IF824 - Otimização
Algoritmo de Newton/Quase-Newton

Gurvan Huiban

13 de maio de 2014

Os trabalhos são realizados na ferramenta Matlab[1] ou na ferramenta Sage[2].
O contexto deste trabalho é o mesmo que do trabalho sobre o algoritmo do gradiente.

1 Descrição do método

Seja o problema de otimização irrestrito seguinte:

$$\begin{array}{ll} \min & f(X) \\ & |X \in \Omega \end{array} \tag{1}$$

com

$$\Omega = \mathbb{R}^n \text{ e } f : \mathbb{R}^n \to \mathbb{R} \text{ diferenciável duas vezes}$$

2 Algoritmo de Newton

Dado um ponto $X_0 \in \mathbb{R}^n$, o algoritmo de Newton (algoritmo 1) vai gerar uma sequência de pontos (X_k) que vai convergir para um mínimo local X^*, caso X_0 seja perto suficiente de X^*.

Data: Função f, ponto inicial X_0
Result: Aproximação numérica de um mínimo local de f
$k \leftarrow 0$;
repeat
  $X_{k+1} \leftarrow X_k - \left[\nabla^2 f(X_k)\right]^{-1} \nabla f(X_k)$
until *Condição de parada*;

Algorithm 1: Algoritmo de Newton

∇f (respectivamente $\nabla^2 f$) representa o gradiente (respectivamente a hessiana) da função f.

3 Algoritmos Quase-Newton

O algoritmo de Newton necessita o cálculo da inversa da hessiana da função f em cada iteração. Este cálculo pode ser custoso é pode ser fonte de instabilidade numéricas.

Os algoritmos Quase-Newton aproximam a inversa da Hessiana com uma sequência de matrizes (S_k), onde S_k é calculada da forma seguinte:

- $S_0 \leftarrow I$
- $S_{k+1} = S_k + C_k$, onde C_k é calculado a partir das fórmulas,

$$\text{BFGS:} \quad C_k = \left(1 + \frac{r_k^T S_k r_k}{v_k^T r_k}\right) \frac{v_k v_k^T}{v_k^T r_k} - \frac{v_k r_k^T S_k + S_k r_k v_k^T}{v_k^T r_k} \tag{2}$$

$$\text{DFP:} \quad C_k = \frac{v_k v_k^T}{v_k^T r_k} - \frac{S_k r_k r_k^T S_k}{r_k^T S_k r_k} \tag{3}$$

[1] ©1994-2013 The MathWorks, Inc.
[2] http://www.sagemath.org/

com $v_k = X_{k+1} - X_k$ e $r_k = \nabla f(X_{k+1}) - \nabla f(X_k)$

Assim podemos definir um algoritmo de descida usando uma estratégia de busca na direção definida por $-S_k \nabla f(X_k)$ (ver algoritmo 2).

Data: Função f, ponto inicial X_0
Result: Aproximação numérica de um mínimo local de f
$k \leftarrow 0$;
Calcular $\nabla f(X_0)$;
$S_0 \leftarrow I$;
repeat
  Calcular $d_k = -S_k \nabla f(X_k)$;
  Resolver o problema de otimização unidimensional: $\alpha_k = \arg\min_{\alpha \geqslant 0} f\left(X_k + \alpha_k d_k\right)$;
  $X_{k+1} \leftarrow X_k + \alpha_k d_k$;
  Calcular $\nabla f(X_{k+1})$;
  Calcular S_{k+1} usando uma das fórmulas BFGS ou DFP;
  $k \leftarrow k + 1$;
until *Condição de parada*;

Algorithm 2: Algoritmo de descida Quase-Newton

4 Implementação

O objetivo do trabalho é a implementação com Matlab ou Sage (sua preferência) do algoritmo de Newton e do algoritmo Quase-Newton BFGS. Nós podemos reusar os mesmos métodos de otimização unidimensional e de busca de intervalo que no trabalho sobre o algoritmo do gradiente. Podemos também usar o mesmo critério de parada.

4.1 Problemas sugeridos

Seguem abaixo alguns problemas que podem ser usados para testar a implementação do algoritmo. Se possível, representar graficamente a função e a solução encontrada pelo algoritmo.

- $f : (x_1, x_2) \mapsto x_1^2 + x_2^2 - 8x_1 + 5x_2 + 22.25$
- $f : (x_1, x_2) \mapsto 10x_1^2 + x_2^2 - 8x_1 + 5x_2 + 22.25$
- $f : (x_1, x_2) \mapsto 9x_1^2 - 3x_1x_2 + 1.25x_2^2 - 24x_1 + 9x_2 + 22.25$
- $f : (x_1, x_2) \mapsto 0.5x_1^2 + 0.25x_2^2 + 20\sin(0.1x_1x_2) + 0.25$

4.2 Comparações

Uma vez os algoritmos implementados, compare:

- o algoritmo do gradiente e o algoritmo de Newton (quantidade de iteração, velocidade de convergência, ...).
- a matriz S_k calculada pelo algoritmo Quase-Newton com o valor de $\left[\nabla^2 f(X_k)\right]^{-1}$.