Universidade Estadual de Campinas

Departamento de Estatística

ME524 - Computação Aplicada à Estatística

Professor Guilherme Ludwig

Atividade 2

Luiz Felipe de Oliveira Barbosa Nunes - 255403

Campinas - SP
Novembro de 2024

1 Introdução

Nesta segunda atividade da disciplina **ME524 - Computação Aplicada à Estatística**, será desenvolvido um estudo baseado em um exercício adaptado de Greene (2008, p.240), utilizando modelos não-lineares disponibilizados pelo **National Institute of Standards and Technology (NIST)**. Esses modelos são projetados para avaliar a precisão e a eficiência de métodos computacionais de otimização. O objetivo principal da atividade é implementar os métodos de **Newton-Raphson**, **Gradient Descent** e **Line Search** para minimizar as funções de perda associadas a cada modelo. Adicionalmente, serão realizadas comparações entre as estimativas obtidas e os valores de referência, analisando não apenas a qualidade dos ajustes, mas também o desempenho computacional de cada método em termos de convergência e eficiência.

Os dois modelos considerados são:

- **Modelo 1: Misra1a**
  O modelo Misra1a (NIST, 2024) descreve estudos de adsorção de fluidos nos dentes e é definido por:

  $$y_i = \beta_1 \left(1 - e^{-\beta_2 x_i}\right) + \epsilon_i, \quad i = 1, \dots, n.$$

  - Parâmetros: β_1 e β_2 (reais e desconhecidos). - Dados disponíveis: listados no apêndice. - Valores iniciais:

  $$\boldsymbol{\beta}_1^{(0)} = (500, 0.0001)^T, \quad \boldsymbol{\beta}_2^{(0)} = (250, 0.0005)^T.$$

- **Modelo 2: Thurber**
  O modelo Thurber (NIST, 2024) representa a mobilidade de elétrons em semicondutores e é definido por:

  $$y_i = \frac{\beta_1 + \beta_2 x_i + \beta_3 x_i^2 + \beta_4 x_i^3}{1 + \beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3} + \epsilon_i, \quad i = 1, \dots, n.$$

  - Parâmetros: $\beta_1, \beta_2, \dots, \beta_7$ (reais e desconhecidos). - Dados disponíveis: listados no apêndice. - Valores iniciais:

  $$\boldsymbol{\beta}_1^{(0)} = (1000, 1000, 400, 40, 0.7, 0.3, 0.03)^T, \quad \boldsymbol{\beta}_2^{(0)} = (1300, 1500, 500, 75, 1, 0.4, 0.05)^T.$$

2 Resposta para o Modelo 1

A função de perda quadrática é definida como:

$$Q(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[y_i - \beta_1 \left(1 - e^{-\beta_2 x_i}\right)\right]^2$$

Gradiente

Derivada em relação a β_1:

$$Q(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[y_i - \beta_1 + \beta_1 e^{-\beta_2 x_i}\right]^2$$

$$\frac{\partial Q}{\partial \beta_1} = 2 \sum_{i=1}^{n} \left[y_i - \beta_1 \left(1 - e^{-\beta_2 x_i}\right)\right] \cdot \frac{\partial}{\partial \beta_1} \left[y_i - \beta_1 \left(1 - e^{-\beta_2 x_i}\right)\right]$$

$$\frac{\partial}{\partial \beta_1} \left[y_i - \beta_1 \left(1 - e^{-\beta_2 x_i}\right)\right] = -\left(1 - e^{-\beta_2 x_i}\right)$$

$$\frac{\partial Q}{\partial \beta_1} = -2 \sum_{i=1}^{n} \left[y_i - \beta_1 \left(1 - e^{-\beta_2 x_i}\right)\right] \cdot \left(1 - e^{-\beta_2 x_i}\right) = (-2 + 2e^{-\beta_2 x_i})(-\beta_1(1 - e^{-\beta_2 x_i}) + y_i)$$

Derivada em relação a β_2:

$$\frac{\partial Q}{\partial \beta_2} = 2\sum_{i=1}^{n}\left[y_i - \beta_1\left(1-e^{-\beta_2 x_i}\right)\right]\cdot\frac{\partial}{\partial \beta_2}\left[y_i - \beta_1\left(1-e^{-\beta_2 x_i}\right)\right]$$

$$\frac{\partial}{\partial \beta_2}\left[y_i - \beta_1\left(1-e^{-\beta_2 x_i}\right)\right] = \beta_1 x_i e^{-\beta_2 x_i}$$

$$\frac{\partial Q}{\partial \beta_2} = -2\sum_{i=1}^{n}\left[y_i - \beta_1\left(1-e^{-\beta_2 x_i}\right)\right]\cdot\beta_1 x_i e^{-\beta_2 x_i} = -2\beta_1 x_i(-\beta_1(1-e^{-\beta_2 x_i}) + y_i)e^{-\beta_2 x_i}$$

Hessiana

A matriz Hessiana é dada por:

$$H(\boldsymbol{\beta}) = \begin{bmatrix} \frac{\partial^2 Q}{\partial \beta_1^2} & \frac{\partial^2 Q}{\partial \beta_1 \partial \beta_2} \\ \frac{\partial^2 Q}{\partial \beta_2 \partial \beta_1} & \frac{\partial^2 Q}{\partial \beta_2^2} \end{bmatrix}$$

$$\frac{\partial^2 Q}{\partial \beta_1^2} = 2\sum_{i=1}^{n}\left(1-e^{-\beta_2 x_i}\right)^2 = (-2+2e^{-\beta_2 x_i})(-1+e^{-\beta_2 x_i})$$

$$\frac{\partial^2 Q}{\partial \beta_1 \partial \beta_2} = 2\sum_{i=1}^{n} x_i e^{-\beta_2 x_i}\left(1-e^{-\beta_2 x_i}\right) = -\beta_1 x_i(-2+2e^{-\beta_2 x_i})e^{-\beta_2 x_i} - 2x_i(-\beta_1(1-e^{-\beta_2 x_i}) + y_i)e^{-\beta_2 x_i}$$

$$\frac{\partial^2 Q}{\partial \beta_2^2} = 2\sum_{i=1}^{n}\beta_1^2 x_i^2 e^{-2\beta_2 x_i} - 2\sum_{i=1}^{n}\left[y_i - \beta_1\left(1-e^{-\beta_2 x_i}\right)\right]\cdot\beta_1 x_i^2 e^{-\beta_2 x_i} = 2\beta_1^2 x_i^2 e^{-2\beta_2 x_i} + 2\beta_1 x_i^2(-\beta_1(1-e^{-\beta_2 x_i}) + y_i)e^{-\beta_2 x_i}$$

2.1 Resultados do Método de Newton-Raphson

Para ajustar os parâmetros do modelo 1, utilizamos o método de Newton-Raphson com dois valores iniciais. O critério de parada foi definido como $\|\Delta\beta\|_2 < \epsilon$, com $\epsilon = 10^{-6}$. Devido à proximidade da matriz Hessiana com singularidade, apliquei uma regularização adicionando um termo γ à diagonal da Hessiana, resultando em:

$$H_{\text{reg}} = H + \gamma I$$

onde H é a matriz Hessiana original e $\gamma = 0.3$ foi escolhido para garantir estabilidade numérica e convergência.

Os resultados obtidos foram:

- Para $\boldsymbol{\beta}^{(0)} = [500, 0.0001]$:
$$\boldsymbol{\beta}^* = \begin{bmatrix} 448.8459 \\ 0.000272 \end{bmatrix}$$
- Para $\boldsymbol{\beta}^{(0)} = [250, 0.0005]$:
$$\boldsymbol{\beta}^* = \begin{bmatrix} 238.9547 \\ 0.0005501 \end{bmatrix}$$

Após testes com diferentes valores de γ, valores entre 0.3 a 1 eram valores bons. Portanto, escolhi $\gamma = 0.3$, pois esse valor garantiu a inversibilidade da Hessiana regularizada.

2.2 Line Search no Método de Newton-Raphson

O método de Newton-Raphson pode ser aprimorado por meio de uma estratégia de *line search*, na qual buscamos um tamanho de passo $\gamma_k \in (0, 1)$ que minimize a função objetivo ao longo da direção de atualização definida pelo método. Essa abordagem é especialmente útil em cenários onde as derivadas da função objetivo apresentam instabilidades ou quando a matriz Hessiana está próxima de ser singular, como identificado neste caso.

As Figuras 1 e 2 ilustram o comportamento da função objetivo em relação aos valores de $\gamma \in [0.01, 1]$, considerando o primeiro e o segundo conjuntos de valores iniciais, respectivamente. A função objetivo foi avaliada como a soma dos quadrados do gradiente regularizado.

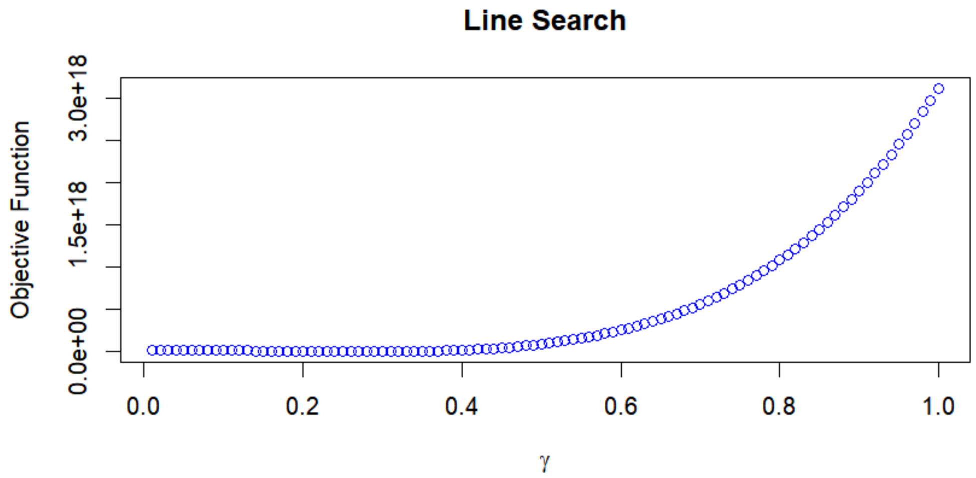

Figura 1: Gráfico do Line Search para diferentes valores de γ considerando o primeiro conjunto de valores iniciais.

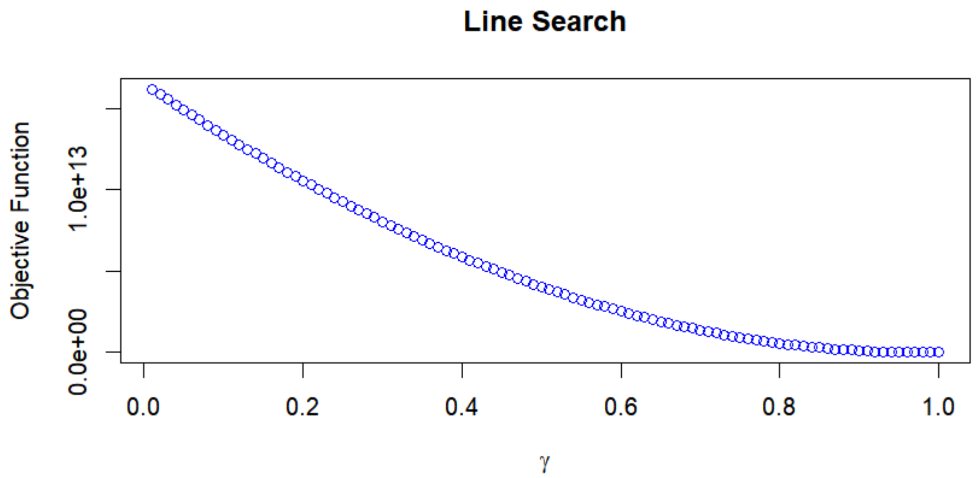

Figura 2: Gráfico do Line Search para diferentes valores de γ considerando o segundo conjunto de valores iniciais.

Com base nos gráficos, verificamos que o valor $\gamma = 0.1$ demonstrou ser um ponto de equilíbrio adequado entre estabilidade e eficiência na Figura 1. Enquanto, na 2, verificamos que o valor $\gamma = 1$ demonstrou ser um ponto de equilíbrio adequado entre estabilidade e eficiência.

Além disso, não foi necessário recorrer ao método da bisseção para buscar $\gamma \in (0, 1)$, pois o procedimento de *line search* permitiu identificar diretamente um valor apropriado de γ para os dois conjuntos de valores iniciais.

O método de Newton-Raphson com *line search* apresentou os seguintes resultados:

- Para o primeiro conjunto de valores iniciais, utilizando $\gamma = 0.1$, a convergência foi alcançada após 285 iterações. O vetor de parâmetros estimado foi:

$$\boldsymbol{\beta}^* = \begin{bmatrix} 238.9421 \\ 0.0005502 \end{bmatrix}$$

- Para o segundo conjunto de valores iniciais, utilizando $\gamma = 0.5$, a convergência foi alcançada após 31 iterações. O vetor de parâmetros estimado foi:

$$\boldsymbol{\beta}^* = \begin{bmatrix} 238.9421 \\ 0.0005502 \end{bmatrix}$$

Utilizando $\gamma = 0.5$ no segundo conjunto de valores iniciais, o método foi eficiente, convergindo rapidamente para o mesmo resultado obtido no primeiro conjunto de valores iniciais. Pois, testes anteriores com $\gamma = 1.0$ levaram à ocorrência de `NA` no *line search*, indicando que o intervalo de candidatos para γ pode não ser apropriado.

2.3 Resultados do Método de Gradient Descent com Line Search

2.3.1 Descida de Gradiente com Tamanho de Passo Dinâmico

Neste método, o tamanho de passo γ_k é ajustado dinamicamente em cada iteração de acordo com a fórmula:

$$\gamma_k = \gamma \cdot 2^{-k},$$

onde $\gamma > 0$ é um valor inicial e k representa o número da iteração atual. A atualização dos parâmetros é feita segundo a seguinte expressão:

$$\boldsymbol{\beta}_k = \boldsymbol{\beta}_{k-1} - \gamma_k \nabla Q(\boldsymbol{\beta}_{k-1}),$$

onde $\nabla Q(\boldsymbol{\beta})$ é o gradiente da função objetivo calculado no ponto atual.

Os resultados para os dois conjuntos de valores iniciais foram:

- Para o primeiro conjunto de valores iniciais ($\boldsymbol{\beta}^{(0)} = [50, 0.001]$), a convergência foi alcançada na iteração 28 com $\gamma_k = 3.725 \times 10^{-10}$. O vetor de parâmetros estimado foi:

$$\boldsymbol{\beta}^* = \begin{bmatrix} 104.8982 \\ 7869687.4451 \end{bmatrix}$$

- Para o segundo conjunto de valores iniciais ($\boldsymbol{\beta}^{(0)} = [100, 0.0005]$), a convergência foi alcançada na iteração 27 com $\gamma_k = 7.450 \times 10^{-10}$. O vetor de parâmetros estimado foi:

$$\boldsymbol{\beta}^* = \begin{bmatrix} 71.16247 \\ 203191.7789 \end{bmatrix}$$

2.3.2 Descida de Gradiente com Tamanho de Passo Fixo

Neste método, o tamanho de passo γ é mantido constante em todas as iterações. A atualização dos parâmetros segue a fórmula:

$$\boldsymbol{\beta}_k = \boldsymbol{\beta}_{k-1} - \gamma \nabla Q(\boldsymbol{\beta}_{k-1}),$$

onde γ é um valor fixo especificado antes do início do algoritmo. Neste caso, utilizamos $\gamma = 0.02$.

Os resultados para os dois conjuntos de valores iniciais foram:

- Para o primeiro conjunto de valores iniciais ($\boldsymbol{\beta}^{(0)} = [50, 0.001]$), a convergência foi alcançada na iteração 26. O vetor de parâmetros estimado foi:

$$\boldsymbol{\beta}^* = \begin{bmatrix} 43.34071 \\ 3147875 \end{bmatrix}$$

- Para o segundo conjunto de valores iniciais ($\boldsymbol{\beta}^{(0)} = [100, 0.0005]$), a convergência foi alcançada na iteração 25. O vetor de parâmetros estimado foi:

$$\boldsymbol{\beta}^* = \begin{bmatrix} 43.34071 \\ 81276.71186 \end{bmatrix}$$

2.4 Comparação com os Resultados Certificados

Os resultados obtidos para o Modelo 1 foram comparados com os valores certificados fornecidos no site de referência. Abaixo estão apresentados os valores para β_1 e β_2, bem como o número de iterações necessário para a convergência.

Os valores certificados são:

$$\beta_1 = 238.94212918, \quad \beta_2 = 5.5015643181 \times 10^{-4}$$

com os seguintes parâmetros adicionais:

- **Soma dos Quadrados dos Resíduos:** 1.2455138894E-01
- **Desvio Padrão:** 1.0187876330E-01
- **Graus de Liberdade:** 12

Os resultados indicam que:

- O método de Newton-Raphson apresentou os valores mais próximos aos resultados certificados, especialmente para o segundo conjunto de valores iniciais. Além disso, convergiu rapidamente no segundo conjunto de valores iniciais (31 iterações).
- O método de gradient descent com tamanho de passo dinâmico ($\gamma_k = \gamma \cdot 2^{-k}$) apresentou valores de $\boldsymbol{\beta}$ que divergem consideravelmente dos valores certificados. Apesar disso, ele mostrou um número moderado de iterações.
- O método de gradient descent com tamanho de passo fixo ($\gamma = 0.02$) também apresentou resultados muito distantes dos valores certificados.

Portanto, o método de Newton-Raphson com regularização ($\gamma = 0.3$) se mostrou o mais robusto e eficiente para este problema, sendo o mais adequado para ajustes do Modelo 1.

3 Resposta para o Modelo 2

A função de perda quadrática para o modelo 2 é definida como:

$$Q(\boldsymbol{\beta}) = \sum_{i=1}^{n} [y_i - f(x_i, \boldsymbol{\beta})]^2$$

onde:

$$f(x_i, \boldsymbol{\beta}) = \frac{\beta_1 + \beta_2 x_i + \beta_3 x_i^2 + \beta_4 x_i^3}{1 + \beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3}$$

Gradiente

Derivada em relação a β_1:

$$\frac{\partial Q}{\partial \beta_1} = -2\sum_{i=1}^{n} \frac{[y_i - f(x_i, \boldsymbol{\beta})]}{1 + \beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3}$$

Derivada em relação a β_2:

$$\frac{\partial Q}{\partial \beta_2} = -2\sum_{i=1}^{n} \frac{x_i\,[y_i - f(x_i, \boldsymbol{\beta})]}{1 + \beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3}$$

Derivada em relação a β_3:

$$\frac{\partial Q}{\partial \beta_3} = -2\sum_{i=1}^{n} \frac{x_i^2\,[y_i - f(x_i, \boldsymbol{\beta})]}{1 + \beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3}$$

Derivada em relação a β_4:

$$\frac{\partial Q}{\partial \beta_4} = -2\sum_{i=1}^{n} \frac{x_i^3\,[y_i - f(x_i, \boldsymbol{\beta})]}{1 + \beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3}$$

Derivada em relação a β_5:

$$\frac{\partial Q}{\partial \beta_5} = 2\sum_{i=1}^{n} \frac{x_i\,[y_i - f(x_i, \boldsymbol{\beta})]\left(\beta_1 + \beta_2 x_i + \beta_3 x_i^2 + \beta_4 x_i^3\right)}{\left(1 + \beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3\right)^2}$$

Derivada em relação a β_6:

$$\frac{\partial Q}{\partial \beta_6} = 2\sum_{i=1}^{n} \frac{x_i^2\,[y_i - f(x_i, \boldsymbol{\beta})]\left(\beta_1 + \beta_2 x_i + \beta_3 x_i^2 + \beta_4 x_i^3\right)}{\left(1 + \beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3\right)^2}$$

Derivada em relação a β_7:

$$\frac{\partial Q}{\partial \beta_7} = 2\sum_{i=1}^{n} \frac{x_i^3\,[y_i - f(x_i, \boldsymbol{\beta})]\left(\beta_1 + \beta_2 x_i + \beta_3 x_i^2 + \beta_4 x_i^3\right)}{\left(1 + \beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3\right)^2}$$

Hessiana

A matriz Hessiana do Modelo 2 é uma matriz 7×7 calculada a partir da função de perda quadrática $Q(\boldsymbol{\beta})$. Cada elemento da matriz é dado por:

$$H_{i,j} = \frac{\partial^2 Q}{\partial \beta_i \partial \beta_j}$$

As derivadas parciais que compõem a matriz Hessiana apresentam padrões relacionados aos termos x, x^2, x^3 e às combinações com os denominadores do modelo. Esses padrões incluem:

1. **Diagonal Principal ($H_{i,i}$):**
   Para os parâmetros $\beta_1, \beta_2, \beta_3$ e β_4, as derivadas são proporcionais a x^k, onde $k = 0, 1, 2, 3$, multiplicados pelo inverso do denominador elevado ao quadrado:

$$H_{i,i} = \sum_{i=1}^{n} \frac{2\,x^{2(k-1)}}{\left(1 + \beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3\right)^2}$$

2. **Cruzamentos com $\beta_5, \beta_6, \beta_7$:**
   Os cruzamentos entre $\beta_1, \beta_2, \beta_3, \beta_4$ e os parâmetros $\beta_5, \beta_6, \beta_7$ incluem produtos entre x^k e os termos associados ao numerador e denominador:

$$H_{i,j} = \sum_{i=1}^{n} \frac{2\, x^{(k+m)}}{(1 + \beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3)^2}$$

3. **Padrão no Denominador:**
   Cada derivada parcial contém o fator $1 + \beta_5 x + \beta_6 x^2 + \beta_7 x^3$ elevado a potências dependentes do grau de interação, como $(1 + \beta_5 x + \beta_6 x^2 + \beta_7 x^3)^{-2}$ ou $(1 + \beta_5 x + \beta_6 x^2 + \beta_7 x^3)^{-3}$. Para elementos relacionados a $\beta_5, \beta_6, \beta_7$, há termos adicionais envolvendo interações com o numerador e os gradientes.

Elemento $H_{1,1}$:

$$H_{1,1} = \sum_{i=1}^{n} \frac{2}{(1 + \beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3)^2}$$

Elemento $H_{1,5}$:

$$H_{1,5} = \sum_{i=1}^{n} \frac{-4x_i\,(y_i - f(x_i, \boldsymbol{\beta}))\left(\beta_1 + \beta_2 x_i + \beta_3 x_i^2 + \beta_4 x_i^3\right)}{(1 + \beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3)^3}$$

Elemento $H_{5,5}$:

$$H_{5,5} = \sum_{i=1}^{n} \left(-4x_i^2 \frac{(y_i - f(x_i, \boldsymbol{\beta}))\left(\beta_1 + \beta_2 x_i + \beta_3 x_i^2 + \beta_4 x_i^3\right)}{(1 + \beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3)^3} + \frac{2x_i^4\left(\beta_1 + \beta_2 x_i + \beta_3 x_i^2 + \beta_4 x_i^3\right)^2}{(1 + \beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3)^4} \right)$$

Esses padrões se repetem para todos os elementos da matriz Hessiana.

3.1 Resultados do Método de Newton-Raphson

O método de Newton-Raphson com regularização foi aplicado para ajustar os parâmetros do Modelo 2. No entanto, o algoritmo não convergiu para uma solução em um número aceitável de iterações, mesmo após várias tentativas de ajustes nos parâmetros e no método.

O método foi configurado para realizar no máximo 10.000 iterações, com um critério de parada baseado na norma da diferença entre os vetores de parâmetros consecutivos ($\|\beta_{k+1} - \beta_k\| < 10^{-6}$). Apesar disso, o algoritmo alcançou o limite máximo de iterações sem convergir para uma solução.

O comportamento dos parâmetros ao longo das iterações foi analisado, e observou-se que os valores de alguns parâmetros cresceram de forma descontrolada, enquanto outros permaneceram relativamente estáveis. Esse comportamento está representado na Figura 3, que demonstra claramente a ausência de convergência.

Além disso, as previsões geradas pelos valores finais dos parâmetros ajustados foram comparadas aos dados observados. A Figura 4 ilustra essa comparação, indicando que as curvas ajustadas não conseguem capturar adequadamente os padrões dos dados.

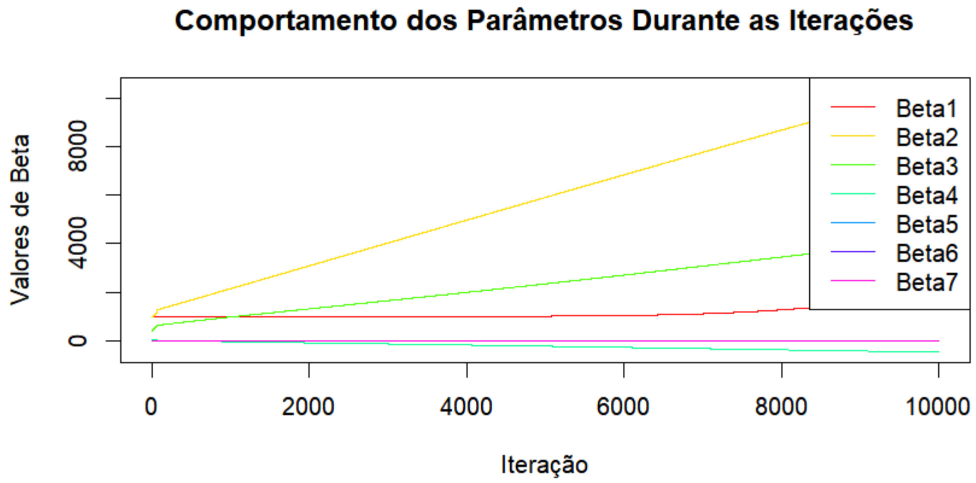

Figura 3: Comportamento dos parâmetros durante as iterações do ajuste do Modelo 2. Observa-se que os parâmetros não convergiram para valores estáveis.

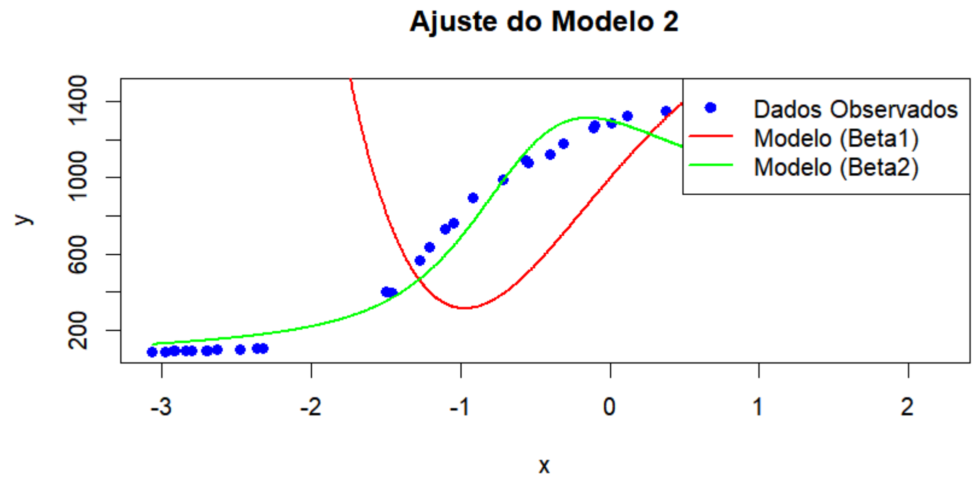

Figura 4: Comparação entre os dados observados e as previsões do Modelo 2 para os conjuntos de valores iniciais β_0 diferentes. Nota-se que o modelo não consegue ajustar corretamente os dados.

3.2 Resultados do Método de Newton-Raphson com Line-Search

O método de Newton-Raphson com **line-search adaptativo** foi aplicado ao Modelo 2, considerando dois conjuntos de valores iniciais: β_{init1} e β_{init2}. Os resultados mostram uma **convergência bem-sucedida** para o primeiro conjunto (β_{init1}), enquanto o segundo conjunto (β_{init2}) não alcançou convergência dentro do número máximo de iterações permitido.

Convergência para β_{init1}

O método convergiu após 7141 iterações para os seguintes valores de parâmetros:

$$\beta_{\text{final1}} = \begin{bmatrix} 1000.0 \\ 1000.0 \\ 400.0 \\ 40.0 \\ 0.7 \\ 0.28 \\ -0.048 \end{bmatrix}.$$

Isso demonstra que, com valores iniciais adequados, o método é capaz de ajustar os dados do modelo de forma satisfatória, conforme representado nos gráficos de ajustes e comparações. A Figura 5 apresenta o comportamento da função objetivo ao longo das iterações, evidenciando uma redução progressiva até a estabilização.

Falha de Convergência para β_{init2}

Para o segundo conjunto de valores iniciais (β_{init2}), o método não conseguiu convergir, alcançando o limite máximo de 10.000 iterações. Os parâmetros finais alcançados foram:

$$\beta_{\text{final2}} = \begin{bmatrix} 1300.0 \\ 1500.0 \\ 500.0 \\ 75.0 \\ 1.0 \\ 0.51 \\ -0.089 \end{bmatrix}.$$

Embora os valores estejam próximos de uma solução razoável, o critério de parada baseado na norma $\|\beta_{k+1} - \beta_k\| < 10^{-6}$ não foi atendido. Isso pode indicar que:

- O conjunto β_{init2} está em uma região onde o método enfrenta dificuldades de convergência.
- Problemas de condicionamento da matriz Hessiana podem ter dificultado o ajuste adequado.

A Figura 5 apresenta a convergência da função objetivo para ambos os conjuntos de valores iniciais. Observa-se que, para β_{init1}, há uma redução contínua e gradual até atingir a convergência. Por outro lado, para β_{init2}, a função objetivo estabiliza sem atingir um ponto de solução satisfatório.

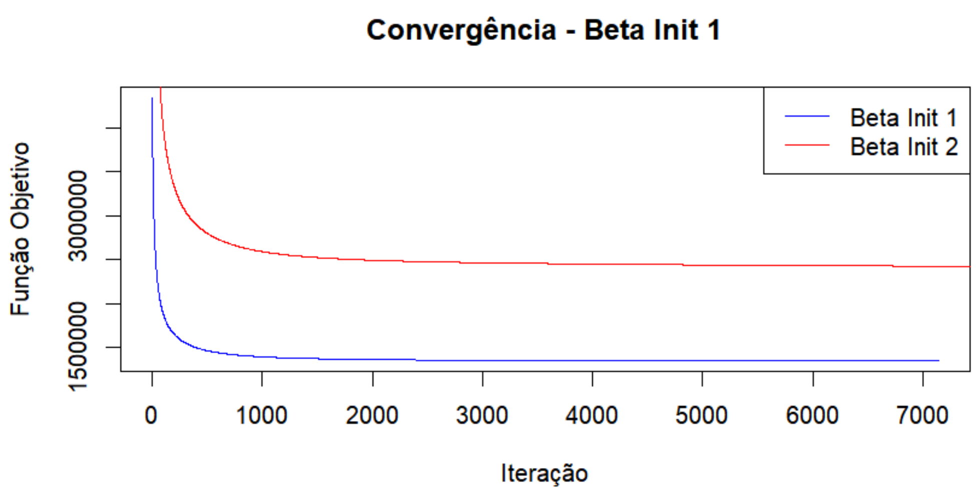

Figura 5: Convergência do Método de Newton-Raphson com Line Search para os dois conjuntos de valores iniciais (β_{init1} e β_{init2}).

A Figura 8 apresenta o movimento da solução ao longo das iterações do método de Newton-Raphson com line-search, sobreposto às curvas de nível da função objetivo. Esse gráfico permite visualizar como a solução inicial (β_{init}) se desloca em direção ao ponto de convergência.

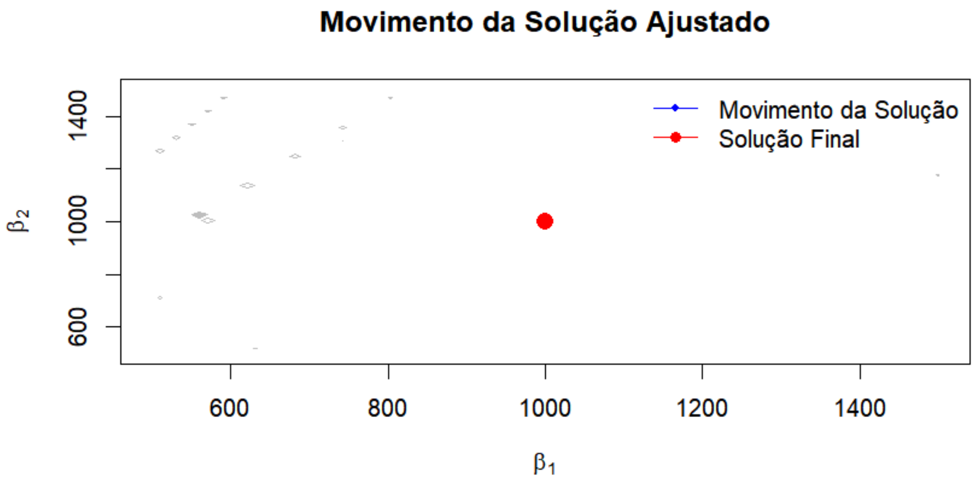

Figura 6: Movimento da solução ao longo das iterações, sobreposto às curvas de nível da função objetivo. O ponto vermelho indica a solução final alcançada pelo método.

3.3 Resultados do Método de Gradient Descent com Line-Search

O método de **Gradient Descent com Line-Search** foi aplicado ao **Modelo 2** utilizando os dois conjuntos de valores iniciais indicados: β_{init1} e β_{init2}. O objetivo era minimizar a função objetivo Q de forma eficiente, ajustando os parâmetros do modelo. Os resultados indicaram que o método **não convergiu** para nenhuma das duas inicializações, mesmo após 10.000 iterações.

Para β_{init1}, os parâmetros finais foram:

$$\beta_{\text{final1}} = \{-742.73, 10273.85, -24937.41, 72928.49, -5818842.0, 15281940.0, -45648860.0\}.$$

Para β_{init2}, os parâmetros finais foram:

$$\beta_{\text{final2}} = \{59516.81, -152048.60, 416498.90, -1143114.0, -383415900.0, 1065535000.0, -2980991000.0\}.$$

A função objetivo, no entanto, não alcançou valores satisfatórios, sugerindo que o método não conseguiu encontrar um mínimo local ou global.

A Figura 7 apresenta o comportamento da função objetivo regularizada ao longo das iterações para os dois conjuntos de valores iniciais.

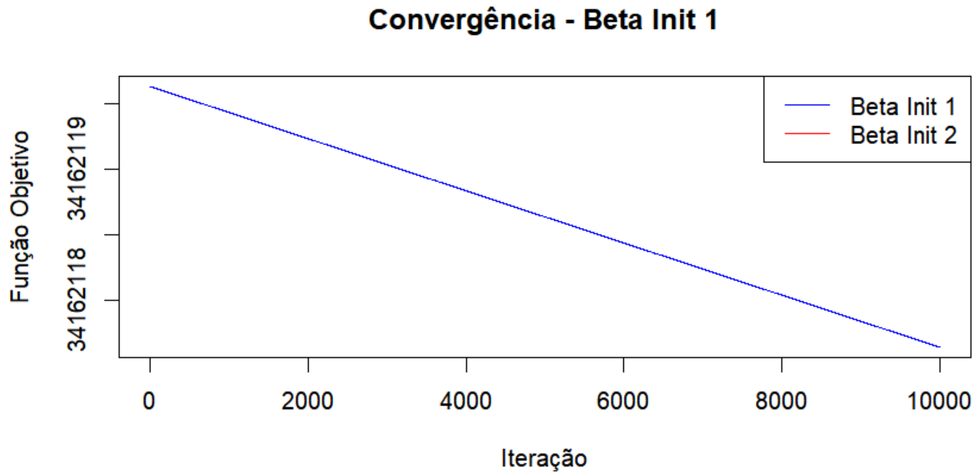

Figura 7: Comportamento da função objetivo ao longo das iterações. O método não alcançou convergência para nenhum dos conjuntos iniciais.

Embora a função objetivo tenha diminuído ligeiramente ao longo das iterações, a convergência para um valor aceitável não foi alcançada. Esse comportamento sugere que os gradientes calculados são insuficientes para guiar o método a um mínimo local ou global.

Adicionalmente, o movimento da solução ao longo das iterações foi analisado. A Figura 8 apresenta as trajetórias de β_1 e β_2 ao longo do processo iterativo:

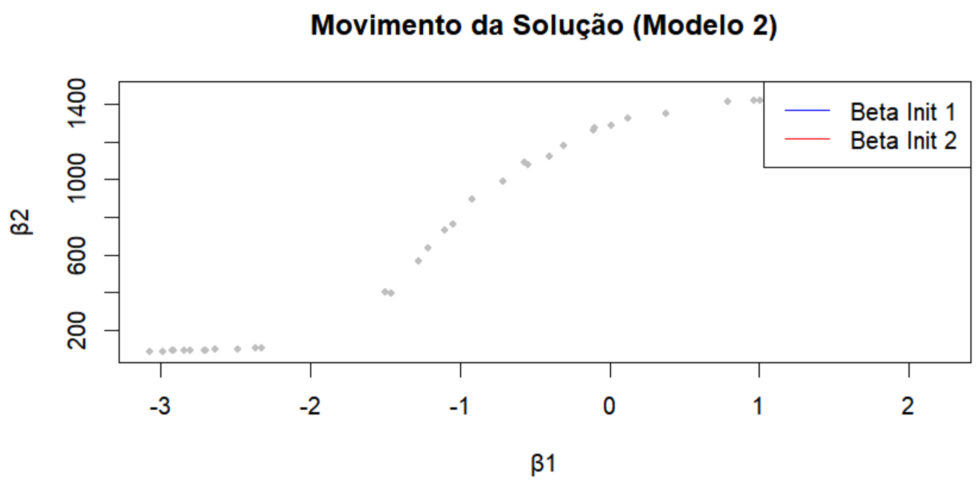

Figura 8: Movimento da solução no espaço de parâmetros (β_1, β_2) ao longo das iterações para os dois conjuntos iniciais.

Com base nos resultados obtidos, realizei as seguintes melhorias no método de Gradient Descent:

- **Diagnóstico do Gradiente**: Foi avaliado se os valores do gradiente diminuem adequadamente durante as iterações, sendo adicionada a verificação da norma do gradiente.
- **Regularização mais Forte**: O termo de regularização λ foi ajustado para controlar os valores extremos dos parâmetros.
- **Ajuste Dinâmico de γ**: Reduzi os valores iniciais dos candidatos de γ e adicionados ajustes dinâmicos para otimizar o tamanho do passo.

- **Normalização dos Dados**: Certifiquei de que as variáveis x e y estavam bem normalizadas para evitar instabilidades no gradiente.
- **Critério de Parada Melhorado**: Adicionei um critério baseado na norma do gradiente e na mudança relativa da função objetivo.
- **Limitação dos Valores de β**: Implementei restrições para evitar que os parâmetros cresçam excessivamente.

Porém, mesmo assim, não consegui obter a convergência nos pontos iniciais.

4 Apêndice - Código Completo

A seguir, segue o código em linguagem R utilizado neste relatório:

```r
# Modelo 1 - Dados Misra1a
Misra1a <- data.frame(y = c(10.07, 14.73, 17.94, 23.93, 29.61,
                            35.18, 40.02, 44.82, 50.76, 55.05,
                            61.01, 66.4, 75.47, 81.78),
                      x = c(77.6, 114.9, 141.1, 190.8, 239.9,
                            289, 332.8, 378.4, 434.8, 477.3,
                            536.8, 593.1, 689.1, 760))
beta_init1 <- c(500, 0.0001)  # Primeiro valor inicial
beta_init2 <- c(250, 0.0005)  # Segundo valor inicial

# Função para calcular o gradiente
gradient <- function(beta, data) {
  y <- data$y
  x <- data$x
  g1 <- sum((-2 + 2 * exp(-beta[2] * x)) * (-beta[1] * (1 - exp(-beta[2] * x)) + y))
  g2 <- sum(-2 * beta[1] * x * (-beta[1] * (1 - exp(-beta[2] * x)) + y) * exp(-beta[2] * x))
  return(c(g1, g2))
}

# Função para calcular a matriz Hessiana
hessian <- function(beta, data) {
  y <- data$y
  x <- data$x
  h11 <- sum((-2 + 2 * exp(-beta[2] * x)) * (-1 + exp(-beta[2] * x)))
  h12 <- sum(-beta[1] * x * (-2 + 2 * exp(-beta[2] * x)) * exp(-beta[2] * x) -
             2 * x * (-beta[1] * (1 - exp(-beta[2] * x)) + y) * exp(-beta[2] * x))
  h21 <- h12
  h22 <- sum(2 * beta[1]^2 * x^2 * exp(-2 * beta[2] * x) +
             2 * beta[1] * x^2 * (-beta[1] * (1 - exp(-beta[2] * x)) + y) * exp(-beta[2] * x))
  return(matrix(c(h11, h12, h21, h22), nrow = 2, byrow = TRUE))
}

# Função do método de Newton-Raphson
newton_raphson <- function(beta_init, data, epsilon = 1e-6, max_iter = 1000, gamma = 0.4) {
  beta <- beta_init
  for (i in 1:max_iter) {
    grad <- gradient(beta, data)
    hess <- hessian(beta, data)

    hess_regularized <- hess + diag(gamma, nrow(hess))


    delta_beta <- solve(hess_regularized) %*% grad

    beta_new <- beta - delta_beta

    if (sqrt(sum((beta_new - beta)^2)) < epsilon) {
      return(beta_new)
    }

    beta <- beta_new
  }

  return(beta)
}


# Resultados do método de Newton-Raphson
result1 <- newton_raphson(beta_init1, Misra1a)
result1

result2 <- newton_raphson(beta_init2, Misra1a)
result2


# Função para calcular a função objetivo
objective_function <- function(beta, data) {
  y <- data$y
  x <- data$x
  residuals <- y - beta[1] * (1 - exp(-beta[2] * x))
  return(sum(residuals^2))
}

# Função para criar o gráfico de line search
line_search_plot <- function(beta, grad, hess, data, gamma_candidates) {
  direction <- -solve(hess) %*% grad
  objective_values <- numeric(length(gamma_candidates))

  # Avalia a função objetivo para cada gamma
  for (i in seq_along(gamma_candidates)) {
    gamma <- gamma_candidates[i]
    beta_new <- beta + gamma * direction
    objective_values[i] <- objective_function(beta_new, data)
  }

  plot(gamma_candidates, objective_values, type = "b", col = "blue",
       main = "Line Search", xlab = expression(gamma),
       ylab = "Objective Function")
}

gamma_candidates <- seq(0.01, 1, by = 0.01)

# Gera o gráfico
line_search_plot(beta_init1, gradient(beta_init1, Misra1a), hessian(beta_init1, Misra1a), Misra1a, gamma_candidates)
line_search_plot(beta_init2, gradient(beta_init2, Misra1a), hessian(beta_init2, Misra1a), Misra1a, gamma_candidates)
```

```r

# Função de line search
line_search <- function(beta, grad, hess, data, gamma_candidates) {
  direction <- -solve(hess) %*% grad
  for (gamma in gamma_candidates) {
    beta_new <- beta + gamma * direction
    if (objective_function(beta_new, data) < objective_function(beta, data)) {
      return(list(beta_new = beta_new, gamma = gamma))
    }
  }
  return(list(beta_new = beta, gamma = NA))
}

# Método de Newton-Raphson com line search ajustado
newton_raphson_ls <- function(beta_init, data, epsilon = 1e-6, max_iter = 10000, gamma_candidates) {
  beta <- beta_init
  for (i in 1:max_iter) {
    grad <- gradient(beta, data)
    hess <- hessian(beta, data)

    # Verifica invertibilidade da Hessiana
    if (det(hess) == 0) {
      stop("A matriz Hessiana não é invertível.")
    }

    # Line search
    ls_result <- line_search(beta, grad, hess, data, gamma_candidates)
    beta_new <- ls_result$beta_new
    gamma <- ls_result$gamma

    cat("Iteração", i, "- Gamma:", gamma, "- Beta:", beta_new, "\n")

    # Critério de parada
    if (sqrt(sum((beta_new - beta)^2)) < epsilon) {
      cat("Convergência atingida na iteração", i, "\n")
      return(list(beta = beta_new, converged = TRUE))
    }

    beta <- beta_new
  }
  cat("Máximo de iterações alcançado sem convergência.\n")
  return(list(beta = beta, converged = FALSE))
}


# Resultados para o primeiro conjunto de valores iniciais
result1 <- newton_raphson_ls(beta_init1, Misra1a, gamma_candidates = 0.1)
result1

# Resultados para o segundo conjunto de valores iniciais
result2 <- newton_raphson_ls(beta_init2, Misra1a, gamma_candidates = 0.5)
result2



# Função de descida de gradiente
gradient_descent_ls <- function(beta_init, data, gamma = 0.1, epsilon = 1e-6, max_iter = 1000) {
  beta <- beta_init
  for (k in 1:max_iter) {
    grad <- gradient(beta, data)
    gamma_k <- gamma * 2^(-k)
    beta_new <- beta - gamma_k * grad

    # Exibe a iteração atual
    cat("Iteração", k, "- Gamma_k:", gamma_k, "- Beta:", beta_new, "\n")

    # Critério de parada
    if (sqrt(sum((beta_new - beta)^2)) < epsilon) {
      cat("Convergência atingida na iteração", k, "\n")
      return(beta_new)
    }

    beta <- beta_new
  }
  cat("Máximo de iterações alcançado sem convergência.\n")
  return(beta)
}

result1 <- gradient_descent_ls(beta_init1, Misra1a)
result1

result2 <- gradient_descent_ls(beta_init2, Misra1a)
result2


# Função de descida de gradiente com gamma fixo
gradient_descent_fixed <- function(beta_init, data, gamma = 0.02, epsilon = 1e-6, max_iter = 1000) {
  beta <- beta_init
  for (k in 1:max_iter) {
    grad <- gradient(beta, data)
    beta_new <- beta - gamma * grad
    beta_new <- pmax(beta_new, 0)

    # Exibe a iteração atual
    cat("Iteração", k, "- Gamma:", gamma, "- Beta:", beta_new, "\n")

    # Critério de parada
    if (sqrt(sum((beta_new - beta)^2)) < epsilon) {
      cat("Convergência atingida na iteração", k, "\n")
      return(beta_new)
    }
    beta <- beta_new
```

```r
  }

  cat("Máximo de iterações alcançado sem convergência.\n")
  return(beta)
}

result1 <- gradient_descent_fixed(beta_init1, Misra1a)
result1

result2 <- gradient_descent_fixed(beta_init2, Misra1a)
result2



#Modelo 2 - Dados Thurber
Thurber <- data.frame(y = c(80.574, 84.248, 87.264, 87.195, 89.076, 89.608,
                            89.868, 90.101, 92.405, 95.854, 100.696, 101.06,
                            401.672, 390.724, 567.534, 635.316, 733.054,
                            759.087, 894.206, 990.785, 1090.109, 1080.914,
                            1122.643, 1178.351, 1260.531, 1273.514, 1288.339,
                            1327.543, 1353.863, 1414.509, 1425.208, 1421.384,
                            1442.962, 1464.35, 1468.705, 1447.894, 1457.628),
                      x = c(-3.067, -2.981, -2.921, -2.912, -2.84, -2.797,
                            -2.702, -2.699, -2.633, -2.481, -2.363, -2.322,
                            -1.501, -1.46, -1.274, -1.212, -1.1, -1.046,
                            -0.915, -0.714, -0.566, -0.545, -0.4, -0.309,
                            -0.109, -0.103, 0.01, 0.119, 0.377, 0.79, 0.963,
                            1.006, 1.115, 1.572, 1.841, 2.047, 2.2))

beta0_chapeu_thurber_1 <- c(1000, 1000, 400, 40, 0.7, 0.3, 0.03)
beta0_chapeu_thurber_2 <- c(1300, 1500, 500, 75, 1, 0.4, 0.05)

# Gradiente
gradient_thurber <- function(beta, data) {
  x <- data$x
  y <- data$y

  num <- beta[1] + beta[2] * x + beta[3] * x^2 + beta[4] * x^3
  denom <- 1 + beta[5] * x + beta[6] * x^2 + beta[7] * x^3
  f <- num / denom

  grad <- numeric(7)
  grad[1] <- sum(-2 * (y - f) / denom)
  grad[2] <- sum(-2 * x * (y - f) / denom)
  grad[3] <- sum(-2 * x^2 * (y - f) / denom)
  grad[4] <- sum(-2 * x^3 * (y - f) / denom)
  grad[5] <- sum(2 * x * (y - f) * (num / denom^2))
  grad[6] <- sum(2 * x^2 * (y - f) * (num / denom^2))
  grad[7] <- sum(2 * x^3 * (y - f) * (num / denom^2))

  return(grad)
}

#Hessiana
hessian_thurber <- function(beta, data) {
  x <- data$x
  y <- data$y
  num <- beta[1] + beta[2] * x + beta[3] * x^2 + beta[4] * x^3
  denom <- 1 + beta[5] * x + beta[6] * x^2 + beta[7] * x^3
  f <- num / denom
  hessian <- matrix(0, nrow = 7, ncol = 7)

  for (i in 1:length(x)) {

    common_term <- y[i] - f[i]
    d1 <- 1 / denom[i]
    d2 <- num[i] / denom[i]^2
    d3 <- denom[i]^3
    d4 <- denom[i]^4

    # Diagonais principais
    hessian[1, 1] <- hessian[1, 1] + 2 * d1^2
    hessian[2, 2] <- hessian[2, 2] + 2 * (x[i] * d1)^2
    hessian[3, 3] <- hessian[3, 3] + 2 * (x[i]^2 * d1)^2
    hessian[4, 4] <- hessian[4, 4] + 2 * (x[i]^3 * d1)^2
    hessian[5, 5] <- hessian[5, 5] + 2 * x[i]^2 * ((-4 * common_term * d2) + (num[i] * x[i]^2 / d4))
    hessian[6, 6] <- hessian[6, 6] + 2 * x[i]^4 * ((-4 * common_term * d2) + (num[i] * x[i]^4 / d4))
    hessian[7, 7] <- hessian[7, 7] + 2 * x[i]^6 * ((-4 * common_term * d2) + (num[i] * x[i]^6 / d4))

    # Termos cruzados
    for (j in 1:7) {
      for (k in j:7) {
        if (j <= 4 && k <= 4) {
          hessian[j, k] <- hessian[j, k] + 2 * (x[i]^(j - 1) * x[i]^(k - 1) * d1^2)
        } else if (j > 4 || k > 4) {
          # Interações cruzadas com beta_5, beta_6, beta_7
          hessian[j, k] <- hessian[j, k] +
            2 * (x[i]^(j - 4) * x[i]^(k - 4) *
                   ((common_term * d2) - (num[i] * x[i]^(j + k - 8) / d4)))
        }
      }
    }
  }

  hessian[lower.tri(hessian)] <- t(hessian)[lower.tri(hessian)]

  return(hessian)
}


# Função do método de Newton-Raphson
newton_raphson_thurber <- function(beta_init, data, epsilon = 1e-6, max_iter = 10000, gamma = 1, max_gamma_iter = 10) {
```

```
  x <- data$x
  y <- data$y

  beta <- beta_init

  for (i in 1:max_iter) {
    grad <- gradient_thurber(beta, data)
    hess <- hessian_thurber(beta, data)

    gamma_adjust <- gamma
    success <- FALSE
    for (gamma_iter in 1:max_gamma_iter) {
      hess_regularized <- hess + diag(gamma_adjust, nrow(hess))
      eigenvalues <- eigen(hess_regularized)$values
      if (all(is.finite(eigenvalues)) && min(abs(eigenvalues)) > 1e-10) {
        success <- TRUE
        break  # Hessiana regularizada é aceitável
      }
      gamma_adjust <- gamma_adjust * 10  # Aumenta gamma
    }

    if (!success) {
      cat("Hessiana regularizada ainda singular na iteração", i, "\n")
      return(beta)
    }

    # Calcula o passo de atualização
    delta_beta <- tryCatch({
      solve(hess_regularized, grad)
    }, error = function(e) {
      cat("Erro ao resolver o sistema na iteração", i, "\n")
      return(rep(NA, length(beta)))
    })

    # Verifica se delta_beta é válido
    if (any(is.na(delta_beta))) {
      cat("Delta beta inválido na iteração", i, "\n")
      return(beta)
    }

    # Reduz o tamanho do passo se necessário
    step_size <- sqrt(sum(delta_beta^2))
    if (step_size > 10) {
      delta_beta <- delta_beta / step_size
    }

    beta_new <- beta - delta_beta

    # Critério de parada
    if (sqrt(sum((beta_new - beta)^2)) < epsilon) {
      cat("Convergência atingida na iteração", i, "\n")
      return(beta_new)
    }

    beta <- beta_new
  }

  cat("Máximo de iterações alcançado sem convergência.\n")
  return(beta)
}

resultado1 <- newton_raphson_thurber(beta0_chapeu_thurber_1, Thurber)
resultado1

resultado2 <- newton_raphson_thurber(beta0_chapeu_thurber_2, Thurber)
resultado2


# Função objetivo
objective_function <- function(beta, data) {
  y <- data$y
  x <- data$x
  num <- beta[1] + beta[2] * x + beta[3] * x^2 + beta[4] * x^3
  denom <- 1 + beta[5] * x + beta[6] * x^2 + beta[7] * x^3
  residuals <- y - (num / denom)
  return(sum(residuals^2))
}

# Função para realizar line-search utilizando um grid de valores para
line_search <- function(beta, grad, hess, data, gamma_candidates) {
  direction <- -solve(hess) %*% grad
  best_gamma <- NA
  best_objective <- Inf

  for (gamma in gamma_candidates) {
    beta_new <- beta + gamma * direction
    obj_value <- objective_function(beta_new, data)
    if (obj_value < best_objective) {
      best_objective <- obj_value
      best_gamma <- gamma
    }
  }

  return(best_gamma)
}

# Função de Newton-Raphson com Line Search
newton_raphson_line_search <- function(beta_init, data, epsilon = 1e-6, max_iter = 10000, gamma_candidates = seq(0.01, 0.1, by
      = 0.01)) {
  beta <- beta_init
  objective_values <- c()
  beta_history <- matrix(NA, nrow = max_iter, ncol = length(beta_init))
```

```r

  for (i in 1:max_iter) {
    grad <- gradient_thurber(beta, data)
    hess <- hessian_thurber(beta, data)

    # Regularização robusta da Hessiana
    hess_regularized <- hess + diag(max(abs(hess)) * 0.1, nrow(hess))

    # Verifica se a Hessiana regularizada é invertível
    if (det(hess_regularized) == 0) {
      cat("Hessiana singular na iteração", i, "\n")
      break
    }

    # Line-search adaptativo
    gamma <- line_search(beta, grad, hess_regularized, data, gamma_candidates)
    direction <- -solve(hess_regularized) %*% grad
    beta_new <- beta + gamma * direction

    # Limita o tamanho do passo
    step_size <- sqrt(sum(direction^2))
    if (step_size > 1) {
      beta_new <- beta + (direction / step_size)
    }
    obj_value <- objective_function(beta_new, data)
    objective_values <- c(objective_values, obj_value)
    beta_history[i, ] <- beta_new

    # Critério de parada
    if (sqrt(sum((beta_new - beta)^2)) < epsilon) {
      cat("Convergência atingida na iteração", i, "\n")
      return(list(beta = beta_new, objective_values = objective_values, beta_history = beta_history[1:i, ]))
    }

    beta <- beta_new
  }

  cat("Máximo de iterações alcançado sem convergência.\n")
  return(list(beta = beta, objective_values = objective_values, beta_history = beta_history))
}


resultado1 <- newton_raphson_line_search(beta0_chapeu_thurber_1, Thurber)
resultado1$beta

resultado2 <- newton_raphson_line_search(beta0_chapeu_thurber_2, Thurber)
resultado2$beta



#Gráfico de convergência
plot(resultado1$objective_values, type = "l", col = "blue", xlab = "Iteração",
     ylab = "Função Objetivo", main = "Convergência - Beta Init 1")
lines(resultado2$objective_values, col = "red")
legend("topright", legend = c("Beta Init 1", "Beta Init 2"), col = c("blue", "red"), lty = 1)



# Função para realizar line-search adaptativo
line_search <- function(beta, grad, data, direction, gamma_candidates) {
  best_gamma <- NA
  best_objective <- Inf

  for (gamma in gamma_candidates) {
    beta_new <- beta + gamma * direction
    obj_value <- objective_function(beta_new, data)
    if (obj_value < best_objective) {
      best_objective <- obj_value
      best_gamma <- gamma
    }
  }

  return(best_gamma)
}


# Função de Normalização dos Dados
normalize_data <- function(data) {
  data$x <- scale(data$x)
  data$y <- scale(data$y)
  return(data)
}

# Função Objetivo Regularizada
objective_function_regularized <- function(beta, data, lambda = 1e-3) {
  y <- data$y
  x <- data$x
  num <- beta[1] + beta[2] * x + beta[3] * x^2 + beta[4] * x^3
  denom <- 1 + beta[5] * x + beta[6] * x^2 + beta[7] * x^3
  residuals <- y - (num / denom)
  return(sum(residuals^2) + lambda * sum(beta^2))  # Penalização L2
}

# Busca de Linha com Grid
line_search <- function(beta, grad, data, direction, gamma_candidates, lambda = 1e-3) {
  best_gamma <- NA
  best_objective <- Inf
  for (gamma in gamma_candidates) {
    beta_new <- beta + gamma * direction
    obj_value <- objective_function_regularized(beta_new, data, lambda)
    if (obj_value < best_objective) {
      best_objective <- obj_value
```

```r
      best_gamma <- gamma
    }
  }
  return(best_gamma)
}

# # Função de Gradient Descent com Line-Search adaptativo
gradient_descent_line_search <- function(beta_init, data, gamma_candidates = seq(0.001, 0.2, by = 0.001),
                                         epsilon = 1e-6, max_iter = 10000, lambda = 1e-3) {
  beta <- beta_init
  objective_values <- c()
  beta_history <- matrix(NA, nrow = max_iter, ncol = length(beta_init))

  for (i in 1:max_iter) {
    grad <- gradient_thurber(beta, data)
    direction <- -grad

    gamma <- line_search(beta, grad, data, direction, gamma_candidates, lambda)
    if (is.na(gamma)) {
      cat("Não foi possível encontrar um tamanho de passo adequado na iteração", i, "\n")
      break
    }

    # Atualização dos Parâmetros
    beta_new <- beta + gamma * direction
    obj_value <- objective_function_regularized(beta_new, data, lambda)
    objective_values <- c(objective_values, obj_value)
    beta_history[i, ] <- beta_new

    # Monitoramento do Gradiente
    cat(sprintf("Iteração %d: Norma do Gradiente = %.5f\n", i, sqrt(sum(grad^2))))

    # Critério de Parada
    if (sqrt(sum((beta_new - beta)^2)) < epsilon) {
      cat("Convergência atingida na iteração", i, "\n")
      return(list(beta = beta_new, objective_values = objective_values, beta_history = beta_history[1:i, ]))
    }

    beta <- beta_new
  }

  cat("Máximo de iterações alcançado sem convergência.\n")
  return(list(beta = beta, objective_values = objective_values, beta_history = beta_history))
}

# Normalização dos Dados
Thurber_normalized <- normalize_data(Thurber)

resultado1 <- gradient_descent_line_search(beta0_chapeu_thurber_1, Thurber_normalized)
resultado2 <- gradient_descent_line_search(beta0_chapeu_thurber_2, Thurber_normalized)

# Gráfico de Convergência
plot(resultado1$objective_values, type = "l", col = "blue", xlab = "Iteração",
     ylab = "Função Objetivo Regularizada", main = "Convergência - Beta Init 1 e Beta Init 2")
lines(resultado2$objective_values, col = "red")
legend("topright", legend = c("Beta Init 1", "Beta Init 2"), col = c("blue", "red"), lty = 1)


#Gráfico do movimento da solução
beta_history1 <- resultado1$beta_history
beta_history2 <- resultado2$beta_history

plot(Thurber$x, Thurber$y, pch = 20, col = "gray", main = "Movimento da Solução (Modelo 2)",
     xlab = " 1 ", ylab = " 2 ")
lines(beta_history1[, 1], beta_history1[, 2], type = "b", col = "blue", lty = 1)
points(beta_history2[, 1], beta_history2[, 2], type = "b", col = "red", pch = 19)
legend("topright", legend = c("Beta Init 1", "Beta Init 2"), col = c("blue", "red"), lty = 1)
```

Referências

[1] Greene, W. H. *Econometric Analysis*. Pearson, New York, NY, 2008.

[2] National Institute of Standards and Technology (NIST). "Statistical Reference Datasets for Nonlinear Regression." Disponível em: https://www.itl.nist.gov/div898/strd/.

[3] Cannon, Ann R., George W. Cobb, Bradley A. Hartlaub, Julie M. Legler, Robin H. Lock, Thomas L. Moore, Allan J. Rossman, and Jeffrey Witmer. *Stat2: Building Models for a World of Data*. Worth, 2012.

[4] Devore, Jay L. *Probability and Statistics for Engineering and the Sciences*. Cengage Learning, 2011.

[5] Nelder, John A., and Roger Mead. "A Simplex Method for Function Minimization." *The Computer Journal*, 7(4): 308–13, 1965.

[6] Nocedal, Jorge, and Stephen Wright. *Numerical Optimization*. Springer, 2006.