Matemática Numérica II
Problema de Março

Afonso Bandeira e Diana Lobo

25 de Março de 2009

Problema 1

(a) Seja $x \in \mathbb{R}$ um minimizante local de $f \in C^\infty$. Sabemos, por um teorema dos apontamentos, que $\nabla f(x) = 0$. Assim, tem-se

$$\nabla f(x) = 0 \Longrightarrow \left\{ \begin{array}{cc} -400x_1(x_2 - x_1^2) - 2(1 - x_1) & = 0 \\ 200(x_2 - x_1^2) & = 0 \end{array} \right. \Longrightarrow \left\{ \begin{array}{cc} x_1 - 1 & = 0 \\ x_2 - x_1^2 & = 0 \end{array} \right.$$

Logo, temos $[x_1 \ x_2]^T = [1 \ 1]^T = x_*$.

Verifiquemos agora que x_* é minimizante local de f, mostrando que a Hessiana $\nabla^2 f(x_*)$ é definida positiva.

Calculando, temos

$$\nabla^2 f(x_*) = \left[\begin{array}{cc} -400x_2 + 1200x_1 + 2 & -400x_1 \\ -400x_1 & 200 \end{array} \right]_{x_*} = \left[\begin{array}{cc} 802 & -400 \\ -400 & 200 \end{array} \right]$$

Usando o software *MATLAB* calculamos os valores próprios da matriz simétrica $\nabla^2 f(x_*)$, fazendo o INPUT: eig([802 -400; -400 200]).

Obtêm-se, assim os valores próprios $\left[\begin{array}{c} 0.399360767488 \\ 1001.600639232512 \end{array} \right]$, ambos positivos. Assim sendo, devido à simetria de $\nabla^2 f(x_*)$, esta matriz é simétrica definida positiva, ver [1]. Logo x_* é minimizante local de f, ver [3].

□

Nas próximas alíneas utilizamos os seguintes m-files:

function $b = grad(a)$
$b(1) = -400 * (a(2) - a(1)^2) * a(1) - 2 * (1 - a(1));$
$b(2) = 200 * (a(2) - a(1)^2);$

(que dá o gradiente da função no ponto $a = [a(1)\ a(2)]^T$)

function $z = BFGS(x, iter, eps)$
$k = 0;$
$syms\ v\ w;$
$B = inv(subs(jacobian(sym('[-400 * (w - v^2) * v - 2 * (1 - v); 200 * (w - v^2)]'), [vw]), vw, x);$
$display('iter \quad | \quad |||x - [11]^T|| \quad |');$
$disp(sprintf(' \%2.0f \%20.8f', k, norm(x - [1; 1])));$
$while norm(grad(x)) > eps$
$k = k + 1;$
$if\ k > iter$
$break$
end
$s = -B * grad(x)';$
$xx = x;$
$x = x + s;$
$y = grad(x) - grad(xx);$
$p = 1/(y * s);$
$B = (eye(2) - p * s * y) * B * (eye(2) - p * y' * s') + p * s * s';$
$disp(sprintf(' \%2.0f \%20.8f', k, norm(x - [1; 1])));$
end

(que implementa o método BFGS, ao gradiente de f, a partir do ponto x, com um máximo de iterações $iter$, um $\epsilon = eps$ e com $B_0 = (\nabla^2 f(x))^{-1}$)

$function z = newton03(z, iter, eps)$
$format\ long$
$syms\ xy$
$deriv = jacobian(sym('[-400*(y-x^2)*x-2*(1-x);200*(y-x^2)]'),[xy]);$
$display('iter \quad | \quad ||x-[11]^T|| \quad |');$
$disp(sprintf('\ \%2.0f\ \%20.8f', 0, norm(z-[1;1])));$
$for\ n = 1 : iter$
$if\ norm(grad(z)) < eps$
$break$
end
$D = (subs(deriv, xy, z));$
$F = (subs('[-400*(y-x^2)*x-2*(1-x);200*(y-x^2)]', x, y, z));$
$z = z - inv(D) * F;$
$disp(sprintf('\ \%2.0f\ \%20.8f', n, , norm(z-[1;1])));$
end

(que implementa o método de Newton para o gradiente da função de Rosenbrock, começando no ponto z, com um máximo de $iter$ iterações e um $\epsilon = eps$)

(b)

Fazendo umas pequenas alterações nos programas para, em vez de o output ser o erro a cada iteração, obtermos o valor de cada iteração de x_k, ficamos com os seguintes programas:

function $z = BFGS2(x, iter, eps)$
$k = 0;$
$syms\ v\ w;$

$B = inv(subs(jacobian(sym('[-400*(w-v^2)*v-2*(1-v);200*(w-v^2)]'),[vw]),vw,x);$

$display('iter \quad | \quad x(1) \quad | \quad x(2)');$

$disp(sprintf(' \%2.0f \%20.8f \%20.8f', k, x(1), x(2)));$

$while norm(grad(x)) > eps$

$k = k + 1;$

$if\ k > iter$

$break$

end

$s = -B * grad(x)';$

$xx = x;$

$x = x + s;$

$y = grad(x) - grad(xx);$

$p = 1/(y * s);$

$B = (eye(2) - p * s * y) * B * (eye(2) - p * y' * s') + p * s * s';$

$disp(sprintf(' \%2.0f \%20.8f \%20.8f', k, x(1), x(2)));$

end

(referente ao método BFGS)

$function z = newton032(z, iter, eps)$

$format\ long$

$syms\ xy$

$deriv = jacobian(sym('[-400*(y-x^2)*x-2*(1-x);200*(y-x^2)]'),[xy]);$

$display('iter \quad | \quad x(1) \quad | \quad x(2)');$

$disp(sprintf(' \%2.0f \%20.8f \%20.8f', 0, z(1), Z(0)));$

$for\ n = 1 : iter$

$if\ norm(grad(z)) < eps$

$break$

end

$D = (subs(deriv, xy, z));$
$F = (subs('[-400 * (y - x^2) * x - 2 * (1 - x); 200 * (y - x^2)]', x, y, z));$
$z = z - inv(D) * F;$
$disp(sprintf(' \%2.0f \%20.8f \%20.8f', n, z(1), z(2)));$
end

(referente ao método de Newton)

Podemos então fazer correr estes programas para ver para que solução convergem estes métodos. Correndo em MATLAB,

INPUT:BFGS2($[-1.2; 1], 80, 0$);

OUPUT:

iter	x	y
0	-1.20000000	1.00000000
⋮	⋮	⋮
80	1.00000000	1.00000000

INPUT:newton032($[-1.2; 1], 10, 0$);

OUTPUT:

iter	x	y
0	-1.20000000	1
⋮	⋮	⋮
10	1.00000000	1

Logo convergem os dois métodos para o único mínimo local da função de Rosenbrock, o ponto $[1\ 1]^T$.

Para fazer as tabelas corremos em MATLAB os comandos (fazemos $eps = -1$ para não ser condição de paragem):

INPUT:

BFGS($[-1.2; 1], 40, -1$);

OUTPUT:

iter	$\|\|x - [1\ 1]^T\|\|$	iter	$\|\|x - [1\ 1]^T\|\|$	iter	$\|\|x - [1\ 1]^T\|\|$	iter	$\|\|x - [1\ 1]^T\|\|$
1	2.20833870	11	2.11122244	21	1.69570354	31	1.64804855
2	2.17489897	12	1.96353961	22	1.67877338	32	1.63137370
3	4.13697539	13	2.77621589	23	3.14300128	33	1.55438708
4	2.16985279	14	1.91048992	24	1.67809969	34	1.48520652
5	2.16598445	15	1.87281924	25	1.67743372	35	1.42875488
6	8.91888170	16	1.82871640	26	5.91680018	36	1.33181458
7	2.16520759	17	1.82995833	27	1.67764057	37	1.27251653
8	2.18433053	18	1.79802117	28	1.67765972	38	1.14156643
9	2.16470844	19	1.72310134	29	1.67550332	39	0.84395509
10	2.16395417	20	1.68641149	30	1.67077908	40	1.12736667

INPUT:

newton03($[-1.2; 1], 40, -1$);

OUTPUT:

iter	$\|\|x - [1\ 1]^T\|\|$
0	2.20000000
1	2.20833870
2	4.18174871
3	0.47958387
4	0.05597268
5	0.00000962
6	0.00000000
⋮	⋮
40	0.00000000

Vemos, como esperaríamos, que o método de Newton é muito mais rápido que o método BFGS. Isso deve-se a este último ser um método quasi-Newton, logo mais lento. No entanto, ao correr os programas, vê-se que o método de Newton é mais pesado computacionalmente (devido à necessidade do calculo de derivadas) do que o BFGS, pois cada iteração demora significativamente mais tempo.

(c)

Para fazer as tabelas corremos em MATLAB os comandos (fazemos $iter = 1000$ para não ser condição de paragem):

INPUT:

BFGS($[-1.2; 1], 1000, 10^{-5}$);

OUTPUT:

iter	$\|\|x - [1\ 1]^T\|\|$	iter	$\|\|x - [1\ 1]^T\|\|$	iter	$\|\|x - [1\ 1]^T\|\|$
0	2.20000000	6	8.91888170	71	0.04476032
1	2.20833870	7	2.16520759	72	0.02702233
2	2.17489897	8	2.18433053	73	0.01010889
3	4.13697539	9	2.16470844	74	0.00031596
4	2.16985279	⋮	⋮	75	0.00005399
5	2.16598445	70	0.04146544	76	0.00000003

INPUT:

newton03($[-1.2; 1], 1000, 10^{-5}$);

OUTPUT:

iter	$\|\|x - [1\ 1]^T\|\|$
0	2.20000000
1	2.20833870
2	4.18174871
3	0.47958387
4	0.05597268
5	0.00000962

Como já tínhamos verificado na alínea anterior, o método de Newton é muito mais rápido que o BFGS (o Newton precisa de 5 iterações e o BFGS de 76), no entanto verificamos aqui que são ambos muitos mais eficientes que o método da descida mais rápida que, segundo o enunciado, necessita de 5264 iterações.

Problema 2

Como A é simétrica garantimos, ver [2], a existência de matrizes U ortogonal e D diagonal tais que,

$$A = UDU^T$$

onde as entradas de D são os valores próprios de A (todos reais devido à simetria de A e positivos por A ser definida positiva).

Seja $D^{\frac{1}{2}}$ a matriz diagonal cujas entradas são as raízes quadradas das entradas de D (reais positivas pois D tem entradas positivas).

Definimos, então, $A^{\frac{1}{2}} := UD^{\frac{1}{2}}U^T$. Como $D^{\frac{1}{2}}$ é diagonal com entradas positivas, $A^{\frac{1}{2}}$ é simétrica definida positiva, ver [1]. Resta apenas verificar que $\left(A^{\frac{1}{2}}\right)^2 = A$. Temos,

$$\left(A^{\frac{1}{2}}\right)^2 = \left(UD^{\frac{1}{2}}U^T\right)^2 = U\left(D^{\frac{1}{2}}\right)^2 U^T = UDU^T = A$$

Mostrámos assim que toda a matriz simétrica definida positiva possui uma raiz quadrada simétrica definida positiva.

□

Problema 3

Como B é positiva definida os seus valores próprios são todos positivos. Sejam então $0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ os valores próprios de B. Temos que $tr(B) = \sum_{i=1}^n \lambda_i$ e $det(B) = \prod_{i=1}^n \lambda_i$, ver [2]. Queremos provar que

$$0 < \psi(B) = tr(B) - \ln(\det(B)) = \sum_{i=1}^n \lambda_i - \ln\left(\prod_{i=1}^n \lambda_i\right)$$

Ou equivalentemente, como e^x é monótona crescente,

$$\prod_{i=1}^n \lambda_i < e^{\left(\sum_{i=1}^n \lambda_i\right)} = \prod_{i=1}^n e^{\lambda_i}$$

Como para $x \in \mathbb{R}^+$, $e^x > x$, tem-se, consequentemente, $\forall_{1 \leq i \leq n}$,

$$\lambda_i < e^{\lambda_i}$$

Logo,

$$\prod_{i=1}^{n} \lambda_i < \prod_{i=1}^{n} e^{\lambda_i}$$

□

Referências

[1] Júdice, J.J. *Texto de Apoio às Aulas de Álgebra Linear Numérica.*

[2] Santana, A.P. , Queiró, J.F. *Álgebra Linear e Geometria Analítica.*

[3] Vicente, L.N. *Matemática Numérica II.*