Tópicos em programação não linear e álgebra linear computacional

Sérgio de A. Cipriano Júnior
Prof. Dr. John Lenon C. Gardenghi
Faculdade do Gama · Universidade de Brasília

Resumo

Este trabalho aborda o método de gradientes conjugados para solução de um subproblema do método de Newton para sistemas não lineares irrestritos. Nosso objetivo é conceber um estudo introdutório a área de métodos computacionais de otimização com o algoritmo de gradientes conjugados, visto que esse integra a classe de algoritmos clássicos para otimização e permanece relevante. Assim, está descrito um estudo detalhado desse método, partindo de uma apresentação à um problema de otimização, uma introdução aos métodos de direções conjugadas, alguns conceitos pré-requisitos para a plena compreensão dos gradientes conjugados, e sua integração com o método de Newton. Além disso, experimentos numéricos implementados em Julia, permitiram a validação do algoritmo estudado e comparações com o método de Newton-Cholesky.

Palavras-chave gradientes conjugados, minimização irrestrita, programação não linear, álgebra linear computacional.

1 Introdução

Neste trabalho, iremos abordar alguns métodos de otimização tendo como objetivo um primeiro contato com a área. Otimização é um importante campo da ciência para tomada de decisões, análise e solução de sistemas.

Na otimização, dado um objetivo, isto é, uma medida quantitativa quanto ao desempenho de um sistema e um conjunto de variáveis que configuram as particularidades do sistema, nossa meta é encontrar valores que otimizem o objetivo.

Matematicamente falando, otimização é a minimização ou maximização de uma determinada função tendo escolhido valores de variáveis dentro de um conjunto viável. Um problema de otimização pode ser descrito como:

$$\begin{aligned} &\text{Minimizar } f(x) \\ &\quad \text{s. a } c_i(x) = 0,\ i \in E, \\ &\qquad c_i(x) \leq 0,\ i \in I, \end{aligned} \tag{1}$$

onde:

- $f : \mathbb{R}^n \to \mathbb{R}$ é a função objetivo;

- x é o vetor de variáveis;
- c_i são funções que definem condições as quais x deve satisfazer;
- E e I são os conjuntos de índices i das restrições de igualdade e desigualdade, respectivamente.

Trabalharemos com problemas de otimização sem restrições, ou seja, problemas nos quais $E = I = \emptyset$ e a função objetivo é suave e duas vezes continuamente diferenciável em $\mathbb{R}^n$.

Assim, vamos descrever o funcionamento dos métodos abordando os conteúdos necessários para compreendê-los e implementá-los.

2 Forma Quadrática

A forma quadrática é uma função quadrática que assume a seguinte forma vetorizada:

$$f(x) = \frac{1}{2}x^T Ax - b^T x + c, \tag{2}$$

onde $x \in \mathbb{R}^n$ é o vetor de incógnitas, $b \in \mathbb{R}^n$ é um vetor conhecido e $A \in \mathbb{R}^{n\times n}$ é uma matriz conhecida, quadrática, simétrica e positiva definida.

Suponha que queiramos minimizar $f(x)$. Como $f(x)$ é uma função convexa, conforme demonstrado por Jorge Nocedal e Stephen J. Wright [2], seu minimizador global é o ponto que anula o gradiente. Para calcular o gradiente de (2), consideremos $n = 2$. Com isso, temos:

$$\begin{aligned}
f(x) &= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} b_1 & b_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + c \\
f'(x) &= \begin{bmatrix} \frac{\partial}{\partial x_1} f(x) \\ \\ \frac{\partial}{\partial x_2} f(x) \end{bmatrix} = \begin{bmatrix} A_{11}x_1 + \frac{1}{2}A_{21}x_2 + \frac{1}{2}A_{12}x_2 - b_1 \\ \\ \frac{1}{2}A_{21}x_1 + \frac{1}{2}A_{12}x_1 + A_{22}x_2 - b_2 \end{bmatrix} \\
f'(x) &= \frac{1}{2}Ax + \frac{1}{2}A^T x - b.
\end{aligned}$$

Considerando que A é uma matriz simétrica e positiva definida, temos que $f'(x) = Ax - b$. Por conseguinte, x^* minimizador de $f(x)$ é o ponto tal que $f'(x^*) = 0$, ou seja, $Ax^* - b = 0$. Logo, minimizar $f(x)$ equivale a resolver o sistema linear:

$$Ax = b. \tag{3}$$

3 Método de Máxima Descida

O método de máxima descida, tal como os demais métodos que serão abordados nesse relatório, funcionam com base em dois pilares principais: direção e passo, que é o quanto percorrer na direção definida. Nesses tipos de métodos tomamos um ponto arbitrário x_0 e executamos k iterações calculando x_1, x_2, ... x_k até que encontremos um $x^* = x_k$ satisfatório.

Antes de prosseguirmos, vamos estabelecer alguns conceitos importantes. O resíduo

$$r_i = -f'(x) = b - Ax_i \tag{4}$$

indica o quão próximos estamos de satisfazer (3), além de ser a direção oposta do gradiente ou direção de máxima descida. O erro

$$e_i = x_i - x^* \tag{5}$$

indica o quão próximos estamos da solução. Substituindo (5) em (4), temos que:

$$\begin{aligned} r_i &= b - Ax_i & (6)\\ r_i &= b - Ax - Ae_i \\ r_i &= -Ae_i. & (7) \end{aligned}$$

A partir da equação (7) é possível ver que o resíduo é também uma combinação linear do erro pela matriz A.

O método de máxima descida consiste em gerar iterandos $x_1, x_2, \ldots$ partindo de um ponto inicial x_0 de forma que

$$x_{i+1} = x_i + \alpha_i r_i, \tag{8}$$

sendo α_i o tamanho do passo que daremos na direção r_i. O procedimento de calcular α_i é chamado de *busca linear*. Há diversas formas de fazer uma busca linear. Uma das formas é determinar um α que minimize $f(x_i + \alpha r_i)$ (2) ao longo da direção r_i. Para isso, tomamos o ponto no qual a derivada direcional da função quadrática $\frac{\partial}{\partial \alpha} f(x_{i+1})$ seja zero.

$$\begin{aligned} \frac{\partial}{\partial \alpha} f(x_{i+1}) &= 0 \\ f'(x_{i+1})^T r_i &= 0 \\ r_i^T r_i &= 0 & (9)\\ (b - Ax_{i+1})^T r_i &= 0 \\ (b - A(x_i + \alpha_i r_i))^T r_i &= 0 \\ r_i^T r_i - \alpha_i r_i^T A r_i &= 0 \\ \alpha_i &= \frac{r_i^T r_i}{r_i^T A r_i}. & (10) \end{aligned}$$

Temos então todas as equações que compõem o método de máxima descida. No entanto, ainda há como aprimorá-lo, sendo que é possível diminuir o número de produtos matriz-vetor que são feitos. Pré-multiplicando $x_{i+1} = x_i + \alpha_i r_i$ por $-A$ e somando b aos dois lados da igualdade, vem que

$$b + (-A)x_{i+1} = b + (-A)x_i + (-A)\alpha_i r_i$$

donde segue que

$$r_{i+1} = r_i - \alpha_i A r_i. \tag{11}$$

Com a equação (11) agora o produto Ar_i se repete duas vezes, precisando ser computado apenas uma vez. Entretanto, ainda é preciso utilizar a equação (6), já que é necessário calcular r_0.

Por fim juntamos todas as peças para montar o Algoritmo 1.

Algoritmo 1: Método de Máxima Descida

Entrada: $A \in \mathbb{R}^{n\times n}$, $x_0 \in \mathbb{R}^n$, $b \in \mathbb{R}^n$, i_{max} e uma tolerância e
Saída: $x \in \mathbb{R}^n$

$r_0 = b - Ax_0$
para $i = 0, 1, \ldots, i_{max}$ **faça**
  $\alpha_i = \frac{r_i^T r_i}{r_i^T A r_i}$
  $x_{i+1} = x_i + \alpha_i r_i$
  $r_{i+1} = r_i - \alpha_i A r_i$
  se $||r_{i+1}|| < e$ **então**
    return x_{i+1}
  fim se
  $i = i + 1$
fim para
return x_{i-1}

4 Método de Direções Conjugadas

No algoritmo de máxima descida é comum uma mesma direção ser tomada repetidas vezes. Com o método de direções conjugadas, tomamos um passo com tamanho suficiente para percorrer determinada direção apenas uma vez. Dessa maneira, um problema *n-dimensional* é resolvido em, no máximo, n passos.

Mais para frente demonstraremos que o resultado será obtido em n passos, no pior caso. Antes disso, a cada iteração queremos definir

$$x_{i+1} = x_i + \alpha_i d_i \tag{12}$$

tal que queremos um valor de α_i que esgote a direção d_i, ou seja, desejamos um passo ótimo para percorrê-la.

Para isso, tomaremos um conjunto de direções $d_0, d_1, \ldots, d_{n-1}$ A-ortogonais entre si. Diz-se que os vetores de um conjunto $\{v_k\}_{k=0}^{n-1}$ são A-ortogonais ou conjugados se:

$$v_i^T A v_j = 0, \forall i \neq j. \tag{13}$$

O fato das direções serem conjugadas também implica que as direções são conjugadas aos erros de índices posteriores, em particular, temos que uma direção d_i é A-ortogonal a e_{i+1}. Antes de mostrarmos a importância desse fato, vamos apresentar um teorema mostrando que d_i é conjugado a e_{i+1} e a todos os demais erros de índices posteriores a i.

Teorema 1. *Seja x a solução do sistema linear (3) e consideremos o método iterativa com iteração definida por (12) que converge à solução x em n iterações utilizando um conjunto de n direções $d_0, d_1, \ldots, d_{n-1}$ A-ortogonais entre si. Então, o erro e_i (5) é A-ortogonal a qualquer direção d_ℓ, para todo $\ell < i$.*

Demonstração. Como o método itera por (12), então

$$x = x_i + \sum_{j=i}^{n-1} \alpha_j d_j,$$

para qualquer i tal que $0 \leq i < n$. Portanto,

$$x - x_i = \sum_{j=i}^{n-1} \alpha_j d_j,$$

que, por (5), implica que

$$e_i = -\sum_{j=i}^{n-1} \alpha_j d_j. \tag{14}$$

Como o conjunto de direções $\{d_0, d_1, \ldots, d_{n-1}\}$ é um conjunto A-ortogonal, pré-multiplicando (14) por $d_\ell^T A$, para algum $\ell < i$, temos que

$$\begin{aligned} d_\ell^T A e_i &= -d_\ell^T A \sum_{j=i}^{n-1} \alpha_j d_j \\ &= 0, \end{aligned}$$

como queríamos demonstrar. □

Com isso, segue o seguinte corolário.

Corolário 1. *Para todo i tal que $0 \leq i < n-1$, e_{i+1} é A-ortogonal a d_i.*

Diante disso, para calcularmos o passo α_i ótimo, usaremos como base a conjugação estabelecida no Corolário 1:

$$d_i^T A e_{i+1} = 0. \tag{15}$$

De (15) e (12), concluímos que:

$$\begin{aligned} d_i^T A(e_i + \alpha_i d_i) &= 0 \\ \alpha_i &= -\frac{d_i^T A e_i}{d_i^T d_i}. \end{aligned} \tag{16}$$

Utilizando a equação (7), que relaciona o resíduo ao erro, podemos reescrever (16):

$$\alpha_i = -\frac{d_i^T r_i}{d_i^T d_i}. \tag{17}$$

O fator α_i assim calculado equivale a encontrar o minimizador da quadrática $f(x)$ definida em (2) ao longo da direção d_i, ou seja, essa relação iguala-se a fazer uma busca linear exata ao longo da direção d_i. Com a derivada direcional de (2) avaliada em (8) valendo zero, temos que:

$$\begin{aligned} \frac{\partial}{\partial\alpha} f(x_{i+1}) &= 0 \\ f'(x_{i+1})^T \frac{\partial}{\partial\alpha}(x_{i+1}) &= 0 \\ -r_{i+1}^T d_i &= 0 \\ d_i^T A e_{i+1} &= 0. \end{aligned}$$

Por fim, para provar que esse processo computa x em n passos, vamos primeiro demonstrar que um conjunto de direções conjugadas é linearmente independente.

Lema 1. *Para toda matriz $A \in \mathbb{R}^{n\times n}$ simétrica e positiva definida existe um conjunto $\{d_i\}_{i=0}^{n-1}$ de direções $d_i \in \mathbb{R}^n$ A-ortogonais ou conjugadas entre si que são linearmente independentes.*

Demonstração. A prova será feita por absurdo. Assuma que podemos representar d_0 como uma combinação linear dos vetores do conjunto $\{d_i\}_{i=1}^{n-1}$. Logo:

$$d_0 = \sum_{i=1}^{n-1} \lambda_i d_i \tag{18}$$

com $\lambda_i \in \mathbb{R}, i = 0, 1, ..., n-1$. Sabendo que a direção d_0 é não nula e conjugada as demais direções, podemos pré-multiplicar (18) por $d_0^T A$. Então:

$$0 < d_0^T A d_0 = \sum_{i=1}^{n-1} \lambda_i d_0^T A d_i = 0 \tag{19}$$

o que nos da uma contradição. □

Isto posto, vamos provar que o método de direções conjugadas computa x em n iterações.

Teorema 2. *Para qualquer ponto inicial $x_0 \in \mathbb{R}^n$, selecionando qualquer conjunto $\{d_i\}_{i=0}^{n-1}$ de direções $d_i \in \mathbb{R}^n$ conjugadas entre si, todo método de direções conjugadas converge para x^*, solução do problema (3) em , no máximo, n passos.*

Demonstração. Representando o erro como uma combinação linear das direções conjugadas, temos:

$$e_0 = x_0 - x^* = \sum_{j=0}^{n-1} \delta_j d_j, \tag{20}$$

com $\delta_j \in \mathbb{R}, j = 0, 1, 2, \ldots, n-1$.

Relacionando (20) com (14) é notório que $\alpha_j = -\delta_j$, $\forall \jmath = 0, 1, \ldots, n-1$. Pré-multiplicando a equação (20) por $d_k^T A$ e sabendo que d_k é A-ortogonal com $d_j, \forall j \neq k$:

$$\begin{aligned}
d_k^T A e_0 &= \delta_k d_k^T A d_k \\
\delta_k &= \frac{d_k^T A e_0}{d_k^T A d_k} \\
&= \frac{d_k^T A(e_0 + \sum_{j=0}^{k-1} \alpha_j d_j)}{d_k^T A d_k}, \quad \text{pela conjugação dos vetores } d \\
&= \frac{d_k^T A e_k}{d_k^T A d_k} \\
&= -\frac{d_k^T r_k}{d_k^T A d_k}, \quad \text{pela Equação (7)} \\
&= -\alpha_k, \quad \text{pela Equação (17) .}
\end{aligned} \tag{21}$$

Temos então a equação (21) com o resultado esperado. Essa relação pode ser interpretada como o processo de eliminar o i-ésimo termo do erro a cada iteração.

$$\begin{aligned}
e_i &= e_0 + \sum_{j=0}^{i-1} \alpha_j d_j \\
&= \sum_{j=0}^{n-1} \delta_j d_j - \sum_{j=0}^{i-1} \delta_j d_j \\
&= \sum_{j=i}^{n-1} \delta_j d_j.
\end{aligned} \tag{22}$$

Após n iterações, todos os componentes são eliminados e $e_n = 0$. □

Enfim, para que possamos resolver o sistema linear com n operações, precisamos de um conjunto de direções conjugadas entre si. Na próxima seção veremos um método para obter esse conjunto.

5 Conjugação de Gram-Schimdt

O método de conjugação de Gram-Schimdt é similar ao processo de ortogonalização de Gram-Schimdt, com a diferença que utilizaremos a matriz A para tornar os vetores conjugados entre si.

Suponha que temos um conjunto de vetores linearmente independentes entre si $u_0, u_1, ...u_{n-1}$. O método consiste em construir a direção d_i subtraindo de u_i as componentes que não forem *A-ortogonais* às direções anteriores. Na prática, definimos $d_0 = u_0$ e para todos $i = 0, 1, 2, 3, ..., n-1$:

$$d_i = u_i + \sum_{k=0}^{i-1} \beta_{ik} d_k, \tag{23}$$

onde $\beta_{ik} \in \mathbb{R}^n$ está definido para todo $i > k$. Para encontrar seu valor realizamos o produto interno entre a equação (23) e Ad_j, sendo $i > j$.

$$\begin{aligned} d_i^T A d_j &= u_i^T A d_j + \sum_{k=0}^{i-1} \beta_{ik} d_k^T A d_j \\ 0 &= u_i^T A d_j + \beta_{ij} d_j^T A d_j \\ \beta_{ij} &= -\frac{u_i^T A d_j}{d_j^T A d_j}. \end{aligned} \tag{24}$$

A dificuldade em utilizar a conjugação de Gram-Schimdt está na necessidade de armazenar todas as direções utilizadas até a i-ésima etapa para se obter a direção da etapa posterior $(i + 1)$.

5.1 Otimalidade do método de direções conjugadas

Como dito no início da Seção 4, o método de direções conjugadas encontra a cada iteração a melhor solução possível no espaço explorado.

O espaço explorado é o subespaço D_i gerado pelas i direções conjugadas na i-ésima iteração, que são linearmente independentes, como mostrado em (19).

$$D_i = span\{d_0, d_1, ..., d_{i-1}\}.$$

Essa solução ótima é pelo fato do método de direções conjugadas escolher um valor $e_0 + D_i$ que minimiza a norma de energia $\|e_i\|_A^2$, sendo definida por $\|e_i\|_A^2 = e_i^T A e_i$.

Para vermos essa propriedade, podemos representar a norma de energia como um somatório em termos das direções conjugadas:

$$\begin{aligned} \|e_i\|_A^2 &= e_i^T A e_i \\ &= (\sum_{j=i}^{n-1} \delta_j d_j)^T A (\sum_{k=i}^{n-1} \delta_k d_k), \text{ pela equação (20)} \\ &= (\sum_{j=i}^{n-1} \sum_{k=i}^{n-1} \delta_j \delta_j d_j^T A d_k) \\ &= (\sum_{j=i}^{n-1} \delta_j^2 d_j^T A d_j), \text{ pela conjugação das direções.} \end{aligned} \tag{25}$$

Veja que as direções na qual a norma de energia, na i-ésima etapa, se relaciona ainda não foram percorridas. Assim, qualquer outro valor do erro tomado no espaço $e_0 + D_i$ será uma combinação dessas e de direções já utilizadas, o que prova que a norma de energia obtida em (25) é mínima. Além disso, como mostrado em (22), a cada iteração uma componente do erro é zerada.

Outra propriedade importante é de que o resíduo r_i é ortogonal ao subespaço D_i. Para constatar essa propriedade, basta pré-multiplicar a expressão (20) por -$d_k^T A$, donde segue que:

$$\begin{aligned} -d_i^T A e_j &= \sum_{j=0}^{n-1} \delta_j - d_i^T A d_j \\ d_i^T r_j &= 0, \quad \text{pela conjugação das direções onde } i < j. \end{aligned} \tag{26}$$

Por fim, o resíduo também pode ser representado de forma a reduzir os produtos matriz-vetor, como foi feito no método de máxima descida (11). Por recorrência temos que:

$$\begin{aligned} r_{i+1} &= -Ae_{i+1} \\ r_{i+1} &= -A(e_i + \alpha_i d_i) \\ r_{i+1} &= r_i - \alpha_i A d_i. \end{aligned} \tag{27}$$

Além de uma melhora no ponto de vista computacional, essa relação mostra que o resíduo na i-ésima iteração é uma combinação linear do resíduo r_{i-1} e do vetor Ad_{i-1}.

6 Método de Gradientes Conjugados

O MGC é basicamente um método de direções conjugadas no qual as direções são criadas pela conjugação dos resíduos, que são o oposto do gradiente, como definimos anteriormente em (4). Em outras palavras, definimos que $u_i = r_i$.

Então, vamos apresentar as propriedades teóricas que fazem a escolha dos resíduos interessante. Primeiramente, sabendo que o resíduo é ortogonal as direções já percorridas (26), é garantido que as novas direções serão linearmente independentes. Aliás, o resíduo é também ortogonal aos resíduos anteriores: efetuando o produto interno entre a equação (23) e r_j, com $j < i$, chegamos

$$\begin{aligned} d_i^T r_j &= u_i^T r_j + \sum_{k=0}^{i-1} \beta_{ik}^T r_j d_k \\ 0 &= u_i^T r_j \\ 0 &= r_i^T r_j, \quad i \neq j. \end{aligned}$$

Essa equação também nos mostra que os gradientes $\nabla f(0)$, $\nabla f(1)$, ..., $\nabla f(n-1)$ compõem um conjunto de vetores ortogonais em $\mathbb{R}^n$.

Diante dessas propriedades podemos ainda repensar o processo de conjugação de Gram-Schmidt, tendo agora um conjunto u formado por resíduos.

$$\beta_{ij} = -\frac{r_i^T A d_j}{d_j^T A d_j}$$

Pré-multiplicando (27), reescrito com índices j, por r_i, temos:

$$\begin{aligned} r_i^T r_{j+1} &= r_i^T r_j - \alpha_j r_i^T A d_j \\ \alpha_j r_i^T A d_j &= r_i^T r_j - r_i^T r_{j+1}. \end{aligned} \tag{28}$$

Da equação (28), segue que:

- quando $i = j$, então $r_i^T r_j \neq 0$ e $r_i^T r_{j+1} = 0$, logo:

$$r_i^T A d_j = \frac{1}{\alpha_i} r_i^T r_i.$$

- quando $i = j + 1$, então $r_i^T r_j = 0$ e $r_i^T r_{j+1} \neq 0$, logo:

$$r_i^T A d_j = -\frac{1}{\alpha_{i-1}} r_i^T r_i.$$

- para quaisquer outros valores de i e j o produto interno dos resíduos será sempre nulo.

Como definimos na Seção 5, $\beta_{ij} \in \mathbb{R}^n$ está definido para todo $i > j$. Assim, teremos um único coeficiente não nulo, em que $i = j + 1$. Portanto:

$$\beta_i \equiv \beta_{i,i-1} = \frac{1}{\alpha_{i-1}} \frac{r_i^T r_i}{d_{i-1}^T A d_{i-1}}. \tag{29}$$

Com isso, temos a maior vantagem do MGC. Não é mais necessário armazenar as direções anteriores para garantir que novas direções sejam *A-ortogonais*, o que nos permite reduzir a equação (23) para:

$$d_i = r_i + \beta_i d_{i-1}, \text{ com } \beta_i \text{ dado por (29).} \tag{30}$$

Por fim, com mais uma relação, poderemos simplificar as equações (29) e (17). Para isso, faremos o produto interno entre r_i e (30):

$$\begin{aligned} r_i^T d_i &= r_i^T r_i + \beta_i r_i^T d_{i-1} \\ r_i^T d_i &= r_i^T r_i, \text{ pela equação (26) .} \end{aligned} \tag{31}$$

Com essa relação definida podemos reduzir (17) para:

$$\alpha_i = \frac{r_i^T r_i}{d_i^T A d_i}. \tag{32}$$

E, como sabemos o valor de α_{i-1}, reduzimos também (29) para:

$$\beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}. \tag{33}$$

Desse modo, foram apresentados todos os elementos que compõem o algoritmo do MGC.

Algoritmo 2: Método de Gradientes Conjugados

Entrada: $A \in \mathbb{R}^{n \times n}$, $x_0 \in \mathbb{R}^n$, $b \in \mathbb{R}^n$, i_{max} e uma tolerância e
Saída: $x \in \mathbb{R}^n$

$d_0 = r_0 = b - Ax_0$
para $i = 0, 1, \ldots, i_{max}$ **faça**
    $\alpha_i = \frac{r_i^T r_i}{d_i^T A d_i}$
    $x_{i+1} = x_i + \alpha_i d_i$
    $r_{i+1} = r_i - \alpha_i A d_i$
    se $||r_{i+1}|| < e$ **então**
        return x_{i+1}
    fim se
    $\beta_i = \frac{r_{i+1}^T r_{i+1}}{r_i^T r_i}$
    $d_{i+1} = r_i + \beta_i d_{i-1}$
    $i = i + 1$
fim para
return x

---

7 Método de Newton-MGC com busca linear

Mesmo que o MGC seja potencialmente mais eficiente que os métodos apresentados anteriormente, a versão que apresentamos é limitada a somente resolver sistemas lineares nos quais a matriz em questão é positiva definida, isto é, sistemas lineares na forma (3).

Para que possamos utilizá-lo para resolver problemas não lineares uma abordagem interessante é usá-lo como um complemento ao método de Newton. Assim, com poucas alterações, podemos utilizá-lo para calcular a direção de busca do método de Newton.

O método de Newton-MGC embasa-se, tal como os demais mostrados, em gerar iterandos $x_1, x_2, \ldots$ partindo de um ponto inicial x_0 de forma que

$$x_{k+1} = x_k + \alpha_k d_k, \tag{34}$$

sendo α_k o tamanho do passo que daremos na direção d_k. O procedimento de busca linear será feito por *Backtracking*, isto é, retroativamente, tendo como condição de parada a seguinte desigualdade:

$$f(x_k + \alpha_i d_k) \leq f(x_k) + c\alpha_i \nabla f_k^T d_k, \tag{35}$$

no qual temos uma constante $c \in (0, 1)$ e $i = 1, 2, \ldots$ representando as iterações da busca linear. Em outros termos, a desigualdade (35), conhecida como condição de *Armijo*, define que a redução em f deve ser proporcional ao passo α_i e a derivada direcional $\nabla f_k^T d_k$.

Como mencionado, a constante c assume valores no intervalo entre 0 e 1, exclusos. O porquê disso é que, no caso de $c = 0$, não ocorreria decréscimo suficiente no valor função e, no caso de $c = 1$, a expressão $f(x_k) + c\alpha_i \nabla f_k^T d_k$ passaria a ser a expressão da reta tangente, logo, em funções convexas, não seria possível calcular um α, visto que a reta tangente fica abaixo da função.

Antes de prosseguirmos, vamos provar que existe um ponto que satisfaz a condição de parada definida em (34). Para isso, é importante relembrar o teorema do valor médio, que será usado na demonstração do Teorema 3.

Dado uma função contínua e diferenciável $\phi : \mathbb{R} \to \mathbb{R}$ e dois números reais α_0 e α_1 que atendem $\alpha_1 > \alpha_0$, temos que

$$\begin{aligned} \phi(\xi)' &= \frac{\phi(\alpha_1) - \phi(\alpha_0)}{\alpha_1 - \alpha_0} \\ \phi(\alpha_1) &= \phi(\alpha_0) + \phi(\xi)'(\alpha_1 - \alpha_0), \end{aligned} \tag{36}$$

com $\xi \in (\alpha_0, \alpha_1)$. Estendendo (36) para funções com n variáveis $f : \mathbb{R}^n \to \mathbb{R}$ e para qualquer vetor $d \in \mathbb{R}^n$, temos

$$f(x + d) = f(x) + \nabla f(x + \alpha d)^T d, \tag{37}$$

para $\alpha \in (0, 1)$, $\phi(\alpha) = f(x + \alpha d)$, $\alpha_1 = 1$ e $\alpha_0 = 0$. Com isso estabelecido, vamos apresentar o teorema.

Teorema 3. *Seja $f : \mathbb{R}^n \to \mathbb{R}$ contínua e diferenciável e d_k uma direção de descida em x_k, assumimos que $f(x_k + \alpha_i d_k)$ está restrita para valores de α_i positivos. Logo, se $0 < c < 1$, existe um intervalo no tamanho do passo ao longo da direção d_k que satisfaz a condição de Armijo (35).*

Demonstração. Temos que $\phi(\alpha) = f(x + \alpha d)$ restrito para $\alpha > 0$. Como $0 < c < 1$, a linha $l(\alpha) = f(x_k) + c\alpha_i \nabla f_k^T d_k$ não está restrita a valores de α e, portanto, intersecta $\phi(\alpha)$ em pelo menos um ponto. Dado um $\alpha' > 0$ o menor valor de α que intersecta $\phi(\alpha)$, temos que:

$$f(x_k + \alpha' d_k) \leq f(x_k) + c\alpha' \nabla f_k^T d_k. \tag{38}$$

A condição de *Armijo* ainda é satisfeita para valores positivos menores que α'. Logo, pelo teorema do valor médio (37), existe um $\alpha'' \in (0, \alpha')$ de modo que

$$f(x_k + \alpha' d_k) = f(x_k) + \alpha' \nabla f(x_k + \alpha'' d_k)^T d_k. \tag{39}$$

Combinando (38) e (39)

$$\nabla f(x_k + \alpha'' d_k)^T d_k = c \nabla f_k^T d_k,$$

vemos que existem valores de α positivos que satisfazem a condição de *Armijo* (35). □

Por fim, após garantir que nossa condição de parada pode ser atendida, podemos calcular α retroativamente até que a condição de *Armijo* seja satisfeita.

Algoritmo 3: Busca linear retroativa

Entrada: $\bar{\alpha} > 0$, $\rho \in (0, 1)$, $c \in (0, 1)$
Saída: α
1: $i = 0$; $\alpha_i = \bar{\alpha}$

2: **enquanto** $f(x_k + \alpha_i d_k) > f(x_k) + c\alpha_i \nabla f_k^T d_k$ **faça**
3: $\alpha_{i+1} = \rho\alpha_i$
4: $i = i + 1$
5: **fim enquanto**
6: **return** α

A constante ρ, responsável por reduzir α, na prática, pode variar a cada iteração da busca linear. Na implementação que fizemos, abordada com mais detalhes na Seção 9, utilizamos duas constantes ρ_{lo} e ρ_{hi}, que satisfazem $0 < \rho_{lo} < \rho_{hi} < 1$, de forma que calculando o mínimo entre α e $\rho_{hi}\alpha$ evitamos reduções muito grandes e calculando máximo entre α e $\rho_{lo}\alpha$ evitamos reduções muito pequenas.

Assim, nos resta calcular a direção de busca. A direção de Newton é uma das mais importantes e é derivada da aproximação de segunda ordem da série de Taylor para a função $f(x_k + d)$, que é

$$f(x_k + d) \approx f_k + d^T \nabla f_k + \frac{1}{2} d^T \nabla^2 f_k d = m_k(d). \tag{40}$$

A direção de Newton é obtida ao encontrar a direção d que minimiza o modelo quadrático $m_k(d)$. De outro modo, queremos definir a direção d que faz a derivada direcional de $m_k(d)$ nula, matematicamente, temos que:

$$d = -(\nabla^2 f_k)^{-1} \nabla f_k. \tag{41}$$

Podemos reescrever (41) para:

$$B_k d = -\nabla f_k, \tag{42}$$

onde B_k é a hessiana $\nabla^2 f_k$. Como dito no início dessa seção, calcularemos a direção de busca utilizando o gradientes conjugados, por consequência, o MGC será usado para solucionar o sistema linear (42).

Em relação ao Algoritmo 2 foram feitas algumas alterações no MGC. Quando B_k for definida positiva, a sequência de iterações z_j, do MGC, convergirá para a direção d que soluciona (42), porém, caso na primeira iteração B_k não for definida positiva, retornaremos a direção de máxima descida $-\nabla f_k$, que é também uma direção de curvatura não positiva para f avaliado em x_k. Além disso, a tolerância e varia a cada iteração para obter uma taxa de convergência mais adequada, como descrito melhor em [2](2006, p.166) por Nocedal e Wright.

Em suma, o Newton-MGC é adequado para solucionar problemas grandes de otimização não linear irrestrita, apresentando dificuldades quando a hessiana B_k for singular ou quase singular.

Algoritmo 4: Newton-MGC

Entrada: $x_0 \in \mathbb{R}^n$
Saída: $x^* \in \mathbb{R}^n$
1: **para** $k = 0, 1, 2, \ldots, k_{max}$ **faça**

$z_0 = 0$; $d_0 = r_0 = -\nabla f_k$
$e_k = min(0.5, \sqrt{||\nabla f_k||})||\nabla f_k||$
para $j = 0, 1, 2, \ldots, j_{max}$ **faça**
  se $d_j^T B_k d_j \leq 0$ **então**
    se $j = 0$ **então**
      return $d = -\nabla f_k$
    senão
      return $d = z_j$
    fim se
  fim se
  $\alpha_j = r_j^T r_j / d_j^T B_k d_j$
  $z_{j+1} = z_j + \alpha_j d_j$
  $r_{j+1} = r_j - \alpha_j B_k d_j$
  se $||r_{j+1}|| < e_k$ **então**
    return $d_k = z_{j+1}$
  fim se
  $\beta_j = \frac{r_{j+1}^T r_{j+1}}{r_j^T r_j}$
  $d_{j+1} = r_{j+1} + \beta_j d_j$
fim para
Calcular α_k retroativamente (Algoritmo 3)
$x_{k+1} = x_k + \alpha_k d_k$
fim para
return $x^* = x_{k+1}$

8 Método de Newton-Cholesky

Quando a hessiana é quase singular, a direção do Newton-MGC pode ser longa e de baixa qualidade, o que exige da busca linear uma quantidade expressiva de iterações. O método Newton-Cholesky apresenta uma abordagem em que, quando a hessiana não for positiva definida, são feitas modificações para que seja positiva definida.

Do mesmo modo que foi utilizado o MGC para calcular a direção de Newton, será usado a fatoração de Cholesky para auxiliar a solucionar o sistema linear (41). O restante do algoritmo será o mesmo, tendo mudado apenas a maneira de obter a direção de busca.

A estratégia que usaremos para alterar a hessiana é de aumentar os valores na sua diagonal principal, quando necessário, para garantirmos que seja positiva o suficiente.

Vamos começar relembrando a decomposição de Cholesky. Toda matriz simétrica definida pode ser reescrita como

$$A = LDL^T, \tag{43}$$

onde L é uma matriz triangular inferior com diagonal unitária e D uma matriz diagonal. Um dos motivos da fatoração de Cholesky ser interessante é que sabemos se uma matriz é positiva definida ou não apenas olhando para a matriz D, que terá valores positivos na diagonal quando for.

Para melhor entendimento, vemos que

$$\begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{bmatrix} \begin{bmatrix} 1 & l_{12} & l_{13} \\ 0 & 1 & l_{23} \\ 0 & 0 & 1 \end{bmatrix}$$
$$= \begin{bmatrix} d_1 & \dots & \dots \\ l_{21}d_1 & l_{21}d_1l_{21} + d_2 & \dots \\ l_{31}d_1 & l_{21}d_1l_{31} + l_{32}d_2 & l_{31}d_1l_{31} + l_{32}d_2l_{32} + d_3 \end{bmatrix},$$

observamos então que, para calcular os elementos das matrizes D e L, temos

$$d_j = c_{jj}, \tag{44}$$
$$l_{ij} = \frac{c_{jj}}{d_j}, \tag{45}$$
$$\text{para } c_{jj} = a_{jj} - \sum_{s=1}^{j-1} l_{js}d_s l_{js},$$

sendo $j = 1, 2, \dots, n$ o iterador da fatoração e n a dimensão da matriz. Seguindo, quando a hessiana for singular, calcularemos a direção do seguinte modo:

$$d_j = max\left(|c_{jj}|, \left(\frac{\theta_j}{\beta}\right)^2, \delta\right), \text{ com } \theta_j = max_{j<i\leq n}|c_{ij}|., \tag{46}$$

sendo β e δ constantes para controlar a qualidade da modificação.

Agora, utilizando as matrizes obtidas na fatoração, iremos solucionar um sistema linear na forma (3), sendo $A = LDL^T$, b o oposto do gradiente e x a direção de busca. Substituindo, temos

$$LDL^T d = -\nabla f_k. \tag{47}$$

Podemos reescrever (47) subdividindo em 3 problemas:

1. $Lz = -\nabla f_k$, para obter z;
2. $Dy = z$, para obter y;
3. $L^T d = y$, para obter a direção d.

Para resolver os sistemas na forma na forma $Lx = b$, sendo L uma matriz triangular inferior com diagonal unitária, basta

$$x_i = b_i - \sum_{j=1}^{i-1} l_{i,j}x_j, \text{ para } i = 1, \dots, n. \tag{48}$$

Por fim, antes de apresentarmos o algoritmo, ainda é possível aprimorar a aproximação da hessiana efetuando permutações. Para fazer isso, fatoramos os maiores elementos das diagonais primeiro, permutando as informações de suas linhas e colunas com as informações da j-ésima linha e coluna. Com isso, teremos que

$$P^T H_k P + E_k = LDL^T, \tag{49}$$

onde P é a matriz de permutação e E_k uma matriz diagonal não negativa que valerá zero quando H_k for suficientemente positiva definida e, quando não, garantirá que H_k seja suficientemente positiva definida.

O sistema linear (47) será solucionado da mesma maneira, exceto pela necessidade de considerar a permutação feita na matriz do sistema. Para tanto, é necessário pré-multiplicar o oposto do gradiente pela matriz P^T e pré-multiplicar o valor resultante do sistema linear por P para obter a direção de busca. Para melhor visualizar essas alterações

$$H_k d = -\nabla f_k \tag{50}$$

é o que queremos solucionar. Pré-multiplicando (50) por P^T, temos que

$$\begin{aligned} P^T H_k d &= -P^T \nabla f_k \\ P^T H_k P y &= -P^T \nabla f_k, \text{ para } d = Py \\ LDL^T y &= b. \end{aligned} \tag{51}$$

Solucionaremos (51) tal qual explicado anteriormente, apenas tendo que se atentar às permutações. Como é possível notar, a matriz E_k não aparece em (51). Isso se deve ao fato de que queremos trabalhar com a hessiana modificada, ao subtrair LDL^T por E_k perderíamos essas alterações.

Em suma, o método Newton-Cholesky é preciso e lida com casos onde a hessiana é singular. Iremos apresentar o algoritmo do método de Newton-Cholesky e o da fatoração de Cholesky separadamente para melhor organização. Segue ambos abaixo.

Algoritmo 5: Cholesky

Entrada: $H \in \mathbb{R}^{n\times n}$
Saída: LDL^T, com $L \in \mathbb{R}^{n\times n}$ e $D \in \mathbb{R}^{n\times n}$

$\nu = max\{1, \sqrt{n^2-1}\}$
$\beta^2 = max\{\gamma, \frac{\xi}{\nu}, \epsilon_m\}$
para $i = 1, \ldots, n$ **faça**
    $c_{ii} = H_{ii}$
fim para
para $j = 1, 2, \ldots, n$ **faça**
    $cmax = q = 0$ ▷ Ache o maior valor da diagonal
    para $i = j, \ldots, n$ **faça**
        se $abs(c_{ii}) > cmax$ **então**
            $q = i$
            $cmax = abs(c_{ii})$
        fim se
    fim para
    $c_{ii} \leftrightarrow c_{qq}$ ▷ Permute informações das linhas e colunas q e j
    para $s = 1, \ldots, j-1$ **faça**
        $l_{js} = c_{js}/d_s$ ▷ Calcule a j-ésima linha de L
    fim para
    $\theta = 0$
    para $i = j+1, \ldots, n$ **faça**
        $soma = 0$

para $s = 1, \ldots, j-1$ **faça**
  $soma = soma + l_{js}c_{is}$
fim para
$c_{ij} = H_{ij} - soma$
se $\theta < ||c_{ij}||$ **então**
  $\theta = ||c_{ij}||$ ▷ Ache o maior $||l_{ij} * d_j||$
fim se
fim para
$d_j = max\left(|c_{jj}|, \left(\frac{\theta}{\beta}\right)^2, \delta\right)$
para $i = j+1, \ldots, n$ **faça**
  $c_{ii} = c_{ii} - \frac{c_{ij}^2}{d_j}$ ▷ Atualize os elementos da diagonal
fim para
fim para
return l, d

Algoritmo 6: Newton-Cholesky

Entrada: $x_0 \in \mathbb{R}^n$, k_{max} e uma tolerância e
Saída: $x^* \in \mathbb{R}^n$

para $k = 1, \ldots, k_{max}$ **faça**
  Fatorar H_k (Algoritmo 5)
  $b = -P^T \nabla f_k$
  para $i = 1, 2, \ldots, n$ **faça**
    $z_i = b_i - \sum_{j=1}^{i-1} l_{i,j} z_j$ ▷ $Lz = b$
  fim para
  $y = D^{-1}z$
  para $i = n, n-1, \ldots, 1$ **faça**
    $d_i = y_i - \sum_{j=1}^{i-1} l_{i,j} d_j$ ▷ $L^T d = y$
  fim para
  $d = Pd$
  Calcular α_k retroativamente (Algoritmo 3)
  $x_{k+1} = x_k + \alpha_k d$
  se $||\nabla f_{k+1}|| < e$ **então**
    return x_{k+1}
  fim se
fim para
return $x_k = x_{k+1}$

9 Experimentos numéricos

Tendo apresentado os algoritmos estudados, iremos mostrar alguns testes numéricos feitos utilizando uma implementação do Newton-MGC, em Julia, e problemas propostos por Moré, Garbow e Hillstrom [3], fornecidos pelo banco de testes do CUTEst.

Todos os experimentos foram realizados numa arquitetura x86, 64 bits, modelo Intel(R) Core(TM) i5-4210U, frequência base de 1.70GHz, no sistema operacional Debian

GNU/linux 10 (buster) com 975,1 GB disponíveis no disco. As bibliotecas utilizadas do Julia 1.0.3 foram: CUTEst v0.7.0, NLPModels v0.10.1, TimerOutputs v0.5.6, LinearOperators v1.1.0 e ForwardDiff v0.10.12.

As funções que aparecem em [3] estão na forma de problemas de quadrados mínimos, ou seja, definidas por $f_1, f_1, \ldots, f_m$. Podemos obter um problema de minimização irrestrita ao definir nossa função objetivo como:

$$f(x) = \sum_{i=1}^{m} f_i^2(x) \tag{52}$$

e resolver

$$\begin{aligned} &min f(x) \\ &x \in \mathbb{R}^n. \end{aligned}$$

Os testes foram divididos em 2 partes:

1. Na **Parte 1**, são apresentados resultados dos 35 problemas de minimização irrestrita propostos em [3] para o algoritmo Newton-MGC;
2. Na **Parte 2**, é feita uma comparação entre o método de Newton-MGC e Newton-Cholesky.

9.1 Parte 1

Nesta parte, serão apresentados os resultados dos 35 problemas propostos por Moré, Garbow e Hillstrom [3], obtidos por meio de uma implementação em Julia[1] do NewtonMGC. Para avaliar o desempenho dessa implementação, seguiremos os critérios apresentados na Tabela 1.

Sigla	Descrição
AF	Quantidade de avaliações da função objetivo.
AG	Quantidade de avaliações do gradiente da função objetivo.
AH	Quantidade de avaliações da matriz hessiana da função objetivo.
IT	Quantidade de iterações do algoritmo principal (Newton).
ITSP	Quantidade de iterações do subproblema (MGC ou Cholesky).
ITBL	Quantidade de iterações da busca linear.
TE	Tempo de execução (em segundos).
VG	Valor de $\|f(x^*)\|$, sendo $f(x^*)$ definida em (52) e x^* o valor do vetor solução.
CP	Critério de parada.

Tabela 1: Descrição dos critérios de avaliação adotados.

Os critérios de parada são 3:

[1] Acesse `https://github.com/SergioAlmeidaCiprianoJr/IC-2019-2020/blob/master/NewtonCG/NewtonCG.jl?ts=4`

1. Algoritmo excedeu o número de iterações (IT), sendo o limite de 1000 iterações.
2. Norma do gradiente menor que a tolerância e, definida como $e = 10^{-8}$.
3. Tempo limite de 10 minutos excedido. Caso o tempo limite seja atingido na primeira iteração, o processo será interrompido e nenhum dado será computado.

Nos testes, serão usadas as mesmas numerações, dimensões n e m (sempre que descritas), e pontos iniciais do artigo [3]. Ademais, serão criadas siglas para cada problema para facilitar sua referência, tal como apresentados na Tabela 2.

Função	Sigla	n	m	Ponto Inicial
1	ROS	2	2	$(-1.2, 1)$
2	FRF	2	2	$(0.5, -2)$
3	PBS	2	2	$(0, 1)$
4	BBS	2	3	$(1, 1)$
5	BEF	2	3	$(1, 1)$
6	JSF	2	10	$(0.3, 0.4)$
7	HVF	3	3	$(-1, 0, 0)$
8	BAF	3	15	$(1, 1, 1)$
9	GAUS	3	15	$(0.4, 1, 0)$
10	MEYE	3	16	$(0.02, 4000, 250)$
11	GULF	3	99	$(5, 2.5, 0.15)$
12	BOX3	3	10	$(0, 10, 20)$
13	PSF	4	4	$(3, -1, 0, 1)$
14	WOOD	4	11	$(-3, -1, -3, -1)$
15	KOF	4	11	$(0.25, 0.39, 0.415, 0.39)$
16	BDF	4	20	$(25, 5, -5, -1)$
17	OB1	5	33	$(0.5, 1.5, -1, 0.01, 0.02)$
18	BIG	6	13	$(1, 2, 1, 1, 1, 1)$
19	OB2	11	65	$(1.3, 0.65, 0.65, 0.7, 0.6, 3, 5, 7, 2, 4.5, 5.5)$
20	WATF	12	31	$(0, ..., 0)$
21	EROS	10	10	$(\xi_j$, onde $\xi_{2j-1} = -1.2$ e $\xi_{2j} = 1)$
22	EPSF	4	4	$(\xi_j$, onde $\xi_{4j-3} = 3$, $\xi_{4j-2} = -1$, $\xi_{4j-1} = 0$ e $\xi_{4j} = 1)$
23	PF1	4	5	$(\xi_j$, onde $\xi_j = j)$
24	PF2	4	8	$(\frac{1}{2}, ..., \frac{1}{2})$
25	VDIM	10	12	$(\xi_j$, onde $\xi_j = 1 - \frac{j}{n})$
26	TRIG	200	200	$(\frac{1}{n}, ..., \frac{1}{n})$
27	BALF	10	10	$(\frac{1}{2}, ..., \frac{1}{2})$
28	DBVF	12	12	$(\xi_j$, onde $\xi_j = t_j(t_j - 1))$
29	DIEF	50	50	$(\xi_j$, onde $\xi_j = t_j(t_j - 1))$
30	BTF	10	10	$(-1, ..., -1)$
31	BBF	10	10	$(-1, ..., -1)$
32	LFFR	200	400	$(1, ..., 1)$
33	LFR1	200	400	$(1, ..., 1)$
34	LFRZ	200	400	$(1, ..., 1)$
35	CHEB	10	10	$(\xi_j$, onde $\xi_j = \frac{j}{n+1})$

Tabela 2: Problemas do Moré, Garbow e Hillstrom.

Segue os resultados dos testes do algoritmo Newton-MGC na Tabela 3.

Função	VG	AF	AG	AH	IT	ITSP	ITBL	TE	CP
ROS	9.103829 E-15	27	65	64	64	111	27	0.77557	2
FRF	1.081813 E-12	0	10	9	9	13	0	0.00016	2
PBS	0.000000 E+00	0	1	0	0	0	0	0.00002	2
BBS	8.881784 E-10	0	5	4	4	5	0	0.00009	2
BEF	1.118009 E-11	2	13	12	12	18	2	0.00020	2
JSF	2.698798 E-07	20678	1000	1000	1000	1992	20678	0.05338	1
HVF	1.332268 E-13	7	20	19	19	40	7	0.00030	2
BAF	1.406650 E-15	1	13	12	12	25	1	0.00027	2
GAUS	0.000000 E+00	0	1	0	0	0	0	0.00002	2
MEYE	7.859909 E+01	15398	1000	1000	1000	1390	15398	0.04705	1
GULF	1.287354 E-10	12	27	26	26	47	12	0.00694	2
BOX3	5.616197 E-12	0	10	9	9	18	0	0.00036	2
PSF	4.482304 E-09	0	26	25	25	80	0	0.00041	2
WOOD	0.000000 E+00	30	386	385	385	1016	30	0.00532	2
KOF	5.831654 E-13	4	14	13	13	40	4	0.00036	2
VDF	1.129537 E-10	19	19	18	18	42	19	0.00055	2
OB1	8.411020 E-09	29	70	69	69	195	29	0.00831	2
BIG	1.261848 E-10	1	46	45	45	98	1	0.00144	2
AB2	1.716796 E-11	19	31	30	30	153	19	0.00886	2
WATF	9.579747 E-09	11	35	34	34	349	11	0.00759	2
EROS	7.626535 E-09	0	9	8	8	11	0	0.00014	2
EPSF	4.482304 E-09	0	26	25	25	80	0	0.00036	2
PF1	9.138767 E-09	23	37	36	36	53	23	0.00045	2
PF2	7.309127 E-09	306	145	144	144	469	306	0.00326	2
VDIM	5.032557 E-12	0	15	14	14	14	0	0.00023	2
TRIG	0.000000 E+00	0	1	0	0	0	0	0.00003	2
BALF	6.751915 E-13	0	8	7	7	13	0	0.00016	2
DBVF	0.000000 E+00	0	1	0	0	0	0	0.00003	2
DIEF	0.000000 E+00	0	1	0	0	0	0	0.00002	2
BTF	0.000000 E+00	0	1	0	0	0	0	0.00002	2
BBF	2.872743 E-11	0	14	13	13	61	0	0.00047	2
LFFR	5.792169 E-13	0	2	1	1	1	0	0.00212	2
LFR1	1.188288 E-02	5960	1000	1000	1000	1000	5960	1.82569	1
LFRZ	0.000000 E+00	0	1	0	0	0	0	0.00003	2
CHEB	2.792677 E-09	5	15	14	14	61	5	0.00229	2

Tabela 3: Resultados dos 35 problemas para o algoritmo Newton-MGC.

Diante desses resultados, podemos fazer algumas observações:

1. A quantidade de avaliações da hessiana é sempre igual ao número de iterações, entretanto, o número de avaliações do gradiente ultrapassa em um, o número de ambos, na maioria dos problemas. Isso se deve ao fato de que avaliamos o gradiente no início de cada iteração e, caso passe no segundo critério de parada, o processo se

encerra. É por esse motivo também que, quando o critério de parada é o primeiro, AG, AH e IT são iguais.

2. Outra relação interessante é acerca das avaliações da função objetivo. Elas ocorrem uma vez por iteração da busca linear, logo, AF e ITBL sempre serão idênticos.

3. Quanto ao tempo de execução de cada teste, vale ressaltar que varia em relação à um conjunto de variáveis como a máquina, o sistema operacional, a quantidade de processos em atividade no momento da execução do teste, entre outros fatores. No entanto, esse valor ainda é interessante quando há uma discrepância significativa entre problemas. Isso será melhor visto na comparação entre o Newton-MGC e o Newton-Cholesky.

4. Por fim, é possível ver que alguns testes já são finalizados na primeira iteração. Esses testes já satisfazem o segundo critério de parada antes mesmo de qualquer avaliação ser realizada e, portanto, não ofereceram muitas informações[2].

9.2 Parte 2

Nesta parte, foram escolhidos 3 problemas: WOOD, EROS e EPSF. Para cada um dos algoritmos, Newton-MGC e Newton-Cholesky, serão executados esses testes variando a quantidade de variáveis n. Além disso, também será feita uma comparação gráfica com o uso de perfis de desempenho. É importante enfatizar que os algoritmos serão executados nas mesmas condições, ou seja, com os mesmos critérios de parada e utilizando o mesmo ambiente de execução.

Os critérios serão os mesmos apresentados na Tabela 1. A única diferença entre ambos os métodos é na solução do sistema linear, logo, características como a quantidade de avaliações da hessiana ser sempre igual ao número de iterações, são vistas nas duas implementações. Assim, as colunas AH e ITBL não serão mais apresentadas, já que ambos valores podem ser deduzidos a partir das colunas IT e AF, respectivamente.

Os resultados dos testes do algoritmo Newton-MGC são apresentados na Tabela 4 e os resultados do Newton-Cholesky na Tabela 5.

Sigla	n	VG	AF	AG	IT	ITSP	TE	CP
WOOD	4	0.000000 E+00	30	386	385	1016	0.00532	2
WOOD	100	9.272824 E+00	3	1000	1000	1956	0.86676	1
WOOD	10000	1.090605 E+00	0	1000	1000	1513	4.62599	1
EROS	10	7.626535 E-09	0	9	8	11	0.00014	2
EROS	50	0.000000 E+00	0	10	9	13	0.98549	2
EROS	500	0.000000 E+00	0	10	9	13	0.76830	2
EROS	10000	0.000000 E+00	0	10	9	13	0.98205	2
EPSF	4	4.482304 E-09	0	26	25	80	0.00046	2
EPSF	20	4.704666 E-09	0	27	26	83	0.80734	2
EPSF	80	9.410469 E-09	0	27	26	84	1.10240	2

[2]Informações detalhadas de cada teste estão disponíveis em `https://github.com/SergioAlmeidaCiprianoJr/IC-2019-2020/tree/master/Tests/NewtonCG/mgh_problems`

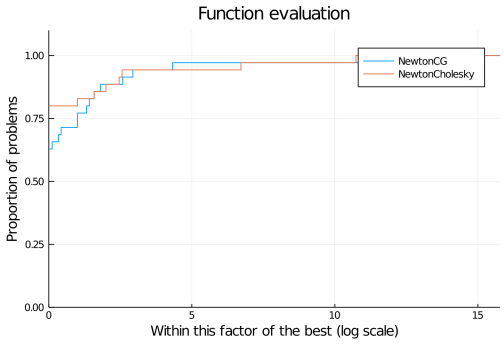
EPSF	10000	8.791239 E-09	0	30	29	100	1.05864	2

Tabela 4: Resultados do Newton-MGC para problemas escalados quanto ao número de variáveis.

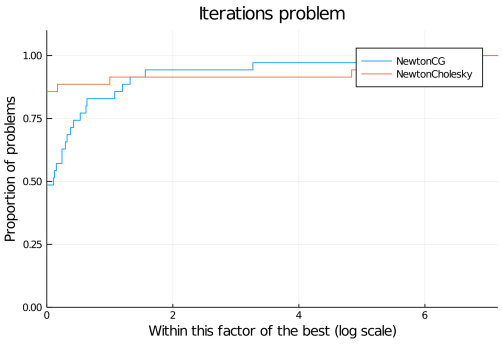
Função	n	VG	AF	AG	IT	ITSP	TE	CP
WOOD	4	6.175618 E-09	22	40	39	780	1.04854	2
WOOD	100	0.000000 E+00	22	41	40	212000	1.20705	2
WOOD	10000	-	-	-	-	-	-	3
EROS	10	3.546281 E-11	1	9	8	640	1.03189	2
EROS	50	7.929725 E-11	1	9	8	11200	1.05868	2
EROS	500	2.507599 E-10	1	9	8	1012000	3.82864	2
EROS	10000	-	-	-	-	-	-	3
EPSF	4	3.647543 E-09	0	22	21	420	1.04371	2
EPSF	20	8.156155 E-09	0	22	21	5460	1.29020	2
EPSF	80	4.833277 E-09	0	23	22	75680	1.06185	2
EPSF	10000	-	-	-	-	-	-	3

Tabela 5: Resultados do Newton-Cholesky para problemas escalados quanto ao número de variáveis.

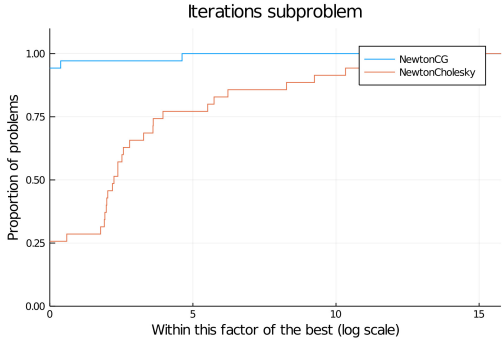
Além disso, foram criados 3 perfis de desempenho:



(a) Avaliações de Função.



(b) Iterações do Problema.



(c) Iterações do Subproblema.

Figura 1: Perfis de desempenho resultantes de testes com os 35 problemas de Moré, Garbow e Hillstrom.

Esses resultados nos permitem fazer as seguintes avaliações:

1. O método de Newton-Cholesky mostrou ser preciso nos casos menores e necessitou de poucas iterações do método de Newton e de poucas avaliações do gradiente e da matriz hessiana. Em contrapartida, o método de Newton-MGC precisou de mais iterações do método de Newton, mais avaliações do gradiente e da matriz hessiana. Mesmo assim, a diferença de precisão entre ambos não foi muito significativa.

2. Sobre a quantidade de iterações, por de tratar de um algoritmo com complexidade cúbica, no algoritmo de Newton-Cholesky as iterações do subproblema escalaram rapidamente a medida que o número de variáveis aumentava. Diante isso, todos os casos com 10000 variáveis excederam o limite de 10 minutos na primeira iteração. Já a performance do Newton-MGC continuou sendo muito boa, mesmo nos casos maiores. Vale notar que, mesmos nos casos com 1000 iterações do método de Newton (limite definido nos critérios de parada), o tempo de execução do Newton-MGC permaneceu pequeno.

3. Em relação ao consumo de memória, o gradientes conjugados necessita de pouca memória, visto que uma de suas propriedades mais interessantes é de poder calcular uma direção d_k usando somente d_{k-1}, sem a necessidade de armazenar todas as direções previamente percorridas. Em contrapartida, o método de Cholesky consome memória proporcional à quantidade de variáveis, podendo ser muito caso a matriz hessiana não seja esparsa.

10 Conclusão

Neste trabalho foi apresentado um estudo introdutório à otimização, onde introduzimos, implementamos e testamos o método de gradientes conjugados.

Com os experimentos numéricos, podemos dizer que aplicar o método de gradientes conjugados ao método de Newton mostrou ser uma estratégia adequada para a solução de problemas na forma (42). Em especial, apresentou bons resultados de performance mesmo em problemas com grande quantidade de variáveis. Diante disso, vimos que o MGC manifestou robustez nos quesitos analisados e, à vista de suas propriedades, mostrou-se um campo fértil para avanços.

Referências

[1] J.R. Shewchuk. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Edition $1\frac{1}{4}$. 1994. School of Computer Science. Carnegie Mellon University, Pittsburgh, PA, 58p. Disponível em `http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf`. Último acesso em 16 de abr. 2020.

[2] J. Nocedal & S.J. Wright, *Numerical Optimization*. 2. ed. 2006. New York: Springer, 664p.

[3] J.J. Moré, B.S. Garbow & K.E. Hillstrom, "Testing unconstrained optimization software", *ACM Transactions on Mathematical Software*, Volume 7, pp. 17-41, 1981.

[4] J.L.C. Gardenghi & S.A. Santos, "Minimização irrestrita usando gradientes conjugados e regiões de confiança". Disponível em: `https://www.ime.unicamp.br/sites/default/files/pesquisa/relatorios/rp-2012-4.pdf`. Último acesso em 9 de jul. 2020.

[5] Gill, Philip E. and Murray, Walter and Wright, Margaret H. *Practical optimization*. 1981. Academic Press Inc. Harcourt Brace Jovanovich Publishers, London. Disponível em `https://www.bibsonomy.org/bibtex/20d69887a67bc0e1844ba347d6b63c296/dwassel`. Último acesso em 26 agos. 2020.

[6] D. Orban and A. S. Siqueira e contribuidores. *NLPModels.jl: Data Structures for Optimization Models*. 2020. Disponível em `https://github.com/JuliaSmoothOptimizers/NLPModels.jl`. Último acesso em 12 de out. 2020.