LIP-BFGS Theory

Jesse Lu

November 11, 2011

Contents

1 Introduction

The acronym LIP-BFGS stands for Limited-memory Interior-Point Broyden-Fletcher-Goldfarb-Shanno. It is simply an interior-point (IP) method which uses the limited-memory BFGS (L-BFGS) algorithm. The main body of the algorithm is described in Chapter 19.3 of [1].

The purpose of this document is to allow the user to understand the accompanying Matlab implementation.

Part I
Theory

2 Theory for the interior-point method

Interior-point methods attempt to minimize $f(x)$ subject to the equality and inequality constraints $c_E(x)$ and $c_I(x)$,

$$\begin{aligned} \text{minimize} \quad & f(x) && \text{(1a)} \\ \text{subject to} \quad & c_E(x) = 0 && \text{(1b)} \\ & c_I(x) \geq 0, && \text{(1c)} \end{aligned}$$

by satisfying the Karush-Kuhn-Tucker (KKT) conditions (see Chapter 12.3 of [1])

$$\begin{aligned} \nabla f(x) - A_E^T(x)y - A_I^T(x)z &= 0 && \text{(2a)} \\ z - \mu s^{-1} &= 0 && \text{(2b)} \\ c_E(x) &= 0 && \text{(2c)} \\ c_I(x) - s &= 0 && \text{(2d)} \\ s &\geq 0 && \text{(2e)} \\ z &\geq 0, && \text{(2f)} \end{aligned}$$

where

$$\begin{aligned} A_E(x) &= \nabla c_E(x), \quad \text{the Jacobian of } c_E(x) && \text{(3a)} \\ A_I(x) &= \nabla c_I(x), \quad \text{the Jacobian of } c_I(x) && \text{(3b)} \end{aligned}$$

y and z are the dual variables for $c_E(x)$ and $c_I(x)$ respectively, and s is the slack variable. Note that the expression s^{-1} refers to the element-wise inverse of the vector s. Also, the expression in (2a) can also be written as $\nabla_x \mathcal{L} = 0$, where $\mathcal{L}$ is the Lagrangian of the problem.

As taken from Chapter 19.3 of [1], the interior point method obtains step direction p by solving

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L} & 0 & A_E^T(x) & A_I^T(x) \\ 0 & \Sigma & 0 & -I \\ A_E(x) & 0 & 0 & 0 \\ A_I(x) & -I & 0 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_s \\ -p_y \\ -p_z \end{bmatrix} = - \begin{bmatrix} \nabla f(x) - A_E^T(x)y - A_I^T(x)z \\ z - \mu s^{-1} \\ c_E(x) \\ c_I(x) - s \end{bmatrix}. \tag{4}$$

where

$$\Sigma = \text{diag}(z/s). \tag{5}$$

This equation can be simplified by first backsubstituting for p_s and then for p_z.

The reduced system is then

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L} + A_I^T(x)\Sigma A_I^T(x) & A_E^T(x) \\ A_E(x) & 0 \end{bmatrix} \begin{bmatrix} p_x \\ -p_y \end{bmatrix} = - \begin{bmatrix} \nabla f(x) - A_E^T(x)y - A_I(x)h \\ c_E(x) \end{bmatrix}, \tag{6}$$

where

$$h = z - \Sigma c_I(x) + \mu s^{-1} \tag{7}$$

and

$$p_s = A_I(x)p_x + c_I(x) - s \tag{8a}$$
$$p_z = -\Sigma A_I(x)p_x - \Sigma c_I(x) + \mu s^{-1}. \tag{8b}$$

We now choose to consider only simple bound inequality constraints $l \leq x \leq u$, and affine equality constraints $Ax - b = 0$. Our problem can then be written down as

$$\begin{bmatrix} \nabla^2 f(x) + \Sigma_0 + \Sigma_1 & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} p_x \\ -p_y \end{bmatrix} = - \begin{bmatrix} \nabla f(x) - A^T y + h_0 + h_1 \\ Ax - b \end{bmatrix}, \tag{9}$$

where

$$\Sigma_0 = \mathrm{diag}(z_0/s_0) \tag{10a}$$
$$\Sigma_1 = \mathrm{diag}(z_1/s_1), \tag{10b}$$

and

$$h_0 = -z_0 + \Sigma_0(x - l) - \mu s_0^{-1} e \tag{11a}$$
$$h_1 = z_1 - \Sigma_1(u - x) + \mu s_1^{-1} e, \tag{11b}$$

and the other components of p are

$$p_{s_0} = p_x + (x - l) - s_0 \tag{12a}$$
$$p_{z_0} = -\Sigma_0 p_x - \Sigma_0(x - l) + \mu s_0^{-1} \tag{12b}$$
$$p_{s_1} = -p_x + (u - x) - s_1 \tag{12c}$$
$$p_{z_1} = \Sigma_1 p_x - \Sigma_1(u - x) + \mu s_1^{-1}. \tag{12d}$$

3 Theory for the limited-memory BFGS algorithm

Practically, computing and solving for $\nabla^2 f(x)$ in (9), the *Hessian* of $f(x)$, is often computationally challenging. For this reason, we use the limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm to approximate $\nabla^2 f(x)$. Specifically, we use the compact or outer-product representation of

$$B \sim \nabla^2 f(x), \tag{13}$$

as described in chapter 7.2 of [1], to efficiently solve for p in (9).

The BFGS algorithm works by approximating the Hessian of function based on a list of the previous values of x and $\nabla f(x)$. The approximate Hessian, B, is recursively updated by the following formula, taken from chapter 6.1 of [1],

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}, \tag{14}$$

where

$$s_k = x_{k+1} - x_k \tag{15a}$$
$$y_k = \nabla f(x_{k+1}) - \nabla f(x_k). \tag{15b}$$

The limited-memory BFGS algorithm simply truncates the list of (s_k, y_k) to the most recent m values, which allows us to store B efficiently in what is called the compact or outer-product representation (see chapter 7.2 of [1]):

$$B_k = B_0 - \begin{bmatrix} B_0 S_k & Y_k \end{bmatrix} \begin{bmatrix} S_k^T B_0 S_k & L_k \\ L_k^T & -D_k \end{bmatrix}^{-1} \begin{bmatrix} S_k^T B_0^T \\ Y_k^T \end{bmatrix}, \tag{16}$$

where B_0 is an initial guess for B,

$$S_k = [s_{k-m}, \ldots, s_{k-1}] \tag{17a}$$
$$Y_k = [y_{k-m}, \ldots, y_{k-1}] \tag{17b}$$

and

$$(L_k)_{i,j} = \begin{cases} s_{i-1}^T y_{j-1} & \text{if } i > j, \\ 0 & \text{otherwise,} \end{cases} \tag{18a}$$
$$D_k = \operatorname{diag}([s_{k-m}^T y_{k-m}, \ldots, s_{k-1}^T y_{k-1}]). \tag{18b}$$

Specifically, we choose

$$B_0 = \delta_k I, \tag{19}$$

where δ_k is a scaling variable, given by

$$\delta_k = \frac{y_{k-1}^T y_{k-1}}{s_{k-1}^T y_{k-1}}. \tag{20}$$

This results in a computationally-efficient diagonal-plus-low-rank structure for B_k,

$$B_k = \delta_k I + W_k M_k W_k^T \tag{21}$$

where

$$W_k = \begin{bmatrix} \delta_k S_k & Y_k \end{bmatrix} \tag{22a}$$
$$M_k = \begin{bmatrix} \delta_k S_k^T S_k & L_k \\ L_k^T & -D_k \end{bmatrix}^{-1}. \tag{22b}$$

Lastly, when $k = 0$ and there are no (s_k, y_k) pairs with which to construct B_k, we simply choose $B_0 = I$.

4 Efficiently solving an arrow-plus-low-rank system

Substituting the expression for B_k in (21) for $\nabla^2 f(x)$ in (9) yields

$$\left(\begin{bmatrix} \delta_k I + \Sigma_0 + \Sigma_1 & A^T \\ A & 0 \end{bmatrix} + \begin{bmatrix} WM \\ 0 \end{bmatrix} \begin{bmatrix} W^T & 0 \end{bmatrix}\right) \begin{bmatrix} p_x \\ -p_y \end{bmatrix} = - \begin{bmatrix} \nabla f(x) - A^T y + h_0 + h_1 \\ Ax - b \end{bmatrix}, \tag{23}$$

which can be efficiently solved by taking advantage of the structure of the matrix

$$\begin{bmatrix} \delta_k I + \Sigma_0 + \Sigma_1 & A^T \\ A & 0 \end{bmatrix} + \begin{bmatrix} WM \\ 0 \end{bmatrix} \begin{bmatrix} W^T & 0 \end{bmatrix}. \tag{24}$$

Such a matrix contains arrow-plus-low-rank structure; in the sense that the

$$\begin{bmatrix} \delta_k I + \Sigma_0 + \Sigma_1 & A^T \\ A & 0 \end{bmatrix} \tag{25}$$

term has “arrow” structure (pointing down and to the left) especially if A is fat ($A \in \mathbb{R}^{m \times n}, m \ll n$), and that the

$$\begin{bmatrix} WM \\ 0 \end{bmatrix} \begin{bmatrix} W^T & 0 \end{bmatrix}. \tag{26}$$

term is a low-rank matrix.

The arrow matrix can be efficiently solved via block substitution; meaning that we solve

$$\begin{bmatrix} \tilde{D} & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{27}$$

where

$$\tilde{D} = \delta_k I + \Sigma_0 + \Sigma_1 \tag{28}$$

by computing, in order,

$$A\tilde{D}^{-1}A^T x_2 = A\tilde{D}^{-1} b_1 - b_2 \tag{29a}$$

$$x_1 = \tilde{D}^{-1}(b_1 - A^T x_2). \tag{29b}$$

This is computationally efficient because the term $A\tilde{D}^{-1}A^T$ is small, if the number of rows in A is small, and therefore easy to invert.

Now that we can compute $\tilde{A}^{-1}b$, where $\tilde{A}$ is the arrow matrix in (25), we employ the matrix inversion lemma (also known as the Sherman-Woodbury-Morrison formula), which states

$$(A + UV^T)^{-1} b = A^{-1}b - A^{-1}U(I + V^T A^{-1} U)^{-1} V^T A^{-1} b, \tag{30}$$

in order to solve the entire arrow-plus-low-rank system in (24),

$$\tilde{A} + \tilde{U}\tilde{V}^T \tag{31}$$

where

$$\tilde{U} = \begin{bmatrix} WM \\ 0 \end{bmatrix} \tag{32a}$$

$$\tilde{V}^T = \begin{bmatrix} W^T & 0 \end{bmatrix}. \tag{32b}$$

Part II
Implementation

5 Outline of LIP-BFGS algorithm

LIP-BFGS requires the following input parameters:

- $\nabla f(x)$, function on $\mathbb{C}^n \rightarrow \mathbb{C}^n$ to evaluate gradient at x,
- $x \in \mathbb{C}^n$, initial value of optimization variable,
- $l, u \in \mathbb{R}^n$, lower and upper bounds on x, and
- $A \in \mathbb{C}^{m \times n}, b \in \mathbb{C}^m$, equality constraint on x.

The basic outline of the LIP-BFGS algorithm is:

1. Determine initial values of $s_{0,1}$, y, and $z_{0,1}$,
2. Check termination condition; if needed, update μ and perform steps 3-5,
3. Form or update B_k using (21),
4. Compute step-direction p by solving (23) and (12),
5. Perform a line-search to determine step-size along p, update x, $s_{0,1}$, y, and $z_{0,1}$.

6 Determining initial values of $s_{0,1}$, y, and $z_{0,1}$

7 Termination condition

The suggested termination condition from chapter 19.2 of [1] is used (with $\mu = 0$),

$$\text{if } E(x, s_0, s_1, y, z_0, z_1) \leq \epsilon_{\text{tol}} \text{ then terminate,} \tag{33}$$

where

$$E(x, s_0, s_1, y, z_0, z_1) = \max\{\|\nabla f(x) - A^T y - z_0 + z_1\|, \\ \|s_0 z_0\|, \|s_1 z_1\|, \|Ax - b\|, \|(x - l) - s_0\|, \|(u - x) - s_1\|\}, \quad (34)$$

and $s_0 z_0$, $s_1 z_1$ are element-wise vector products.

8 L-BFGS update of B_k

9 Computing step direction p

10 Line-search and variable update

Lastly, inspired from section 11.7.3 of [2], we perform a backtracking line search (see section 9.2 or [2]) in order to guarantee decrease of the residual $r(x^+, s_0^+, s_1^+, y^+, z_0^+, z_1^+, \mu)$ where,

$$x^+ = x + t\alpha_p p_x \quad (35a)$$
$$s_0^+ = s_0 + t\alpha_p p_{s_0} \quad (35b)$$
$$s_1^+ = s_1 + t\alpha_p p_{s_1} \quad (35c)$$
$$y^+ = y + \alpha_d p_y \quad (35d)$$
$$z_0^+ = z_0 + \alpha_d p_{z_0} \quad (35e)$$
$$z_1^+ = z_1 + \alpha_d p_{z_1} \quad (35f)$$

and,

$$r(x, s_0, s_1, y, z_0, z_1, \mu) = \left\| \begin{bmatrix} \nabla f(x) - A^T y + h_0 + h_1 \\ Ax - b \end{bmatrix} \right\|_2 . \quad (36)$$

The exit condition for the line search is

$$r(x^+, s_0^+, s_1^+, y^+, z_0^+, z_1^+, \mu) \leq (1 - \alpha t) r(x, s_0, s_1, y, z_0, z_1, \mu). \quad (37)$$

where t is initially set to $t = \alpha_p$.

References

[1] Nocedal and Wright, Numerical Optimization, Second Edition (Cambridge 2004)

[2] Boyd and Vandenberghe, Convex Optimization (Cambridge 2004)