

Programação Não-Linear

Algoritmos de Busca Linear

Lucas Magno
7994983

Introdução

Otimização Irrestrita

Dado $f : \mathbb{R}^n \rightarrow \mathbb{R}$, um problema de otimização irrestrita pode ser escrito

$$\begin{array}{ll} \text{minimizar} & f(x) \\ \text{sujeito a} & x \in \mathbb{R}^n \end{array}$$

ou seja, queremos encontrar x^* que minimize f (dito minimizador), o que pode se dar em duas formas:

Minimizador global $f(x^*) \leq f(x) \quad \forall x \in \mathbb{R}^n$

Minimizador local $f(x^*) \leq f(x) \quad \forall x \mid \|x - x^*\| \leq \epsilon$ para algum $\epsilon > 0$ (isto é, existe uma vizinhança na qual $f(x^*)$ tem valor mínimo).

Analogamente a problemas unidimensionais, sendo f diferenciável, pode-se mostrar que, se x^* é um minimizador (local ou global), então

$$\nabla f(x^*) = 0$$

o que nos dá uma forma de encontrar os minimizadores, pois basta encontrar os pontos que anulam o gradiente (dito estacionários). No entanto, de forma geral, não temos como determinar localmente se um ponto é mínimo global, então os métodos implementados se contentarão em encontrar mínimos locais (sejam eles globais ou não).

Busca Linear

Os métodos utilizados neste trabalho seguem todos o mesmo paradigma: a *busca linear*. Tal paradigma consiste basicamente em se escolher, a partir de um ponto x , uma direção de busca d e então escolher um passo α nessa direção, de forma que o valor da função f que queremos minimizar diminua adequadamente (cujo significado será discutido adiante).

Ou seja, a forma básica da busca linear é (dado x^0):

Algorithm 1 Busca Linear

```

1:  $k \leftarrow 0$ 
2: while  $\nabla f(x^k) \neq 0$  do
3:   Escolher  $d^k \in \mathbb{R}^n$  tal que  $\nabla f(x^k)^\top d^k < 0$ 
4:   Escolher  $\alpha_k \in \mathbb{R}$  tal que  $f(x^k + \alpha_k d^k) < f(x^k)$ 
5:    $x^{k+1} \leftarrow x^k + \alpha_k d^k$ 
6:    $k \leftarrow k + 1$ 
7: end while

```

Em que a linha 4 impõe o decréscimo da função e a 5 é a atualização da iteração. A linha 3, entretanto, introduz a condição

$$\nabla f(x^k)^\top d^k < 0 \quad (1)$$

cuja motivação pode ser vista através da expansão em Taylor da função:

$$f(x + \alpha d) = f(x) + \alpha \nabla f(x)^\top d + \frac{1}{2} \alpha^2 d^\top \nabla^2 f(x) d + o(\|\alpha d\|^2)$$

assim, sempre podemos escolher um passo α suficientemente pequeno tal que o termo de primeira ordem domine os seguintes e, por ele ser negativo (restringindo $\alpha > 0$), vale que

$$f(x + \alpha d) < f(x)$$

Portanto, direções que satisfaçam essa condição garantem que seja sempre possível decrescer o valor da função e por isso são chamadas de *direções de descida*.

Esse algoritmo gera a sequência

$$\{x^k\}_{k=0}^{\infty}$$

para a qual *esperaríamos* que valesse

$$\lim_{k \rightarrow \infty} x^k = x^*$$

mas não é o caso, pois, dados

$$\begin{aligned} f(x) &= x^2 \\ x^k &= 1 + \frac{1}{k} \\ d^k &= -1 \end{aligned}$$

para todo $k > 0$ e notando que $x^{k+1} < x^k$, temos

$$\begin{aligned} \alpha_k &= x^k - x^{k+1} &> 0 \\ \nabla f(x^k)^\top d^k &= -2x^k &< 0 \quad \forall x > 0 \\ f(x^{k+1}) - f(x^k) &= (x^{k+1})^2 - (x^k)^2 &< 0 \end{aligned}$$

Logo, essa é uma sequência perfeitamente válida ao que concerne o algoritmo 1 e apesar disso é fácil notar que

$$\lim_{k \rightarrow \infty} x^k = 1$$

que não é nem ponto estacionário de f , então essas condições não são suficientes para garantir a convergência do algoritmo.

Condições

Felizmente, com apenas algumas condições a mais podemos resolver o problema da convergência, a saber:

Condição de Armijo

É natural que exijamos que a função deva decrescer a cada iteração, mas somente isso basta? *Quanto* f deve decrescer? Uma forma de exigir um decréscimo mínimo é através da *condição de Armijo*:

Dado $0 < \gamma < 1$ (normalmente $\gamma = 10^{-4}$)

$$f(x + \alpha d) < f(x) + \gamma \alpha \nabla f(x)^\top d$$

que, sendo d direção de descida (vale a equação 1), impõe um limite superior no valor da função no novo ponto melhor do que a condição anterior (a qual equivale a fazermos $\gamma = 0$ aqui).

Além disso, sempre existe um α suficientemente pequeno para o qual Armijo vale, então podemos utilizar a seguinte técnica, conhecida como *backtracking* para determinar um passo satisfatório: partindo de um passo candidato¹, se este não satisfizer Armijo, diminua ele e tente novamente. Repetindo essas etapas, chegaremos a um valor que o satisfaça e então o adotamos.

Condição do Passo

É importante, entretanto, que esses passos não sejam muito pequenos, pois, se estes diminuírem rápido demais, a sequência pode convergir a algum ponto de acumulação que não tem propriedade de mínimo ou ponto estacionário. É o que acontece no exemplo anterior, em que os passos α_k convergem rapidamente para zero, levando o algoritmo a um ponto qualquer (no ponto de vista da minimização).

De forma similar, queremos evitar passos muito longos, pois o algoritmo pode ficar preso num ziguezague ao redor do ponto estacionário.

Podemos evitar esses problemas, então, limitando o valor de um novo passo com base no anterior. Por exemplo:

$$0.1\alpha_k \leq \alpha_{k+1} \leq 0.9\alpha_k$$

E como escolher o valor do passo? Um jeito simples e inteligente é usando interpolação polinomial nos pontos anteriores, podendo aproveitar os valores da função e seu gradiente já calculados. Neste trabalho foram utilizadas a interpolação quadrática e a cúbica para comparação.

Condição da Norma

Ainda assim, como o avanço em cada iteração

$$x_{k+1} - x_k = \alpha_k d^k$$

depende tanto do passo quanto da direção, valem as mesmas observações do item anterior sobre a norma de d^k , portanto também queremos restringir ela e podemos exigir que:

$$\|d^k\| \geq \sigma \|\nabla f(x^k)\|$$

para algum $\sigma > 0$.

Assim, limitamos inferiormente o avanço e mesmo assim permitindo que este seja suficientemente pequeno perto do ponto estacionário (∇f pequeno).

¹É importante que esse passo candidato seja $\alpha = 1$ para os métodos de Newton e Quasi-Newton, pois para pontos suficientemente próximos da solução ele é sempre aceito e isso que dá origem a convergência quadrática e superlinear dos métodos, respectivamente.

Condição do Ângulo

Outro problema é a própria direção de busca. Analogamente ao tamanho do passo, apenas exigir que ela seja direção de descida não é suficiente, pois ela pode ser arbitrariamente próxima à ortogonalidade com o gradiente, apontado “para o lado”, ao invés de para o ponto estacionário, fazendo com que o avanço na direção deste seja desprezível.

Então podemos simplesmente exigir que d faça um ângulo máximo com o sentido de $-\nabla f$, isto é (através da relação entre cosseno e produto interno):

$$\nabla f(x)^\top d \leq -\theta \|\nabla f(x)\| \|d\|$$

para algum $0 < \theta \leq 1$.

Métodos

Como visto, a busca linear é um paradigma razoavelmente simples, mas como escolhemos a direção de busca? Há várias estratégias para isso e é isso que diferencia os diversos métodos existentes.

Método do Gradiente

Talvez o método mais simples, consiste em se escolher a direção de busca como sendo no sentido oposto ao gradiente:

$$d = -\nabla f(x)$$

Então, ao menos localmente, d aponta para onde a função mais decresce e por isso este método também é conhecido como *máxima descida*.

É fácil verificar que esta direção satisfaz as condições de norma (com $\sigma \leq 1$) e ângulo, então só é necessário verificar Armijo e o tamanho do passo.

Esta simplicidade, entretanto, tem suas desvantagens: apesar de ter convergência global (para qualquer ponto inicial) garantida, por sua direção ser sempre de descida, a taxa dessa é apenas linear. Pior ainda, problemas com diferenças de escala muito grandes dificultam a convergência e o método pode se tornar ineficiente.

Método de Newton

Mais complexo que o método do gradiente, o método de Newton se baseia em pegar a direção que minimiza uma aproximação quadrática da função objetiva, isto é, os primeiros termos de expansão de f em série de Taylor:

$$f(x + d) = f(x) + \nabla f(x)^\top d + \frac{1}{2} d^\top \nabla^2 f(x) d$$

e é simples ver que tal direção é solução do sistema²

$$\nabla^2 f(x) d = -\nabla f(x)$$

Por utilizar informações de segunda ordem sobre a função, podemos esperar um melhor comportamento para este método, comparando com o anterior. De fato, o método de Newton não apresenta problemas com escalas e no geral sua convergência é quadrática, ao menos para pontos suficientemente próximos da solução.

Por outro lado, ele exige o cálculo da hessiana a cada iteração e ainda a resolução de um sistema linear, então não é um método barato. Além disso, a hessiana pode ser singular, já que não temos restrição nenhuma sobre ela, e logo não existir solução para o sistema, ou então esta pode não satisfazer a condição do ângulo³.

²Portanto o método de Newton exige $f \in C^2$.

³Pode não satisfazer a condição da norma também, mas isso exigiria apenas uma normalização e não o cálculo de uma nova direção.

Nesses casos, porém, podemos utilizar no sistema uma forma alterada da hessiana:

$$B = \nabla^2 f(x) + \rho I$$

com $\rho > 0$, o que é chamado de *shift* nos autovalores, pois cada autovalor da hessiana terá o valor ρ somado a ele.

Assim, para ρ suficientemente grande, todos os autovalores serão positivos e portanto B será definida positiva, o que já garante a existência de uma solução e que esta será uma direção de descida. Melhor ainda, note que quanto maior a constante, mais B se aproxima da identidade e portanto do método do gradiente, o qual sabemos que satisfaz a condição do ângulo.

Método BFGS (Quasi-Newton)

A fim de ser menos custoso que o método de Newton, mas mais esperto que o de máxima descida, o método BFGS⁴ (um dos métodos conhecidos como *Quasi-Newton*) é análogo ao método de secantes para se achar zero de funções, no sentido de que constrói uma aproximação da hessiana, de segunda ordem, a partir de diversas informações de primeira ordem.

Dessa forma, não é necessário o cálculo da hessiana a cada iteração, apenas uma aproximação, e ainda podemos exigir que tal aproximação a cada ponto seja uma atualização da anterior, exigindo menos cálculos ainda. Mais ainda, podemos aproximar diretamente a *inversa* da hessiana e eliminar de vez a necessidade de se resolver um sistema linear.

Essas exigências, mais o fato de que essas aproximações sejam definidas positivas, determinam o método BFGS, que, embora não seja simples mostrar, consiste em calcular, a cada iteração

$$H_{k+1} = H_k + \left(\frac{1 + q_k^\top H_k q_k}{p_k^\top q_k} \right) \frac{p_k p_k^\top}{p_k^\top q_k} - \frac{p_k q_k^\top H_k + H_k q_k p_k^\top}{p_k^\top q_k}$$

em que⁵

$$\begin{aligned} p_k &= x^{k+1} - x^k \\ q_k &= \nabla f(x^{k+1}) - \nabla f(x^k) \end{aligned}$$

e então, na próxima iteração, calcular

$$d_k = -H_k \nabla f(x^k)$$

Como H_k é dada por uma fórmula iterativa, é necessário definir um valor inicial H_0 , contanto que seja uma matriz definida positiva, para manter as propriedades desejadas. Escolhas comuns são a matriz identidade (simples, mas nenhuma relação com a hessiana) e a própria hessiana (cujo cálculo pode ser caro).

Como no método de Newton, a direção de busca calculada pode não satisfazer a condição do ângulo e poderíamos lidar com isso similarmente. Mas, como temos liberdade de se afastar da hessiana, podemos utilizar a matriz identidade como novo H e repetir o cálculo de d , o que efetivamente é o método do gradiente e portanto satisfaz a condição do ângulo.

Devido a isso e à simplicidade, a identidade é uma escolha razoável para H_0 .

Novamente de forma análoga ao método das secantes, a convergência do BFGS fica entre os métodos de primeira e segunda ordem, isto é, é superlinear.

Vale notar também que essa fórmula para H_{k+1} nos permite a calcular sem o uso de memória auxiliar, já que ela contém escalares ($q_k^\top H_k q_k$, $p_k^\top q_k$), produtos externos ($p_k p_k^\top$, $p_k q_k^\top$, $q_k p_k^\top$) e seus produtos com H_k , que são simples de descrever. A dificuldade se encontra na ordem em que os elementos de H_{k+1} devem ser gravados sobre H_k , já que cada elemento daquela depende de todos desta.

No entanto, observando que a matriz é sempre simétrica (pela fórmula), ou seja, a parte triangular inferior é a transposta da superior, podemos sobrescrever primeiro a parte triangular inferior de H_k (sem a diagonal principal), tomando cuidado para ler somente da triangular superior. Depois, podemos sobrescrever a diagonal principal, novamente lendo somente da parte triangular superior e finalmente basta copiar os valores calculados na triangular inferior para a superior, obtendo a matriz H_{k+1} completa.

⁴Broyden–Fletcher–Goldfarb–Shanno.

⁵Vale notar que H_k também deve satisfazer $H_k q_j = p_j$ para todo $j \leq k$, que, não por coincidência, é a chamada *equação secante*.

O Programa

Para implementar os métodos foi feito um programa em FORTRAN 2008 que testa o conjunto de 18 problemas implementados em [Moré, Garbow, and Hillstrom, 1981].

Módulos

O programa em si é dividido em vários módulos, a saber:

mgh Faz a interface com os códigos em F77 de [Moré, Garbow, and Hillstrom, 1981], que fornecem as subrotinas (e o que calculam, para cada um dos 18 problemas):

INITPT x^0
OBJFCN $f(x)$
GRDFCN $\nabla f(x)$
HESFCN $\nabla^2 f(x)$

ls Contém os algoritmos que realizam a escolha do passo α através da busca linear por interpolação quadrática (**lsquad**) e cúbica (**lscube**).

methods Implementa os métodos (**grad**, **newt**, **bfgs**) de fato.

cholesky, **trisys**, **utils** Conjunto de módulos para a decomposição de matrizes em fatores de Cholesky e a resolução de sistemas lineares que as contenham.

stats Fornece *wrappers* e subrotinas para automatizar a contagem e a impressão de diversas etapas durante a execução dos algoritmos.

Arquivos

No diretório fonte ficam os arquivos:

EP1.f08 Código principal, responsável por juntar os módulos e iniciar a execução.

EP1 Executável, criado com o comando **make**.

EP1.pdf Relatório, criado com o comando **make tex**.

Makefile

Os demais arquivos são distribuídos nos seguintes diretórios:

src/ Contém os respectivos arquivos de cada módulo (**.f08**), bem como os das subrotinas MGH (**.f**).

obj/ Criado durante a compilação, guarda os arquivos objetos (**.o**) e módulos (**.mod**). É excluído com o comando **make clean**.

tex/ Contém o código fonte deste relatório (**.tex**) e a bibliografia (**.bib**).

aux/ Criado durante a compilação, guarda os arquivos auxiliares do L^AT_EX e do BibT_EX. É excluído com o comando **make clean**.

Resultados

Conclusão

Referências

J. J. Moré, Burton S. Garbow, and Kenneth E. Hillstom. Algorithm 566: Fortran subroutines for testing unconstrained optimization software [c5], [e4]. *ACM Trans. Math. Softw.*, 7(1):136–140, March 1981. ISSN 0098-3500. doi: 10.1145/355934.355943. URL <http://doi.acm.org/10.1145/355934.355943>.