Reduced-Hessian Methods for Constrained Optimization

Philip E. Gill

University of California, San Diego

Joint work with:
Michael Ferry & Elizabeth Wong

11th US & Mexico Workshop on Optimization and its Applications
Huatulco, Mexico, January 8–12, 2018.

Our honoree . . .

Outline

1. Reduced-Hessian Methods for Unconstrained Optimization
2. Bound-Constrained Optimization
3. Quasi-Wolfe Line Search
4. Reduced-Hessian Methods for Bound-Constrained Optimization
5. Some Numerical Results

Reduced-Hessian Methods for Unconstrained Optimization

Definitions

Minimize $f : \mathbb{R}^n \mapsto \mathbb{R} \in C^2$ with quasi-Newton line-search method:

Given x_k, let $f_k = f(x_k)$, $g_k = \nabla f(x_k)$, and $H_k \approx \nabla^2 f(x_k)$.

Choose p_k such that $x_k + p_k$ minimizes the quadratic model

$$q_k(x) = f_k + g_k^T(x - x_k) + \frac{1}{2}(x - x_k)^T H_k (x - x_k)$$

If H_k is positive definite then p_k satisfies

$$H_k p_k = -g_k \qquad \text{(qN step)}$$

Definitions

Define $x_{k+1} = x_k + \alpha_k p_k$ where α_k is obtained from line search on

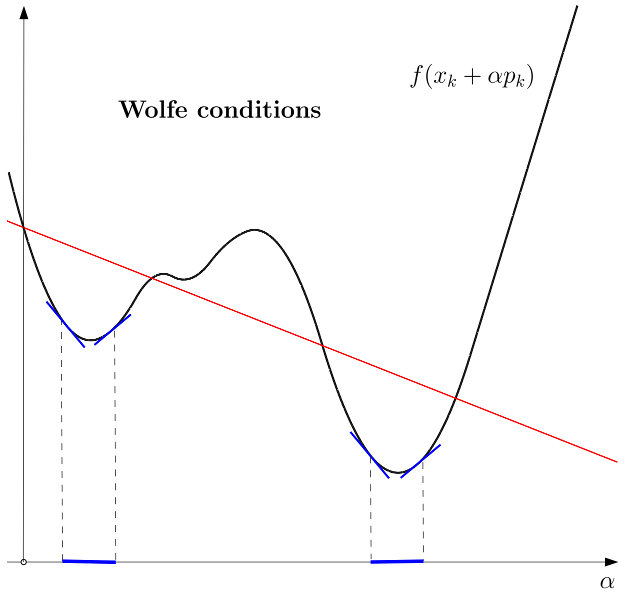
$$\phi_k(\alpha) = f(x_k + \alpha p_k)$$

- *Armijo condition*:

$$\phi_k(\alpha) < \phi_k(0) + \eta_A \alpha \phi_k'(0), \quad \eta_A \in (0, \tfrac{1}{2})$$

- *(strong) Wolfe conditions*:

$$\begin{aligned} \phi_k(\alpha) &< \phi_k(0) + \eta_A \alpha \phi_k'(0), & \eta_A &\in (0, \tfrac{1}{2}) \\ |\phi_k'(\alpha)| &\le \eta_W |\phi_k'(0)|, & \eta_W &\in (\eta_A, 1) \end{aligned}$$


Wolfe conditions
$f(x_k + \alpha p_k)$
α

Quasi-Newton Methods

Updating H_k:

- $H_0 = \sigma I_n$ where $\sigma > 0$
- Compute H_{k+1} as the BFGS update to H_k, i.e.,

$$H_{k+1} = H_k - \frac{1}{s_k^T H_k s_k} H_k s_k s_k^T H_k + \frac{1}{y_k^T s_k} y_k y_k^T,$$

  where $s_k = x_{k+1} - x_k$, $y_k = g_{k+1} - g_k$, and $y_k^T s_k$ approximates the curvature of f along p_k.

- Wolfe condition guarantees that H_k can be updated.

One option to calculate p_k:

- Store upper-triangular Cholesky factor R_k where $R_k^T R_k = H_k$

Quasi-Newton Methods

Updating H_k:

- $H_0 = \sigma I_n$ where $\sigma > 0$
- Compute H_{k+1} as the BFGS update to H_k, i.e.,

$$H_{k+1} = H_k - \frac{1}{s_k^T H_k s_k} H_k s_k s_k^T H_k + \frac{1}{y_k^T s_k} y_k y_k^T,$$

  where $s_k = x_{k+1} - x_k$, $y_k = g_{k+1} - g_k$, and $y_k^T s_k$ approximates the curvature of f along p_k.
- Wolfe condition guarantees that H_k can be updated.

One option to calculate p_k:

- Store upper-triangular Cholesky factor R_k where $R_k^T R_k = H_k$

Reduced-Hessian Methods

(Fenelon, 1981 and Siegel, 1992)

Let $\mathcal{G}_k = \text{span}(g_0, g_1, \ldots, g_k)$ and $\mathcal{G}_k^{\perp}$ be the orthogonal complement of $\mathcal{G}_k$ in $\mathbb{R}^n$.

Result

Consider a quasi-Newton method with BFGS update applied to a general nonlinear function. If $H_0 = \sigma I$ ($\sigma > 0$), then:

- $p_k \in \mathcal{G}_k$ for all k.
- If $z \in \mathcal{G}_k$ and $w \in \mathcal{G}_k^{\perp}$, then $H_k z \in \mathcal{G}_k$ and $H_k w = \sigma w$.

Reduced-Hessian Methods

(Fenelon, 1981 and Siegel, 1992)

Let $\mathcal{G}_k = \text{span}(g_0, g_1, \ldots, g_k)$ and $\mathcal{G}_k^\perp$ be the orthogonal complement of $\mathcal{G}_k$ in $\mathbb{R}^n$.

Result

Consider a quasi-Newton method with BFGS update applied to a general nonlinear function. If $H_0 = \sigma I$ ($\sigma > 0$), then:

- $p_k \in \mathcal{G}_k$ for all k.
- If $z \in \mathcal{G}_k$ and $w \in \mathcal{G}_k^\perp$, then $H_k z \in \mathcal{G}_k$ and $H_k w = \sigma w$.

Reduced-Hessian Methods

(Fenelon, 1981 and Siegel, 1992)

Let $\mathcal{G}_k = \text{span}(g_0, g_1, \ldots, g_k)$ and $\mathcal{G}_k^\perp$ be the orthogonal complement of $\mathcal{G}_k$ in $\mathbb{R}^n$.

Result

Consider a quasi-Newton method with BFGS update applied to a general nonlinear function. If $H_0 = \sigma I$ ($\sigma > 0$), then:

- $p_k \in \mathcal{G}_k$ for all k.
- If $z \in \mathcal{G}_k$ and $w \in \mathcal{G}_k^\perp$, then $H_k z \in \mathcal{G}_k$ and $H_k w = \sigma w$.

Reduced-Hessian Methods

(Fenelon, 1981 and Siegel, 1992)

Let $\mathcal{G}_k = \text{span}(g_0, g_1, \ldots, g_k)$ and $\mathcal{G}_k^{\perp}$ be the orthogonal complement of $\mathcal{G}_k$ in $\mathbb{R}^n$.

Result

Consider a quasi-Newton method with BFGS update applied to a general nonlinear function. If $H_0 = \sigma I$ ($\sigma > 0$), then:

- $p_k \in \mathcal{G}_k$ for all k.
- If $z \in \mathcal{G}_k$ and $w \in \mathcal{G}_k^{\perp}$, then $H_k z \in \mathcal{G}_k$ and $H_k w = \sigma w$.

Reduced-Hessian Methods

Significance of $p_k \in \mathcal{G}_k$:

- No need to minimize the quadratic model over the full space.
- Search directions lie in an expanding sequence of subspaces.

Significance of $H_k z \in \mathcal{G}_k$ and $H_k w = \sigma w$:

- Curvature stored in H_k along any unit vector in $\mathcal{G}_k^\perp$ is σ.
- All nontrivial curvature information in H_k can be stored in a smaller $r_k \times r_k$ matrix, where $r_k = \dim(\mathcal{G}_k)$.

Reduced-Hessian Methods

Significance of $p_k \in \mathcal{G}_k$:

- No need to minimize the quadratic model over the full space.
- Search directions lie in an expanding sequence of subspaces.

Significance of $H_k z \in \mathcal{G}_k$ and $H_k w = \sigma w$:

- Curvature stored in H_k along any unit vector in $\mathcal{G}_k^{\perp}$ is σ.
- All nontrivial curvature information in H_k can be stored in a smaller $r_k \times r_k$ matrix, where $r_k = \dim(\mathcal{G}_k)$.

Reduced-Hessian Methods

Given a matrix $B_k \in \mathbb{R}^{n \times r_k}$, whose columns span $\mathcal{G}_k$, let

- $B_k = Z_k T_k$ be the QR decomposition of B_k;
- W_k be a matrix whose orthonormal columns span $\mathcal{G}_k^{\perp}$;
- $Q_k = \begin{pmatrix} Z_k & W_k \end{pmatrix}$.

Then, $\quad H_k p_k = -g_k \Leftrightarrow (Q_k^T H_k Q_k) Q_k^T p_k = -Q_k^T g_k$, where

$$Q_k^T H_k Q_k = \begin{pmatrix} Z_k^T H_k Z_k & Z_k^T H_k W_k \\ W_k^T H_k Z_k & W_k^T H_k W_k \end{pmatrix} = \begin{pmatrix} Z_k^T H_k Z_k & 0 \\ 0 & \sigma I_{n-r_k} \end{pmatrix}$$

$$Q_k^T g_k = \begin{pmatrix} Z_k^T g_k \\ 0 \end{pmatrix}.$$

Reduced-Hessian Methods

Given a matrix $B_k \in \mathbb{R}^{n \times r_k}$, whose columns span $\mathcal{G}_k$, let

- $B_k = Z_k T_k$ be the QR decomposition of B_k;
- W_k be a matrix whose orthonormal columns span $\mathcal{G}_k^\perp$;
- $Q_k = \begin{pmatrix} Z_k & W_k \end{pmatrix}$.

Then, $\quad H_k p_k = -g_k \Leftrightarrow (Q_k^T H_k Q_k) Q_k^T p_k = -Q_k^T g_k$, where

$$Q_k^T H_k Q_k = \begin{pmatrix} Z_k^T H_k Z_k & Z_k^T H_k W_k \\ W_k^T H_k Z_k & W_k^T H_k W_k \end{pmatrix} = \begin{pmatrix} Z_k^T H_k Z_k & 0 \\ 0 & \sigma I_{n-r_k} \end{pmatrix}$$

$$Q_k^T g_k = \begin{pmatrix} Z_k^T g_k \\ 0 \end{pmatrix}.$$

Reduced-Hessian Methods

Given a matrix $B_k \in \mathbb{R}^{n \times r_k}$, whose columns span $\mathcal{G}_k$, let

- $B_k = Z_k T_k$ be the QR decomposition of B_k;
- W_k be a matrix whose orthonormal columns span $\mathcal{G}_k^{\perp}$;
- $Q_k = \begin{pmatrix} Z_k & W_k \end{pmatrix}$.

Then, $\quad H_k p_k = -g_k \Leftrightarrow (Q_k^T H_k Q_k) Q_k^T p_k = -Q_k^T g_k$, where

$$Q_k^T H_k Q_k = \begin{pmatrix} Z_k^T H_k Z_k & Z_k^T H_k W_k \\ W_k^T H_k Z_k & W_k^T H_k W_k \end{pmatrix} = \begin{pmatrix} Z_k^T H_k Z_k & 0 \\ 0 & \sigma I_{n-r_k} \end{pmatrix}$$

$$Q_k^T g_k = \begin{pmatrix} Z_k^T g_k \\ 0 \end{pmatrix}.$$

Reduced-Hessian Methods

A reduced-Hessian (RH) method obtains p_k from

$$p_k = Z_k q_k \quad \text{where} \quad q_k \quad \text{solves} \quad Z_k^T H Z_k q_k = -Z_k^T g_k, \qquad \text{(RH step)}$$

which is equivalent to (qN step).

In practice, we use a Cholesky factorization $R_k^T R_k = Z_k^T H_k Z_k$.

- The new gradient g_{k+1} is accepted iff $\|(I - Z_k Z_k^T) g_{k+1}\| > \epsilon$.
- Store and update Z_k, R_k, $Z_k^T p_k$, $Z_k^T g_k$, and $Z_k^T g_{k+1}$.

Reduced-Hessian Methods

A reduced-Hessian (RH) method obtains p_k from

$$p_k = Z_k q_k \text{ where } q_k \text{ solves } Z_k^T H Z_k q_k = -Z_k^T g_k, \quad \text{(RH step)}$$

which is equivalent to (qN step).

In practice, we use a Cholesky factorization $R_k^T R_k = Z_k^T H_k Z_k$.

- The new gradient g_{k+1} is accepted iff $\|(I - Z_k Z_k^T) g_{k+1}\| > \epsilon$.
- Store and update Z_k, R_k, $Z_k^T p_k$, $Z_k^T g_k$, and $Z_k^T g_{k+1}$.

$$
\begin{aligned}
H_k &= Q_k Q_k^T H_k Q_k Q_k^T \\
&= \begin{pmatrix} Z_k & W_k \end{pmatrix} \begin{pmatrix} Z_k^T H_k Z_k & 0 \\ 0 & \sigma I_{n-r_k} \end{pmatrix} \begin{pmatrix} Z_k \\ W_k \end{pmatrix} \\
&= Z_k (Z_k^T H_k Z_k) Z_k^T + \sigma (I - Z_k Z_k^T).
\end{aligned}
$$

$\Rightarrow$ any z such that $Z_k^T z = 0$ satisfies $H_k z = \sigma z$.

Reduced-Hessian Method Variants

Reinitialization: If $g_{k+1} \notin \mathcal{G}_k$, the Cholesky factor R_k is updated as

$$R_{k+1} \leftarrow \begin{pmatrix} R_k & 0 \\ 0 & \sqrt{\sigma_{k+1}} \end{pmatrix},$$

where σ_{k+1} is based on the latest estimate of the curvature, e.g.,

$$\sigma_{k+1} = \frac{y_k^T s_k}{s_k^T s_k}.$$

Lingering: restrict search direction to a smaller subspace and allow the subspace to expand only when f is suitably minimized on that subspace.

Reduced-Hessian Method Variants

Reinitialization: If $g_{k+1} \notin \mathcal{G}_k$, the Cholesky factor R_k is updated as

$$R_{k+1} \leftarrow \begin{pmatrix} R_k & 0 \\ 0 & \sqrt{\sigma_{k+1}} \end{pmatrix},$$

where σ_{k+1} is based on the latest estimate of the curvature, e.g.,

$$\sigma_{k+1} = \frac{y_k^T s_k}{s_k^T s_k}.$$

Lingering: restrict search direction to a smaller subspace and allow the subspace to expand only when f is suitably minimized on that subspace.

Reduced-Hessian Method Variants

Limited-memory: instead of storing the full approximate Hessian, keep information from only the last m steps ($m \ll n$).

Key differences:

- Form Z_k from *search directions* instead of the *gradients*.
- Must store T_k from $B_k = Z_k T_k$ to update most quantities.
- Drop columns from B_k when necessary.

Reduced-Hessian Method Variants

Limited-memory: instead of storing the full approximate Hessian, keep information from only the last m steps ($m \ll n$).

Key differences:

- Form Z_k from *search directions* instead of the *gradients*.
- Must store T_k from $B_k = Z_k T_k$ to update most quantities.
- Drop columns from B_k when necessary.

Main idea:

Maintain $B_k = Z_k T_k$, where B_k is the $m \times n$ matrix

$$B_k = \begin{pmatrix} p_{k-m+1} & p_{k-m+2} & \cdots & p_{k-1} & p_k \end{pmatrix}$$

with $m \ll n$ (e.g., $m = 5$).

$B_k = Z_k T_k$ is not available until p_k has been computed.

However, if Z_k is a basis for $\operatorname{span}(p_{k-m+1}, \ldots, p_{k-1}, p_k)$

then it is also a basis for $\quad \operatorname{span}(p_{k-m+1}, \ldots, p_{k-1}, g_k)$,

i.e., only the triangular factor associated with the (common) basis for each of these subspaces will be different.

Main idea:

Maintain $B_k = Z_k T_k$, where B_k is the $m \times n$ matrix

$$B_k = \begin{pmatrix} p_{k-m+1} & p_{k-m+2} & \cdots & p_{k-1} & p_k \end{pmatrix}$$

with $m \ll n$ (e.g., $m = 5$).

$B_k = Z_k T_k$ is not available until p_k has been computed.

However, if Z_k is a basis for $\operatorname{span}(p_{k-m+1}, \ldots, p_{k-1}, p_k)$

then it is also a basis for $\operatorname{span}(p_{k-m+1}, \ldots, p_{k-1}, g_k)$,

i.e., only the triangular factor associated with the (common) basis for each of these subspaces will be different.

At the start of iteration k, suppose we have the factors of

$$B_k^{(g)} = \begin{pmatrix} p_{k-m+1} & p_{k-m+2} & \cdots & p_{k-1} & g_k \end{pmatrix}$$

- Compute p_k.
- Swap p_k with g_k to give the factors of

$$B_k = \begin{pmatrix} p_{k-m+1} & p_{k-m+2} & \cdots & p_{k-1} & p_k \end{pmatrix}$$

- Compute x_{k+1}, g_{k+1}, etc., using a Wolfe line search.
- Update and reinitialize the factor R_k.
- If g_{k+1} is accepted, compute the factors of

$$B_{k+1}^{(g)} = \begin{pmatrix} p_{k-m+2} & p_{k-m+2} & \cdots & p_k & g_{k+1} \end{pmatrix}$$

At the start of iteration k, suppose we have the factors of

$$B_k^{(g)} = \begin{pmatrix} p_{k-m+1} & p_{k-m+2} & \cdots & p_{k-1} & g_k \end{pmatrix}$$

- Compute p_k.
- Swap p_k with g_k to give the factors of

$$B_k = \begin{pmatrix} p_{k-m+1} & p_{k-m+2} & \cdots & p_{k-1} & p_k \end{pmatrix}$$

- Compute x_{k+1}, g_{k+1}, etc., using a Wolfe line search.
- Update and reinitialize the factor R_k.
- If g_{k+1} is accepted, compute the factors of

$$B_{k+1}^{(g)} = \begin{pmatrix} p_{k-m+2} & p_{k-m+2} & \cdots & p_k & g_{k+1} \end{pmatrix}$$

At the start of iteration k, suppose we have the factors of

$$B_k^{(g)} = \begin{pmatrix} p_{k-m+1} & p_{k-m+2} & \cdots & p_{k-1} & g_k \end{pmatrix}$$

- Compute p_k.
- Swap p_k with g_k to give the factors of

$$B_k = \begin{pmatrix} p_{k-m+1} & p_{k-m+2} & \cdots & p_{k-1} & p_k \end{pmatrix}$$

- Compute x_{k+1}, g_{k+1}, etc., using a Wolfe line search.
- Update and reinitialize the factor R_k.
- If g_{k+1} is accepted, compute the factors of

$$B_{k+1}^{(g)} = \begin{pmatrix} p_{k-m+2} & p_{k-m+2} & \cdots & p_k & g_{k+1} \end{pmatrix}$$

At the start of iteration k, suppose we have the factors of

$$B_k^{(g)} = \begin{pmatrix} p_{k-m+1} & p_{k-m+2} & \cdots & p_{k-1} & g_k \end{pmatrix}$$

- Compute p_k.
- Swap p_k with g_k to give the factors of

$$B_k = \begin{pmatrix} p_{k-m+1} & p_{k-m+2} & \cdots & p_{k-1} & p_k \end{pmatrix}$$

- Compute x_{k+1}, g_{k+1}, etc., using a Wolfe line search.
- Update and reinitialize the factor R_k.
- If g_{k+1} is accepted, compute the factors of

$$B_{k+1}^{(g)} = \begin{pmatrix} p_{k-m+2} & p_{k-m+2} & \cdots & p_k & g_{k+1} \end{pmatrix}$$

At the start of iteration k, suppose we have the factors of

$$B_k^{(g)} = \begin{pmatrix} p_{k-m+1} & p_{k-m+2} & \cdots & p_{k-1} & g_k \end{pmatrix}$$

- Compute p_k.
- Swap p_k with g_k to give the factors of

$$B_k = \begin{pmatrix} p_{k-m+1} & p_{k-m+2} & \cdots & p_{k-1} & p_k \end{pmatrix}$$

- Compute x_{k+1}, g_{k+1}, etc., using a Wolfe line search.
- Update and reinitialize the factor R_k.
- If g_{k+1} is accepted, compute the factors of

$$B_{k+1}^{(g)} = \begin{pmatrix} p_{k-m+2} & p_{k-m+2} & \cdots & p_k & g_{k+1} \end{pmatrix}$$

At the start of iteration k, suppose we have the factors of

$$B_k^{(g)} = \begin{pmatrix} p_{k-m+1} & p_{k-m+2} & \cdots & p_{k-1} & g_k \end{pmatrix}$$

- Compute p_k.
- Swap p_k with g_k to give the factors of

$$B_k = \begin{pmatrix} p_{k-m+1} & p_{k-m+2} & \cdots & p_{k-1} & p_k \end{pmatrix}$$

- Compute x_{k+1}, g_{k+1}, etc., using a Wolfe line search.
- Update and reinitialize the factor R_k.
- If g_{k+1} is accepted, compute the factors of

$$B_{k+1}^{(g)} = \begin{pmatrix} p_{k-m+2} & p_{k-m+2} & \cdots & p_k & g_{k+1} \end{pmatrix}$$

Summary of key differences:

- Form Z_k from search directions instead of gradients.
- Must store T_k from $B_k = Z_k T_k$ to update most quantities.
- Can store B_k instead of Z_k.
- Drop columns from B_k when necessary.
- Half the storage of conventional limited-memory approaches.

Retains finite termination property for quadratic f

(both with and without reinitialization).

Summary of key differences:

- Form Z_k from search directions instead of gradients.
- Must store T_k from $B_k = Z_k T_k$ to update most quantities.
- Can store B_k instead of Z_k.
- Drop columns from B_k when necessary.
- Half the storage of conventional limited-memory approaches.

Retains finite termination property for quadratic f

(both with and without reinitialization).

Bound-Constrained Optimization

Bound Constraints

Given ℓ, $u \in \mathbb{R}^n$, solve

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \;\; f(x) \;\; \text{subject to} \quad \ell \leq x \leq u.$$

Focus on line-search methods that use the BFGS method:

- Projected-gradient [Byrd, Lu, Nocedal, Zhu (1995)]
- Projected-search [Bertsekas (1982)]

These methods are designed to move on and off constraints rapidly and identify the active set after a finite number of iterations.

Projected-Gradient Methods

Definitions

$$\mathcal{A}(x) = \{i : x_i = \ell_i \ \text{ or } \quad x_i = u_i\}$$

Define the projection $P(x)$ componentwise, where

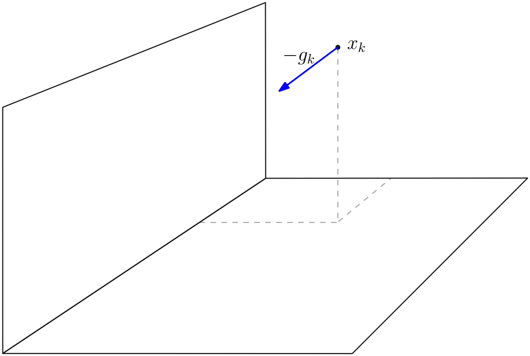
$$[P(x)]_i = \begin{cases} \ell_i & \text{if } \ x_i < \ell_i, \\ u_i & \text{if } \ x_i > u_i, \\ x_i & \text{otherwise.} \end{cases}$$

Given an iterate x_k, define the piecewise linear paths

$$x_{-g_k}(\alpha) = P(x_k - \alpha g_k) \ \text{ and } \quad x_{p_k}(\alpha) = P(x_k + \alpha p_k)$$

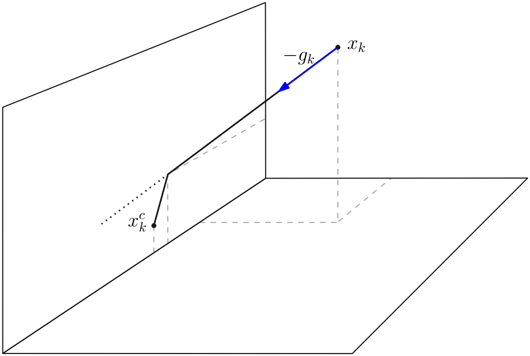
Algorithm L-BFGS-B

Given x_k and $q_k(x)$, a typical iteration of L-BFGS-B looks like:



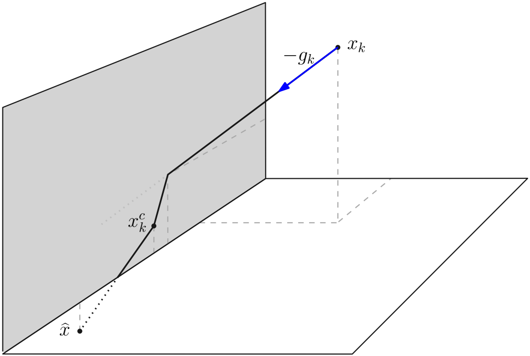
Move along projected path $x_{-g_k}(\alpha)$

Algorithm L-BFGS-B



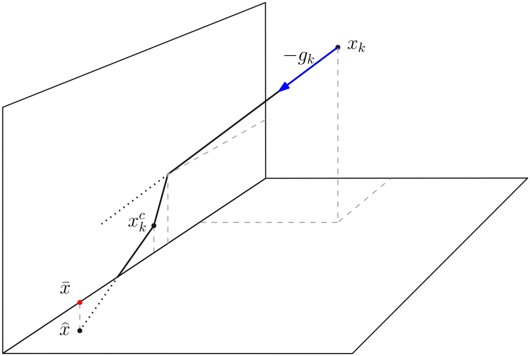
Find x_k^c, the first point that minimizes $q_k(x)$ along $x_{-g_k}(\alpha)$

Algorithm L-BFGS-B



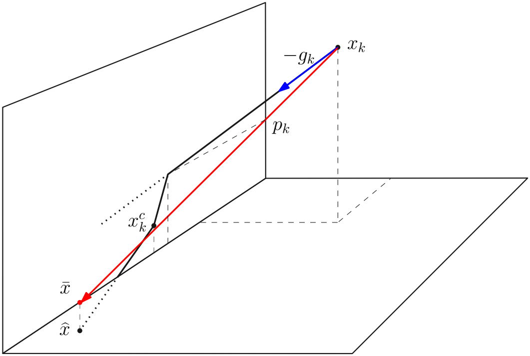
Find $\widehat{x}$, the minimizer of $q_k(x)$ with x_i fixed for every $i \in \mathcal{A}(x_k^c)$

Algorithm L-BFGS-B



Find $\bar{x}$ by projecting $\widehat{x}$ onto the feasible region

Algorithm L-BFGS-B



Wolfe line search along p_k with $\alpha_{\max} = 1$ to ensure feasibility

Projected-Search Methods

Definitions

$$\mathcal{A}(x) = \{i : x_i = \ell_i \ \text{ or } \ x_i = u_i\}$$

$$\mathcal{W}(x) = \{i : (x_i = \ell_i \ \text{ and } \ g_i > 0) \ \text{ or } \ (x_i = u_i \ \text{ and } \ g_i < 0)\}$$

Given a point x and vector p, the projected vector of p at x, $P_x(p)$, is defined componentwise, where

$$[P_x(p)]_i = \begin{cases} 0 & \text{if } \ i \in \mathcal{W}(x), \\ p_i & \text{otherwise.} \end{cases}$$

Given a subspace $\mathcal{S}$, the projected subspace of $\mathcal{S}$ at x is defined as

$$P_x(\mathcal{S}) = \{P_x(p) : p \in \mathcal{S}\}.$$

Definitions

$$\mathcal{A}(x) = \{i : x_i = \ell_i \ \text{ or } \ x_i = u_i\}$$

$$\mathcal{W}(x) = \{i : (x_i = \ell_i \ \text{ and } \ g_i > 0) \ \text{ or } \ (x_i = u_i \ \text{ and } \ g_i < 0)\}$$

Given a point x and vector p, the projected vector of p at x, $P_x(p)$, is defined componentwise, where

$$[P_x(p)]_i = \begin{cases} 0 & \text{if } \ i \in \mathcal{W}(x), \\ p_i & \text{otherwise.} \end{cases}$$

Given a subspace $\mathcal{S}$, the projected subspace of $\mathcal{S}$ at x is defined as

$$P_x(\mathcal{S}) = \{P_x(p) : p \in \mathcal{S}\}.$$

Projected-Search Methods

A typical projected-search method updates x_k as follows:

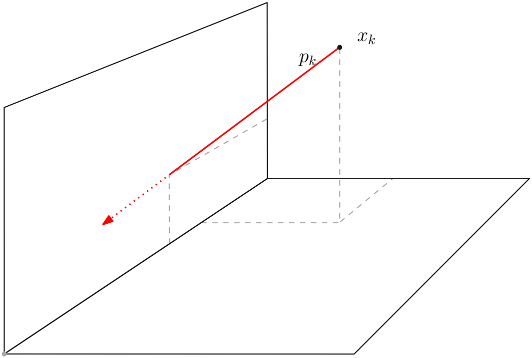
- Compute $\mathcal{W}(x_k)$
- Calculate p_k as the solution of

$$\min_{p \in P_{x_k}(\mathbb{R}^n)} q_k(x_k + p)$$

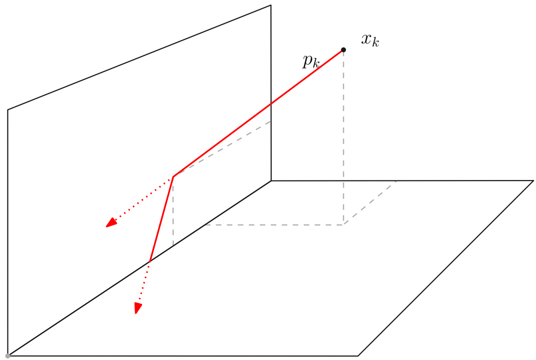
- Obtain x_{k+1} from an Armijo-like line search on $\psi(\alpha) = f(x_{p_k}(\alpha))$

Projected-Search Methods

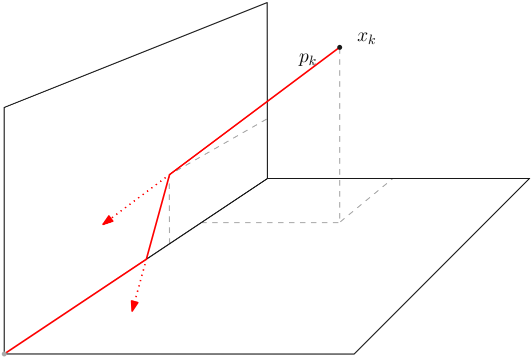
Given x_k and $f(x)$, a typical iteration looks like:



Projected-Search Methods



Projected-Search Methods



Armijo-like line search along $P(x_k + \alpha p_k)$

Quasi-Wolfe Line Search

Can't use Wolfe conditions: $\psi(\alpha)$ is a continuous, piecewise differentiable function with cusps where $x_{p_k}(\alpha)$ changes direction.

As $\psi(\alpha) = f\big(x_p(\alpha)\big)$ is only piecewise differentiable, it is not possible to know when an interval contains a Wolfe step.

Definition

Let $\psi'_-(\alpha)$ and $\psi'_+(\alpha)$ denote left- and right-derivatives of $\psi(\alpha)$.

Define a [strong] quasi-Wolfe step to be an Armijo-like step that satisfies at least one of the following conditions:

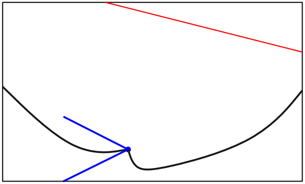
- $|\psi'_-(\alpha)| \leq \eta_W |\psi'_+(0)|$
- $|\psi'_+(\alpha)| \leq \eta_W |\psi'_+(0)|$
- $\psi'_-(\alpha) \leq 0 \leq \psi'_+(\alpha)$

Definition

Let $\psi'_-(\alpha)$ and $\psi'_+(\alpha)$ denote left- and right-derivatives of $\psi(\alpha)$.

Define a [strong] quasi-Wolfe step to be an Armijo-like step that satisfies at least one of the following conditions:

- $|\psi'_-(\alpha)| \le \eta_W |\psi'_+(0)|$
- $|\psi'_+(\alpha)| \le \eta_W |\psi'_+(0)|$
- $\psi'_-(\alpha) \le 0 \le \psi'_+(\alpha)$

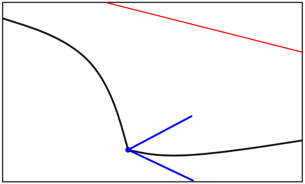


Definition

Let $\psi'_-(\alpha)$ and $\psi'_+(\alpha)$ denote left- and right-derivatives of $\psi(\alpha)$.

Define a [strong] quasi-Wolfe step to be an Armijo-like step that satisfies at least one of the following conditions:

- $|\psi'_-(\alpha)| \le \eta_W |\psi'_+(0)|$
- $|\psi'_+(\alpha)| \le \eta_W |\psi'_+(0)|$
- $\psi'_-(\alpha) \le 0 \le \psi'_+(\alpha)$

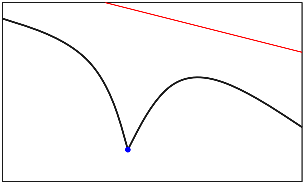


Definition

Let $\psi'_-(\alpha)$ and $\psi'_+(\alpha)$ denote left- and right-derivatives of $\psi(\alpha)$.

Define a [strong] quasi-Wolfe step to be an Armijo-like step that satisfies at least one of the following conditions:

- $|\psi'_-(\alpha)| \leq \eta_W |\psi'_+(0)|$
- $|\psi'_+(\alpha)| \leq \eta_W |\psi'_+(0)|$
- $\psi'_-(\alpha) \leq 0 \leq \psi'_+(\alpha)$



Theory

Analogous conditions for the existence of the Wolfe step imply the existence of the quasi-Wolfe step.

Quasi-Wolfe line searches are ideal for projected-search methods.

If ψ is differentiable the quasi-Wolfe and Wolfe conditions are identical.

In rare cases, H_k cannot be updated after quasi-Wolfe step.

Reduced-Hessian Methods for Bound-Constrained Optimization

Motivation

Projected-search methods typically calculate p_k as the solution to

$$\min_{p \in P_{x_k}(\mathbb{R}^n)} q_k(x_k + p)$$

If N_k has orthonormal columns that span $P_{x_k}(\mathbb{R}^n)$:

$$p_k = N_k q_k \;\text{ where } q_k \text{ solves }\; (N_k^T H_k N_k) q_k = -N_k^T g_k$$

An RH method for bound-constrained optimization (RH-B) solves

$$p_k = Z_k q_k \;\text{ where }\; q_k \;\text{ solves }\; (Z_k^T H_k Z_k) q_k = -Z_k^T g_k$$

where the orthonormal columns of Z_k span $P_{x_k}(\mathcal{G}_k)$.

Motivation

Projected-search methods typically calculate p_k as the solution to

$$\min_{p \in P_{x_k}(\mathbb{R}^n)} q_k(x_k + p)$$

If N_k has orthonormal columns that span $P_{x_k}(\mathbb{R}^n)$:

$$p_k = N_k q_k \text{ where } q_k \text{ solves } (N_k^T H_k N_k) q_k = -N_k^T g_k$$

An RH method for bound-constrained optimization (RH-B) solves

$$p_k = Z_k q_k \text{ where } q_k \text{ solves } (Z_k^T H_k Z_k) q_k = -Z_k^T g_k$$

where the orthonormal columns of Z_k span $P_{x_k}(\mathcal{G}_k)$.

Details

Builds on limited-memory RH method framework, plus:

- Update values dependent on B_k when $\mathcal{W}_k \neq \mathcal{W}_{k-1}$
- Use projected-search trappings (working set, projections, ...).
- Use line search compatible with projected-search methods.
- Update H_k with curvature in direction restricted to range(Z_{k+1}).

Cost of Changing the Working Set

An RH-B method can be explicit or implicit. An explicit method stores and uses Z_k and an implicit method stores and uses B_k.

When $\mathcal{W}_k \neq \mathcal{W}_{k-1}$, all quantities dependent on B_k are updated:

- Dropping n_d indices: $\approx 21m^2 n_d$ flops if implicit
- Adding n_a indices: $\approx 24m^2 n_a$ flops if implicit
- Using an explicit method: $+6mn(n_a + n_d)$ flops to update Z_k

Some Numerical Results

Results

LRH-B (v1.0) and LBFGS-B (v3.0)

LRH-B implemented in Fortran 2003; LBFGS-B in Fortran 77.

373 problems from CUTEst test set, with n between 1 and $192,627$.

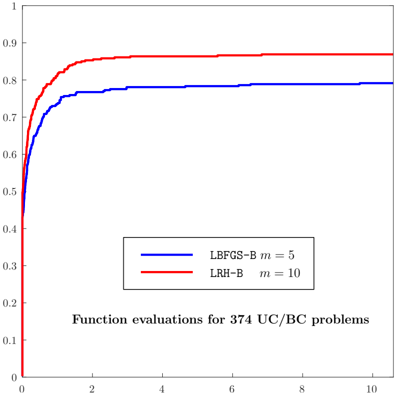
Termination: $\|P_{x_k}(g_k)\|_\infty < 10^{-5}$ or 300K itns or 1000 cpu secs.

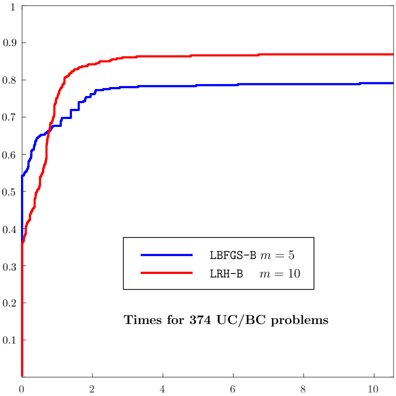
Implementation

Algorithm LRH-B (v1.0):

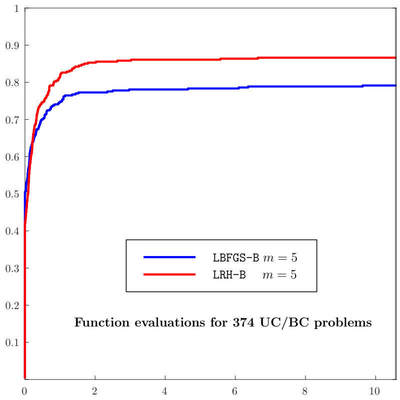
- Limited-memory: restricted to m preceding search directions.
- Reinitialization: included if $n > \min(6, m)$.
- Lingering: excluded.
- Method type: implicit.
- Updates: B_k, T_k, R_k updated for all $\mathcal{W}_k \neq \mathcal{W}_{k-1}$.

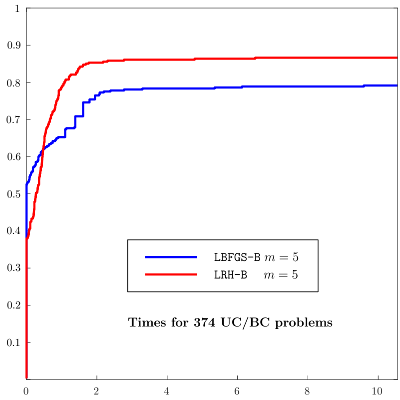
Default parameters



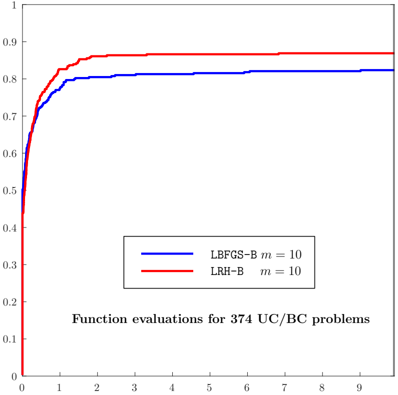


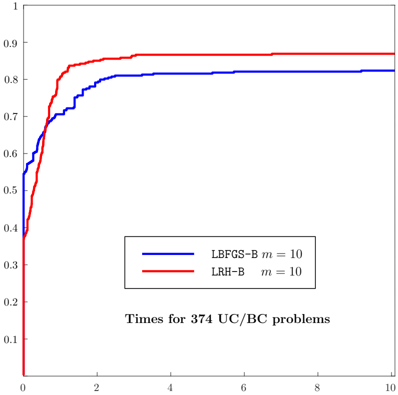
Name	m	*Failed*
LBFGS-B	5	*78*
LRH-B	10	*49*



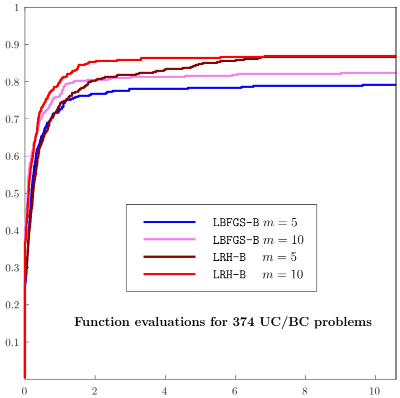


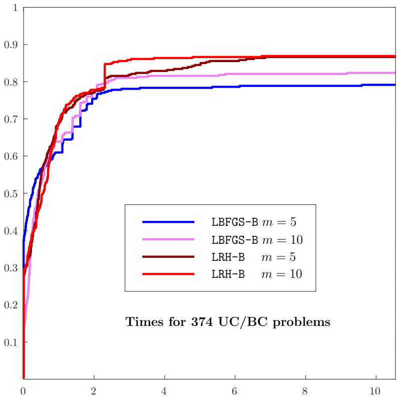
Name	*m*	*Failed*
LBFGS-B	5	*78*
LRH-B	5	*50*





Name	m	*Failed*
`LBFGS-B`	10	*78*
`LRH-B`	10	*49*


LBFGS-B $m = 5$
LBFGS-B $m = 10$
LRH-B $m = 5$
LRH-B $m = 10$
Function evaluations for 374 UC/BC problems


LBFGS-B $m = 5$
LBFGS-B $m = 10$
LRH-B $m = 5$
LRH-B $m = 10$
Times for 374 UC/BC problems

Outstanding Issues

- Projected-search methods:
    - When to update/factor?

      Better to refactor when there are lots of changes to $\mathcal{A}$.
    - Implement plane rotations via level-two BLAS.
- How complex should we make the quasi-Wolfe search?

Thank you!

References

D. P. Bertsekas, *Projected Newton methods for optimization problems with simple constraints*, SIAM J. Control Optim., 20:221–246, 1982.

R. Byrd, P. Lu, J. Nocedal and C. Zhu, *A limited-memory algorithm for bound constrained optimization*, SIAM J. Sci. Comput. 16:1190–1208, 1995.

M. C. Fenelon, *Preconditioned conjugate-gradient-type methods for large-scale unconstrained optimization*, Ph.D. thesis, Department of Operations Research, Stanford University, Stanford, CA, 1981.

M. W. Ferry, P. E. Gill and E. Wong, *A limited-memory reduced Hessian method for bound-constrained optimization*, Report CCoM 18-01, Center for Computational Mathematics, University of California, San Diego, 2018 (to appear).

P. E. Gill and M. W. Leonard, *Reduced-Hessian quasi-Newton methods for unconstrained optimization*, SIAM J. Optim., 12:209–237, 2001.

P. E. Gill and M. W. Leonard, *Limited-memory reduced-Hessian methods for unconstrained optimization*, SIAM J. Optim., 14:380–401, 2003.

J. Morales and J. Nocedal, *Remark on "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization"*, ACM Trans. Math. Softw., 38(1)7:1–7:4, 2011.

D. Siegel, *Modifying the BFGS update by a new column scaling technique*, Math. Program., 66:45–78, 1994.