THE UNIVERSITY OF CHICAGO

Section 7 – short

Methods for Large Scale Optimization

N & W: Section 7

Mihai Anitescu

What if we apply these methods on truly large scale problems?

- We may be unwilling to factorize systems of that size but perhaps we are willing to computer Hessian vector products (which you can incidentally approximate with finite differences of gradients or use automatic differentiation)
  - Solution (a) Solve the linear system INEXACTLY
  - Solution (b): versions of Conjugated gradients IF it can deal with indefinite matrices.
- We may be unwilling to compute the Hessian altogether and using BFGS gets us out of memory!
  - Solution limited-memory BFGS.

CHICAGO | SSA

Framework for Early termination: Inexact Newton Methods

- We modify the original Newton method: $\nabla^2 f_k p_k = -\nabla f_k \Rightarrow \nabla^2 f_k p_k \approx -\nabla f_k$
- Residual: $r_k = \nabla^2 f_k p_k + \nabla f_k,$
- Inner Loop termination rule $\|r_k\| \leq \eta_k \|\nabla f_k\|,$
- Ensures at least linear convergence:

Theorem 7.1.

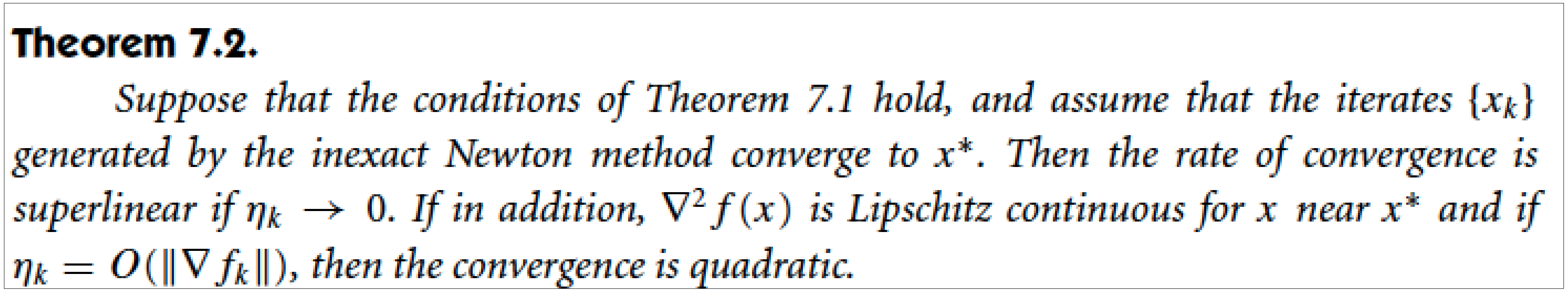
Suppose that $\nabla^2 f(x)$ exists and is continuous in a neighborhood of a minimizer x^*, with $\nabla^2 f(x^*)$ is positive definite. Consider the iteration $x_{k+1} = x_k + p_k$ where p_k satisfies (7.3), and assume that $\eta_k \leq \eta$ for some constant $\eta \in [0, 1)$. Then, if the starting point x_0 is sufficiently near x^*, the sequence $\boxed{\{x_k\} \text{ converges to } x^*}$ and satisfies

$$\boxed{\|\nabla^2 f(x^*)(x_{k+1} - x^*)\| \leq \hat{\eta}\|\nabla^2 f(x^*)(x_k - x^*)\|,} \tag{7.4}$$

for some constant $\hat{\eta}$ with $\eta < \hat{\eta} < 1$.

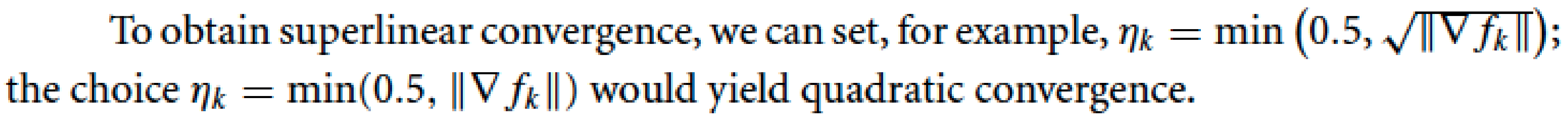
Superlinear Convergence

- If the forcing sequence goes to 0, I will solve the subproblem exactly and approach the Newton step.



Theorem 7.2.

Suppose that the conditions of Theorem 7.1 hold, and assume that the iterates $\{x_k\}$ generated by the inexact Newton method converge to x^*. Then the rate of convergence is superlinear if $\eta_k \to 0$. If in addition, $\nabla^2 f(x)$ is Lipschitz continuous for x near x^* and if $\eta_k = O(\|\nabla f_k\|)$, then the convergence is quadratic.



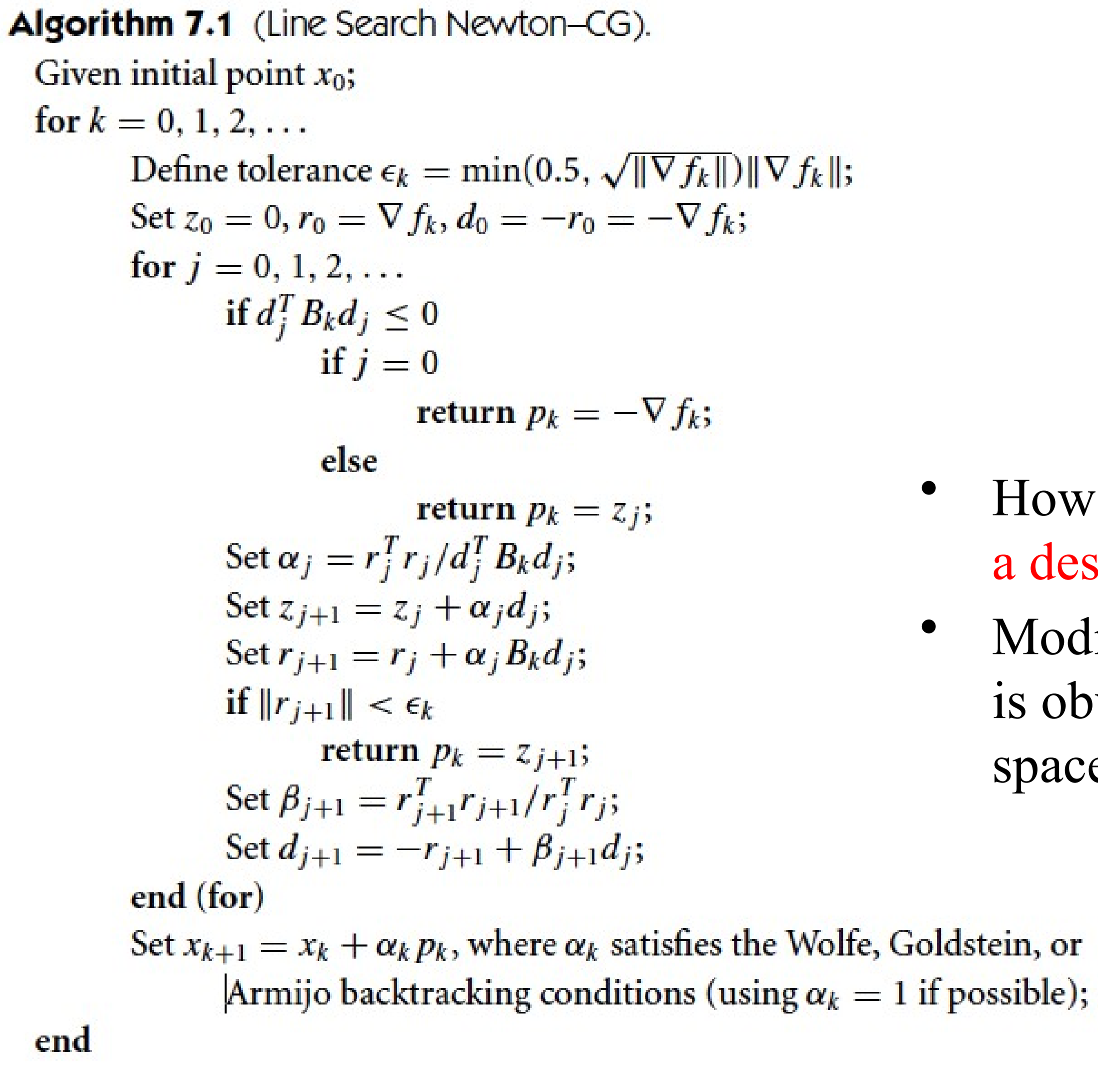
To obtain superlinear convergence, we can set, for example, $\eta_k = \min\left(0.5, \sqrt{\|\nabla f_k\|}\right)$; the choice $\eta_k = \min(0.5, \|\nabla f_k\|)$ would yield quadratic convergence.

- Note, however, that the second one is too conservative on tolerance so it is likely to require many iterations in the beginning

7.2 Krylov Methods

- How do I deal with indefiniteness of the matrices, (CG works only for positive definite matrices?)
- I***dea: Proceed with CG until we find a negative inner product!***
- In a line search framework break when negative curvature occurs.
- In a trust-region approach, break if the inner iteration exceeds the trust region or if negative curvature occurs.
- As the forcing sequence goes to 0, I will pick up the Newton step and recover superlinear convergence.

Line-Search CG



Algorithm 7.1 (Line Search Newton–CG).

```
Given initial point $x_0$;
for $k = 0, 1, 2, \ldots$
        Define tolerance $\epsilon_k = \min(0.5, \sqrt{\|\nabla f_k\|})\|\nabla f_k\|$;
        Set $z_0 = 0, r_0 = \nabla f_k, d_0 = -r_0 = -\nabla f_k$;
        for $j = 0, 1, 2, \ldots$
                if $d_j^T B_k d_j \le 0$
                        if $j = 0$
                                return $p_k = -\nabla f_k$;
                        else
                                return $p_k = z_j$;
                Set $\alpha_j = r_j^T r_j / d_j^T B_k d_j$;
                Set $z_{j+1} = z_j + \alpha_j d_j$;
                Set $r_{j+1} = r_j + \alpha_j B_k d_j$;
                if $\|r_{j+1}\| < \epsilon_k$
                        return $p_k = z_{j+1}$;
                Set $\beta_{j+1} = r_{j+1}^T r_{j+1} / r_j^T r_j$;
                Set $d_{j+1} = -r_{j+1} + \beta_{j+1} d_j$;
        end (for)
        Set $x_{k+1} = x_k + \alpha_k p_k$, where $\alpha_k$ satisfies the Wolfe, Goldstein, or
                Armijo backtracking conditions (using $\alpha_k = 1$ if possible);
end
```

- How come it works? CG itself is a descent method !!!
- Modification for Preconditioning is obvious, it happens only in D-space.

$$\boxed{\eta_k = \min(0.5, \sqrt{\|\nabla f_k\|})}$$

CG-Trust Region (STEIHAUG)

$$\min_{p\in\mathbb{R}^n} m_k(p) \overset{\text{def}}{=} f_k + (\nabla f_k)^T p + \tfrac{1}{2} p^T B_k p \quad \text{subject to } \|p\| \le \Delta_k,$$

Algorithm 7.2 (CG–Steihaug).

```
Given tolerance $\epsilon_k > 0$;
Set $z_0 = 0$, $r_0 = \nabla f_k$, $d_0 = -r_0 = -\nabla f_k$;
if $\|r_0\| < \epsilon_k$
        return $p_k = z_0 = 0$;
for $j = 0, 1, 2, \ldots$
        if $d_j^T B_k d_j \le 0$
                Find $\tau$ such that $p_k = z_j + \tau d_j$ minimizes $m_k(p_k)$ in (4.5)
                        and satisfies $\|p_k\| = \Delta_k$;
                return $p_k$;
        Set $\alpha_j = r_j^T r_j / d_j^T B_k d_j$;
        Set $z_{j+1} = z_j + \alpha_j d_j$;
        if $\|z_{j+1}\| \ge \Delta_k$
                Find $\tau \ge 0$ such that $p_k = z_j + \tau d_j$ satisfies $\|p_k\| = \Delta_k$;
                return $p_k$;
        Set $r_{j+1} = r_j + \alpha_j B_k d_j$;
        if $\|r_{j+1}\| < \epsilon_k$
                return $p_k = z_{j+1}$;
        Set $\beta_{j+1} = r_{j+1}^T r_{j+1} / r_j^T r_j$;
        Set $d_{j+1} = -r_{j+1} + \beta_{j+1} d_j$;
end (for).
```

- Only inner loop.
- +Define forcing sequence as LS-CG
- +Iterate in x.

7.4 Limited Memory BFGS. Data Assimilation

- Future: Hidden Markov Models with Model Error (Present: models are perfect).

$$x_{t_{k+1}} = M(x_{t_k}) + \eta_k,\ y_k = \mathcal{H}(x_{t_k}) + \varepsilon_k,\ \eta_k \sim \mathcal{N}(\mathbf{0}, Q_k)\ \varepsilon_k \sim \mathcal{N}(\mathbf{0}, R_k).$$

- Resulting minus log-likelihood

$$\begin{aligned}\mathcal{J}(x_{t_0}, x_{t_1}, \ldots, x_{t_N}) =& \frac{1}{2}(x_{t_0} - x_B)^T Q_B^{-1}(x_{t_0} - x_B) + \\ & \frac{1}{2}\sum_{k=0}^{N}(\mathcal{H}_k(x_{t_k}) - y_k)^T R_k^{-1}(\mathcal{H}_k(x_{t_k}) - y_k) + \\ & \frac{1}{2}\sum_{k=0}^{N-1}(x_{t_{k+1}} - \mathcal{M}_k(x_{t_k}))^T Q_k^{-1}(x_{t_{k+1}} - \mathcal{M}_k(x_{t_k})).\end{aligned}$$

Memory Issues

- Weakness of quasi-Newton: storage.
- Atmospheric state estimation – "regularized" least squares, with DOF== dimension of atmospheric states space x time.
- If we have a 10 km by 10 km by 1 km grid ~ 15M cell points.
- About 10 state variables: 150M variables per time step.
- 100 time steps: 15B variables per assimilation.
- BFGS storage == 15Bx 15B == 2 x 10^20 === 2 x 10^11 GB == 16 x 10^11 $ ☺
- Not working. What may be the alternatives?

BFGS recap

$$x_{k+1} = x_k - \alpha_k H_k \nabla f_k, \tag{7.15}$$

where α_k is the step length and H_k is updated at every iteration by means of the formula

$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T \tag{7.16}$$

(see (6.17)), where

$$\rho_k = \frac{1}{y_k^T s_k}, \qquad V_k = I - \rho_k y_k s_k^T, \tag{7.17}$$

and

$$s_k = x_{k+1} - x_k, \qquad y_k = \nabla f_{k+1} - \nabla f_k. \tag{7.18}$$

Setup of L-BFGS

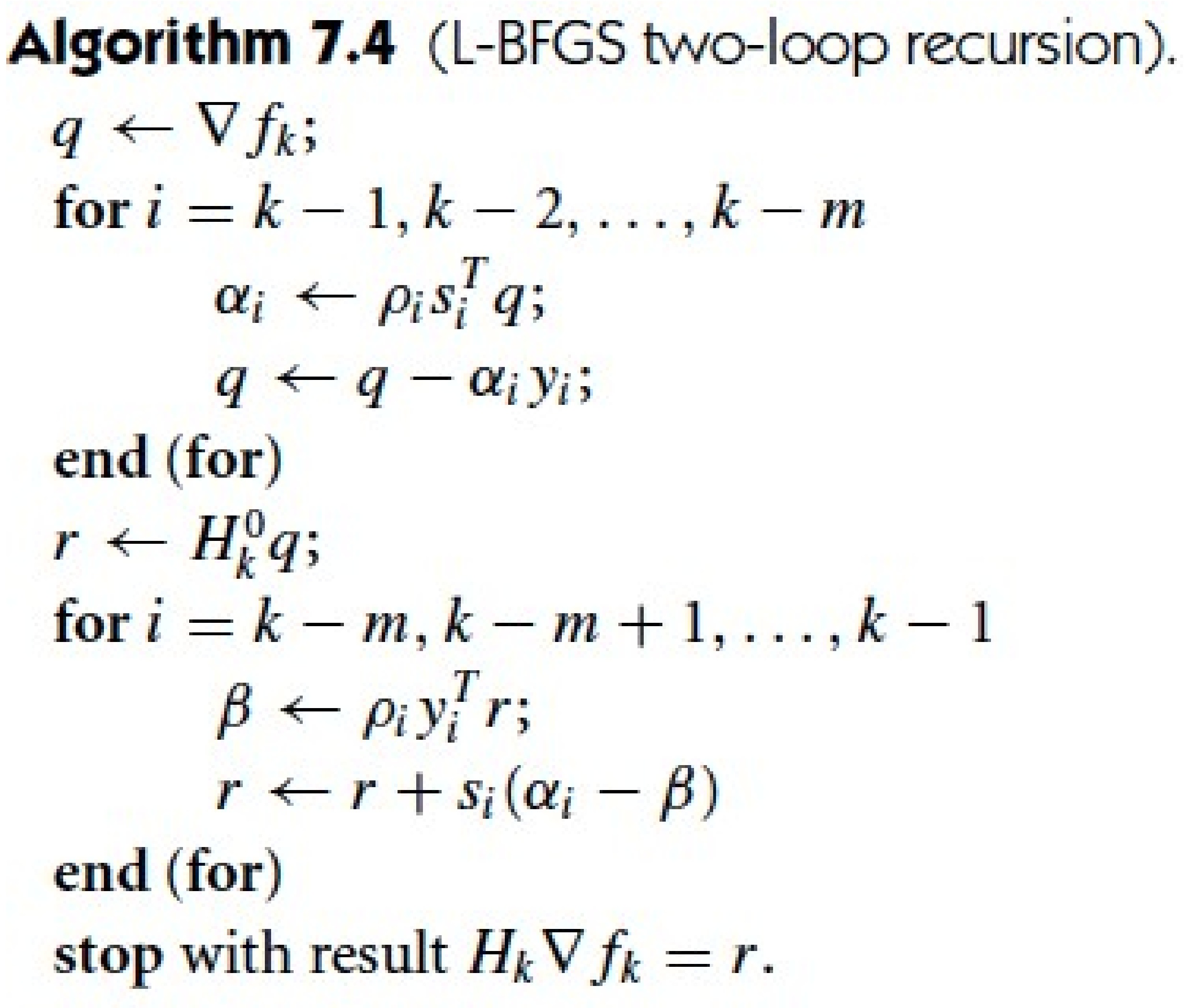
- At iteration k, and iterate x_k
- Store only the last m s,y pairs: $\{s_i, y_i, \quad i = k-m, \ldots, k-1$
- (For Example) Choose $H_k^0 = \gamma_k I$ where

$$\gamma_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}$$

(other starts are an issue of research and possible note that this approximates the inverse hessian along the last direction)

- Use the BFGS update for m steps and compute the search direction.
- Do BFGS line search or safeguarded BFGS line search and determine x_{k+1} and then s_{k+1}, y_{k+1}
- Restart.

Two Loop Evaluation of H times vector for L-BFGS

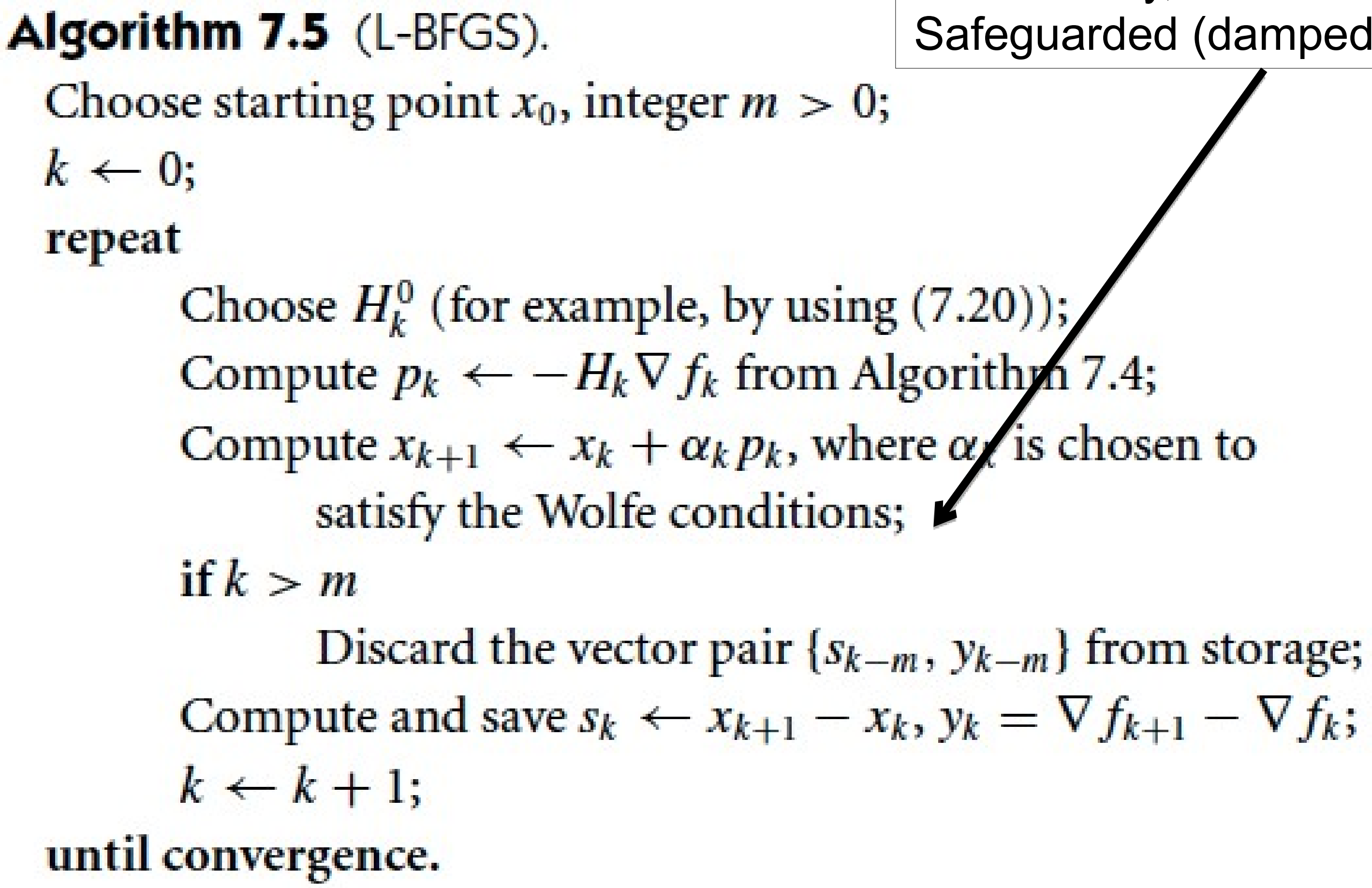


Algorithm 7.4 (L-BFGS two-loop recursion).

```
$q \leftarrow \nabla f_k$;
for $i = k-1, k-2, \ldots, k-m$
        $\alpha_i \leftarrow \rho_i s_i^T q$;
        $q \leftarrow q - \alpha_i y_i$;
end (for)
$r \leftarrow H_k^0 q$;
for $i = k-m, k-m+1, \ldots, k-1$
        $\beta \leftarrow \rho_i y_i^T r$;
        $r \leftarrow r + s_i(\alpha_i - \beta)$
end (for)
stop with result $H_k \nabla f_k = r$.
```

L-BFGS outer loop

Alternatively, use
Safeguarded (damped) BFGS rule



Algorithm 7.5 (L-BFGS).

```
Choose starting point $x_0$, integer $m > 0$;
$k \leftarrow 0$;
repeat
        Choose $H_k^0$ (for example, by using (7.20));
        Compute $p_k \leftarrow -H_k \nabla f_k$ from Algorithm 7.4;
        Compute $x_{k+1} \leftarrow x_k + \alpha_k p_k$, where $\alpha_k$ is chosen to
                satisfy the Wolfe conditions;
        if $k > m$
                Discard the vector pair $\{s_{k-m}, y_{k-m}\}$ from storage;
        Compute and save $s_k \leftarrow x_{k+1} - x_k$, $y_k = \nabla f_{k+1} - \nabla f_k$;
        $k \leftarrow k + 1$;
until convergence.
```

L-BFGS

- L-BFGS is the method of choice for variational data assimilation in weather forecast.
- It is used for example by ECMWF in their operations.
- On well conditioned problems it tends to beat Newton-CG, where the Hessian-vector is computed with AD.
- On ill-conditioned problems, not so much, but it is rarely run to convergence for huge scale problems.