4. TRICKS OF THE TRADE
4.1 Structure of an optimization program

Elements of an optimization problem and "templates".

Why would I want to do this?
It is easier to "debug" and verify and reuse. The number of errors per functional block may be larger; but they are easier to find. Also easier to grade.

Elements:
1) The problem
2) The iteration
3) Stopping criterion
4) Update of parameters

Good software practices: functional forms and comments, consistent tests.
4.12 Element:

1) Problem (e.g. febhu_wrap.m)

function [f, g, h] = myprob (x, param_list

% param_list: noderivs

% problem_version

2) One iteration (e.g. newtonLikeIT)

function [xout, inb] = myiteration

(function Handle, xin, param_list

% param_list = iteration_type

% iteration_index

[f, g, h] = functionHandle (x, ...)

inb = grad value?

3) Stopping criterion:

Tricky.
4.1.3 4) Update of parameters (e.g. trust region). Some people would write a separate function for it.

5) main

while not_stopping (4)

[ xout, ... ] = uniktin (Duemproblem, x, 

update_par (4)

Statistics
4. Special topics.
4.2. Using Sparse

Most Hessian matrices of interest in applications are not dense.
E.g., Building Control Problem.
Variational discretization of PDE:
\[ \text{Laplacian in 1d: } V_{xx} - 2V_x + V_{x-1} = 2 \]

Therefore, can store them only by
\[
[i, j, s] = \text{find}(M) \quad :=
\]
\[
\text{Column index}
\]
\[
\text{row index}
\]
\[
\text{value}
\]

It can be reduced even further; some of the row and column indices are redundant ("compressed row storage")
Advantage: If \( \text{null}(A) \sim O(n) \), then \( A + x \) takes only \( O(n) \) and \( A \) needs only \( O(n) \) storage.

Disadvantage: Now all operations must be aware of sparsity but most math libraries support this format.

Problem: \( A \) may be sparse, but \( A = LL^T \Rightarrow L \) may be dense; so storage will be a problem.

Idea: Use iterative methods such as CG, which needs only Matrix-Vector multiplication and terminates early...
4. Special topics
4.3. Conjugate gradient

1. Idea: for a p.d. solve \(Ax=b\) is equivalent to min \(\frac{1}{2}x^T Ax - b^T x = \phi(x)\)

2. Steepest descent is slow: "2s - 2s - 2s - 2s". I tend to repeat search directions. How do I prevent this?

3. Idea: use orthogonal search directions and minimize error along them. \(\langle d_i, d_j \rangle = 0\) if

\[\text{Then } e^0 = Ax^0 - b = -\sum_{i=1}^{n} \alpha_i \hat{d}_i \]

\[\Rightarrow \langle e^0, d_i \rangle = -\alpha_i \langle d_i^T d_i \rangle \Rightarrow \alpha_i = \frac{\langle e^0, d_i \rangle}{\langle d_i^T, d_i \rangle} \]

Define \(e(d) = e^0 + \sum_{i=1}^{n} \alpha_i d_i \)

\[\Rightarrow d_i = \frac{\langle e^0, d_i \rangle}{\langle d_i^T, d_i \rangle} \Rightarrow e = e^0 + d_i \]

and minimize
Problem: I don't know $e^0$! If I use Euclidean $\langle, \rangle$,

But I do know $Ae^0 = A(x^0 - x^*) = A\tilde{x}$.

$$\langle e^0, d^0 \rangle_A = e^{T} \tilde{A} d = -\tilde{r}^T d \approx e^0$$

Idea: Require $A$-ORTHOGONAL search

directions: $$(d^i)^T \hat{A} d^j = 0 \ (i \neq j)$$

$$e^{d+1} = e^d + \alpha^j d^j$$

$$x^{d+1} = x^d + \alpha^j d^j$$

$$-r^{d+1} = -r^d + \alpha^j d^j$$

Redo calculation $\Rightarrow$

$$z^i = \frac{1}{n} (d^i)^T A(d^j)$$

Note: $\min_{\alpha^d} \frac{1}{2} (x^j + \alpha^j d^j)^T A(x^j + \alpha^j d^j) - b^j(x^j + \alpha^j d^j)$
4.3.3 Optimality conditions:

\[ \nabla \mathbf{A}(\mathbf{x}^0 + \mathbf{u}) - \mathbf{d}^T \mathbf{b} = 0 \quad \mathbf{G} \]

\[ \mathbf{d}^T \mathbf{A} \mathbf{d}^j - (\mathbf{c}^T)^T \mathbf{r}^j = 0 \]

\[ \Rightarrow \mathbf{d}^j = \frac{(\mathbf{d}^T \mathbf{r}^j) \mathbf{A}^T \mathbf{d}^j}{(\mathbf{d}^T \mathbf{A} \mathbf{d}^j)} \]

* Observation: Each CG step reduces the objective function. Terminates in \( n \) iterations.

* Nice, but; how do I choose conjugate directions? (Note: This may be a misprint; the original text seems to be cut off.)

Idea: start with \( \mathbf{u}^1, \mathbf{u}^2, \ldots, \mathbf{u}^n \) linearly independent. Choose \( \mathbf{d}^1 \) by Gram-Schmidt with \( \langle \cdot, \cdot \rangle \) a inner product.
4.3.4.

\[ d_i^* = a_i^* + \sum_{k=0}^{i-1} \beta_k d_k \]

\[ \Rightarrow d_i^* A_{ij}^0 = d_i^* A_{ui}^i + \beta_{ij} d_j^* A_{ij}^j = 0 \]

\[ \Rightarrow \beta_{ij} = - \frac{d_i^* A_{ui}^i}{d_j^* A_{ij}^j} \quad j = 0, \ldots, i-1 \]

**But... need to store ALL d... \Rightarrow O(n^3)**

Effort increases at each iteration.

**Observations:**

\[ e_0 + \sum_{i=1}^{n-1} e_i = 0 \Rightarrow e_j + \sum_{i=0}^{j-1} \alpha_i d_i = 0 \]

\[ \Rightarrow d_0^* A_{e_j}^j + \sum_{i=j+1}^{n} d_i^* A_{ei}^i d_i^* = 0 \]

\[ i = j + 1 \quad 0 \quad (e < j) \]

\[ (e) = \sqrt{d_i^* r_j^*} = 0; \quad e < j \]

\[(P) \]

\[ \sum_{i=j}^{j} a_{ij} A_{ij}^i = 0 \]
4.3.5

5. Return now to Gram-Schmidt.

\[ r_i^T d_i = r_i^T u_i + \sum_{k=0}^{i-1} \beta_{ik} r_i^T d_k \]

If \( i < j \Rightarrow \) by (1) we have

\[ r_j^T u_i = 0 \quad (\beta) \quad \text{for} \quad i < j \]

\[ r_i^T u_i = r_i^T d_i \]

6. Recursion

\[ x_{k+1} = x_k + d_k \quad (k \geq 0) \quad (v = b - Ax) \]

\[ -r_{k+1} = -r_k + d_k Ad_k \]

\[ \Rightarrow r_{k+1} = r_k - \alpha_k Ad_k \quad (2k) \]
4.3.6 Recap

If solving $Ax = b$ (for, with $\frac{1}{2} x^T A x - b^T x$)

by $x^{k+1} = x^k + t_k d_k$ where $d^T A d = 0$; $i + j$.

And obtained by GS from $u^i$:

\[
d^i = u^i + \sum_{k=0}^{i-1} \beta_{i-k} d^k
d^i = \frac{d^i}{d^T A A^i} \beta_{ij} = \frac{d^i}{d^T A A^i}
\]

by exact minimization:

\[
r^i = b - A x^i
\]

Then:

\[
d^T r^i = 0; i \leq j\]
\[
d^T r^i = (d^i)^T r^i
\]
\[
r^{i+1} = r^i - \alpha^i d^i A d^i; i = 0,\ldots
\]
4.3.7 What if I choose $u_d = r_i$?

Then $r_i + u = 0$.

$$(d_j)^T B_{ij} = -d_j^T A u_i \quad j = 0, 1, \ldots, i-1$$

$$= - \frac{1}{d^T_i} (r^{j}_i - r^{j+1}_i)^T \frac{d^T_i}{d^T_j} \quad r_i^j = \frac{1}{d^T_j} \left( \begin{array}{c} r_i^j \\ \vdots \\ r_i^{j-1} \end{array} \right)$$

$$= \left\{ \begin{array}{ll}
\frac{1}{d^T_j} r_i^j & j = i-1 \\
0 & \text{otherwise}
\end{array} \right.$$
\[ r_0 = b_0 - Ax_0 = d_0 \]

"Line search" \[ d_i = \frac{r_i^\top r_i}{d_i^\top A d_i}; x_{i+1} = x_i + \alpha d_i \]

\[ r_{i+1} = r_i - x_i A d_i \]

"Search direction update" \[ \beta_{i+1} = (r_{i+1})^\top r_{i+1} \]

\[ d_{i+1} = r_{i+1} + \beta_{i+1} d_i \]

Variant (Nocedal and Wright)

\[ r_0 = Ax_0 - b_0 = -d_0 \]

"Line search" \[ d_i = \frac{r_i^\top r_i}{d_i^\top A d_i}; x_{i+1} = x_i + \alpha d_i \]

\[ r_{i+1} = r_i - x_i A d_i \]

"Search direction update" \[ \beta_{i+1} = (r_{i+1})^\top r_{i+1} \]

\[ d_{i+1} = r_{i+1} + \beta_{i+1} d_i \]

Only \( 1 \) Matrix-vector multiplication
4. Special topics

4.4 Early termination of CG

Claim. Solve \( \min \frac{1}{2} x^T A x - b^T x \) by CG

s.t. \( d^T A d = 0 \), \( i \neq j \) for \( k \)

with \( d^0 = x^0 = b \). Then \( x^k \) is a \( b \) solution.

Proof:

\[
\frac{1}{2} x^T A x = \frac{1}{2} \sum (d_i^T)^2 d_i^T A d_i = \\
\frac{1}{2} \sum (d_i^T)^2 (r_i^T r_i) = \frac{1}{2} \sum \frac{(r_i^T r_i)^2}{d_i^T A d_i} > 0
\]

\[ \frac{1}{2} x^T A x - b^T x \leq \phi(101) = 0 \]

\( \Rightarrow b^T x > 0 \)

Q.E.D.
4.4.2 When applied to optimization problem, produces descent direction.

\[ b = -\nabla f \Rightarrow -\nabla f^T p_k \geq 0. \]

(Still not exact with Hessian (or approx).)
4.5 Special topics
4.5 Inexact Newton Methods

Idea: Framework for solving the subproblem inexactly; i.e., by iterative methods.

\[ \nabla^2 \mathbf{x} = \mathbf{H}(\mathbf{p_x} + \nabla f(x^k)) \rightarrow \text{Inexact NM} \]

Terminate CG iterations, when

\[ 1 - \| r^k \| \leq \gamma \cdot 1 - \| \nabla f(x^k) \| \]

\( r^k \) is called the force sequence.

**Theorem 7.1.** Local convergence. Suppose \( f^2 \) is strictly convex and \( \Delta \).
4.5.2 Assume

1) \( \nabla^2 f(x) \in C^2(N(x^k)) \)

2) \( x_{k+1} - x_k + \gamma_k \)

3) \( 0 < \gamma_k \leq \gamma < 1 \) \( \forall k \)

Then \( \gamma_k \leq \frac{1}{\eta} \)

\[
\| \nabla^2 f(x^k) (x_{k+1} - x_k) \| \leq \eta \| \nabla^2 f(x^k) (x_{k+1} - x_k) \| 
\]

\[\boxdot\]

**Remark**: You will at least courage !!

---

**Theorem 7.2**: Assume \( x_k \rightarrow x^* \)

Then \( \gamma_k \rightarrow 0 \implies \) convergence is hyperlinear

\( \gamma_k = O(\| \nabla f(x_k) \|) \implies \) convergence is quadratic

Superlinear, e.g., \( \gamma_k = (\min(0.5, \sqrt{\| f(x_k) \|})) \)
4. Special topics
4.6 Line Search Newton - CG

**Idea:** Solve

\[ B^k p + \nabla f(x^k) = 0 \quad \text{approximately} \quad B^k = \nabla^2 f(x^k) \]

by CG

Define \( s^k = B^k p + \nabla f(x^k) \); \( \| s^k \| \leq \epsilon \)

\[ = \min \left( 0.5 \sqrt{ \frac{\| s^k \|}{\| s^k \|} } \right) \|

\[ \alpha^k = \frac{1}{\| s^k \|} \left[ \begin{array}{c} \alpha^k_1 \\ \vdots \\ \alpha^k_{n+1} \end{array} \right] \]
4.62 Given initial $x^0$

\[ f(x) = \begin{cases} 
& \text{for } k = 0, 1, 2 \\
& \text{Set } z^0 = 0; \quad r^0 = -\nabla f(x^0); \quad d^0 = -\nabla f(x^0) \\
& \text{for } j = 0, 1, 2, \ldots \\
& \quad \text{if } d^j \cdot r^j \leq 0 \\
& \quad \quad \text{if } j = 0 \\
& \quad \quad \text{else return } p^k = -\nabla f(x^0) \\
& \quad \quad \text{else return } p^k = x^j \\
& \quad \text{while } \|r^j\| \leq \varepsilon \\
& \quad \quad \text{if } r^j \cdot d^j \leq 0 \\
& \quad \quad \quad \beta^j = (r^j + \beta^j d^j) \cdot (r^j + \beta^j d^j) \\
& \quad \quad \quad p^k = x^j + \beta^j d^j \\
& \quad \quad \text{else } p^k = x^j + \alpha^j d^j \\
& \quad \quad \text{end while} \\
& \quad \text{end for} \\
& \text{end for} \\
\text{Set } x^{k+1} = x^k + \alpha^k p^k; \text{ only Amijo } (\varepsilon) \]
4.6.3. By my lemma; it ALWATS decreases

If near SOSC => superlinear convergence

Weakness: in $\mathbb{R}^k$ is nearly singular

but IPD; direction is poor; and need a lot to compute it. Trust region prevents this...
4. Special topics

4.7 CG-Stehauy

\[ \min \limits_{p \in \mathbb{R}^n} \, w_k(p) = f^k + (a^k + c^k) p^k + \frac{1}{2} p^k B_k^k \]

s.t. \( \| p \| \leq \Delta_k \)

\[ B_k = \nabla^2 \sigma(x^k) \]

- I will show only how to solve the subproblem; the rest of the machinery is the same.
Algorithm 7.2 (CG - Steihaug)

Set $z^0 = 0; s^0 = -A^{-1} b; d^0 = -A f(x^k)$

if $\|z^0\| < \varepsilon_k$ then return $u^k = 0$

for $j = 0, 1, 2, \ldots$

if $d^j \perp z^j$ then

find $\beta$ such that $p_k^j = 2^{j+1}d_t^j$ minimizing $\|z^{j+1}\|$ and satisfying $\|p_k^j\| = \Delta_k$; return $p_k^j$

end if

$\beta^j = \frac{d^j \flat B_k d^j}{d^j \flat B^T T_k B_k d^j}$

$z^{j+1} = 2^{j+1}d^j$

$z^{j+1} = z^j + \alpha p_k^j$

if $\|z^{j+1}\| \geq \Delta_k$

find $\beta > 0$ such that $z^{j+1} = 2^{j+1}d^j$ has up to $\delta_k$ return $p_k^j$

$\delta^j + 1 = \beta^j + \beta^j$

if $\|z^{j+1}\| \leq \Delta_k$

return $p_k^j = 2^{j+1}$

$\beta^j = \frac{r^j T r^j}{r^j T r^j}$

$d^{j+1} = -r^{j+1} + p_k^j$

end if

end for
4.7.3 Observation.

If $\nabla f(x_k) \geq \varepsilon_k$; algorithm terminates at $w_k(x_k) = w_k(x^*)$

$w_k(x_k) = w_k(x^*)$)

Proof: If first test fails =>

$\nabla f(x_k) B_k \nabla f(x_k) < 0$; so search of

walked along $d_0$ goes all the way to

boundary which is exactly CP!! $p_k = -d_k$;

If not, $z_1 = d_0 \cdot d_0 = \frac{r_{i-1}}{d_0 \cdot B_k \cdot d_0}$

$= \frac{\nabla f(x_k) \cdot \nabla f(x_k)}{\nabla f(x_k) \cdot B_k \cdot \nabla f(x_k)}$ .

If $z_1 \leq \theta_k$ => candy point

If not; its truncation is candy point
4.74 Either way \( m(2') \leq m(p^c) \);

So algorithm is globally convergent! =

(Same not true about CG)- LS

\[ \sum \text{If the first of triggers:} \]

\[ m(p^k) \leq m(2^k) \]

\[ \text{Proof:} \]

\[ m(2^{i+2}) = m(2^i) + \] 

\[ + v_f(x^i)^T \left[ 2 \cdot 2^i + \sum d_{ij} B d_{ij} \right] + \frac{1}{2} \sum d_{ij} \]

\[ \sum \text{If } Z = -5\ln (\partial f(x^i)^T d_{ij} + 2^{k} B d_{ij}) \]

at least one of these solves:

\[ \left[ m(2^{i+2}) < m(2^i) \right] \]
4. Special topics

4.8 Preconditioning

Idea: If we apply CG, cannot afford to let it run for n steps if n is large

(1) Worst-case convergence: Same as S.D

\[ \|x^k - x^*\|_A \leq 2 \left( \frac{\sqrt{k+1}}{\sqrt{k+1}+1} \right)^k \|x_0 - x^*\|_A \]

\[ k(A) = \frac{dn}{d} \] : condition number

Idea: cluster eigenvalues

(2) \[ Ax = b \quad Cx = \delta \tilde{x} \]

\[ C^{-1}AC^{-1}x = C^{-1}b \]

But \( \frac{dn_1}{d_1} \ll \frac{dn}{d} \), so convergence is faster
4.8.2 Idea: Rescale TR...

\[ \text{with } \| \varepsilon \| = \| f(x^k) + \nabla f(x^k) \|_p + \frac{1}{2} \| B - B^k \|_p \]

\[ \| Dp \| \leq \Delta \]

\( S \hat{p} = Dp; \quad g = D^{-1} \nabla f(x^k) \)

\( h^k = D^{-T} (\nabla f(x^k))^T D^{-1} \)

(=)

\[ \text{with } \| \varepsilon \| = \| f(x^k) + \nabla f(x^k) \|_p + \frac{1}{2} \| B - B^k \|_p \]

\[ \| \hat{p} \| \leq \Delta \]

And now use [Steilhang CG]
4.8.3 How to choose D?

- Eigenvectors of $B^*$ as clustered as possible.
- Fast convergence

1) Orthogonally Modified Diagonalize Closely

2) Diagonal preconditioning

$$B_k = LL^T - W$$

$L$ is obtained by force $e$-step, pattern drop tolerance.
4.9. Obtain differentiation information

\[ \frac{df}{dx_i} \approx \frac{f(x+3\varepsilon_i) - f(x)}{3\varepsilon} \]

Requires \( n+1 \) function evaluations.

From Taylor's formula, upper bound on

\[ \left\| \frac{\partial f(x_0 + 3\varepsilon_i)}{\partial x_i} \right\| \approx \frac{3\varepsilon}{6} \]

Assume relative error is unit round

\[ \left\| \text{comp}(f+\varepsilon_i) - f(x) \right\| \leq u\cdot\varepsilon \]

\[ \left\| \text{comp}(f+3\varepsilon_i) - f(x + 3\varepsilon_i) \right\| \leq u\cdot3\varepsilon \]

\[ \Rightarrow \text{error} \approx \frac{2u\cdot\varepsilon}{3} + \frac{1}{2} \varepsilon \]
4.9.2 How do I choose $\varepsilon$? by minimizing

$$\frac{2uLf}{\varepsilon^2} + L/2 = 0 \Rightarrow \varepsilon = \sqrt{\frac{2uLf}{L}}$$

Case $\varepsilon = L$

$$\varepsilon^2 = 4uLf = 0 \Rightarrow \sqrt{\frac{2uLf}{L}} = 0$$

No descent $|\varepsilon| < 1^\circ$

Approximately the Hessian

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \varepsilon^2 + O(\varepsilon^3)$$

$$\Rightarrow \text{Need } f(x + 3e_i + 3e_j) \text{ for } \varepsilon = \frac{u(4+1)}{2}$$

$$\left[ \frac{u(4-1)}{2} + u = \frac{u(4+1)}{2} \right] + u = \frac{u(4+3)}{2} = \left[ u^2 + \frac{u^2}{2} \right] + 0$$
4.9.3

\[ Error(3) = \frac{c}{\varepsilon^2} \cdot u + 3.7 \]

\[ \Rightarrow \varepsilon = 3 \approx 0.1 \left( \sqrt[3]{u} \right) \approx 10^{-6}. \]
4.10.2 AD: Forward Mod.

\[ D_p x_i = \frac{d}{dx_i} \hat{p} = \sum_{j=1}^{3} \frac{\partial x_i}{\partial x_j} \hat{p}_j \quad [\text{e.g.} \, 1, 0, 0] \]

\[ D_p x_i = \sum_{j \text{ parent}(i)} \frac{d}{dx_j} \frac{\partial x_i}{\partial x_j} \hat{p}_j \]

[Rush parents kids. Have all kids parents with]

Computed / push
4.10.3 The fact that I have to do it per component suggests effort = O(n)
or, u = time constant found on the other hand: once I hit child

\[ \frac{df}{dx_i} = \sum_{j} \frac{df}{dx_j} \frac{dx_j}{dx_i} \]

AD: Reorder Mode

\[ \frac{f}{dx_i} = 1 \]

Problem: I then to store all variables.

Traverse forward \rightarrow \text{obtain } x \text{ value }

backwards \rightarrow \frac{df}{dx_j}; "adjoint variables"
4.10.4. But, in our sweep, I have all of them.

It seems effort is comparable to for value.

Theorem (Freewaft). There is a reverse mode strategy which will compute the

\( f \) in a time no larger than 5 times

number of operations

The one needed for \( f \)
4. Tricks of the trade

4.10

Automatic differentiation

The initial package, free for download.

\[ f(x) = \frac{x_1 x_2 \sin x_3 + e^{x_1 x_2}}{x_3} \]

\[ x_4 = x_1 \cdot x_2 \]

\[ x_5 = \sin x_3 \]

\[ x_6 = e^{x_4} \]

\[ x_7 = x_4 \cdot x_5 \]

\[ x_8 = x_7 + x_6 \]

\[ x_9 = \frac{x_8}{x_3} \]

\[ x_9 \]

\[ x_3 \]
4. TRICKS OF THE TRADE

4.11 Quasi-Newton methods

Q: What if I am willing to compute the gradients but not the Hessian?

Idea: use secant method

\[ \nabla^2 f(x_j) \cdot (x_k - x_j) = \nabla f(x_k) - \nabla f(x_j) + O(x_k - x_j) \]

\[ \Rightarrow \nabla^2 f(x_j) = \frac{1}{x_k - x_j} \left[ \nabla f(x_k) - \nabla f(x_j) \right] \nabla f(x_k) + O(x_k - x_j) \]

In multiple dimensions, the secant condition becomes:

\[ \nabla^2 f(x_j) \cdot H = \nabla f(x_j) - \nabla f(x_i) + O \left( \left| x_j - x_i \right|^2 \right) \]

\[ j = k-1, \ldots, k-n \]

\[ \Rightarrow \nabla^2 f(x_j) \cdot H = [d_k^{k-n}, \ldots, d_k^{k-2}] \cdot \begin{bmatrix} k-n & \cdots & k-1 \end{bmatrix} \cdot [d_k^{k-n}, \ldots, d_k^{k-1}] \]

\[ B_k \cdot d_j = g_j, \quad j = k-1, \ldots, k-n \]

\[ \Rightarrow \text{No need to compute } B_k^{k-1} \]
4.11.2 Problems: 1) What if $d^j$ is short. (Why is it not a good idea to use a longer one?)
2) What if not symmetric? (PSD)

Idea: define it iteratively; be sure each iterate is of high enough quality.

Discussion: line Search: $B_k p^k = -f^k$

Two ways:

So, I would like $p^k = -H_k f^k$, so it is the inverse I may like to approximate:

$H_k^* q = d^j$ ... and decent

So: two approaches.

1. $\min_B \| B - B_k \|_w$
2. $\min_H \| H - H_k \|_w$

$B = B^T \quad B d^k = g^k$

$H = H^T$; $H = H_k$

$H^* g^k = d^k$
4.11.3

\[ H_s^k = d_k \]

\[ H_{k+1} = (1 - \gamma d_k^T k) H_k (1 - \gamma^2 \tilde{d}_k \tilde{d}_k^T) + \gamma d_k \tilde{d}_k^T \]

\[ s_k = d_k \tilde{d}_k \]

\[ SMW: (A + ab^T)^{-1} = A^{-1} - \frac{A^{-1} a b^T A^{-1}}{1 + b^T A^{-1} a} \]

Notes

1) Rule 1 update: "The minimum modification"

2) Symmetry obvious; so is causality

3) \( s_k > 0 \) \( \Rightarrow \) \( H_k \) is p.d. \( \Rightarrow \) \( H_k \) is p.d.
   (if not, perturb \( d_k, s_k \); it is true near convergence)

4) \( B^k \) obtained by SMW; after working out \( H \)

5) Update can be done in \( O(n^2) \) steps!
   (homework)

6) Memory is \( O(n^3) \)
4.11.4 BFGS methods (w. line search) = all
Compute rank 1 updates of B or H
Properties (a lot of them: SR1; BFGS; DFP, BFG)

1) Need $O(n^2)$ per iteration; $O(n^3)$ storage
2) Need only 1 gradient evaluation per step
   (no Hessian evaluation)
3) Nevertheless; if $n$ is large and $\nabla f(x)$ is sparse: both computation AND storage
   may kill you.
4) However, under some conditions

$$\lim_{k \to \infty} \frac{\| B_k p_k - \nabla f(x_k + p_k) \|}{\| p_k \|} = 0 \implies \text{Superlinear convergence}.$$