Programação Não-Linear
Algoritmos de Busca Linear

Lucas Magno
7994983

Introdução

Seja uma pessoa querendo controlar gastos ou um cientista buscando o estado de menor energia de um sistema, a miniminização (e seu equivalente, maximimização) é um problema ubíquo e não é surpreendente que pessoas buscassem desenvolver técnicas sofisticadas para sua resolução.

Essa classe de problemas, conhecida como otimização, apresenta diversas abordagens diferentes e neste trabalho será explorada uma delas: a otimização contínua irrestrita. Isto é, focaremos na minimização de funções contínuas, sem se preocupar em que região do seu domínio as soluções podem estar.

Mais especificadamente, exploraremos o paradigma de busca linear, um dos mais conhecidos juntamente com o de região de confiança (que não será tratado aqui). Com isso se espera determinar a eficiência dos algoritmos mais conhecidos e em que contexto eles são melhor aplicados.

Otimização Irrestrita

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

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

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 [Nocedal and Wright, 2006] 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

$$\nabla f(x^k)^\top d^k < 0 \tag{1}$$

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 descrescer 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

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

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

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

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 $\gamma = 10^{-4}$)

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

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

Além disso, sempre existe um α suficientemente pequeno para o qual Armijo vale, o que sugere a seguinte técnica, conhecida como *backtracking* para determinar um passo satisfatório: partindo de um passo candidato[1], 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 diminuirem 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 $\sigma > 0$.

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

[1] É 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 tal 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 descresce, o que dá ao método o nome de *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 sistema[2]

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

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 ângulo[3].

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.

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

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

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 BFGS[4] (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 [Friedlander, 1994] [5], consiste em calcular, a cada iteração

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

em que[6]

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

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 de 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.

[4] Broyden–Fletcher–Goldfarb–Shanno.

[5] Embora a referência tenha sido o livro da Ana, na Wikipédia (BFGS) encontrei essa expressão com o fator $p_k^\top q_k$ ao invés da constante 1 e, na prática, o algoritmo se mostrou muito mais eficiente nesse caso, então escolhi essa fórmula.

[6] Vale 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 divido em vários módulos, a saber:

mgh Faz a interface com os códigos em `F77` de MGH, 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 durante a compilação.

`EP1.pdf` Relatório, criado durante a compilação.

`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 na limpeza.

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

`aux/` Criado durante a compilação, guarda os arquivos auxiliares do LaTeXe do `BibTeX`. É excluído na limpeza.

Compilação

A partir do diretório raiz, os seguintes comandos são possíveis:

make Compila o programa, criando o executável.

make tex Compila o relatório, criando o arquivo `.pdf`.

make clean Realiza a limpeza, deletando os arquivos de saída.

Execução

Ao executar o programa, para cada um dos problemas, os três métodos apresentados serão testados (e tanto com interpolação quadrática, quanto cúbica). Então, para cada problema, 6 processos diferentes são criados e rodados em paralelo, através da chamada para o sistema `fork()` e o processo principal aguarda o término destes antes de passar para o próximo problema.

O critério de parada escolhido para cada método foi a norma[7] do gradiente final, ou seja, fornecido um $\epsilon > 0$, o programa para quando

$$\|\nabla f(x^k)\| < \epsilon$$

já que desejamos que essa norma seja nula.

Cada um dos processos, ao terminar, imprime para a tela um resumo dos dados do algorítmo, como norma final do gradiente, número de iterações, chamadas à função objetiva, etc. Assim, os resultados de todos os métodos ficam organizados por problema.

Como exemplo de saída, podemos olhar o seguinte problema:

1 Helical Valley	$\|\nabla f_*\|$	it	f	∇f	$\nabla^2 f$	armijo	norma	ângulo
newtcube	0.11E-05	12	25	13	12	1	4	50
bfgscube	0.11E-05	74	263	61	0	143	0	14
bfgsquad	0.58E-05	93	327	79	0	171	0	15
newtquad	0.11E-05	12	25	13	12	1	4	50
gradquad	0.98E-05	2296	4638	2297	0	46	0	0
gradcube	0.98E-05	2287	4618	2288	0	44	0	0

Tabela 1: Saída do programa para o primeiro problema de Moré-Garbow-Hillstrom (*Helical Valley*), com $\epsilon = \theta = 10^{-5}$, $\gamma = \sigma = 10^{-4}$.

em que cada coluna representa:

- $\|\nabla f_*\|$ A norma do gradiente no último ponto calculado (solução).
- **it** O número de iterações do algorítmo.
- f O número de chamadas da função objetiva realizadas.
- ∇f O número de chamadas do gradiente realizadas.
- $\nabla^2 f$ O número de chamadas da hessiana realizadas.
- **armijo** O número de vezes em que a condição de Armijo não foi satisfeita.
- **norma** O número de vezes em que a condição da norma não foi satisfeita.
- **ângulo** O número de vezes em que a condição do ângulo não foi satisfeita.

Além disso, como a execução é em paralelo, não existe uma ordem fixa entre os métodos, isto é, cada um imprime seus resultados assim que terminar. Consequentemente, a ordem final em que os métodos aparecem pode indicar o tempo gasto por cada um. No entanto, questões como o *buffer* para a saída podem alterar essa ordem, então ela não deve ser tomada literalmente e por isso daqui em diante eles serão apresentados em uma ordem fixa, para facilitar a leitura.

Assim, indicadores melhores do gasto de cada algorítmo são os contadores. O tempo de execução de fato não foi registrado pois esse se mostrou muito abaixo da precisão disponível (milissegundo), não sendo útil para comparação.

[7] Todas as normas utilizadas nesse trabalho são euclidianas.

Resultados

Para começar com um exemplo simples, tomemos o parabolóide dado por

$$f(x) = x_1^2 + 1000x_2^2$$

e cujo mínimo é

$$f(0,0) = 0$$

e o ponto inicial

$$x^0 = \begin{bmatrix} 100 \\ 100 \end{bmatrix}$$

Rodando essa função no programa, temos os resultados

Paraboloid	$\|\nabla f_*\|$	it	f	∇f	$\nabla^2 f$	armijo	norma	ângulo
gradquad	0.63E-05	7	36	8	0	22	0	0
gradcube	0.99E-05	5926	11907	5927	0	55	0	0
bfgsquad	0.19E-10	2	9	3	0	5	0	0
bfgscube	0.16E-07	7	18	8	0	4	0	0
newtquad	0.57E-13	1	2	2	1	0	0	0
newtcube	0.57E-13	1	2	2	1	0	0	0

Tabela 2: Saída do programa para o parabolóide, com $\epsilon = \theta = 10^{-5}$, $\gamma = \sigma = 10^{-4}$.

em que fica claro que todos os métodos convergiram para um ponto estacionário (dada a precisão exigida).

Mas é interessante notar que o método Newton converge em apenas uma iteração, enquanto que os outros não.

Há diversos fatores que levam a isso, mas os principais são:

1. A função é quadrática, ou seja, a aproximação que Newton faz utilizando a hessiana acaba sendo a própria função, e então o método é capaz de escolher a direção que de fato aponta para a solução, permitindo a convergência instantânea.

2. O BFGS fica logo atrás, mas como inicialmente ele ainda não tem informação suficiente para aproximar a hessiana, ele não consegue atingir diretamente a solução. Entretanto, ele não demora muito para tal. Nota-se também que a interpolação quadrática é mais eficiente, justamente pela função também o ser.

3. O método do gradiente, entretanto, apesar de ter uma boa eficiência com a interpolação quadrática (que faz grande parte do trabalho em minimizar a função), não se dá muito bem com a interpolação cúbica. Isso se dá em grande parte devido à diferença de escalas do problema (o fator 1000 em x_2), com a qual esse algorítmo não lida muito bem. E, uma vez removida a contribuição da interpolação, isso se torna bem evidente.

Para mostrar mais claramente a contribuição da diferença de escala para o problema, tomando o parabolóide regular:

Regular Paraboloid	$\|\nabla f_*\|$	it	f	∇f	$\nabla^2 f$	armijo	norma	ângulo
gradquad	0.00E+00	1	3	2	0	1	0	0
gradcube	0.00E+00	1	3	2	0	1	0	0
bfgsquad	0.00E+00	1	3	2	0	1	0	0
bfgscube	0.00E+00	1	3	2	0	1	0	0
newtquad	0.80E-13	1	2	2	1	0	0	0
newtcube	0.80E-13	1	2	2	1	0	0	0

Tabela 3: Saída do programa para o parabolóide regular, com $\epsilon = \theta = 10^{-5}$, $\gamma = \sigma = 10^{-4}$.

ou seja, a interpolação sozinha é capaz de minimizar a função, pois a direção de busca inicial de todos já aponta para a solução.

Outro exemplo interessante é o seguinte:

4 Powell Badly Scaled	$\|\nabla f_*\|$	it	f	∇f	$\nabla^2 f$	armijo	norma	ângulo
gradquad	FC							
gradcube	FC							
bfgsquad	0.60E-05	122	530	93	0	346	6	30
bfgscube	0.91E-05	309	1412	229	0	956	15	81
newtquad	0.94E-05	297	979	298	297	385	169	452
newtcube	0.94E-05	297	979	298	297	385	169	452

Tabela 4: Saída do programa para o problema 4 de Moré-Garbow-Hillstrom (*Powell Badly Scaled*) , com $\epsilon = \theta = 10^{-5}$, $\gamma = \sigma = 10^{-4}$.

cuja análise não é tão simples, mas o interessante é que o método do gradiente não convergiu (FC, *failure to converge*, indica que o algorítimo não conseguiu convergir em menos de 1.000.000 avaliações da função objetiva).

Novamente, como o nome sugere, isso se dá pelas diferenças de escala do problema. Vale também notar que o método de Newton apresenta grandes problemas com a condição do ângulo, pois o melhor que podemos fazer para corrigir isso é criar uma aproximação da hessiana que "tende" à identidade (que nos daria a direção de máxima descida), enquanto que o BFGS tem a liberdade de a tomar imediatamente.

Convergência

Analisando agora a função de Rosenbrock:

$$f(x) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$$

com os resultados

Rosenbrock	$\|\nabla f_*\|$	it	f	∇f	$\nabla^2 f$	armijo	norma	ângulo
gradquad	0.99E-05	10668	21373	10669	0	37	0	0
gradcube	0.99E-05	10526	21093	10527	0	41	0	0
bfgsquad	0.79E-05	103	299	76	0	149	0	28
bfgscube	0.67E-05	32	151	27	0	99	0	6
newtquad	0.40E-05	11	23	12	11	1	0	13
newtcube	0.40E-05	11	23	12	11	1	0	13

Tabela 5: Saída do programa para a função de Rosenbrock , com $\epsilon = \theta = 10^{-5}$, $\gamma = \sigma = 10^{-4}$.

e cujos 11 últimos valores[8] de $\|\nabla f(x)\|$ para todos os algorítmos com interpolação quadrática são:

Gradiente	BFGS	Newton
0.0000120	0.0107588	447.2135955
0.0000280	0.0094878	176.8260698
0.0000103	0.0094878	15.9468161
0.0000168	0.0003073	3.0784906
0.0000230	0.0001685	5.2304703
0.0000123	0.0027361	4.5961242
0.0000287	0.0015423	1.9377240
0.0000105	0.0015423	1.7015645
0.0000214	0.0000150	0.3193714
0.0000146	0.0003777	0.1137346
0.0000271	0.0003777	0.0017722
0.0000099	0.0000079	0.0000040

[8]No caso, para o método de Newton, são todos os valores da norma do gradiente, já que ele teve somente 11 iterações.

podemos observar algumas coisas:

1. O método do gradiente não avança muito, apenas flutua um pouco o valor, eventualmente chegando na precisão exigida. Mas, como demonstrado pelo seu número de iterações, demora muito.
2. O BFGS já é melhor, começando com um valor três ordens de grandeza maior e ainda assim atingindo uma precisão melhor no mesmo número de iterações. Isso sem contar com os valores duplicados, que são devido à condição do ângulo, a qual força o algoritmo a repetir a iteração com a matriz identidade. Mesmo assim, os valores variam bastante, então é difícil dizer qual sua taxa de convergência.
3. Já o método de Newton apresenta de longe a melhor convergência, ainda mais nas últimas três iterações, nas quais a ordem de grandeza dos valores cai quase quadraticamente (10^{-1}, 10^{-3}, 10^{-5}), que era o esperado.

Entretanto, nem sempre se observam essas diferenças, como no seguinte problema:

8 Penalty I	$\|\nabla f_*\|$	it	f	∇f	$\nabla^2 f$	armijo	norma	ângulo
gradquad	0.89E-05	7	26	8	0	12	0	0
gradcube	0.90E-05	8	33	9	0	17	0	0
bfgsquad	0.90E-05	10	22	11	0	2	0	0
bfgscube	0.90E-05	10	22	11	0	2	0	0
newtquad	0.17E-05	33	78	34	33	12	0	0
newtcube	0.17E-05	33	78	34	33	12	0	0

Tabela 6: Saída do programa para o problema 11 de Moré-Garbow-Hillstrom (*Penalty I*), com $\epsilon = \theta = 10^{-5}$, $\gamma = \sigma = 10^{-4}$.

no qual, curiosamente, a ordem dos métodos por número de iterações é o contrário do que se esperaria!

Além disso, houve também casos patológicos:

11 Brown and Dennis	$\|\nabla f_*\|$
gradquad	FC
gradcube	FC
bfgsquad	NaN
bfgscube	NaN
newtquad	FC
newtcube	FC

Tabela 7: Saída do programa para o problema 11 de Moré-Garbow-Hillstrom (*Brown and Dennis*), com $\epsilon = \theta = 10^{-5}$, $\gamma = \sigma = 10^{-4}$.

em que `NaN` indica que o método convergiu para um ponto que não era estacionário (e portanto aparece um `NaN` durante o cálculo de H). No entanto, não foi econtrada nenhuma referência que explicasse os detalhes dessa função, nem o que levaria o método a isso.

Conclusão

Há uma grande liberdade em se adotar estratégias dentro do paradigma de busca linear, e no geral é necessário balancear simplicidade e eficiência. Mesmo assim, isso varia muito com o problema em questão, o que impossibilita a escolha do "melhor" método, já que um método supreendentemente pode se mostrar o mais eficaz naquele contexto.

Mas no geral há métodos que se mostram eficazes em boa parte dos problemas e acabam se tornando padrões, como o BFGS, que evita os problemas do método do gradiente e tem a vantagem de não exigir informações de segunda ordem do problema, como o de Newton.

Portanto, a escolha de um algorítmo às vezes se mostra por tentativa e erro, a não ser que o problema em questão tenha características em conhecidas que possam ser exploradas por algum método particular.

Referências

Ana Friedlander. *Elementos de Programação Não-Linear*. Editora UNICAMP, 1 edition, 1994.

J. J. Moré, Burton S. Garbow, and Kenneth E. Hillstrom. 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.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.