Universidade Federal do Paraná

Newton Modificado usando Busca Linear Inexata

Aluno: Fillipe Rafael Bianek Pierin

Orientador: Prof. Abel Soares Siqueira

Curitiba

18 de dezembro de 2018

Sumário

1 Introdução

A otimização é uma vertente da matemática para resolução de problemas, onde buscamos encontrar uma opção menos custosa dentre as disponíveis. Essa opção é chamada de minimizadores de uma função objetivo do problema em que se esteja analisando. Problemas de otimização podem ou não possuir restrições.

O problema de minimização irrestrita consiste em

$$\min_{x} f(x), \tag{1}$$

em que $f : \mathbb{R}^n \to \mathbb{R}$, com $f \in C^2$.

Uma das estratégias mais comuns de resolução deste problema são os métodos de busca linear, que consistem em determinar uma direção sobre a qual o valor de função decresce, e escolher um tamanho de passo que determine o quanto andamos nesta direção. O método de Newton consiste em escolher a direção que minimiza uma aproximação quadrática da função. Dentre as maneiras de escolher o tamanho do passo, a busca de Armijo é uma das mais conhecidas.

O objetivo deste trabalho é entendermos o método de Newton, a busca linear inexata de Armijo e os motivos que levam a fazer mudanças no método de Newton para resolver diferentes tipos de problemas, isso problemas irrestritos. Em seguida, fazemos a comparação do método de Newton Modificado com busca de Armijo com outros métodos:

- Newton sem busca vs Newton Modificado com busca;
- Newton Modificado vs BFGS ambos com busca de Armijo;
- Newton Modificado com busca e diferentes estratégias de atualização.

Este trabalho esta organizado da seguinte maneira. No capítulo 2, apresentamos a teoria da decomposição de Cholesky, que será usada em outros capítulos para verificar se a matriz Hessiana é definida positiva. No capítulo 3, explicamos o método de Newton e alguns aspectos teóricos do mesmo. No capítulo 4, apresentamos o método de Newton modificado, onde se modifica a Hessiana $\nabla^2 f(x)$ de forma a obter uma matriz definida positiva. No capítulo 5 fazemos a comparação entre o Método de Newton Modificado e outros métodos de otimização irrestrita.

Para a implementação dos algoritmos e a posterior comparação, através do gráfico Perfil de Desempenho, utilizamos a linguagem de programação Julia versão 0.6.2.

2 Decomposição de Cholesky

A decomposição de Cholesky é uma decomposição de uma matriz definida positiva no produto de uma matriz triangular inferior e sua matriz transposta. Ou seja, podemos decompor A em G e G^T. Essa decomposição é também usada para verificar se uma matriz é definida positiva. Usamos no estudo da decomposição de Cholesky as referências (GOLUB; LOAN, 2012), (TSUMURA, 2017), (RUGGIERO; LOPES, 1997) e (WATKINS, 2004).

Mostramos algumas definições e teoremas importantes que usamos na demonstração da decomposição de Cholesky. Em seguida provamos a decomposição em si.

Definição 1 *A matriz A é simétrica se $A^T = A$.*

Definição 2 *A matriz A é dita simétrica definida positiva, se*

$$x^T A x > 0, \quad \forall \quad x \in \mathbb{R}^n \setminus \{0\}.$$

Teorema 1 *Se a matriz A é simétrica definida positiva então seus autovalores são positivos.*

Demonstração: Sejam λ um autovalor real de A e x seu respectivo autovetor. Isto é, tem-se que

$$Ax = \lambda x.$$

Então, multiplicando por x^T no lado esquerdo, obtém-se

$$\begin{aligned} x^T A x &= \lambda x^T x \\ &= \lambda \|x\|^2 . \end{aligned}$$

O lado esquerdo é positivo, pois a matriz A é definida positiva e x um vetor não nulo (pois, x é um autovetor). Seque que, como $\|x\|^2$ é positivo, deve-se ter que λ é positivo. Logo, todo autovalor λ de A é positivo. $\square$

Definição 3 (Ortogonal) *Uma matriz $Q \in \mathbb{R}^{n\times n}$ é dita ortogonal quando $Q^{-1} = Q^T$.*

Teorema 2 *Se os autovalores de uma matriz A simétrica real são positivos, então A é definida positiva.*

Demonstração: Note que uma matriz simétrica real é diagonalizável por uma matriz ortogonal. Então, existe uma matriz ortogonal Q tal que $Q^T A Q = D$, onde D é uma matriz diagonal onde as entradas da diagonal principal são os autovalores λ_i de A, que por hipótese são positivos.

Seja x um vetor arbitrário não nulo em $\mathbb{R}^n$. Segue que, como $A = QDQ^T$ tem-se que

$$x^T A x = x^T Q D Q^T x.$$

Tomando $y = Q^T x$, escreve-se a equação anterior como

$$x^T A x = y^T D y.$$

então, encontra-se que

$$\begin{aligned} x^T A x &= y^T D y \\ &= \begin{bmatrix} y_1 & y_2 & \dots & y_n \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ \vdots & \dots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \\ &= \lambda_1 y_1^2 + \lambda_2 y_2^2 + \dots + \lambda_n y_n^2 > 0, \end{aligned}$$

logo, como x é um vetor não nulo e Q é invertível, então $y = Q^T x$ é vetor não nulo. Consequentemente, a soma da expressão acima é positiva, o que implica que $x^T A x$ é positivo.

Portanto, a matriz A é definida positiva. $\square$

2.1 Fatoração LDL^T

Primeiramente precisamos provar o Teorema que diz que a matriz A pode ser fatorada de forma única em duas matrizes L e U.

Teorema 3 *Seja A uma matriz quadrada de ordem n, e A_k o menor principal k. Assumimos que $det(A_k) \neq 0$, para $k = 1, 2, \ldots, n-1$. Então, existe uma única matriz triangular inferior L, com $l_{11}, l_{22}, \ldots, l_{nn} = 1$, e uma única matriz triangular superior U, tal que $LU = A$.*

Demonstração: Prova-se usando indução.

1. (caso base): Prova-se que o teorema é verdadeiro para $n = 1$. Seja $A = [a_{11}]$. As únicas matrizes L e U que satisfazem são $L = [1]$ e $U = [a_{11}]$, que implica em $LU = a_{11}$.

2. (passo indutivo): $k-1 \mapsto k$. Prova-se que o teorema é verdadeiro para $n = k-1$, e o teorema será verdadeiro para $n = k$. Sejam A uma matriz, de ordem k, escrita da seguinte forma:
$$A = \begin{pmatrix} A_{k-1} & r \\ s & a_{kk} \end{pmatrix},$$
L_{k-1} e U_{k-1} a decomposição LU de A_{k-1}, e m, p e u_{kk} definidos por
$$\begin{cases} p & = L_{k-1}^{-1} r \\ m & = s U_{k-1}^{-1} \\ u_{kk} & = a_{kk} - mp \end{cases}.$$

   Pela hipótese do teorema, existe a decomposição LU para A_{k-1}. De fato, definindo L e U da seguinte forma: $L = \begin{pmatrix} L_{k-1} & 0 \\ m & 1 \end{pmatrix}; U = \begin{pmatrix} U_{k-1} & p \\ 0 & u_{kk} \end{pmatrix}$, e fazendo LU, obtemos que
$$\begin{aligned} LU &= \begin{pmatrix} L_{k-1} & 0 \\ m & 1 \end{pmatrix} \begin{pmatrix} U_{k-1} & p \\ 0 & u_{kk} \end{pmatrix} \\ &= \begin{pmatrix} L_{k-1}U_{k-1} & L_{k-1}p \\ mU_{k-1} & mp + u_{kk} \end{pmatrix} \\ &= \begin{pmatrix} A_{k-1} & r \\ s & a_{kk} \end{pmatrix} \\ &= A. \end{aligned}$$

Logo, construímos a decomposição L e U da matriz A. E concluímos o passo indutivo.

Portanto, podemos ver que LU é a fatoração de A. □

Agora podemos fatorar LU em $LD\bar{U}$, pois pelo Teorema 3 temos que o determinante dos menores principais é diferente de zero, ou seja, $det(A_k) \neq 0$, em que D é uma matriz diagonal de ordem n e $\bar{u}_{ij} = \dfrac{u_{ij}}{u_{ii}}$.

Considere $A = L D \bar{U}$ a decomposição LDU da matriz A simétrica definida positiva. Então,

$$A = A^T = (LD\bar{U})^T = \bar{U}^T D^T L^T = \bar{U}^T D L^T.$$

Como $\bar{U}^T$ e L^T são matrizes unitárias triangular inferior e superior, respectivamente, obtemos duas decomposições $LD\bar{U}$ de A

$$A = \bar{U}^T D L^T$$

e

$$A = LD\bar{U}.$$

Então, $\bar{U}^T D L^T = LD\bar{U}$. Mostraremos agora que $\bar{U}^T = L$, ou seja, $\bar{u}_{ij} = l_{ji}$.

Teorema 4 *Sejam L_i (U_i) matrizes triangulares inferiores (superiores) com diagonal unitária, $i \in (1,2)$, e D uma matriz diagonal, de tamanho $n \times n$. Então, temos que*

(i) a multiplicação de L_1 e L_2 (U_1 e U_2) é uma matriz triangular inferior (superior) com diagonal unitária;

(ii) a multiplicação de L e D (U e D) é uma matriz triangular inferior (superior);

(iii) a inversa de uma matriz L (U) é uma matriz triangular inferior (superior) com diagonal unitária.

Demonstração: (i) Considere L_1 e L_2 duas matrizes triangulares inferiores com diagonal unitária. Sabe-se que uma matriz é triangular inferior se e somente se $L_{ij} = 0$ para todo $i < j$. Se $W = L_1 L_2$, então

$$\begin{aligned} W_{ij} &= \sum_{k=1}^{n} L_{1_{ik}} L_{2_{kj}}, \text{ por definição} \\ &= \sum_{j \le k \le i} L_{1_{ik}} L_{2_{kj}}, \text{ pois } L_1 \text{ e } L_2 \text{ são triangulares inferiores.} \end{aligned}$$

Logo, se $i < j$, não existe índices k com $j < k < i$ tal que $W_{ij} \neq 0$.
Agora, consideramos o caso em que $i = j$. Logo,

$$\begin{aligned} W_{ii} &= \sum_{k=1}^{n} L_{1_{ik}} L_{2_{ki}}, \text{ por definição} \\ &= L_{1_{ii}} L_{2_{ii}} \\ &= 1. \end{aligned}$$

Portanto, W é uma matriz triangular inferior com diagonal unitária. (ii) Seja D uma matriz diagonal e L uma matriz triangular inferior. Se V = LD, então

$$\begin{aligned} V_{ij} &= \sum_{k=1}^{n} L_{ik} D_{kj}, \text{ por definição} \\ &= \sum_{j \le k \le i} L_{ik} D_{kj}, \text{ pois } L \text{ é triangular inferior e } D \text{ é diagonal} \\ &= L_{ij} D_{jj}. \end{aligned}$$

Logo, se $i < j$ não há índice k com $j < k < i$ tal que $V_{ij} \neq 0$.
Consideramos o caso em que $i = j$. Logo,

$$\begin{aligned} V_{ii} &= \sum_{k=1}^{n} L_{ik} D_{ki}, \text{ por definição} \\ &= L_{ii} D_{ii}. \end{aligned}$$

Portanto, V é uma matriz triangular inferior.

(iii) Assuma que L é uma matriz triangular inferior com diagonal unitária e L^{-1} sua respectiva inversa.

Primeiro mostra-se que L não tem zeros na diagonal principal. Considere por absurdo que $L_{ii} = 0$, e que i é o maior valor com essa propriedade. Consequentemente $Lx = e_i$ não tem solução, pois por substituição temos que $x_n = \cdots = x_{i+1} = 0$ e $L_{ii} x_i = 1$ não possui solução. Logo, conclui-se que L não pode ter zeros na diagonal principal.

Agora, assuma por absurdo que $L_{ij}^{-1} \neq 0$, para algum $i < j$. Tomando o menor valor de i com $L_{ij}^{-1} \neq 0$, com j fixo. Logo, obtemos que

$$0 = I_{ij} = \sum_{k \leq j} L_{kj} L_{ik}^{-1} = L_{ii} L_{ij}^{-1}.$$

Como $L_{ii} \neq 0$, tem-se que $L_{ij}^{-1} \neq 0$.

Considerando $i = j$, temos

$$0 = I_{ii} = L_{ii} L_{ii}^{-1} = 1,$$

pois L é uma matriz triangular inferior com diagonal unitária.
Portanto, L_{ij}^{-1} é uma matriz triangular inferior.
Analogamente, podemos mostrar os itens acima para matrizes triangulares superiores com diagonal unitária. $\square$

Voltando,

$$\bar{U}^T D L^T = L D \bar{U},$$

$$L^{-1} \bar{U}^T D L^T = D \bar{U}$$

e

$$L^{-1} \bar{U}^T D = D \bar{U} \left(L^T\right)^{-1}. \tag{2}$$

Pelo Teorema 4, sabemos que $L^{-1} \bar{U}^T$ é uma matriz triangular inferior, $\bar{U} \left(L^T\right)^{-1}$ é uma matriz triangular superior. Consequentemente, temos também pelo Teorema 4 que $L^{-1} \bar{U}^T D$ e $D \bar{U} \left(L^T\right)^{-1}$ são triangular inferior e superior, respectivamente. Desse modo, para que a igualdade (2) seja verdadeira, ou seja, para que uma matriz triangular inferior seja igual a uma matriz triangular superior, precisamos que as duas matrizes sejam diagonais. Logo, $L^{-1} U^T$ é diagonal unitária, ou seja, a matriz I. Portanto,

$$L^{-1} \bar{U}^T = I,$$

e

$$\bar{U}^T = L.$$

Assim, como A é simétrica concluimos que $A = LD\bar{U} = LDL^T$.

Em resumo:

$$\begin{aligned} A &= LU \text{ (pelo Teorema 3)} \\ &= LD\bar{U} \text{ (pois, } det(A_k) \neq 0) \\ &= LDL^T \text{ (pois, } A \text{ é simétrica).} \end{aligned}$$

2.2 Decomposição GG^T

A partir da decomposição LDU da matriz A obtemos a decomposição de Cholesky.

$$\begin{aligned} A = LDL^T &= L\bar{D}\bar{D}L^T \text{ (pois, } A \text{ é definida positiva e } D \text{ é diagonal)} \\ &= L\bar{D}(L\bar{D})^T \text{ (pois, } D \text{ é diagonal} \Rightarrow D^T = D) \\ &= GG^T. \end{aligned}$$

onde $\bar{d}_{ii} = \sqrt{d_{ii}}$.

Teorema 5 *Seja $A \in \mathrm{R}^{n\times n}$ uma matriz invertível admitindo uma decomposição de Cholesky $A = GG^T$. Então, A é definida positiva se, e somente se, tem decomposição de Cholesky.*

Demonstração:

($\Rightarrow$) A demonstração do fato de A ser definida positiva implica em A ter decomposição de Cholesky veem do Teorema 3 juntamente com a decomposição em da matriz A em GG^T demonstrado anteriormente.

($\Rightarrow$) Considere que a matriz A tem decomposição de Cholesky, isto é, $A = GG^T$, mostraremos que A é definida positiva. Por outro lado, se $x \in \mathbb{R}^n \setminus \{0\}$, então

$$x^T Ax = x^T GG^T x = (G^T x)^T G^T x = \|G^T x\|_2^2.$$

Como a matriz A é assumido como invertível, então a matriz G e também a matriz G^T são invertíveis. Segue disso, que $0 \neq det(A) = det(GG^T) = det(G)^2$. Desde que $x \neq 0$, implica que também $G^T x \neq 0$, e consequentemente $\|G^T x\|_2^2 > 0$, provando que a matriz A é definida positiva. $\square$

Em termos matemáticos, se $a_{ij} \in \mathbb{R}$ e $l_{ij} \in \mathbb{R}$, $i, j \in \{1, 2, \ldots, n\}$, então

$$
\begin{aligned}
A = GG^T &= \begin{bmatrix} g_{11} & 0 & 0 & 0 & 0 \\ g_{21} & g_{22} & 0 & 0 & 0 \\ g_{31} & g_{32} & g_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_{n1} & g_{n2} & g_{n3} & g_{n4} & g_{nn} \end{bmatrix} \begin{bmatrix} g_{11} & g_{21} & g_{31} & \cdots & g_{n1} \\ 0 & g_{22} & g_{32} & \cdots & g_{n2} \\ 0 & 0 & g_{33} & \cdots & g_{n3} \\ 0 & 0 & 0 & \ddots & \vdots \\ 0 & 0 & 0 & 0 & g_{nn} \end{bmatrix} \\
&= \begin{bmatrix} g_{11}^2 & g_{11}g_{21} & g_{11}g_{31} & \cdots & g_{11}g_{n1} \\ g_{21}g_{11} & g_{21}^2 + g_{22}^2 & g_{21}g_{31} + g_{22}g_{32} & \cdots & g_{21}g_{n1} + g_{22}g_{n2} \\ g_{31}g_{11} & g_{31}g_{21} + g_{32}g_{22} & g_{31}^2 + g_{32}^2 + g_{33}^2 & \cdots & g_{31}g_{n1} + g_{32}g_{n2} + g_{33}g_{n3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_{n1}g_{11} & g_{n1}g_{21} + g_{n2}g_{22} & g_{n1}g_{31} + g_{n2}g_{32} + g_{n3}g_{33} & \cdots & g_{n1}^2 + g_{n2}^2 + g_{n3}^2 + \cdots + g_{nn}^2 \end{bmatrix}.
\end{aligned}
$$

Doravente, concluímos que

$$
g_{ij} = \begin{cases} \dfrac{a_{ij} - \sum_{k=1}^{j-1} g_{ik}g_{jk}}{g_{jj}}, \text{se i} \neq \text{j} \\ (a_{ii} - \displaystyle\sum_{k=1}^{j-1} g_{ik}^2)^{\frac{1}{2}}, \text{se i = j} \end{cases}.
$$

2.3 Pseudocódigo da Decomposição de Cholesky

Por último, nesta seção apresentamos o pseudocódigo da decomposição de Cholesky.

Algoritmo 1: Decomposição de Cholesky

Entrada: $A \in \mathbb{R}^{n \times n}$ simétrica e definida positiva

for $k = 1, \ldots, n$ **do**
  soma $= 0$
  for $j = 1, \ldots, (k-1)$ **do**
    soma $=$ soma $+ g_{kj}^2$
  end
  $r = a_{kk}$ - soma
  $g_{kk} = \sqrt{r}$
  for $i = (k+1), \ldots, n$ **do**
    soma $= 0$
    for $j = 1, \ldots, (k-1)$ **do**
      soma $=$ soma $+ g_{ij}g_{kj}$
    end
    $g_{ik} = (a_{ik}$ - soma$)$ / g_{kk}
  end
end

3 Newton

O método de Newton para o caso irrestrito é um método de otimização baseado na linearização da condição de otimalidade. Para o estudo do método e Newton utilizamos as seguintes referências: (ATTUX; ZUBEN, 2017) e (RONCHI, 2017).

Seja $f : \mathbb{R}^n \to \mathbb{R}$, com $f \in C^2$. Este método resolve o problema de forma iterativa, aproximando a função $f(x)$ por uma função quadrática, que é minimizada. Considere a expansão de Taylor de 2º ordem de $f(x)$:

$$f(x^k + d^k) \cong f(x^k) + \underbrace{\nabla f(x^k)^T(x^{k+1} - x^k) + \frac{1}{2}(x^{k+1} - x^k)^T \nabla^2 f(x^k)(x^{k+1} - x^k)}_{Q(d^k)}, \quad (3)$$

onde $\nabla f(x^k)$ é o vetor gradiente, $\nabla^2 f(x^k)$ é a matriz Hessiana de $f(x)$ e $d^k = x^{k+1} - x^k$.

Daí, utilizamos a condição de linearidade de primeira ordem, isto é, minimizamos $f(x^{k+1}) = f(x^k + d^k)$ dado pela equação 3 $\Big(\nabla Q(d^k) = 0\Big)$. Logo, obtemos que

$$\begin{aligned} \nabla Q(d^k) = \nabla f(x^k) + \nabla^2 f(x^k)^T(x^{k+1} - x^k) &= 0 \\ \nabla^2 f(x^k)^T(x^{k+1} - x^k) &= -\nabla f(x^k) \\ x^{k+1} - x^k &= -\left[\nabla^2 f(x^k)\right]^{-1} \nabla f(x^k) \\ x^{k+1} &= x^k - \left[\nabla^2 f(x^k)\right]^{-1} \nabla f(x^k), \end{aligned}$$

então temos o seguinte sistema:

$$\nabla^2 f(x^k) d^k = -\nabla f(x^k), \quad (4)$$

com $x^{k+1} = x^k + d^k$, que resolvemos para encontrar a direção d^k.

Com respeito ao sistema (4), podemos formalizar o método de Newton para minimizar uma função f. Considerando $x^{k+1} = x^k + t_k d^k$, para este método existem três variações:

- Newton "puro" (sem busca), onde fixa-se $t_k = 1, \forall k \in \mathbb{N}$;
- busca exata $t_k = \arg\min_t \{f(x^k + t_k d^k)\}$, no qual t_k será o valor otimal numa otimização unidimensional;
- busca inexata, em que encontramos um valor aproximado para t_k usando a condição de Armijo.

3.1 Pseudocódigo do método de Newton

Agora apresentamos o algoritmo do método de Newton.

Algoritmo 2: MÉTODO DE NEWTON

Entrada: $x_0 \in \mathbb{R}$
Saída: $x^* = x^k$ ótimo

início
  Faça $k \leftarrow 0$
  while $\nabla f(x^k) \neq 0$ **do**
    Defina $d^k \leftarrow -(\nabla^2 f(x^k))^{-1} \nabla f(x^k)$
    Determine o tamanho do passo $t_k > 0$
    Faça $x^{k+1} \leftarrow x^k + t_k d^k$
    $k \leftarrow k + 1$
  end
fim
retorna $x^* = x^k$ *ótimo*

Aqui discutimos as propriedades locais da ordem de convergência do método de Newton. Sabemos que para todo x na vizinhança da solução, x^* ótimo, tal que $\nabla^2 f(x^*)$ é definida positiva, temos que a $\nabla^2 f(x^k)$ também será definida positiva. Nesta vizinhança, o método de Newton está bem definido e converge quadraticamente, desde que os tamanhos dos passos t_k sejam igual a 1, a partir de certa iteração.

Teorema 6 *Seja $f : \mathbb{R}^n \to R$, C^2 e $\nabla^2 f(x^k)$ definida positiva. Então, a direção d^k é de descida.*

Demonstração: Seja d^k resolvida pelo sistema (4) e $\nabla^2 f(x^k)$ definida positiva, ou seja, $x^T \nabla^2 f(x^k) x > 0, \forall x$. Então, temos que

$$\nabla f(x^k)^T d^k = -\nabla f(x^k)^T (\nabla^2 f(x^k))^{-1} \nabla f(x^k) < 0.$$

Portanto, d^k é uma direção de descida. □

Porém, nem sempre se consegue a convergência global, pois a direção d^k do método de Newton pode estar longe da solução. Então, utilizamos um controlador de passos com busca linear inexata, a condição de Armijo.

3.2 Busca Linear Inexata: Condição de Armijo com *backtracking*

Seja x^k e d^k dados, buscamos $t_k \in (0, 1]$ tal que

$$f(x^k + t_k d^k) < f(x^k) + \alpha_k t_k \nabla f(x^k)^T d^k,$$

onde $\alpha \in (0, 1)$ é o parâmetro de Armijo e $\sigma \in (0, 1)$ é o parâmetro de Backtracking.

Na busca através de *backtracking*, encontramos a menor potência $t_k = \sigma^p, p = 0, \ldots, n$ que satisfaça a condição de Armijo, ou seja, testamos $t_k = 1, \sigma, \sigma^2, \sigma^3, \ldots$ até encontrar o primeiro que satisfaça a condição de Armijo.

Teorema 7 *Considere uma função diferenciável $f : \mathbb{R}^n \to R$, um ponto $x^k \in \mathbb{R}^n$, uma direção de descida $d^k \in \mathbb{R}^n$ e $\alpha \in (0, 1)$. Então existe $\beta > 0$ tal que*

$$f(x^k + t_k d^k) < f(x^k) + \alpha_k t_k \nabla f(x^k)^T d^k$$

para todo $t_k \in [0, \beta)$.

Demonstração: Caso $\nabla f(x^k)^T d^k = 0$, segue da definição de descida a desigualdade. Assuma que $\nabla f(x^k)^T d^k < 0$. Como $\alpha < 1$,

$$\lim_{t_k \to 0} \frac{f(x^k + t_k d^k) - f(x^k)}{t_k} = \nabla f(x^k)^T d^k < \alpha \nabla f(x^k)^T d^k$$

Logo, existe $\beta > 0$ tal que

$$\frac{f(x^k + t_k d^k) - f(x^k)}{t_k} < \alpha \nabla f(x^k)^T d^k$$

para todo $t_k \in [0, \beta)$. Assim,

$$f(x^k + t_k d^k) < f(x^k) + \alpha_k t_k \nabla f(x^k)^T d^k$$

□

Em seguida mostramos o algoritmo para implementação da busca linear inexata com *backtracking*, que é usado no método de Newton com busca de Armijo e no Newton modificado.

Algoritmo 3: BUSCA LINEAR INEXATA COM BACKTRACKING

Escolha $\alpha, \sigma \in (0, 1)$
Faça $t := \sigma$
while $f(x^k + \alpha d^k) \geq f(x^k) + \alpha t \nabla f(x^k)^T d^k$ **do**
  $t := t\sigma$
end
retorna $t_k = t$

4 Newton Modificado

A busca inexata para Newton, Armijo com *backtracking*, visto na seção anterior só funciona se a matriz Hessiana $\nabla^2 f(x^k)$ for definida positiva (mostrado no Teorema 6). Quando isso não ocorre, a direção d^k pode ser de subida, para evitar esse problema, sugerimos a aproximação do sistema linear da matriz Hessiana, com B_k da forma $B_k d^k = \nabla^2 f(x^k) + \rho_k I$, de forma que se torne definida positiva. Esse método é chamado de método de Newton modificado. Neste estudo usamos a referência (WRIGHT; NOCEDAL, 1999).

Apresentamos agora o algoritmo do método de Newton modificado que é usado em testes, e comparado com outros métodos nas próximas seções.

Algoritmo 4: MÉTODO DE NEWTON MODIFICADO COM BUSCA LINEAR ARMIJO

Entrada: x_0, $\epsilon > 0$
Saída: $x^* = x^k$ ótimo
1 **início**
2   Faça $k \leftarrow 0$
3   **while** $\left\|\nabla f(x^k)\right\| > \epsilon$ **do**
4   Faça $B_k \leftarrow \nabla^2 f(x^k) + \rho_k I$, com $\rho_k > 0$, de forma que B_k seja definida positiva.
5   Fatore B_k e resolva $B_k d^k = -\nabla f(x^k)$.
6   Faça $x^{k+1} \leftarrow x^k + t_k d^k$, sendo t_k calculado usando backtracking com Armijo.
7   **end**
8   Faça $k \leftarrow k + 1$ e volte a linha 3.
9 **fim**
10 **retorna** $x^* = x^k$ *ótimo*

Existem várias maneiras de se encontrar ρ_k, apresentamos algumas delas a seguir.

Primeira estratégia.

Algoritmo 5: ESTRATÉGIA 1

Faça $\rho_k = 0$
while B_k *não é definida positiva* **do**
    if $\rho_k = 0$ **then**
        $\rho_{k+1} = \rho_{min}$, $\rho_{min} > 0$
    else
        $\rho_{k+1} = 2\rho_k$
    end
end

Segunda estratégia.

Algoritmo 6: ESTRATÉGIA 2

Faça $\rho_k = 0$
if $\lambda_{min} > 0$ **then**
    $B = A$
end
if $\lambda_{min} \leq 0$ **then**
    $B = A + (\epsilon - \lambda_{min})I$, $\epsilon > 0$
end

Terceira estratégia.

Algoritmo 7: ESTRATÉGIA 3

Faça $\rho_{k+1} = \dfrac{\rho_k}{7}$
while B_k *não é definida positiva* **do**
  if $\rho_k = 0$ **then**
    $\rho_{k+1} = \rho_{min}$, $\rho_{min} > 0$
  else
    $\rho_{k+1} = 2\rho_k$
  end
end

No algoritmo implementado do método de Newton modificado usa-se $\rho_{min} = 0.1$ e $\epsilon = 0.1$.

Uma maneira para verificar se a matriz B_k é definida positiva, é usando a decomposição de Cholesky, pois esta só existe se a matriz é definida positiva.

5 Comparação entre Algoritmos

Para comparações entre os métodos usamos o gráfico Perfil de Desempenho, com relação ao tempo computacional e a quantidade de avaliações de funções, e empregamos os problemas do CUTEst, usando problemas com até 100 variáveis. Para entender o gráfico Perfil de Desempenho utilizamos das seguintes referências (RIBEIRO; KARAS, 2013) e (BIRGIN; CASTILLO; MARTÍNEZ, 2005). Comparamos o método de Newton modificado com os métodos: Newton "puro" e BFGS com busca de Armijo. Também comparamos o método de Newton modificado entre as três diferentes estratégias de atualização utilizadas.

Nas Figuras (1) e (2) analisamos a comparação entre as três estratégias propostas. A terceira estratégia apresentou resultado melhor em termos de avaliação de funções, porém a segunda estratégia se aproxima. Já em relação ao tempo computacional o desempenho se mantém. A segunda estratégia que usamos que precisa do cálculo do menor autovalor da matriz Hessiana se apresenta resultado pior comparado com a estratégia 1 e o BFGS, porque para calcular autovalores precisa de maior tempo computacional e espaço de memória.

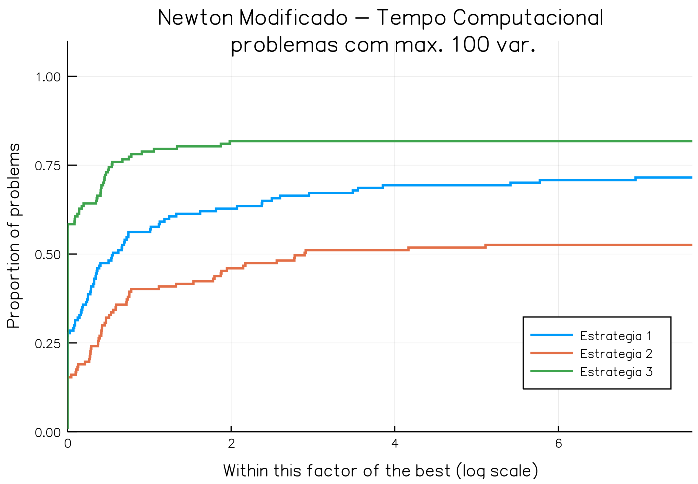

Figura 1 – Relação ao tempo computacional.

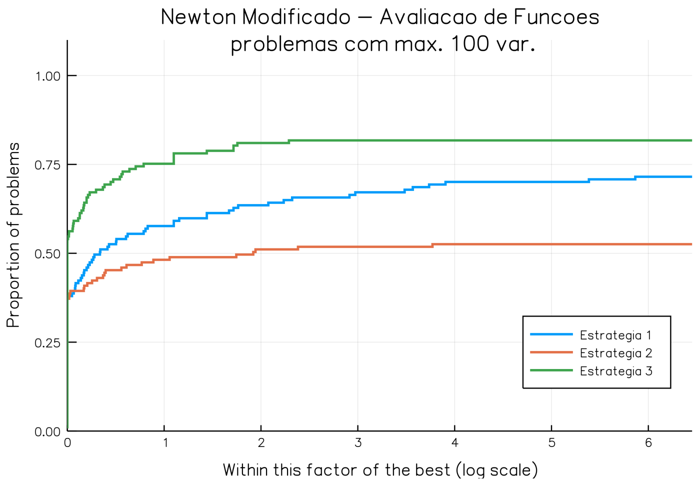

Figura 2 – Relação ao número de avaliações de funções.

Em comparação com o método de Newton “Puro”, as três estratégias apresentam um desempenho melhor tanto com relação a robustez quanto a eficiência (ver nas Figuras (3) e (4)). Quando vamos aumentando o número de problemas com mais variáveis, até 100, o Newton “Puro” apresenta piora no desempenho.

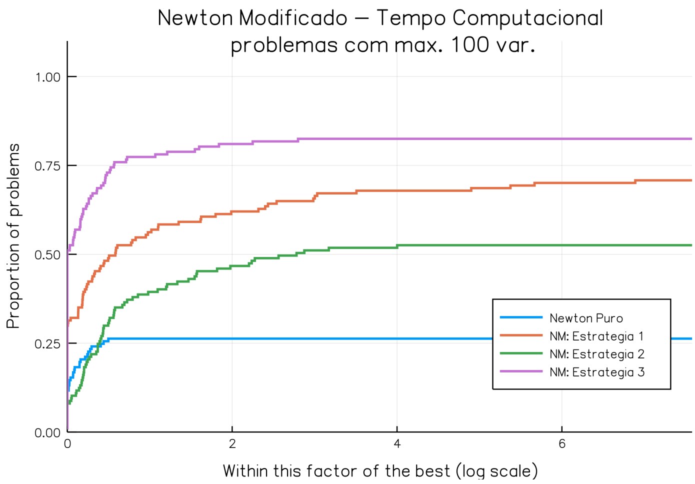

Figura 3 – Relação ao tempo computacional.

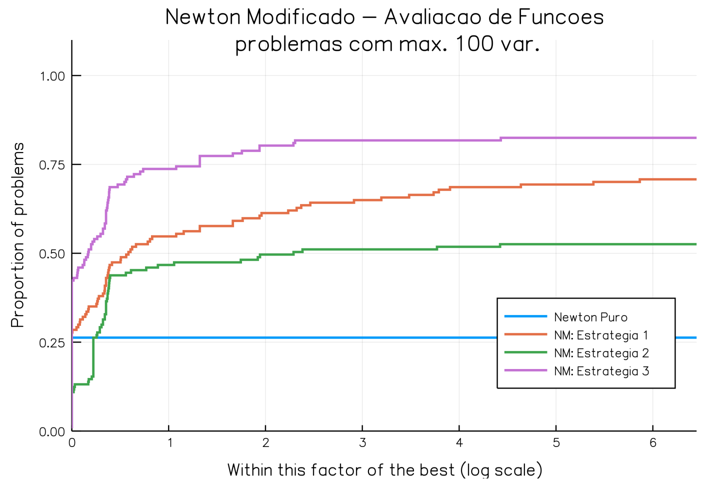

Figura 4 – Relação ao número de avaliações de funções.

Analisando os métodos junto com o método BFGS, verificamos que o BFGS se apresentou melhor com relação a robustez. Apesar de ser melhor em relação a avaliação de funções e pior ao tempo computacional. Isso vemos nas Figuras (5) e (6).

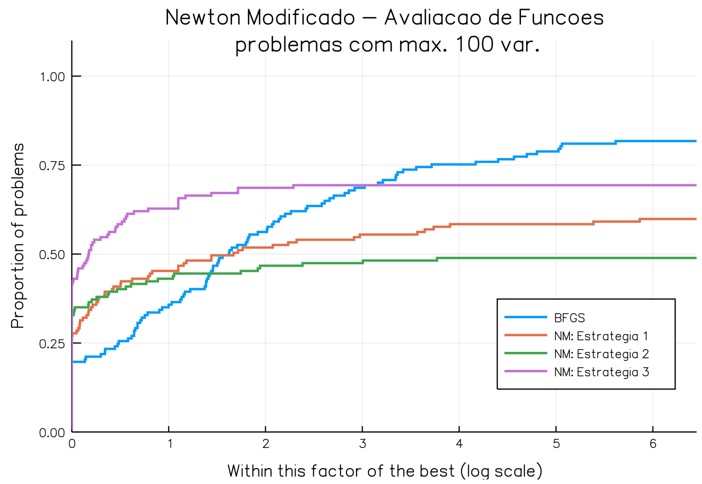

Figura 5 – Relação ao tempo computacional.

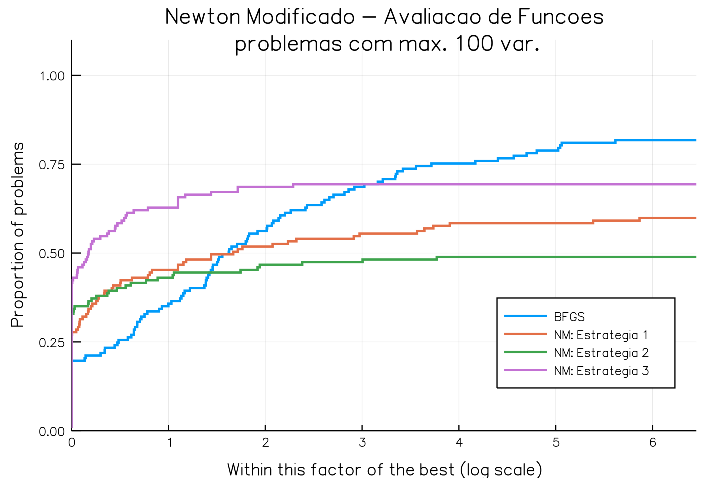

Figura 6 – Relação ao número de avaliações de funções.

Apesar dos métodos de BFGS e Newton modificado com as três estratégias sejam um dos métodos que mais demoram em resolver os problemas, e terem menos avaliações de funções, entre os métodos comparados, estes métodos resolvem mais problemas na prática.

6 Conclusão

O trabalho propôs a implementação do método de Newton modificado, entender o método de Newton com as possíveis mudanças com Armijo e o modificação na Hessiana. Com as análises feitas com o programa Julia na Seção 5, concluímos que o método de Newton modificado, ficou em segundo lugar entre os métodos comparados usando a terceira estratégia, analisando a robustez e a eficiência. Por causa do cálculo do menor autovalor da matriz Hessiana, a segunda estratégia no método de Newton modificado não apresentou melhor resultado entre as estratégias.

Ainda obtemos que os métodos BFGS e Newton modificado resolvem mais problemas na prática, mas demandam de maior tempo computacional para resolvê-los. O método de Newton modificado apresentou melhor desempenho que o Newton “Puro”, porque resolve mais problemas apesar de usar maior tempo computacional e avaliação de funções.

Assim, caso se queira maior resolução de problemas aconselhamos optar pelo método BFGS, e quando for necessário resolver alguns problemas em tempo mais rápido parece mais adequado usar o método de Newton Modificado. Apesar disso, podemos ajustar o BFGS para ficar mais rápido, sendo mais eficiente. Mas isso depende dos problemas que se desejam resolver e do objetivo.

Para os próximos passos, consideramos fazer a melhoria nos parâmetros de Armijo α, de Backtracking σ, e também a procura de outras estratégias para o ρ_k.

Referências

ATTUX, R.; ZUBEN, F. V. *Métodos de otimização paramétrica não-linear irrestrita*. 2017. Disponível em: <ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/ia353_1s07/topico6_07comp2.pdf>. Citado na página 7.

BIRGIN, E. G.; CASTILLO, R.; MARTÍNEZ, J. M. Numerical comparison of augmented lagrangian algorithms for nonconvex problems. *Computational Optimization and Applications*, Springer, v. 31, n. 1, p. 31–55, 2005. Citado na página 12.

FIEDLER, M. *Special matrices and their applications in numerical mathematics*. [S.l.]: Courier Corporation, 2008. Nenhuma citação no texto.

GOLUB, G. H.; LOAN, C. F. V. *Matrix computations*. [S.l.]: JHU Press, 2012. v. 3. Citado na página 1.

RIBEIRO, A. A.; KARAS, E. W. Otimização contínua: aspectos teóricos e computacionais. *Sao Paulo: Cengage Learning*, 2013. Citado na página 12.

RONCHI, C. H. V. *Estudo matemático do reconhecimento de carecteres*. 43 f. Monografia — Universidade Federal do Paraná, Curitiba, 2017. Citado na página 7.

RUGGIERO, M. A. G.; LOPES, V. L. d. R. *Cálculo numérico: aspectos teóricos e computacionais*. [S.l.]: Makron Books do Brasil, 1997. Citado na página 1.

TSUMURA, Y. *Positive definite Real Symmetric Matrix and its Eigenvalues*. 2017. Disponível em: <https://yutsumura.com/positive-definite-real-symmetric-matrix-and-its-eigenvalues/>. Citado na página 1.

WATKINS, D. S. *Fundamentals of matrix computations*. John Wiley & Sons, 2004. v. 64. Disponível em: <https://davidtabora.files.wordpress.com/2015/01/david_s-_watkins_fundamentals_of_matrix_computat.pdf>. Citado na página 1.

WRIGHT, S.; NOCEDAL, J. Numerical optimization. *Springer Science*, v. 35, n. 67-68, p. 7, 1999. Citado na página 10.

Anexo

Códigos das implementações usadas no Método de Newton Modificado.

Decomposição de Cholesky $A = GG^T$

```
function decomp_cholesky(A::Matrix; tol = 1e-6)
    n = size(A)[1]
    G = zeros(size(A))
    for k in 1:n
        soma = 0
        for j in 1:k-1
            soma += G[k, j]^2
        end
        if ((A[k,k] - soma) < tol)
            return G, :falha
        else
            g = A[k,k] - soma
        end
        G[k, k] = sqrt(g)
        for i in (k + 1):n
            soma = 0
            for j in 1:(k - 1)
                soma += G[i, j] * G[k, j]
            end
            G[i, k] = (A[i,k] - soma) / G[k, k]
        end
    end
    return G, :sucesso
end
```

Método de Newton Puro

```
function newton_puro(nlp;
                     tol_abs = 1e-8,
                     tol_rel = 1e-6,
                     max_time = 60.0,
                     max_evals = 10000)
    f(x) = obj(nlp, x)
    g(x) = grad(nlp, x)
    H(x) = Symmetric(hess(nlp, x), :L)
    x = copy(nlp.meta.x0)

    tempo_0 = time()
```

```
    Δt = time() - tempo_0

    fx = f(x)
    gx = g(x)

    normgx = norm(gx)
    ϵ = tol_abs + normgx * tol_rel

    exitflag = :desconhecido
    sucesso = normgx < ϵ
    cansado = Δt > max_time || sum_counters(nlp) > max_evals

    while !(sucesso || cansado)
        local G
        try
            G = chol(H(x))
        catch ex
            if isa(ex, LinAlg.PosDefException)
                exitflag = :nao_eh_pos_def
            else
                # Erro desconhecido
                exitflag = :excecao
            end
            break
        end
        d = -( G \ (G' \ gx) )
        x = x + d
        gx = g(x)
        normgx = norm(g(x))

        Δt = time() - tempo_0
        sucesso = normgx < ϵ
        cansado = Δt > max_time || sum_counters(nlp) > max_evals
    end
    if sucesso
        exitflag = :sucesso
    elseif cansado
        if Δt > max_time
            exitflag = :max_time
        else
            exitflag = :max_evals
        end
    end

    return x, f(x), normgx, Δt, sum_counters(nlp), exitflag
end
```

Método BFGS com Busca Linear Inexata

```
function bfgs_busca_linear(nlp;
                           tol_abs = 1e-8,
                           tol_rel = 1e-6,
```

```
                        armijo_param = 0.5,
                        backtrack_param = 0.5,
                        max_time = 60.0,
                        max_evals = 10000)
    f(x) = obj(nlp, x)
    g(x) = grad(nlp, x)

    x = copy(nlp.meta.x0)
    n = length(x)

    H = eye(n)

    tempo_0 = time()
    Δt = time() - tempo_0

    fx = f(x)
    gx = g(x)

    normgx = norm(gx)
    ϵ = tol_abs + normgx * tol_rel

    exitflag = :desconhecido
    sucesso = normgx < ϵ
    cansado = Δt > max_time || sum_counters(nlp) > max_evals

    while !(sucesso || cansado)
        d = -H * gx
        t = 1.0
        xt = x + d
        ft = f(xt)
        prodint = dot(d, gx)
        # Devemos ter d^T  f  (x) < 0
        if prodint ≥ 0
            exitflag = :direcao_nao_descida
            break
        end
        while !(ft < fx + armijo_param * t * prodint)
            t = t * backtrack_param
            xt = x + t * d
            ft = f(xt)
            if t < 1e-20
                exitflag = :passo_muito_pequeno
                break
            end
        end
        if exitflag != :desconhecido
            break
        end

        x .= xt
        fx = ft
        gt = g(x)

        y = gt - gx
        y^Ts = t * dot(y, d)
        if y^Ts > 0
            ρ = 1 / y^Ts
            M = eye(n) - t * ρ * d * y'
            H = M * H * M' + (t^2 * ρ) * d * d'
        end
```

```
        gx .= gt
        normgx = norm(g(x))

        Δt = time() - tempo_0
        sucesso = normgx < ϵ
        cansado = Δt > max_time || sum_counters(nlp) > max_evals
    end

    if sucesso
        exitflag = :sucesso
    elseif cansado
        if Δt > max_time
            exitflag = :max_time
        else
            exitflag = :max_evals
        end
    end

    return x, f(x), normgx, Δt, sum_counters(nlp), exitflag
end
```

Método de Newton Modificado

```
function newton_modificado(nlp;
                           strategy = 1,
                           tol_abs = 1e-8,
                           tol_rel = 1e-6,
                           max_time = 60.0,
                           max_evals = 10000,
                           armijo_param = 0.1,
                           backtrack_param = 0.5,
                           ρmin = 0.1)
    f(x) = obj(nlp, x)
    g(x) = grad(nlp, x)
    H(x) = Symmetric(hess(nlp, x), :L)

    x = copy(nlp.meta.x0)
    n = nlp.meta.nvar

    tempo_0 = time()
    Δt = time() - tempo_0

    fx = f(x)
    gx = g(x)

    normgx = norm(gx)
    ϵ = tol_abs + normgx * tol_rel

    exitflag = :desconhecido
    sucesso = normgx < ϵ
    cansado = Δt > max_time || sum_counters(nlp) > max_evals
```

```
    ρ = 0
    while !(sucesso || cansado)
        Hx = H(x)
        if strategy == 1
            ρ = 0
            Bk = Hx + ρ * eye(n)
            eh_pos_def = false
            while eh_pos_def == false
                if ρ == 0
                    ρ = ρmin
                elseif ρ > 0
                    ρ = 2 * ρ
                end
                Bk = Hx + ρ * eye(n)
                G, status = decomp_cholesky(Bk)
                if status == :sucesso
                    eh_pos_def = true
                else
                    eh_pos_def = false
                end
            end
        elseif strategy == 2
            eh_pos_def = false
            k = 0
            mu_menor = eigmin(full(Hx))
            if mu_menor > 0
                ρ = 0.0
            else
                eps = 0.1
                ρ = eps - mu_menor
            end

            if ρ > 100
                exitflag = :ρ_grande_Hessiana_mal_condicionada
            end

            Bk = Hx + ρ * eye(n)
            G, status = decomp_cholesky(Bk)
            if status == :sucesso
                eh_pos_def = true
            else
                eh_pos_def = false
            end
        elseif strategy == 3
            ρ = ρ / 7
            Bk = Hx + ρ * eye(n)
            eh_pos_def = false
            while eh_pos_def == false
                if ρ == 0
                    ρ = ρmin
                elseif ρ > 0
                    ρ = 2 * ρ
                end

                Bk = Hx + ρ * eye(n)
                G, status = decomp_cholesky(Bk)
                if status == :sucesso
                    eh_pos_def = true
                else
```

```
                    eh_pos_def = false
                end
            end
        end

        d = - ( G' \ (G \ gx) )
        t = 1.0
        xt = x + d
        ft = f(xt)
        gx = g(x)
        normgx = norm(gx)
        prodint = dot(d,gx)

        # Devemos ter d^T f (x) < 0
        if prodint ≥ 0
            exitflag = :direcao_nao_descida
            break
        end
        while !(ft < fx + armijo_param * t * prodint)
            t = t * backtrack_param
            xt = x + t * d
            ft = f(xt)
            if t < 1e-20
                exitflag = :passo_muito_pequeno
                break
            end
        end
        if exitflag != :desconhecido
            break
        end

        x = xt
        fx = ft

        gx = g(x)
        normgx = norm(gx)

        Δt = time() - tempo_0
        sucesso = normgx < ϵ
        cansado = Δt > max_time || sum_counters(nlp) > max_evals
    end

    if sucesso
        exitflag = :sucesso
    elseif cansado
        if Δt > max_time
            exitflag = :max_time
        else
            exitflag = :max_evals
        end
    end

    return x, f(x), normgx, Δt, sum_counters(nlp), exitflag
end
```