 Estudo a partir da metodologia Bayesiana

5.1 O método de Monte Carlos por Cadeias de Markov

O processo de Monte Carlo com cadeias de Markov consiste em simular amostras de densidades posteriores muito complexas para serem obtidas analiticamente. Para tal, define-se um ponto inicial da distribuição, sobre o qual se gera uma amostra. Utiliza-se, na próxima iteração, do resultado do processo anterior de amostragem como ponto de partida, criando uma cadeia de Markov. Parte-se da proposição de que as cadeias de Markov representam um processo estocástico "sem memória", isto é, que dependem apenas do estado atual dos dados, e não da sequência de eventos precedente. Supõe-se, portanto a convergência para uma distribuição estacionária e independente do ponto inicial de partida.

5.2 O algoritmo Metropolis-Hasting

Realizar simulações estocásticas de uma distribuição sem um algoritmo de aceitação promove o fenômeno chamado de "caminhada aleatória", no qual o espaço dos parâmetros é explorado de forma aleatória, sem a perspectiva de convergência para uma priori desejada.

Utiliza-se, portanto, de um critério de aceitação baseado na razão de verossimilhança a fim de decidir se um novo estado é válido ou não para a simulação de Monte Carlo.

Na implementação em questão, em cada etapa do MCMC o algoritmo Metropolis propõe um novo valor para β com base no valor atual. A magnitude desta proposta é influenciada pelo "step_scale".

Calcula-se, portanto, a razão de verossimilhança (nesse caso, exprime qual dos estados tem maior probabilidade de ter gerado os dados) a qual, quando maior do que 1 -sugerindo melhor adequação do sistema- é aceita e quando menor que 1 não necessariamente é descartada, sob controle de um evento aleatório em função da razão. Esse processo é repetido ao longo do chamado "burn-in", no qual os valores iniciais são descartados a fim de que as cadeias de Markov "esqueçam"as condições iniciais.

5.3 Simulação

5.3.1 A priori uniforme

Diante da natureza simples e não informativa, foi utilizada, em primeira análise, uma priori uniforme para o parâmetro β da distribuição A. Supõe-se, nessa análise, que a escolha da priori uniforme reflita a falta de conhecimento prévio acerc do problema, estimulando uma maior significância dos dados observados na formação da posteriori.

Considera-se, portanto, que a variável aleatória X segue a distribuição A, com $X \sim A(\beta)$, e o parâmetro β segue uma distribuição Uniforme, $\beta \sim \text{Unif}(a, b)$. A formulação Bayesiana é dada por:

$$\pi(\beta|x) \propto f(x|\beta) \times \pi(\beta)$$

$$\propto \frac{1}{x^2} e^{\frac{1}{\beta}(1-e^{\frac{\beta}{x}})+\frac{\beta}{x}} \times \frac{1}{b-a}$$

$$\propto e^{\frac{1}{\beta}(1-e^{\frac{\beta}{x}})+\frac{\beta}{x}}$$

Optou-se pelos valores de $a = 0$ e $b = 100$ ou 1000, a fim de explorar a vaguidão da priori no estudo.

5.3.2 Priori Gama

Para efeito de comparação, utilizou-se também como priori uma distribuição gama, explorando também, por meio do espaço paramétrico, uma abordagem pouco informativa da priori. Destaca-se a natureza positiva e unimodal da distribuição.

Considera-se por conseguinte, que a variável aleatória X segue a distribuição A, com $X \sim A(\beta)$, e o parâmetro β segue uma distribuição Gamma, $\beta \sim \text{Gamma}(a, b)$. A formulação Bayesiana é dada por:

$$\pi(\beta|x) \propto f(x|\beta) \times \pi(\beta)$$

$$\propto \frac{1}{x^2} e^{\frac{1}{\beta}(1-e^{\frac{\beta}{x}})+\frac{\beta}{x}} \times \frac{b^a}{\Gamma(a)} \beta^{a-1} e^{-\beta x}$$

$$\propto \beta^{a-1} e^{\frac{1}{\beta}(1-e^{\frac{\beta}{x}})+\beta(\frac{1}{x}-b)}$$

Optou-se por uma priori Gama "flat", isto é, cuja abordagem é pouco informativa e menos restritiva, dando enfoque aos dados observados para os processos MCMC. Tem-se, portanto, os parâmetros $a = [1, 0.1]$ $b = [0.001, 0.0001]$. Observa-se, ainda, que a posteriori nesse caso específico não possui um núcleo explícito.

5.3.3 Priori de Jeffreys

A fim de obter a priori de Jeffreys - e futuramente a variância do estimador - faz-se necessário o cálculo da matriz de informação de Fisher, cuja formulação é dada pela derivada segunda da função log-verossimilhança:

1. Primeira Derivada da Log-Verossimilhança:

$$\frac{dl(\beta)}{d\beta} = \frac{1}{x} - \frac{1}{\beta^2}\left(\frac{\beta}{x} e^{\frac{\beta}{x}} - e^{\frac{\beta}{x}} + 1\right)$$

2. Segunda Derivada da Log-Verossimilhança:

$$\frac{d^2 l(\beta)}{d\beta^2} = \frac{2e^{\frac{\beta}{x}}}{\beta^3 x} - \frac{2}{\beta^3 x} - \frac{2e^{\frac{\beta}{x}}}{\beta^2 x^2} + \frac{e^{\frac{\beta}{x}}}{\beta x^3}$$

3. Informação de Fisher:

$$E\left[-\frac{d^2 l(\beta)}{d\beta^2}\right] = E\left[\frac{2e^{\frac{\beta}{x}}}{\beta^3 x} - \frac{2}{\beta^3 x} - \frac{2e^{\frac{\beta}{x}}}{\beta^2 x^2} + \frac{e^{\frac{\beta}{x}}}{\beta x^3}\right]$$

Em decorrência da impossibilidade de encontrar analiticamente a esperança desejada, segue-se a partir de métodos numéricos.

Diante desse panorama, não foi utilizada a priori de Jeffreys. Simulou-se, porém, o comportamento do modelo am cenários pouco informativos a partir de outras distribuições.

5.4 Resultados e discussões

Realizou-se experimentos para os $\beta = [0.125; 0.6; 1]$, de modo a comparar a abordagem bayesiana à clássica. O objetivo, novamente, é calcular as mesmas métricas de viés e variância, dessa vez obtidos através das simulações de Monte Carlo.

Tabela 7: Resultados da Simulação com Priori Gamma (a = 1, b = 0.001) para $\beta = 0.125$

Size	Mean	Var	Bias	MSE	RMSE	PC
25	0.255756	0.028168	0.130756	0.045265	0.212756	0.929
50	0.195230	0.013095	0.070230	0.018027	0.134266	0.937
100	0.153991	0.005144	0.028991	0.005985	0.077362	0.953
200	0.136739	0.002824	0.011739	0.002961	0.054419	0.936
400	0.128950	0.001574	0.003950	0.001590	0.039873	0.927

Tabela 8: Resultados da Simulação com Priori Uniforme (a = 0, b = 1000.0) para $\beta = 0.125$

Size	Mean	Var	Bias	MSE	RMSE	PC
25	0.255122	0.028153	0.130122	0.045085	0.212332	0.931
50	0.194912	0.013087	0.069912	0.017974	0.134069	0.939
100	0.153862	0.005150	0.028862	0.005983	0.077350	0.949
200	0.136699	0.002827	0.011699	0.002964	0.054441	0.934
400	0.128953	0.001574	0.003953	0.001590	0.039874	0.928

Tabela 9: Resultados da Simulação com Priori Gamma (a = 1, b = 0.001) para $\beta = 0.6$

Size	Mean	Var	Bias	MSE	RMSE	PC
25	0.666222	0.066290	0.066222	0.070676	0.265849	0.929
50	0.629221	0.033831	0.029221	0.034685	0.186240	0.934
100	0.607921	0.013651	0.007921	0.013713	0.117104	0.942
200	0.604802	0.006706	0.004802	0.006729	0.082032	0.949
400	0.602869	0.003295	0.002869	0.003303	0.057473	0.941

Tabela 12: Resultados da Simulação com Priori Uniforme (a = 0, b = 1000.0) para $\beta = 1.0$

Size	Mean	Var	Bias	MSE	RMSE	PC
25	1.055836	0.094798	0.055836	0.097915	0.312914	0.939
50	1.028129	0.047484	0.028129	0.048275	0.219717	0.930
100	1.008404	0.019111	0.008404	0.019182	0.138498	0.955
200	1.005010	0.009459	0.005010	0.009484	0.097387	0.947
400	1.002869	0.004708	0.002869	0.004716	0.068673	0.948

Observa-se que a probabilidade de cobertura é calculada de maneira diferente no paradigma

Tabela 10: Resultados da Simulação com Priori Uniforme (a = 0, b = 1000.0) para $\beta = 0.6$

Size	Mean	Var	Bias	MSE	RMSE	PC
25	0.666282	0.066443	0.066282	0.070837	0.266151	0.929
50	0.629130	0.033829	0.029130	0.034678	0.186219	0.933
100	0.607899	0.013640	0.007899	0.013702	0.117055	0.944
200	0.604710	0.006701	0.004710	0.006723	0.081992	0.948
400	0.602876	0.003297	0.002876	0.003305	0.057492	0.942

Tabela 11: Resultados da Simulação com Priori Gamma (a = 1, b = 0.001) para $\beta = 1.0$

Size	Mean	Var	Bias	MSE	RMSE	PC
25	1.056040	0.094632	0.056040	0.097773	0.312686	0.940
50	1.028186	0.047459	0.028186	0.048254	0.219667	0.931
100	1.008461	0.019106	0.008461	0.019178	0.138483	0.960
200	1.005019	0.009451	0.005019	0.009477	0.097348	0.947
400	1.002835	0.004702	0.002835	0.004710	0.068627	0.945

bayesiano: em contraposição à metodologia clássica, em cuja probabilidade de cobertura representa captura a a frequência com que o verdadeiro valor do parâmetro pertence ao intervalo de confiança calculado dado um grande número de repetições, aqui PC é a probabilidade de que o intervalo de credibilidade contenha o verdadeiro valor do parâmetro, dada a distribuição posterior.

5.5 Dados do artigo

Com o intuito de repetir os passos realizados no artigo original, analisa-se, sob o enfoque bayesiano, o mesmo conjunto de dados para as prioris apresentadas:

5.5.1 Priori Gama

O histograma abaixo exibe a distribuição das amostras fornecendo uma visão da distribuição posterior do parâmetro. Percebe-se a tendência central das amostras à simetria em torno de 125, indicando bons resultados.

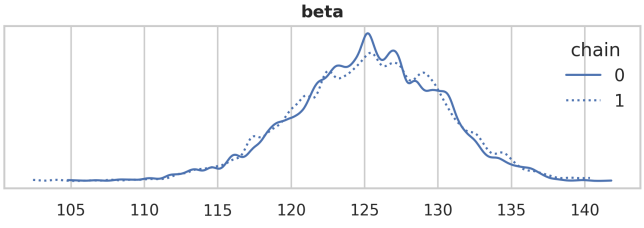

Figura 8: Plot do valor da distribuição estimada de beta para a priori Gama.

O trace plot mostra as trajetórias das amostras ao longo das iterações da MCMC para cada

Tabela 13: Estatísticas Resumidas - Parte 1

Parâmetro	Mean	SD	HDI 3%	HDI 97%
beta	125.208	5.031	115.859	134.685

Tabela 14: Estatísticas Resumidas - Parte 2

Parâmetro	MCSE Mean	MCSE SD	ESS Bulk	ESS Tail	R-hat
beta	0.078	0.055	4224.0	4553.0	1.0

cadeia. Ele permite visualizar a convergência da simulação ao longo do tmepo. Percebe-se, nesse exemplo uma convergência adequada, indicada por cadeias que parecem misturar-se bem, sem tendências ou padrões discerníveis.

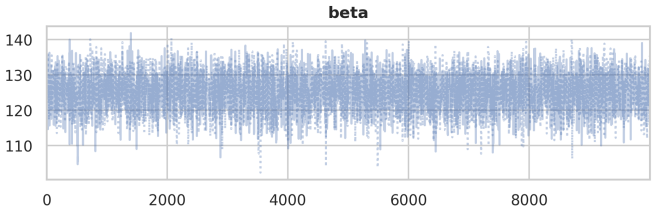

Figura 9: Plot do valor de beta em função da cadeia de Markov para a priori Gama

No plot de autocorrelação, observa-se a tendência à redução da autocorrelação em função das iterações, revelando uma estabilização próxima a zero, indicando que após o período original de burn-in, as amostras se tornam independentes.

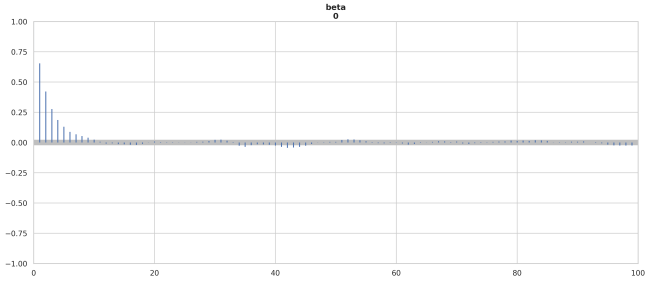

Figura 10: Plot da autocorrelação.

5.5.2 Priori uniforme

De forma análoga à priori Gama flat, a priori uniforme mostra sua tendência de centralidade em torno de 125, sugerindo adequação.

Tabela 15: Estatísticas Resumidas - Parte 1

Parâmetro	Mean	SD	HDI 3%	HDI 97%
beta	125.147	5.035	115.455	134.27

Tabela 16: Estatísticas Resumidas - Parte 2

Parâmetro	MCSE Mean	MCSE SD	ESS Bulk	ESS Tail	R-hat
beta	0.083	0.059	3683.0	3981.0	1.0

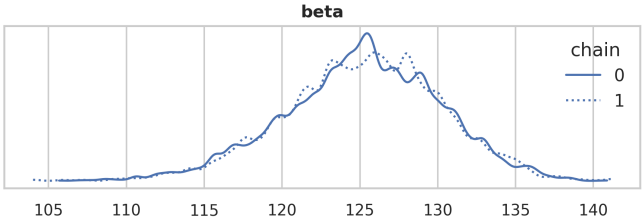

Figura 11: Plot do valor da distribuição estimada de beta para a priori uniforme.

A trajetória das amostras também não revela padrões, ciclos ou tendências.

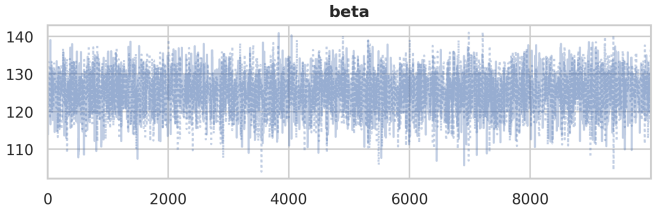

Figura 12: Plot do valor de beta em função da cadeia de Markov para a priori uniforme.

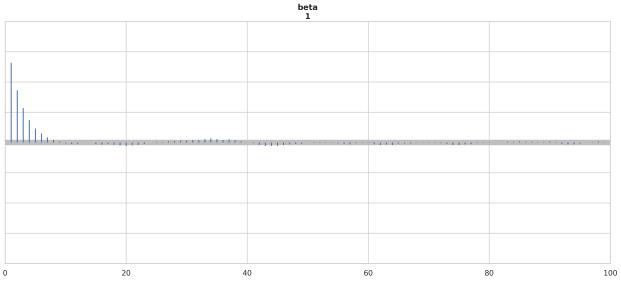

Figura 13: Plot da autocorrelação.

6 Conclusão

Inicialmente, é importante observar que a seleção da priori e dos parâmetros para a priori constitui uma etapa crucial, uma vez que essa escolha impacta diretamente o comportamento do modelo bayesiano. Opta-se, portanto, por utilizar prioris pouco informativas, caracterizadas por uma alta variância, dada a ausência de um especialista no tema.

6.1 Simulações

Ao analisar as simulações, torna-se evidente que, quanto menor o valor do parâmetro beta, pior é a convergência do modelo bayesiano em relação ao cálculo da log-verossimilhança, como ilustrado nas tabelas 1, 7 e 8. Por outro lado, para valores mais elevados de beta, os parâmetros estimados pelo modelo convergem mais rapidamente para o valor real do parâmetro, conforme apresentado nas tabelas 3, 9 e 10. Entretanto, é válido ressaltar que a alteração nos valores dos parâmetros da priori influencia esses resultados, indicando que é possível obter resultados mais precisos com betas menores.

6.2 Dados do Artigo

Ao utilizar as mesmas prioris para estimar o valor de beta, obtivemos os seguintes resultados: Priori Gamma: 125.208 Priori Uniforme: 125.147 Ambos os valores são próximos, entretanto, são inferiores ao valor previsto utilizando um Estimador de Máxima Verossimilhança (EMV). Essa observação reforça, conforme destacado nas tabelas das simulações bayesianas, que ambas as prioris pouco informativas apresentam comportamentos semelhantes.

É notável que ambas as prioris convergiram rapidamente, como evidenciado nas figuras 9 e 12, além de gerarem vários estimadores pouco autocorrelacionados, conforme demonstrado nas figuras 10 e 13.

6.3 Considerações Finais

Conforme observado, os modelos bayesianos proporcionam resultados robustos e significativos. No entanto, é crucial realizar a escolha adequada de prioris e parâmetros para cada problema. Além disso, é importante ressaltar que esses modelos tendem a ser computacionalmente mais onerosos em comparação com métodos clássicos. Portanto, dominar ambas as ferramentas e empregá-las nos momentos mais apropriados torna-se de extrema importância para uma abordagem eficaz e eficiente.

Apêndice A: Códigos utilizados

A.1 Definição das funções

```
def Fx(x:float, beta: float) -> float:
  return np.power(np.e, 1/beta * (1 - np.power(np.e, beta/x)))


def fx(x:float, beta: float) -> float:
  return (1/np.power(x, 2)) * np.power(np.e, 1/beta * (1 - np.power(np.e, beta/x)) + beta/x)
```

A.2 Plots

```
betas = np.array([30, 20, 10, 3, .1])
x = np.arange(0.1, 40, 0.1)

fig, ax = plt.subplots(figsize=(15, 5), ncols=2)
for beta in betas:
  sns.lineplot(x=x, y=Fx(x, beta), label=r'$\beta :: $' + str(beta), ax=ax[0])
  sns.lineplot(x=x, y=fx(x, beta), label=r'$\beta :: $' + str(beta), ax=ax[1])

ax[0].set_title('CDF')
ax[1].set_title('PDF')
plt.legend()
plt.show()
```

A.3 Geração dos dados

```
# Define uma semente para a geração de números aleatórios,
# garantindo reprodutibilidade nos resultados.
np.random.seed(42)

# Tamanhos das amostras que serão geradas.
sizes = [25, 50, 100, 200, 400]

# Número de réplicas para cada tamanho de amostra.
replicas = 1000

# Itera sobre os diferentes tamanhos de amostra.
for size in sizes:
    # Gera dados aleatórios uniformemente distribuídos entre 0 e 1.
    data = np.random.uniform(0, 1, (replicas, size))

    # Salva os dados gerados em um arquivo numpy (.npy).
    with open(f'uniform_data_{size}.npy', 'wb') as f:
        np.save(f, data)
```

```
-----------------------------------------------------------------------------------------------
def negative_likelihood(beta: float, data: np.ndarray):
        return - (beta * np.sum(1/data) - (1/beta) * np.sum((np.exp(beta/data) - 1)) - 2*np
        .sum(np.log(data)))

-----------------------------------------------------------------------------------------------
```

A.4 Classe para os experimentos

```
-----------------------------------------------------------------------------------------------
def create_statistics(self, alpha: float = 5e-2):

        z = stats.norm.ppf(1 - (alpha/2))

        for key in self.betas_hat:
            data = np.array(self.betas_hat[key])

            self.statistics['size'].append(key)
            self.statistics['mean'].append(np.mean(data))
            self.statistics['var'].append(np.var(data))
            self.statistics['bias'].append(np.mean(data) - self.beta)
            MSE = np.sum((self.beta - data)**2) / len(data)
            self.statistics['MSE'].append(MSE)
            self.statistics['RMSE'].append(np.sqrt(MSE))

            cout = 0
            for current_beta, variance in zip(data, self.fisher[key]):
                upper = current_beta  + z * np.sqrt(variance)
                lower = current_beta  - z * np.sqrt(variance)

                cout += 1 if lower < self.beta < upper else 0

            self.statistics['PC'].append(cout / self.replicas)

        self.statistics = pd.DataFrame(self.statistics)
-----------------------------------------------------------------------------------------------
```

A.5 Simulações

```
-----------------------------------------------------------------------------------------------
def create_statistics(self, alpha: float = 5e-2):

        z = stats.norm.ppf(1 - (alpha/2))

        for key in self.betas_hat:
            data = np.array(self.betas_hat[key])

            self.statistics['size'].append(key)
            self.statistics['mean'].append(np.mean(data))
            self.statistics['var'].append(np.var(data))
            self.statistics['bias'].append(np.mean(data) - self.beta)
```

```
            MSE = np.sum((self.beta - data)**2) / len(data)
            self.statistics['MSE'].append(MSE)
            self.statistics['RMSE'].append(np.sqrt(MSE))

            cout = 0
            for current_beta, variance in zip(data, self.fisher[key]):
                upper = current_beta  + z * np.sqrt(variance)
                lower = current_beta  - z * np.sqrt(variance)

                cout += 1 if lower < self.beta < upper else 0

            self.statistics['PC'].append(cout / self.replicas)

        self.statistics = pd.DataFrame(self.statistics)
```

A.6 Modelos consolidados

```
def Exponencial(x_values, **kwargs):
    mle = len(x_values) / sum(x_values)
    ll = len(x_values)*np.log(mle) - np.sum(x_values)*mle
    sample_cont = np.linspace(min(x_values), max(x_values), 100)

    def CDF(x):
        cdf = 1 - np.exp(-mle*x)
        return cdf

    def PDF(x):
        pdf = mle * np.exp(-mle*x)
        return pdf

    ks_test = stats.kstest(x_values, CDF)
    return ['Exponencial', mle, -ll, ks_test.statistic, ks_test.pvalue, *get_stats(x_values,
    ll)], (sample_cont, [CDF(x) for x in sample_cont], [PDF(x) for x in sample_cont])

def Inverse_Exponencial(x_values, **kwargs):
    mle = len(x_values) / sum(x**-1 for x in x_values)
    ll = len(x_values) * np.log(mle) - 2  * np.sum(np.log(x_values))  - mle*sum(x**-1 for x
    in x_values)
    sample_cont = np.linspace(min(x_values), max(x_values), 100)

    def CDF(x):
        cdf = np.exp(-mle/x)
        return cdf

    def PDF(x):
        pdf = mle * np.exp(-mle/x) / (x**2)
        return pdf

    ks_test = stats.kstest(x_values, CDF)
```

```
    return ['Inverse Exponencial', mle, -ll, ks_test.statistic,
    ks_test.pvalue, *get_stats(x_values, ll)], (sample_cont, [CDF(x) for x in sample_
    cont], [PDF(x) for x in sample_cont])

def Lindley(x_values, **kwargs):
    mean = sum(x_values) / len(x_values)
    mle = (((mean - 1)**2 + 8*mean)**.5 - (mean - 1)) / (2*mean)
    ll = len(x_values) * (2 * np.log(mle) - np.log(1 + mle)) - mle*np.sum(x_values) + sum(np.
    log(1 + x) for x in x_values)
    sample_cont = np.linspace(min(x_values), max(x_values), 100)

    def CDF(x):
        cdf = 1 - ((1 + mle + mle*x) / (1 + mle)) * np.exp(-mle * x)
        return cdf

    def PDF(x):
        pdf = (np.exp(-mle*x) * (x * mle**2 + mle**2)) / (mle + 1)
        return pdf

    ks_test = stats.kstest(x_values, CDF)
    return ['Lindley', mle, -ll, ks_test.statistic, ks_test.pvalue, *get_stats(x_values,
    ll)], (sample_cont, [CDF(x) for x in sample_cont], [PDF(x) for x in sample_cont])


def Rayleigh(x_values, **kwargs):
    mle = len(x_values) / (sum(x**2 for x in x_values))
    ll = len(x_values) * np.log(2*mle)  +  np.sum(np.log(x_values)) - mle * sum(x**2 for x
    in x_values)
    sample_cont = np.linspace(min(x_values), max(x_values), 100)

    def CDF(x):
        cdf = 1 - np.exp(-mle*(x**2))
        return cdf

    def PDF(x):
        pdf = 2 * mle * x * np.exp(-mle*(x**2))
        return pdf

    ks_test = stats.kstest(x_values, CDF)
    return ['Rayleigh', mle, -ll, ks_test.statistic, ks_test.pvalue, *get_stats(x_values, ll)],


def Inverse_Rayleigh(x_values, **kwargs):
    mle = len(x_values) / sum(x**-2 for x in x_values)
    ll = len(x_values) * np.log(2*mle)  - 3 * np.sum(np.log(x_values)) - mle * sum(x**-2 for
    x in x_values)
    sample_cont = np.linspace(min(x_values), max(x_values), 100)

    def CDF(x):
        cdf = np.exp(-mle/(x**2))
        return cdf
```

```
    def PDF(x):
        pdf = 2 * mle * np.exp(-mle/(x**2)) / (x**3)
        return pdf

    ks_test = stats.kstest(x_values, CDF)

     return ['Inverse Rayleigh', mle, -ll, ks_test.statistic, ks_test.pvalue, *get_stats
     (x_values, ll)], (sample_cont, [CDF(x) for x in sample_cont], [PDF(x) for x in samp
     le_cont])

    diff = abs(results_ll[i] - results_ll[j])
        if diff > diff_ll:
                diff_ll = diff

    if (diff_mle > tol) or (diff_ll > tol):
        print("Erro de convergência!\nFunção abortada!")
        return None

    mle = results_mle[0]
    ll = results_ll[0]

    def CDF(x):
        result = np.zeros_like(x)
        mask_positive = x > 0

        c = mle / x[mask_positive]
        log_cdf = (1 - np.exp(c)) / mle
        result[mask_positive] = np.exp(log_cdf)
        return result

    def PDF(x):
        result = np.zeros_like(x)
        mask_positive = x > 0

        c = mle / x[mask_positive]
        result[mask_positive] = (x**-2) * np.exp(((1 - np.exp(c))/mle) + c)
        return result

    sample_cont = np.linspace(min(x_values), max(x_values), 100)

    ks_test = stats.kstest(x_values, CDF)
    return ['A', mle, -ll, ks_test.statistic, ks_test.pvalue, *get_stats(x_values, ll)],
    (sample_cont, [CDF(x) for x in sample_cont], [PDF(x) for x in sample_cont])
```

A.7 Plots auxiliares (confiabilidade e risco)

```
prob_empirica = np.arange(1, len(sample) + 1) / len(sample)
sample_space = np.linspace(min(sample), max(sample), 150)

patt = plt.get_cmap('Spectral', 6)

fig, axs = plt.subplots(1, 2, figsize = (20,6))

ax = axs[0]
ax.step(sample, 1 - prob_empirica, where='post',  color = 'blue', label = 'Empirical')
ax.plot(cdf_A[0], [1 - x for x in cdf_A[1]], color = 'red', label = 'A')
ax.plot(cdf_exp[0], [1 - x for x in cdf_exp[1]], ls = (5, (10, 3)), color = patt(1),
label = 'Exponential')
ax.plot(cdf_invexp[0], [1 - x for x in cdf_invexp[1]], ls = (5, (7, 3, 3, 3)), color =
patt(2), label = 'Inv. Exponential')
ax.plot(cdf_lind[0], [1 - x for x in cdf_lind[1]], ls = (5, (10, 10)), color = patt(3), label
= 'Lindley')
ax.plot(cdf_ray[0], [1 - x for x in cdf_ray[1]], ls = (5, (1, 3)), color = patt(4), label
= 'Rayleigh')
ax.plot(cdf_invray[0], [1 - x for x in cdf_invray[1]], ls = (5, (2, 2)), color = patt(5),
label = 'Inv. Rayleigh')

ax.set_xlabel('Sample')
ax.set_ylabel('Probabability')
ax.set_title('Reliability functions')

ax = axs[1]

bin_edges = np.linspace(18, 46, 10)

ax.hist(sample, bins = bin_edges, density = True, histtype='step', edgecolor='black', linest
yle='dashed', label = 'Empirical')
ax.plot(cdf_A[0], [x/(1-y) for x, y in zip(cdf_A[2], cdf_A[1])], color = 'red', label = 'A')
ax.plot(cdf_exp[0], [x/(1-y) for x, y in zip(cdf_exp[2], cdf_exp[1])], ls = (5, (10, 3)),
color = patt(1), label = 'Exponential')
ax.plot(cdf_invexp[0], [x/(1-y) for x, y in zip(cdf_invexp[2], cdf_invexp[1])], ls = (5, (7, 3,
3, 3)), color = patt(2), label = 'Inv. Exponential')
ax.plot(cdf_lind[0], [x/(1-y) for x, y in zip(cdf_lind[2], cdf_lind[1])], ls = (5, (10, 10))
, color = patt(3), label = 'Lindley')
ax.plot(cdf_ray[0], [x/(1-y) for x, y in zip(cdf_ray[2], cdf_ray[1])], ls = (5, (1, 3)),
color = patt(4), label = 'Rayleigh')
ax.plot(cdf_invray[0], [x/(1-y) for x, y in zip(cdf_invray[2], cdf_invray[1])], ls = (5, (2,
2)), color = patt(5), label = 'Inv. Rayleigh')

ax.set_xlabel('Sample')
ax.set_ylabel('Probabability')
ax.set_title('Hazard Rate')


plt.legend()
plt.grid(True)
plt.show()
```

A.8 MCMC

```
    #@title Create Experiments
def logp(x: TensorVariable, beta: TensorVariable) -> TensorVariable:
    return beta/x - (1/beta) * (exp(beta/x) - 1) - 2*log(x)


def ppf(q: float, beta: float) -> float:
    return np.divide( beta, np.log(1 - beta * np.log(q)) )


def create_betas(target_beta, unifom_data,
                 size, seed = 42, alpha = .05,
                 beta_guess = 1e-2, data_guess = 1e-2,
                 a_gamma = 1, b_gamma = 1e-3,
                 a_unif = 0, b_unif = 1e+3,
                 burn  = 5000, trace_size = 5000,
                 step_size = 5):

    new_data = copy.deepcopy(uniform_data)

    for key in tqdm(new_data, desc='Transform Dataset'):
      for idx, data in enumerate(new_data[key]):
        sample = [ppf(x, target_beta) for x in data]
        new_data[key][idx] = sample


    gamma_info = []
    unif_info = []

    for sample in tqdm(new_data[size], desc='Estimating Betas'):

      with pm.Model():
          beta = pm.Gamma('beta', a_gamma, b_gamma)
          data = pm.CustomDist('data', beta, logp=logp, observed=sample)

          start = {'beta': beta_guess, 'data': data_guess}
          step = pm.Metropolis(step_scale=step_size)
          gam_trace = pm.sample(trace_size, tune=burn, cores=4,
                                random_seed=seed, initvals=start,
                                step=step, chains=1, progressbar=False)

      gam_beta = np.mean(gam_trace.posterior.beta[0].to_numpy())
      gam_values = pm.hdi(gam_trace, hdi_prob=(1 - alpha))
      gam_values = gam_values.beta.to_numpy()
      gam_values = np.append(gam_values, gam_beta)

      gamma_info.append(gam_values)
```

```
      with pm.Model():
            beta = pm.Uniform('beta', a_unif, b_unif)
            data = pm.CustomDist('data', beta, logp=logp, observed=sample)

            start = {'beta': beta_guess, 'data':data_guess}
            step = pm.Metropolis(step_scale=step_size)
            unif_trace = pm.sample(trace_size, tune=burn, cores=4,
                                   random_seed=seed, initvals=start,
                                   step=step, chains=1, progressbar=False)

      unif_beta = np.mean(unif_trace.posterior.beta[0].to_numpy())
      unif_values = pm.hdi(unif_trace, hdi_prob=(1 - alpha))
      unif_values = unif_values.beta.to_numpy()
      unif_values = np.append(unif_values, unif_beta)

      unif_info.append(unif_values)

    return gamma_info, unif_info
```

A.9 Estatísticas dos experimentos bayesianos

```
#@title Function to create statistics
def create_statistics(beta: float, sizes: np.ndarray):

  # Dicionários para armazenar estatísticas das distribuições Gamma e Uniforme.
  gam_statistics ={'size': [], 'mean': [], 'var': [],
                   'bias': [], 'MSE': [], 'RMSE': [], 'PC': []}

  uni_statistics ={'size': [], 'mean': [], 'var': [],
                   'bias': [], 'MSE': [], 'RMSE': [], 'PC': []}


  # Itera sobre os diferentes tamanhos de amostra.
  for size in sizes:
    # Cria uma chave baseada no valor de beta para encontrar o arquivo CSV correspondente.
    if beta < 1:
      key = '0_' + str(beta).split('.')[1]

    else:
      spt = str(beta).split('.')
      key = spt[0] + '_' + spt[1]

    # Lê os dados do arquivo CSV gerado.
    df = pd.read_csv(f'beta{key}_n{size}.csv')

    # Estatísticas para as distribuições
    gam_data = df['Gamma'].to_numpy()

    gam_statistics['size'].append(size)
    gam_statistics['mean'].append(np.mean(gam_data))
```

```
    gam_statistics['var'].append(np.var(gam_data))
    gam_statistics['bias'].append(np.mean(gam_data) - beta)
    MSE = np.sum((beta - gam_data)**2) / len(gam_data)
    gam_statistics['MSE'].append(MSE)
    gam_statistics['RMSE'].append(np.sqrt(MSE))

    cout = 0
    for lower, upper in df[['Lower-Gamma', 'Upper-Gamma']].to_numpy():
      cout += 1 if lower < beta < upper else 0
    gam_statistics['PC'].append(cout / gam_data.shape[0])


    uni_data = df['Uniform'].to_numpy()

    uni_statistics['size'].append(size)
    uni_statistics['mean'].append(np.mean(uni_data))
    uni_statistics['var'].append(np.var(uni_data))
    uni_statistics['bias'].append(np.mean(uni_data) - beta)
    MSE = np.sum((beta- uni_data)**2) / len(uni_data)
    uni_statistics['MSE'].append(MSE)
    uni_statistics['RMSE'].append(np.sqrt(MSE))

    cout = 0
    for lower, upper in df[['Lower-Uniform', 'Upper-Uniform']].to_numpy():
      cout += 1 if lower < beta < upper else 0
    uni_statistics['PC'].append(cout / uni_data.shape[0])

  # Retorna DataFrames contendo as estatísticas para as distribuições Gamma e Uniforme.
  return pd.DataFrame(gam_statistics), pd.DataFrame(uni_statistics)
```