1 General

1.1 Wolfe conditions

Finding a step length α_k given a search direction p_k.

- Sufficient Decrease/ Armijo Condition

  Decrease proportional to α_k step length and directional derivative $\nabla f_k^T p_k$:

$$f_{k+1} \leq f_k + c_1 \alpha_k \nabla f_k^T p_k \tag{1}$$
$$\nabla f_k^T p_k \leq 0 \ //by\ construction \tag{2}$$
$$c_1 \in (0,1) \tag{3}$$

- Curvature Condition

  Reject extremely short steps:

$$\nabla f_{k+1}^T p_k \geq c_2 \nabla f_k^T p_k \tag{4}$$
$$\frac{\partial \phi(\alpha_k)}{\partial \alpha_k} \geq \frac{\partial \phi(0)}{\partial \alpha_k} \tag{5}$$
$$c_1, \alpha_k \in (0,1) \tag{6}$$
$$0 < c_1 < c_2 < 1 \tag{7}$$

where :
$$f_k = f(x_k)$$
$$f_{k+1} = f(x_k + \alpha_k p_k)$$
$$\phi(\alpha_k) = f(x_k + \alpha_k p_k)$$
$$\frac{\partial \phi(\alpha_k)}{\partial \alpha_k} = \nabla f_{k+1}^T p_k$$
$$\frac{\partial \phi(0)}{\partial \alpha_k} = \nabla f_k^T p_k //initial\ slope$$

1.2 Strong Wolfe Condition

Modify curvature condition to reject large positive derivative:

$$f_{k+1} \leq f_k + c_1 \alpha_k \nabla f_k^T p_k \tag{8}$$
$$|\nabla f_{k+1}^T p_k| \leq c_2 |\nabla f_k^T p_k| \tag{9}$$
$$0 < c_1 < c_2 < 1 \tag{10}$$

1.3 Form of Search Direction

$$p_k = -B_k^{-1} \nabla f_k \tag{11}$$

B_k symmetric, non-singular, postive definite $\implies$ p_k is a descent direction:

$$\nabla f_k^T (-B_k^{-1} \nabla f_k) < 0 \tag{12}$$

1.4 Goldstein Condition

$$f_k + (1-c)\alpha_k \nabla f_k^T p_k \leq f_{k+1} \leq f_k + c\alpha \nabla f_k^T p_k$$
$$c \in (0, \frac{1}{2})$$

may miss minimizer of f, commonly used in Newtom algo

1.5 Back Tracking Line Search

Using sufficient decrease condition and back tracking for step length search:

Algorithm 1: Line Search

$f, x, d, c_1, \alpha, \beta$: function, x, direction, gradient threshold, initial step length, contraction

α : found step length

```
while f(x + αd) > f(x) + c1 α ∇f(x)^T d do
    α ← α * β
return α
```

2 Conjugate Gradient

2.1 linear method

Assuming unconstrained problem with strict convex quadratic objective function:

$$\frac{1}{2}x^TAx - b^Tx, A \succ 0, A^T = A$$

$\nabla(\frac{1}{2}x^TAx - b^Tx) = Ax - b$, thus $\min_x x^TAx - b^Tx$ transformed to solving $Ax - b = 0$.

let $x_{k+1} = x_k + \alpha_k p_k$, solve for α:

$$\begin{aligned}
&Ax_{k+1} - b = 0\\
&A(x_k + \alpha_k p_k) - b = 0\\
&\alpha_k Ap_k = b - Ax_k\\
&r_k = Ax - b\\
&\alpha_k Ap_k = -r_k\\
&\alpha_k p_k^T Ap_k = -p_k^T r_k\\
&\alpha_k = -\frac{p_k^T r_k}{p_k^T Ap_k}
\end{aligned}$$

2.2 Conjugate Direction

Enforce by construction for search directions linearly independent wrt. A.:

$$(\forall i \neq j) p_i^T Ap_j = 0$$

Properties:

- Residual elimnated one direction at a time, resulting in max of n iterations.
- Optimal if Hessian is diagonal, if not can try preconditioning.
- Current residual is orthogonal to all previous search directions.
- Any set of conjugate directions can be used:
  - eigenvectors
  - Gram-Schmidt
  - conjugate gradient algo.

2.2.1 Termination Steps

Theorem 2.1 (Conjugate Direction Termination)**.** *Conjugate direction algorithm converges to solution $x \in \mathbb{R}^n$ of linear system in n steps. Theorem [T5.1] in Num. Opt. book.*

Proof.

given:

$x_{k+1} = x_k + \alpha_k p_k$

$\alpha_k = -\frac{r_k^T p_k}{p_k^T Ap_k}$

$(\forall i \neq j) p_i^T Ap_j = 0$ [A conjugate]

then:

$x^* - x_0 = \sum_{i=0}^{n-1} \sigma_i p_i$

$p_k^T A(x^* - x_0) = p_k^T A(\sum_{i=0}^{n-1} \sigma_i p_i)$

$\sigma_i = \frac{p_k^T A(x^* - x_0)}{p_k^T Ap_k}$ (from conjugacy)

$x_k = x_0 + \sum_{i=0}^{k-1} \alpha_i p_i$

$p_k^T Ax_k = p_k^T A(x_0 + \alpha_0 p_0 + ..)$

$p_k^T Ax_k = p_k^T Ax_0$ (from conjugacy)

$p_k^T A(x_k - x_0) = 0$

$p_k^T A(x^* - x_0) = p_k^T A(x^* - x_k)$

$p_k^T A(x^* - x_0) = p_k^T (b - Ax_k) = -p_k^T r_k$

$\sigma_i = \frac{-p_k^T r_k}{p_k^T Ap_k} = \alpha_k$

α_k is 1D minimizer of kth coordinate by construction

Conjugate direction algo. terminates in n steps. □

If A is not diagonal, can transform coordinates:

$$\begin{aligned}
S\hat{x} &= x\\
S &= [p_0\ p_1\ ..\ p_{n-1}],\ (\forall i \neq j)\ p_i Ap_j = 0\\
\phi(x) &= \frac{1}{2}x^TAx - b^Tx\\
\phi(\hat{x}) &= \frac{1}{2}\hat{x}^T S^T AS\hat{x} - (S^Tb)^T\hat{x}
\end{aligned}$$

S^TAS *diagonal* $\implies$ can optimize one coordinate at a time

2.2.2 Expanding subspace minimizer

Using conjugate directions to generate sequence $\{x\}$, then:
$r_k^T p_i = 0, \forall i < k$, x_k is minimizer of $\frac{1}{2}x^T Ax - b^T x$ over $\{x | x = x_0 + span\{p_0, ...p_{k-1}\}$

Proof.

$$\tilde{x} = x_0 + \sum_i \sigma_i p_i$$

$\tilde{x}$ minimizes over $\{x_0 + span\{p_0, ...p_{k-1}\}\} \iff r(\tilde{x})^T p_i = 0$
$h(\sigma) = \phi(\tilde{x})$

$$\phi(x) = \frac{1}{2}x^T Ax - b^T x$$

h is also strictly convex quadratic,
with unique σ^* satisfying:

$$\frac{\partial h(\sigma^*)}{\partial \sigma_i} = 0, i = [0, k-1]$$
$$\frac{\partial h(\sigma^*)}{\partial \sigma_i} = \nabla\phi(\tilde{x})^T p_i = 0, i = [0, k-1]$$
$$\nabla\phi(x) = Ax - b = r$$
$$r(\tilde{x})^T p_i = 0, i = [0, k-1]$$

□

$p_i^T r_k = 0, i = [0, k-1]$ via induction:

Proof.

base case of k=1, i=0 : $x_1 = x_0 + \alpha_0 p_0$
minimizes ϕ along p_0
$\implies r_1^T p_0 = (r_0 + \alpha_0 A p_0)^T p_0 = 0$
case: $r_{k-1}^T p_i = 0, i = [0, k-2]$:
$r_k = r_{k-1} + \alpha_{k-1} A p_{k-1}$
$(\forall i \in [0, k-2]) p_i^T r_k = p_i^T r_{k-1} + \alpha_{k-1} p_i^T A p_{k-1}$
(A-conjugacy by construction):
$(\forall i = [0, k-2]) p_i^T A p_{k-1} = 0$
$(\forall i = [0, k-2]) p_i^T r_{k-1} = 0$ (by induction hypothesis)
$p_i^T r_k = 0, i = [0, k-1]$

□

2.3 Conjugate Gradient Method

Idea:

- uses only previous search direction to compute current search direction
- p_k set to linear combination of $-r_k$ and p_{k-1}
- impose $p_k^T A p_{k-1} = 0$

$$p_k = -r_k + \beta_k p_{k-1}$$
$$p_{k-1}^T A p_k = -p_{k-1}^T A r_k + \beta p_{k-1}^T A p_{k-1}$$
$$0 = -p_{k-1}^T A r_k + \beta p_{k-1}^T A p_{k-1}$$
$$\beta = \frac{p_{k-1}^T A r_k}{p_{k-1}^T A p_{k-1}}$$
$$p_0 = -(Ax_0 - b) = -r_0$$

Algorithm 2: Basic Conjugate Gradient Algorithm

```
x_0 = ..
r_0 ← Ax_0 − b
p_0 = −r_0
for k = [0, ..n − 1] do
    if r_k == 0 then
        return x_k
    else
        α_k ← (−r_k^T p_k)/(p_k^T A p_k) = (r_k^T r_k)/(p_k^T A p_k)
        x_{k+1} ← x_k + α_k p_k
        r_{k+1} ← Ax_{k+1} − b = r_k + α_k A p_k
        β_{k+1} ← (p_k^T A r_{k+1})/(p_k^T A p_k) = (r_{k+1}^T r_{k+1})/(r_k^T r_k)
        p_{k+1} ← −r_{k+1} + β_{k+1} p_k
```

p_k and r_k is within krylov subspace:
$K(r_0; k) = span\{r_0, Ar_0, ..A^k r_0\}$
if $r_k \neq 0$:
$r_k^T r_i = 0, i = [0, k-1]$
$span\{r_0, .., r_k\} = span\{r_0, Ar_0, .., A^k r_0\}$
$span\{p_0, .., p + k\} = span\{r_0, Ar_0, .., A^k r_0\}$
$p_k^T A p_i = 0, i = [0, k-1]$
then, $\{x_k\} \to x^*$ in at most n steps.

Simplification:

$$p_{k+1} \leftarrow -r_{k+1} + \beta_{k+1} p_k$$
$$\alpha_k \leftarrow \frac{-r_k^T p_k}{p_k^T A p_k}$$
$$\alpha_k \leftarrow \frac{-r_k^T(-r_k + \beta_k p_{k-1})}{p_k^T A p_k}$$
$$(\forall i = [0, k-1]) r_k^T p_i = 0 \implies \beta_k r_k^T p_{k-1} = 0$$
$$\alpha_k \leftarrow \frac{r_k^T r_k}{p_k^T A p_k} \text{ (simplified)}$$

$$r_{k+1} = r_k + \alpha_k A p_k$$
$$A p_k = \frac{r_{k+1} - r_k}{\alpha_k}$$
$$\beta = \frac{p_k^T A r_{k+1}}{p_k^T A p_k}$$
$$p_k^T A p_k = p_k^T \frac{r_{k+1} - r_k}{\alpha_k} = \frac{-p_k^T r_k}{\alpha} \text{(conjugacy)}$$
$$p_k^T A p_k = -\frac{(-r_k + \beta_k p_{k-1})^T r_k}{\alpha} = \frac{r_k^T r_k}{\alpha} \text{(conjugacy)}$$
$$p_k^T A r_{k+1} = r_{k+1}^T A p_k$$
$$p_k^T A r_{k+1} = r_{k+1}^T \frac{r_{k+1} - r_k}{\alpha_k}$$
$$r_k \in span\{p_k, p_{k-1}\} \; and \; r_{k+1}^T p_i = 0, i = [0, k] \implies$$
$$p_k^T A r_{k+1} = \frac{r_{k+1}^T r_{k+1}}{\alpha_k}$$
$$\beta_{k+1} \leftarrow \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \text{ (simplified)}$$

2.4 Nonlinear Method

Minimize general convex function or nonlinear function. Variants: FR, PR.

2.4.1 FR (Fletcher Reaves)

Modify linear CG by:

- replace residual by gradient of nonlinear objective, $r_k \to \nabla f_k$
- replace α_k computation by a linear search to find approx. minimum along search direction of f

Equivalent to linear CG if objective is strongly convex quadratic.

Linear search for α_k with strong Wolfe condition to ensure p_k's are descent directions wrt. objective function.

2.4.2 PR

Replace β_{k+1} computation in FR with:

$$\beta_{k+1}^{PR} \leftarrow \frac{\nabla f_{k+1}^T (\nabla f_{k+1} - \nabla f_k)}{\nabla f_k^T \nabla f_k}$$
$$\beta_{k+1}^{+} \leftarrow \max(\beta_{k+1}^{PR}, 0)$$

CG Gradient Algo. Property [T.5.3]:

$$(\forall i = [0, k-1]) \; r_k^T r_i = 0$$
$$span\{r_0, .., r_k\} = span\{r_0, Ar_0, .., A^k r_0\}$$
$$span\{p_0, .., p_k\} = span\{r_0, Ar_0, .., A^k r_0\}$$
$$(\forall i = [0, k-1]) \; p_k^T A p_i = 0$$
$$\implies \{x_k\} \text{ converges to } x^* \text{ in at most n steps}$$

Proof by induction to show generated search direcitons are A-conjugate. Then apply Theorem 2.1 to conclude algo terminates within n steps.

3 Quasi Newton

3.1 Concept

properties: $O(n^2)$, self correcting, slightly more iterations than Newton Method, linear convergence order and superlinear rate of convergence

Derivation, using 2nd order model, with B_k SPSD:

$$m_k(p) = f_k + \nabla f_k^T p + \frac{1}{2} p^T B_k p$$

taking gradient wrt. p and solve, assuming minimum exists:

$$0 = \nabla f_k + B_k p$$
$$p = -B_k^{-1} \nabla f_k$$

update equation:

$$x_{k+1} = x_k + \alpha_k p_k$$
$$x_{k+1} = x_k - \alpha_k B_k^{-1} \nabla f_k$$

pick α to satisfy Wolfe conditions

updated model at next iterate x_{k+1}:

$$m_{k+1}(p) = f_{k+1} + \nabla f_{k+1}^T p + \frac{1}{2} p^T B_{k+1} p$$

impose reasonable conditions:

1. gradient of m_{k+1} matches gradient of objective function f at iterate x_k
2. gradient of m_{k+1} matches gradient of objective function f at iterate x_{k+1}

$\nabla m_{k+1}(0) = \nabla f_{k+1} \implies$ condition 2 satisfied

for condition 1 (gradient of m_{k+1} match gradient of objective f at iterate x_k):

$$\nabla m_{k+1}(-\alpha_k p_k) = \nabla f_{k+1} - \alpha_k B_{k+1} p_k = \nabla f_k$$
$$\nabla f_{k+1} - \nabla f_k = \alpha_k B_{k+1} p_k$$
$$y_k = \nabla f_{k+1} - \nabla f_k$$
$$s_k = x_{k+1} - x_k = \alpha_k p_k$$
$$y_k = B_{k+1} s_k \text{ [secant equation]}$$

B_{k+1} SPD $\implies s_k^T B_{k+1} s_k = s_k^T y_k > 0$
f strongly convex $\implies s_k^T y_k > 0$ satisfied for any 2 points x_k and x_{k+1}.
f nonconvex $\implies$ need to enforce $s_k^T y_k$ by Wolfe conditions, and use line search for step length α.

Using line search ensures curvature condition:

Proof.

$s_k = x_{k+1} - x_k = \alpha_k p_k$
$y_k = \nabla f_{k+1} - \nabla f_k$
Wolfe curvature condition:
$\nabla f_{k+1}^T p_k \geq c_2 \nabla f_k^T p_k, c_2 \in (0,1)$
$y_k^T s_k = (\nabla f_{k+1} - \nabla f_k)^T (\alpha_k p_k)$
$y_k^T s_k = \alpha_k (\nabla f_{k+1} - \nabla f_k)^T p^k$
$y_k^T s_k = \alpha_k (c_2 \nabla f_k - \nabla f_k)^T p^k$
$y_k^T s_k = \alpha_k (c_2 - 1) \nabla f_k^T p^k$
$\alpha > 0 \wedge c_2 - 1 < 0 \wedge \nabla f_k^T p_k < 0 (p_k \text{ is a descent dir})$
$\implies y_k^T s_k > 0$

Curvature condition holds when Wolfe line search is performed. □

Choosing approximate Hessian, B:

Force a unique solution among infinite many due to extra degrees of freedom in the matrix, few (n) constraining conditions imposed by secand equation, few (n) constraining conditions of PD. One approach is solving a optimization problem to make row rank update to previous iterate:

$$\min_B ||B - B_k||$$
$$s.t.\ B = B^T$$
$$B s_k = y_k \text{ [secant equation]}$$
$$B \succ 0$$

alternatively, constrain B's inverse, H:

$$\min_H ||H - H_k||$$
$$s.t.\ H = H^T$$
$$H y_k = s_k \quad H_{k+1} y_k = s_k \text{ [secant equation]}$$

Different norms can be used. One choice: weighted Frobenius norm:

$$||A||_W := ||W^{1/2} A W^{1/2}||_F$$
$$||X||_F := (\sum_i \sum_j (X_{ij})^2)^{1/2}$$

3.2 DFP update algorithm

let $\bar{G_k}$ be the average Hessian:

$$\bar{G_k} = \int_0^1 \nabla^2 f(x_k + \tau\alpha_k p_k) d\tau$$

using Taylor's theorem:
$y_k = \bar{G_k} s_k = \bar{G_k}\alpha_k p_k$
let $W = {\bar{G_k}}^{-1}$
solve optimization problem:

$$\begin{aligned} &\min_B \|B - B_k\|_W \\ s.t.\ &W = {\bar{G_k}}^{-1} \\ &Bs_k = y_k \\ &B = B^T \end{aligned}$$

Solution:

$$\begin{aligned} B_{k+1} &= (I - \rho_k y_k s_k^T) B_k (I - \rho_k s_k y_k^T) + \rho_k y_k y_k^T \\ \rho_k &= \frac{1}{y_k s_k} \end{aligned}$$

Computation simplification: optimize for inverse Hessian instead which is used in update of search direction: $p_k = -B_k^{-1}\nabla f_k$
Use Sherman-Morrison-Woodbury formula to obtain inverse.
let $H_k = B_k^{-1}$, then DFP update becomes:

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

This is a rank 2 modification to previous iterate for efficiency.

3.3 BFGS update algorithm

Idea: impose condition on inverse of Hessian instead of Hessian approximation.

let H be approximate inverse of Hessian, then impose:

$$\begin{aligned} H_{k+1} &\succ 0 \\ H_{k+1} &= H_{k+1}^T \\ H_{k+1} y_k &= s_k \end{aligned}$$

solve for H in the optimization problem:

$$\begin{aligned} &\min_H \|H - H_k\|_W \\ s.t.\ &H = H^T \\ &Hy_k = s_k \end{aligned}$$

where $y_k = W s_k$

let W be the average Hessian:
$\bar{G_k} = \int_0^1 \nabla^2 f(x_k + \tau\alpha_k p_k) d\tau$

solve to obtain:

$$\begin{aligned} H_{k+1} &= (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T \\ \rho_k &= \frac{1}{y_k^T s_k} \end{aligned}$$

Deduce from rank 2 update:
$B_{k+1} = B_k + U_k + V_k$
U_k, V_k are rank 1 symmetric matrices:
$U_k = \alpha u u^T, V_k = \beta v v^T$
Let $u = y_k, v = B_k s_k$
$B_{k+1} = B_k + \alpha y_k y_k^T + \beta B_k s_k (B_k s_k)^T$
Impose secant condition:
$B_{k+1} s_k = y_k = B_k s_k + \alpha y_k y_k^T s_k + \beta B_k s_k (B_k s_k)^T s_k$
Solve for α, β:
$\alpha = \frac{1}{y_k^T s_k}$
$\beta = \frac{-1}{(B_k s_k)^T s_k}$
$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k (B_k s_k)^T}{s_k^T B_k^T s_k}$
Use Sherman-Morrison formula to get B^{-1} and optimize with inverse Hessian in order to avoid computing inverses in the actual algorithm:
$B_{k+1}^{-1} = H_{k+1} = (I - \frac{s_k y_k^T}{y_k^T s_k}) H_k (I - \frac{y_k s_k^T}{y_k^T s_k}) + \frac{s_k s_k^T}{y_k^T s_k}$

initial H_0 can be chosen approximately:

- finite diferences
- αI

Algorithm 3: BFGS Algorithm

$H_0, x_0, \epsilon > 0$: inverse Hessian approx., initial point, convergence tolerance
x : solution

$k \leftarrow 0$
//until convergence
while $||\nabla f_k|| > \epsilon$ **do**
  $p_k \leftarrow -B_k^{-1}\nabla f(x_k) = -H_k \nabla f(x_k)$
  $\alpha_k \leftarrow \text{LineSearch}(p_k, f, \nabla f)$
  $x_{k+1} \leftarrow x_k + \alpha_k p_k$
  $s_k \leftarrow x_{k+1} - x_k$
  $y_k \leftarrow \nabla f_{k+1} - \nabla f_k$
  $\rho_k \leftarrow \frac{1}{y_k^T s_k}$
  //update inverse Hessian approximation for next iterate
  $H_{k+1} \leftarrow (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T$
  $k \leftarrow k + 1$
return x

Cost: $O(n^2)$ + cost of eval $f(.)$ + cost of eval $\nabla f(.)$.
Order of convergence: superlinear, worse than Newton but more computationally efficient than Newton.

Using Sherman-Morrison-Woodbury formula to obtain Hessian update equation,

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

, but is it more efficient to use the inverse version.

Proper line search is required so that BFGS algo captures curvature information.

Inaccurate line search can be used to reduce computation cost.

3.4 Sherman-Morrison

A^{-1} *exists* $\implies$

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^T A^{-1}}{1 + v^T A^{-1} u} \iff 1 + v^T A^{-1} u \neq 0$$

4 Trust Region Methods

idea:

- models local behaviour of the objective function (eg: 2nd order Taylor series)
- set local region to explore, then simultaneously find direction and step size to take
- region size adaptively set using results from previous iterations, using ratio of function value decrease vs. model value decrease
- step may fail due to inadequately set region, which need to be adjusted
- superlinear convergence when approximate model Hessian is equal to true Hessian

using 2nd order Taylor series model with symmetric matrix approximating Hessian

$$\min_{p\in\mathbb{R}^n} m_k(p) = f_k + g_k^T p + \frac{1}{2}p^T B_k p,\ st.\ ||p|| \leq \Delta_k$$

$$\begin{aligned}\Delta_k &:= \text{trust region radiius}\\ g_k &= \nabla f(x_k)\\ B_k &\succeq 0\\ B_k^T &= B_k\end{aligned}$$

full step is ($p_k = -B_k^{-1}g_k$) taken when $B \succ 0$ and $||B_k^{-1}g_k|| \leq \Delta_k$

evaluate goodness of model with $\frac{\text{actual function value change}}{\text{predicted model change}}$:

$$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k) \geq 0}$$

$$action \leftarrow \begin{cases}\text{expand trust region} & ,\rho_k \approx 1\ (agreement)\\ \text{shrink trust region} & ,\rho_k < 0 + \text{thresh}\\ \text{keep trust region} & ,o/w\end{cases}$$

Algorithm 4: Trust Region Algorithm

```
k ← 0
while ||∇f_k|| > ε do
    p_k ← argmin_p f_k + g_k^T p + ½ p^T B_k p, st. ||p|| ≤ Δ_k
    ρ_k = (f(x_k) − f(x_k + p_k)) / (m_k(0) − m_k(p_k)) //ratio test
    //change trust region for next iterate
    if ρ_k < γ(: ¼) then
        Δ_{k+1} ← α(: ¼)Δ_k //shrink
    else if ρ_k > β(: ¾) and ||p_k|| == Δ_k then
        Δ_{k+1} ← min(2Δ_k, Δ̂) //expand
    else
        Δ_{k+1} ← Δ_k //keep
    //accept/reject for current iterate
    if ρ_k > η(:∈ [0, ¼)) then
        x_{k+1} ← x_k + p_k
    else
        x_{k+1} ← x_k
```

Minimizer of the 2nd order Taylor series satisfy the following as a global solution for trust region iff:

$$\begin{aligned}(B + \lambda I)p^* &= -g\\ \text{complementary slackness:}&\\ \lambda(\Delta - ||p^*||) &= 0\\ (B + \lambda I) &\succeq 0\\ \lambda &\geq 0\end{aligned}$$

$\lambda \geq 0 \implies p$ parallel to negative gradient of model

Solving 2nd order Taylor series using approx methods, which reduce at least as much as Cauchy Point:

- dogleg (positive definite)
- 2-D subspace minimization (require smallest eigenvalue estimation if not positive definite)
- conjugate gradient based, effective when B large, sparse

Cauchy Point (sufficient reduction in model value): Use 1st order approx. of model and steepest descent direciton to get next iterate, bounded within trust region.

4.1 Cauchy Point

Idea: find an approximate step that would still achieve convergence in a sequence of steps.
Use linear version of the model:

$$p_k^s = \underset{p}{\operatorname{argmin}}\, f_k + g_k^T p$$
$$s.t.\ \|p\| \leq \Delta_k$$

$p_k^s = -\frac{\|\Delta_k\|}{\|g_k\|} g_k$

Solve for step length in the original model with trust region:

$$\tau_k = \underset{\tau \geq 0}{\operatorname{argmin}}\, m_k(\tau p_k^s)$$
$$s.t.\ \|\tau p_k^s\| \leq \Delta_k$$

Cauchy point: $p_k^c = \tau_k p_k^s$

Solutions for τ_k:

$$\begin{cases} g_k^T B_k g_k \leq 0 & , g_k \neq 0 \implies m_k(\tau_k p_k^s) \\ & \textit{monotonically decreasing}, \\ & \tau_k = 1, \\ g_k^T B_k g_k > 0 & , m_k(\tau_k p_k^s)\ \textit{convex} \\ & \textit{quadratic}, \\ & \tau_k = min(1, \frac{\|g_k\|^3}{\Delta_k g_k^T B_k g_k}), \\ & \textit{either : boundary val of } \Delta_k, \\ & \textit{or unconstrained minimizer of} \\ & \textit{quadratic} \end{cases}$$

Cauchy point p_k^c inexpensive to calculate.

[TODO] derive τ_k for case of $g_k^T B_k g_k > 0$.

4.2 Dogleg method

Idea: improve on Cauchy point in considering curvature information. Requires $B \succ 0$.

if $B \succ 0$:
$p^B = -B^{-1} g$
$p^* = p^B$ if $\Delta \geq \|p^B\|$
$p^U = \frac{-g^T g}{g^T B g} g$ (intermediate point along direction of steepest descent)
Resulting algo, interpolate between p^U and p^B:

$$\tilde{p}(\tau) = \begin{cases} \tau p^U & , \tau_{\in}[0,1] \\ p^U + (\tau - 1)(p^B - p^U) & , \tau \in [1,2] \end{cases}$$

$$B \succ 0 \implies \|\tilde{p}(\tau)\| \text{increases wrt. } \tau \wedge m(\tilde{p}(\tau)) \text{decreases wrt. } \tau$$

if $\|p^B\| \leq \Delta$: p chosen at p^B
else p chosen at intersection of $\tilde{p}(\tau)$ and trust region boudnary by solving:
$\|p^U + (\tau - 1)(p^B - p^U)\|^2 = \|\Delta^2\|$

$p_k^S = \underset{p}{\operatorname{argmin}}\, f_k + g_k^T p, \|p\| \leq \Delta_k$
$\tau_k = \underset{\tau \geq 0}{\operatorname{argmin}}\, m_k(\tau p_k^S), \|\tau p_k^S\| \leq \Delta_k$
$p_k^S = \frac{-\Delta_k g_k|}{\|g_k\|}$
$p_k^C = \tau_k p_k^S$
$p_k^C = -\tau_k \frac{\Delta_k g_k}{\|g_k\|}$

$$\tau_k = \begin{cases} 1 & , g_k^T B_k g_k \leq 0 \\ min(\frac{\|g_k|^3}{\Delta_k g_k^T B_k g_k}, 1) & , o/w \end{cases}$$

[TODO: add section on 2D subspace minimization method]

4.3 Iterative Solution

Idea: solve subproblem $\min_{\|p\| \leq \Delta} m(p)$ by applying Newton's method to find λ that matches trust region radius. This is slightly more accurate per step compared to Dogleg. Use $(B + \lambda I)p^* = -g$ to solve $\min_{\|p\| \leq \Delta} m(p)$ for λ.

If $\lambda = 0$ and $(B + \lambda I)p^* = -g, \|p^*\| \leq \Delta$ and $(B + \lambda I) \succeq 0$: return λ
Else: find λ s.t. $(B + \lambda I) \succeq 0$ and $\|p(\lambda)\| = \Delta, p(\lambda) = -(B + \lambda I)^{-1} g$. Solve and return λ.

Solve $\|p(\lambda)\| - \Delta = 0, \lambda > \lambda_1$ via Newton's method (root finding). Approx. this to nearly a linear problem for easy solving:

Algorithm 5: Subproblem Algo

```
for l = 0, 1, .. do
    solve B + λ^l I = R^T R
    R^T R p_l = -g
    R^T q_l = p_l
    λ^{l+1} ← λ^l + (||p_l|| / ||q_l||)^2 (||p_l|| − Δ) / Δ) check λ ≥ λ_1
```

4.4 Trust Region Newton CG Method

Idea: use trust region algo for the outer iteration, use iterative CG based algorithm to solve for inner optimization problem.

Outer iteraion: Algo 4.

Inner optimization problem:

$$\min_{p\in\mathbb{R}^n} m(p) = f + g^T p + \frac{1}{2}p^T Bp$$
$$s.t.\ \|p\| \leq \Delta$$

Algorithm 6: Trust Region Newton-CG Subproblem (CG Steinhaug)

$\epsilon_k = \eta_k\|\nabla f_k\|$
$z_0 = 0$
$r_0 = \nabla f_k$
$d_0 = -r_0 = \nabla f_k$
if $\|r_0\| < \epsilon_k$ **then**
  return $p_k = z_0 = 0$
for $j = 0, 1, ..$ **do**
  //dir of nonpositive curvature test
  if $d_j^T B_k d_j \leq 0$ **then**
    return $\text{argmin}_{p_k}\, m_k(p_k = z_j + \tau d_j)$ s.t. $\|p_k\| = \Delta_k$
  $\alpha_j = \frac{r_j^T r_j}{d_j^T B_k d_j}$
  $z_{j+1} = z_j + \alpha_j d_j$
  //trust region check
  if $\|z_{j+1}\| \geq \Delta_k$ **then**
    return $p_k : p_k = z_j + \tau d_j$ *s.t.* $\|p_k\| = \Delta_k, \tau \geq 0$
  $r_{j+1} = r_j + \alpha_j B_k d_j$
  if $\|r_{j+1}\| < \epsilon_k = \eta_k\|\nabla f_k\|, \eta_k = min(\frac{1}{2}, \|\nabla f_k\|^{\frac{1}{2}})$ **then**
    return $p_k = z_{j+1}$
  $B_{j+1} = \frac{r_{j+1}^T r_{j+1}}{r_j^T r_j}$
  $d_{j+1} = -r_{j+1} + B_{j+1} d_j$

ϵ_j can be chosen similarly as in Line Search Newton-CG method, where $\{\eta_k\}$ is the forcing sequence.

5 Proximal Algorithm

Idea:

- reliance on easy to evaluate proximal operators
- separability allows parallel evaluation
- generalization of projection based algorithms

$$prox_{\lambda f}(v) = \underset{x}{\operatorname{argmin}}\, f(x) + \frac{1}{2\lambda}\|x - v\|_2^2$$

Resolvent of subdifferential operator:

$$z = prox_{\lambda f}(x) \implies z \in (I + \lambda \partial f)^{-1}(x)$$
$$(I + \lambda \partial f)^{-1} := \text{resolvent of operator } \partial f$$

5.1 Proximal Gradient Method

Solve $\min_x g(x) + f(x)$, where f, g are closed, convex functions and f differentiable.

Idea: combine Proximal point algorithm with gradient projection method. Gradient step on f, and proximal point algo step on g.

$$x^* = prox_{\lambda^k g}(x^k - \lambda^k \nabla f(x^k))$$
$$= \underset{x}{\operatorname{argmin}}\, g(x) + \frac{1}{2\lambda^k}\|x - (x^k - \lambda^k \nabla f(x^k))\|_2^2$$

tradeoff between g and and gradient step

$g = I_C(x) \implies$ projected gradient step

$g = 0 \implies$ gradient descent

$f = 0 \implies$ proximal minimization

Relation to Pixed Point:
x^* is a fixed point solution of $\min_x g(x) + f(x)$ iff $0 \in \nabla f(x^*) + \partial g(x^*)$ iff $x^* = (I + \lambda \partial g)^{-1}(I - \lambda \nabla f)(x^*)$

Forward Euler, Backward Euler stepping is same as the proximal gradient iteration, $prox_{\lambda^k g}(x^k - \lambda^k \nabla f(x^k))$

5.2 Accelerated Proximal Gradient Method

Introduce extrapolation:

$$y^{k+1} = x^k + w^k(x^k - x^{k-1})$$
$$x^{k+1} = prox_{\lambda^k g}(y^{k+1} - \lambda^k \nabla f(y^{k+1}))$$
$$w^k \in [0, 1)$$

Example: $w^k = \frac{k}{k+3}, w^0 = 0, \lambda^k \in (0, 1/L], L :=$ Lipschitz constant of ∇f, *or* λ^k found via line search. Line search for λ^k (Beck and Teboulle):

Algorithm 7: Proximal Gradient Algorithm

```
1 f̂(x, y) := f(y) + ∇f(y)^T(x − y) + (1/2λ)‖x − y‖_2^2
2 while True do
3 |   z = prox_{λg}(y^k − λ∇f(y^k))
4 |   if f(x) ≤ f̂(z, y^k) then
5 |   |   break
6 |   λ = βλ
7 return λ^k := λ, x^{k+1} := z
```

5.3 Types of Proximal Operators

- quadratic functions

$$f = \frac{1}{2}\|.\|_x^2 \implies prox_{\lambda f}(v) = (\frac{1}{1+\lambda})v$$
$$f = \frac{1}{2}x^T Ax + b^T x + c, A \in S_+^n \implies$$
$$prox_{\lambda f}(v) = (I + \lambda)^{-1}(v - \lambda b)$$

- unconstrained problem: use gradient methods such as Newton, Quasi-Newton
- constrained: use projected subgradient for non-smooth, projected gradient or interior method for smooth
- separable function: if scalar, may be solved analytically, eg: L1 norm separable to:

$$f(x) = |x| \implies prox_{\lambda f}(v) = \begin{cases} v - \lambda, & v \geq \lambda \\ 0, & |v| \leq \lambda \\ v + \lambda, & v \leq -\lambda \end{cases}$$

$$f(x) = -log(x) \implies prox_{\lambda f}(v) = \frac{v + \sqrt{v^2 + 4\lambda}}{2}$$

- general scalar function
  - localization: using a subgradient oracle and bisection algorithm
  - twice continuously differentiable: guarded Newton method
- polyhedra constraint, quadratic objective: solve as QP problem
  - duality to reduce number of variables to solve if possible
  - gram matrix caching

- affine constraint($Ax = b$): use pseudo-inverse, A^+:

$$\Pi_C(v) = v - A^+(Av - b)$$
$$A \in R^{m\times n}, m < n \implies A^+ = A^T(AA^T)^{-1}$$
$$A \in R^{m\times n}, m > n \implies A^+ = (A^TA)^{-1}A^T$$

- hyperplane constaint($a^Tx = b$):

$$\Pi_C(v) = v + (\frac{b - a^Tb}{\|a\|_2^2})a$$

- halfspace

$$\Pi_C(v) = v - \frac{max(a^Tv - b, 0)}{\|a\|_2^2}a$$

- box($l \le x \le u$)

$$\Pi_C(v)_k = min(max(v_k, l_k), u_k)$$

- probability simplex($1^Tx = 1, x \ge 0$)
  bisection algo on ν:

$$\Pi_C(v) = (v - \nu 1)_+$$
$$\text{intial } [l_k, u_k] = [\max_i v_i - 1, \max_i vi]$$

  analytically solve when bounded in between 2 adjacent v¨i's

- cones (κ: proper cone)
  problem of the form:

$$min_x \|x - v\|_2^2$$
$$s.t. : x \in \kappa$$

$$x \in \kappa$$
$$v = x - \lambda$$
$$\lambda \in \kappa^*$$
$$\lambda^Tx = 0$$

  - cone $C = \mathbb{R}^n_+$

$$\Pi_C(v) = v_+$$

  - 2nd order cone $C = \{(x,t) \in \mathbb{R}^{n+1} : \|x\|_2 \le t\}$

$$\Pi_C(v,s) = \begin{cases} 0, & \|v\|_2 \le -s \\ (v,s), & \|v\|_2 \le s \\ \frac{1}{2}(1 + \frac{s}{\|v\|_2})(v, \|v\|_2), & \|v\|_2 \ge |s| \end{cases}$$

  - PSD cone S^n_+

$$\Pi_C(V) = \sum_i (\lambda_i)_+ u_i u_i^T$$
$$V = \sum_i \lambda_i u_i u_i^T \ (eigendecomp)$$

  - exponential cone
    Todo

- pointwise supremum
  - max function
  - support function
- norms
  - L2
  - L1
  - L-inf
  - elastic net
  - sum of norms
  - matrix norm
- sublevel set
- epigraph
- matrix functions Todo

6 Smoothness

β smoothness:

$\frac{\beta}{2}\|x\|_2^2 - f(x)$ is convex (fit quadratic on top)

$f(y) \leq f(x) + \nabla f(x)^T(y-x) + \frac{\beta}{2}\|y-x\|_2^2$

$\implies$ f is smooth

α strong convexity:

$f(x) - \frac{\alpha}{2}\|x\|_2^2$ is convex (fit quadratic below)

$f(y) \geq f(x) + \nabla f(x)^T(y-x) + \frac{\alpha}{2}\|y-x\|_2^2$

f may not be smooth

Smoothness near optimal point: there always exists a step (length $< \frac{2}{\beta}$) that is a descent direction.

Property of f wrt. optimality value:
f is β-smooth $\implies$

$$\frac{1}{2\beta}\|\nabla f(x)\|_2^2 \leq f(x) - f(x^*) \leq \frac{\beta}{2}\|x-x^*\|_2^2$$

f is α-strongly convex $\implies$

$$\frac{\alpha}{2}\|x-x^*\|_2^2 \leq f(x) - f(x^*) \leq \frac{1}{2\alpha}\|\nabla f(x)\|_2^2$$

Co-coercivity for β-smooth f:

$$\left(\nabla f(x) - \nabla f(y)\right)^T (x-y) \geq \frac{1}{\beta}\|\nabla f(x) - \nabla f(y)\|_2^2$$

Coercivity for α-strongly convex f:

$$\left(\nabla f(x) - \nabla f(y)\right)^T (x-y) \geq \alpha\|x-y\|_2^2$$

6.1 Oracle based lower bounds

Lipschitz convex function: error=$\mathcal{O}(\frac{1}{\sqrt{T}})$
Smooth convex function: $\epsilon = \mathcal{O}(\frac{1}{T^2})$
Smooth + strongly convex function: $\epsilon = \mathcal{O}(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1})$

6.2 Gradient descent rate of convergence

β-smooth f: $\mathcal{O}(\frac{1}{T})$
β-smooth + α-strongly convex f: $\mathcal{O}((\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1})^T)$

6.3 Gradient descent with momentum

f β-smooth, α-s.c.:

$$x_1 = y_1 = x_{init}$$
$$y_{t+1} = x_t - \frac{1}{\beta}\nabla f(x_t)$$
$$x_{t+1} = \left(1 + \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right) y_{t+1} - \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right) y_t$$

f β-smooth:

$x_{t+1} = x_t - \eta\nabla f(x_t)$ [normal GD]
momentum, nesterov accel.:
$d_{t+1} = \gamma_{t+1}(x_{t+1} - x_t)$
$x_{t+1} = (x_t + d_t) - \eta\nabla f(x_t + d_t)$
$\epsilon = \mathcal{O}\left(\frac{1}{T^2}\right)$

7 Common Algorithms

7.1 Projected Gradient

$$y_{t+1} = x_t - \eta g_t, g_t \in \partial f(x_t)$$
$$x_{t+1} = proj_X(y_{t+1})$$
$x^* = proj_X(x^*)$ [fixed point at optimality]

7.2 Proximal Gradient

$$prox_{\eta h}(y) = \underset{x}{\operatorname{argmin}}\, h(x) + \frac{1}{2\eta}\|x-y\|_2^2$$
$$x_{t+1} = prox_{\eta h}(x_t - \eta \nabla f(x_t))$$

Special case of projected gradient:
let $h(x) = I_X(x) \implies prox_{\eta h} = proj_X$
where $I_X(x) = \begin{cases} 0 & , x \in X \\ +\infty & , o/w \end{cases}$

7.2.1 Common proximal operators

$h(x) = \|x\|_1$:

$$(prox_{\eta h}(x))_i = \begin{cases} x_i - \eta & , x_i \geq \eta \\ 0 & , |x_i| \leq \eta \\ x_i + \eta & , x_i \leq -\eta \end{cases}$$
$$= \max(|x_i| - \eta, 0)\ sign(x_i)$$

$h(x) = \frac{1}{2}x^TQx + q^Tx + q_0, Q \succeq 0$:

$$prox_{\eta h} = (I + \eta Q)^{-1}(x - \eta q)$$

$h(x) = \sum_i h_i(x_i)$:

$$(prox_{\eta h}(x))_i = prox_{\eta h_i}(x_i) \text{ [parallelism]}$$

Composite function:
$\min_x f(x) = g(x) + h(x)$, g smooth, h not smooth but has prox. operator. Convergence due to non-smooth function: $\mathcal{O}(\frac{1}{\sqrt{T}})$

7.3 L1 regularization with subgradient descent

$$\min_x \|Ax-y\|_2^2 + \|x\|_1$$
$$g = \|Ax-y\|_2^2$$
$$h = \lambda\|x\|_1$$
$$\partial_x h(x) = \begin{cases} -1 & , x_i < 0 \\ [-1,1] & , x_i = 0 \\ 1 & , x_i > 0 \end{cases}$$
$$x_{t+1} = x_t - \eta(\nabla g(x_t) + \lambda \partial h(x))$$
$$x_{t+1} = x_t - \eta(2A^T(Ax-y) + \lambda z)$$
$$z = \begin{cases} -1 & , x_i < 0 \\ [-1,1] & , x_i = 0 \\ 1 & , x_i > 0 \end{cases}$$

Convergence: $\epsilon = \mathcal{O}\left(\frac{1}{\sqrt{T}}\right)$

7.4 ISTA (with proximal gardient)

$$\min_x g(x) + h(x), g\ \beta\text{-smooth}$$
$$g(x) = \|Ax-y\|_2^2$$
$$h(x)\lambda\|x\|_1$$
$$x_{t+1} = prox_{\eta h}(x_t - \eta \nabla g(x_t))$$
$$g\ \beta\text{-smooth, select } \eta = \frac{1}{\beta}$$
$$x_{t+1} = prox_{\frac{1}{\beta}\lambda\|\dot{\|}_1}(x_t - \frac{1}{\beta}\nabla g(x_t))$$
$$x_{t+1} = \underset{x}{\operatorname{argmin}}\, \lambda\|x\|_1 + \frac{\beta}{2}\|x - (x_t - \frac{1}{\beta}\nabla g(x_t))\|_2^2$$
$$\nabla g(x_t) = 2A^T(Ax-y)$$

Convergence: $\mathcal{O}\left(\frac{1}{\sqrt{T}}\right)$

7.5 FISTA

$$\min_x g(x) + h(x), g\ \beta\text{-smooth}$$
$$\mu_0 = 0$$
$$\mu_t = \frac{1 + \sqrt{1 + 4\mu_t^2 - 1}}{2}$$
$$\gamma_t = \frac{1 - \mu_t}{\mu_{t+1}}$$
$$x_1 = z_1 = \text{arbitrary}$$
$$x_{t+1} = (1-\gamma_t)x_{t+1} + \gamma_t z_t$$
$$z_{t+1} = prox_{\frac{\lambda}{\beta}\|\dot{\|}_1}(x_t - \frac{1}{\beta}\nabla g(x_t))$$

Convergence: $\epsilon = \mathcal{O}\left(\frac{1}{T^2}\right)$

8 Subgradient Method

TODO

9 Large Scale Optimization

1. memory reduction techniques with quasi-Newton methods.

2. Inexact Newton methods with superlinear local convergence properties, indefinite Hessian, Hessian-free impl.

3. function that is partial separable to use smaller subspaces.

9.1 Inexact Newton Methods

Iterative method with local convergence properties.
Residual: $r_k = \nabla^2 f_k p_k + \nabla f_k$
Termination condition for iterative solver: $\|r_k\| \leq \eta_k \|\nabla f_k\|, \eta_k \in (0,1)$, where η_k is the forcing sequence.
Convergence properties: see Thm 7.1, 7.2 (Nocedal)

2 such algorithms, using CG as the iterative solver:

1. Line Search Newton-CG

2. Trust Region Newton-CG (Section 4.4)

9.1.1 Line Search Newton-CG

Augment original CG algorithm with line search, curvature check and forcing function check.

Algorithm 8: Line Search Newton-CG Algo

$x_0 = ..$
for $k = 0, 1, ..$ **do**
  $\eta_k = \min(\frac{1}{2}, \|\nabla f_k\|^{\frac{1}{2}})$
  $z_0 = 0, r_0 = \nabla f_k, d_0 = -r_0 = -\nabla f_k$
  for $j = 0, 1, ..$ **do**
    //choose direction
    if $d_j^T B_k d_j \leq 0$ **then**
      if $j = 0$ **then**
        break $p_k = -\nabla f_k$
      else
        break $p_k = z_j$
    $\alpha_j = \frac{r_j^T r_j}{d_j^T B_k d_j}$
    $z_{j+1} = z_j + \alpha_j d_j$
    $r_{j+1} = r_j + \alpha_j B_k d_j$
    //forcing function constraint
    if $\|r_{j+1}\| < \epsilon_k = \eta_k \|\nabla f_k\|$ **then**
      break $p_k = z_{j+1}$
    $B_{j+1} = \frac{r_{j+1}^T r_{j+1}}{r_j^T r_j}$
    $d_{j+1} = -r_{j+1} + B_{j+1} d_j$
  $\alpha_k =$ Wolfe/Goldstein/Armijo step search
  $x_{k+1} = x_k + \alpha_k p_k$

9.2 Partially Separable Functions

Decompose into sum of element functions with a large number of variables being linearly independent.

Ideally: $\nabla f(x) = \sum_i \nabla f_i(x), \nabla^2 f(x) = \sum_i \nabla^2 f_i(x)$.

Use Hessian approximations for individual element functions.

Use chain rule to transform into msaller spaces and back to original space via compactifying matrices.

eg: $\nabla^2 f(_1(x) \approx U_i^T B_{[1]} U_i$ mapping $B_{[1]}$ back into correct position in the full Hessian approximation.

$$x_{[i]} = U_i x$$
$$f(x) = \sum_i \phi_i(U_i x)$$
$$\nabla f_i(x) = U_i^T \nabla \phi_i(U_i x)$$
$$\nabla^2 f_i(x) = U_i^T \nabla^2 \phi_i(U_i x) U_i$$

Use a Quasi-Newton to approximate $\nabla^2\phi_i$ which require smaller resources by updating a new approximation $B_{[i]}$ iteratively:

$$\begin{aligned} s_{[i]} &= x^+_{[i]} - x_{[i]} \\ y_{[i]} &= \nabla\phi_i(x^+_{[i]}) - \nabla\phi_i(x_{[i]}) \\ B_{[i]} &\approx \nabla^2\phi_i(U_i x) = \nabla^2\phi_i(x_{[i]}) \end{aligned}$$

Map back to place in original matrix:

$\nabla^2 f_i(x) \approx U_i^T B_{[i]} U_i$

Complete Hessian approximation then becomes:

$B = \sum_i U_i^T B_{[i]} U_i$

which can be used in trust region algo to get p_k:

$B_k p_k = -\nabla f_k$.

10 Quadratic Programming

TODO

11 Sequential QP

TODO

12 Nonlinear Penalty Based Methods

12.1 Quadratic Penalty

Inexact, iterative method. Idea: add a weighted quadratic penalty term per constraint to the objective. Form a sequence of probems with increasing weights to penalty terms.

$\min_x f(x)\ s.t.\ c_i(x) = 0, i \in \mathcal{E}$
$Q(x;\mu) = f(x) + \frac{\mu}{2}\sum_i c_i(x)^2$
$\mu_k \to +\infty\ as\ k \to +\infty$

Use unconstrained optimization techniques and use previous iterate as starting point for current iteration.

May diverges when penalty weight if not big enough.

Ill conditioning of Hessian when μ_k becomes large at the minimizer. CG and Quasi-Newton not suitable. Newton's method may still have problems: numerical inaccuracies when solving linear equations to calculate step, and Taylor series 2nd order approximation onyl accurate in a small neighbourhood of x so it may not generate useful step unless μ_{k+1} is chosen slightly larger than μ_k.

let $A(x)^T = [\nabla c_i(x)]_{i\in\mathcal{E}}$
$Q(x;\mu_k) = f(x) + \sum_{i\in\mathcal{E}} \mu_k c_i(x_k)$
$\nabla_x Q(x;\mu_k) = \nabla f(x) + \sum_{i\in\mathcal{E}} \mu_k c_i(x_k)\nabla c_i(x_k)$
$\nabla^2_{xx} Q(x;\mu_k) = \nabla^2 f(x) + \sum_{i\in\mathcal{E}} \mu_k c_i(x)\nabla^2 c_i(x) + \mu_k A(x)^T A(x)$
$\nabla^2_{xx} Q(x;\mu_k) \approx \nabla^2_{xx} L(x;\lambda^*) + \mu_k A(x)^T A(x)$ near minimizer where $\mu_k A(x)^T A(x)$ may be ill conditioned as $\mu_k \to \infty$

For the case of equality and inequality constrained problem:

$$\min_x f(x)$$
$$s.t.\ c_i(x) = 0, i \in \mathcal{E}$$
$$s.t.\ c_i(x) \geq 0, i \in \mathcal{I}$$
$$Q(x;\mu) = f(x) + \frac{\mu}{2}\sum_i c_i(x)^2 + \frac{\mu}{2}\sum_i max(-c_i(x), 0)^2$$

Algorithm 9: Quadratic Penalty Algo

```
x_0^s = ..
μ_0 > 0
{τ_k} → 0
for k = 0, 1, .. do
    solve x_k = argmin_x Q(x; μ_k) with initial guess x_k^s with termination condition ‖∇_x Q(x; μ_k)‖ ≤ τ_k
    if final convergence criteria passes then
        return approx solution
    update μ_{k+1} : μ_{k+1} > μ_k (eg: chosen adaptively)
    update x_{k+1}^s ← x_k
```

12.2 Nonsmooth Penalty Methods

12.2.1 L1 Penalty

Exact method, not rely on updating stratefor penalty parameter. Penalty function:

$$\phi(x;\mu) = f(x) + \mu\sum_{i\in\mathcal{E}}\|c_i(x)\| + \mu\sum_{i\in\mathcal{I}}\max(0, -c_i(x))$$

If x^* is a strict local minimizer then x^* is a local minimizer of $\phi(x;\mu)$ for $\forall\mu > \mu^*$, where $\mu^* = ||\lambda^*||_\infty = \max_{i\in\mathcal{I}\cup\mathcal{E}}|\lambda_i^*|$, λ_is being lagrange multiplers. This generates enough penalization when going into infeasible resion to keep x^* as local minimizer.

Nonsmooth function, but has directional derivatives $D(\phi(x;\mu);p)$ along p.

$\hat{x}$ is a stationary point for $\phi(x;\mu)$ if $D(\phi(\hat{x},\mu);p) \geq 0, \forall p$

$\hat{x}$ is a stationary point of measure of infeasibility $h(x) = \sum_{i\in\mathcal{E}}|c_i(x)| + \sum_{i\in\mathcal{I}} max(0, -c_i(x))$ if $D(\phi(\hat{x},\mu);p) \geq 0, \forall p$

Algorithm 10: Classic L1 Penalty Algo

```
x_0^s = ..
μ_0 > 0
τ = ..
for k = 0, 1, .. do
    solve approximately x_k = argmin_x φ(x; μ_k) with initial guess x_k^s
    if h(x_k) ≤ τ then
        return x_k
    update μ_{k+1} : μ_{k+1} > μ_k
    update new starting point: x_{k+1}^s
```

Practical L1 penalty method: linearize constraints, replace nonlinear objective by a quadratic. Convert to a smooth QP problem by new slack variables and solve with standard QP solver.

$$\begin{aligned}\min_{p,r,s,t} & f(x) + \frac{1}{2}p^T W p + \nabla f(x)^T p \\ & + \mu \sum_{i\in\mathcal{E}}(r_i + s_i) + \mu \sum_{i\in\mathcal{I}} t_i \\ s.t. & \nabla c_i(x)^T p + c_i(x) = r_i - s_i, i \in \mathcal{E} \\ & \nabla c_i(x)^T p + c_i(x) \geq -t_i, i \in \mathcal{I} \\ & r, s, t \geq 0\end{aligned}$$

Possible issue: μ_k is critical to success of iteration. Strategy: choose μ slightly larger than $\|\lambda^*\|_\infty$ but that depends on lagrange multipler estimate.

12.3 Augmented Lagrangian

Add estimated Lagrange multipliers term to quadratic penalty formulation. Differs from standard Lagrangian with the extra quadratic terms.
$Q(x, \lambda; \mu) = f(x) - \sum_i \lambda_i c_i(x) + \frac{\mu}{2}\sum_i c_i(x)^2, \forall i \in \mathcal{E}$

Idea: fix μ_k, λ_k at current iteration, minimized wrt. x, update estimate of Lagrangian variables.

$x_k = \text{argmin}_x L(x, \lambda_k; \mu_k)$
Optimality condition for uncstained minimization:
$0 \approx \nabla_x L(x_k, \lambda_k; \mu_k) = \nabla f(x_k) - \sum_{i\in\mathcal{E}}(\lambda_{k,i} - \mu_k c_i(x_k))\nabla c_i(x_k)$
$\lambda_i^* \approx \lambda_{k,i} - \mu_k c_i(x_k), \forall i \in \mathcal{E}$
Update of Lagrangian variables follow naturally.
$\lambda_{k+1,i} \leftarrow \lambda_{k,i} - \mu_k c_i(x_k), \forall i \in \mathcal{E}$
$c_i(x_k) \approx \frac{\lambda_{k,i} - \lambda_i^*}{\mu_k}$
If $\lambda_{k,i}$ close to λ_i^*, then $c_i(x_k) \ll \frac{1}{\mu_k}$, this is better than quadratic penalty method's $c_i(x_k) \approx \frac{-\lambda_i^*}{\mu_k}, \forall i \in \mathcal{E}$

Even with modest penalty weight, μ_k, $c_i(x_k)$ is more like 0 which is require for optimality condition, this allows convergence without having $\mu_k \to +\infty$. Less ill conditioning and initial starting point is less critical.

Tolerance τ_k can be chosen on infeasibility, $\sum_i |c(x_k)|$, and can be increased if infeasibility reduction in insufficient.

Algorithm 11: Augmented Lagrangian Algo

```
x_0^s = ..
λ_0 = ..
τ_0 > 0
μ_0 > 0
for k = 0, 1, .. do
    solve x_k = argmin_x L(x, λ_k; μ_k) starting
      at x_k^s with termination condition
      ‖∇_x Q(x, λ_k; μ_k)‖ ≤ τ_k
    if c_i(x) = 0, ∀i ∈ E then
        return x_k
    λ_{k+1,i} ← λ_{k,i} − μ_k c_i(λ_{k,i})
    update μ_{k+1} : μ_{k+1} > μ_k (eg: chosen
      adaptively)
    update x_{k+1}^s ← x_k
    update τ_{k+1}^s ← ...
```

Augmented Lagrangian Approaches:

- Bound Constained

- Linearly Constrained Lagrangian
- Unconstrained

12.3.1 Bound Constrained

Introduce slack variables for inequalities, transform into equality constrained problem with variable bound constraints.

$\min_x f(x),\ s.t.\ c_i(x) = 0\ \forall i,\ l \le x \le u$

Solve the subprobem using augmented Lagrangian with nonlinear gradient probjection method by fixing λ, μ.
$\min_x L(x, \lambda_k; \mu_k) = f(x) - \sum_i \lambda_i c_i(x) + \frac{\mu}{2} \sum_i c_i(x)^2$

1st order necessary conditions for x:
$x - Proj_{[l,u]}(x - \nabla_x L(x, \lambda; \mu), l, u) = 0$

TODO: [insert algo 17.4 nocedal]

12.3.2 Linearly Constrained Lagrangian

Idea: lienarize constraints and generate a step by minimizing Lagragian.

$$\min_x F_k(x) = f(x) - \sum_i \lambda_i^k \bar{c}_i^k(x) + \frac{\mu}{2} \sum_i [\bar{c}_i^k(x)]^2$$
$$s.t.\ c(x_k) + A_k(x - x_k) = 0$$
$$\bar{c}_i^k(x) = c_i(x) - (c_i(x_k) + \nabla c_i(x_k)^T (x - x_k))$$
$$l \le x \le u$$

where A_k is the constraint Jacobian.

12.3.3 Unconstrained

Idea: use proximal point approach for inequality constrained problems.

$$\min_x F(x) = \max_{\lambda \ge 0} \{f(x) - \sum_{i \in \mathcal{I}} \lambda_i c_i(x)\}$$
$$= \begin{cases} f(x) & , if\ x\ feasible \\ \infty & , o/w \end{cases}$$

$$\begin{cases} x \text{ feasible} & , (c_i(x) < 0),\ select\ \lambda_i\ large,\ \lambda_j = 0, \forall j \ne i \\ & \implies F(x) \to \infty \\ x \text{ infeasible} & , (c_i(x) \ge 0), \forall i \in \mathcal{I},\ select\ (\forall i \in \mathcal{I}) \lambda_i = 0 \\ & \implies F(x) = f(x) \end{cases}$$

Then $\min_x F(x) = \min_{x\ feasible} f(x)$

Use an approximation $\hat{F}(x; \lambda^k, \mu_k)$ to F that is smooth for practical optimization:

$\hat{F}(x; \lambda$
$, \mu_k) = \max_{\lambda \ge o} \{f(x) - \sum_{i \in \mathcal{I}} \lambda_i c_i(x) - \frac{1}{2\mu_k} \sum_{i \in \mathcal{I}} (\lambda_i - \lambda_i^k)^2\}$

This is a bound constrained quadratic program in *lambda*, separable in individual components λ_i. The additional proximal term helps staying close to λ_i^k.

We perform maximization explicitly:

$$\lambda_i = \begin{cases} 0 & , -c_i(x) + \frac{\lambda_i^k}{\mu_k} \le 0 \\ \lambda_i^k - \mu_k c_i(x) & , o/w \end{cases}$$

Substitute λ_i back in $\hat{F}$:

$$\hat{F}(x; \lambda^k, \mu_k) = f(x) + \sum_{i \in \mathcal{I}} \psi(c_i(x), \lambda_i^k; \mu_k)$$

$$\psi(t, \sigma; \mu) = \begin{cases} -\sigma t + \frac{\mu}{2} t^2 & , t - \frac{\sigma}{\mu} \le 0 \\ -\frac{1}{2} \mu \sigma^2 & , o/w \end{cases}$$

New iterate x_k is found by minimizing $\hat{F}$:

$$x_K = \text{argmin}_x \hat{F}(x; \lambda_k, \mu_k)$$

New Lagrangian multiplier update rule:

$$\lambda_i^{k+1} = \begin{cases} 0 & , -c_i(x) + \frac{\lambda_i^k}{\mu_k} \le 0 \\ \lambda_i^k - \mu_k c_i(x) & , o/w \end{cases}$$

13 Linear System, Iterative

13.1 Idea

Solve $Ax = b$
Idea: split A into $M + N$ and use back/forward substitutions to solve:
$Mx^{k+1} = Nx^k + b$

13.2 Jacobi

13.2.1 Splitting

$$\begin{aligned} A &= M - N \\ M &= D = Diagonal \\ N &= M - A = D - A = -(L + U) \\ L &= \text{Lower left} \\ U &= \text{Upper right} \\ (M - N)x &= b \\ Mx &= Nx + b \\ Dx &= -(L + U)x + b \\ x &= D^{-1}(b - (L + U)x) \end{aligned}$$

13.2.2 Algo

$$x^{k+1} = D^{-1}(b - (L + U)x^k)$$

13.3 Gauss-Seidel

Use most recently updated values

13.3.1 Splitting

$$\begin{aligned} A &= M - N \\ M &= D + L \\ N &= -U \\ Mx &= Nx + b \\ (D + L)x &= -Ux + b \\ x &= (D + L)^{-1}(b - Ux) \end{aligned}$$

13.3.2 Algo

$$\begin{aligned} x^{k+1} &= (D + L)^{-1}(b - Ux^k) \\ (D + L)x^{k+1} &= (b - Ux^k) \\ (I + D^{-1}L)x^{k+1} &= D^{-1}(b - Ux^k) \\ x^{k+1} &= D^{-1}(b - Ux^k) - D^{-1}Lx^{k+1} \\ x^{k+1} &= D^{-1}(b - Lx^{k+1} - Ux^k) \end{aligned}$$

$$(\forall i)\ x_i^{k+1} \leftarrow \frac{1}{A_{ii}}\left(b_i - \sum_{j<i} A_{ij}x_j^{k+1} - \sum_{j>i} A_{ij}x_j^k\right)$$

13.4 SOR(Successive Over-Relaxation)

$$\begin{aligned} x^{k+1} &\leftarrow x^k + w(x_{GS}^{k+1} - x^k) \\ x^{k+1} &\leftarrow (1 - w)x^k + wx_{GS}^{k+1} \end{aligned}$$

$$\begin{cases} w > 1 & \text{, over-relaxation} \\ w \in (0, 1) & \text{, under-relaxation} \\ w = 1 & \text{, Gauss-Seidel} \end{cases}$$

13.4.1 Splitting

$$\begin{aligned} &A = M - N \\ &M = \frac{1}{w}D + L \\ &N = (\frac{1}{w} - 1)D - U \\ &Mx = Nx + b \\ &x = M^{-1}(Nx + b) \\ &(\frac{1}{w}D + L)x^{k+1} = ((\frac{1}{w} - 1)D - U)w^k + b \\ &x^{k+1} = (D + wL)^{-1}(((1 - w)D - wU)x^k + wb) \end{aligned}$$

13.4.2 Algo

$$\begin{aligned} &(\frac{1}{w}D + L)x^{k+1} = ((\frac{1}{w} - 1)D - U)w^k + b \\ &(\frac{1}{w}I + D^{-1}L)x^{k+1} = ((\frac{1}{w} - 1)I - D^{-1}U)x^k + D^{-1}b \\ &\frac{1}{w}x^{k+1} = -D^{-1}Lx^{k+1} + ((\frac{1}{w} - 1)I - D^{-1}U)x^k + D^{-1}b \\ &x^{k+1} = w(D^{-1}(b - Lx^{k+1} - Ux^k) + (\frac{1}{w} - 1)x^k) \\ &x^{k+1} = wD^{-1}(b - Lx^{k+1} - Ux^k) + (1 - w)x^k \\ &x^{k+1} = wx_{Gauss-Seidel}^{k+1} + (1 - w)x^k \end{aligned}$$

14 Factorization

14.1 LU

By design:

$$\begin{aligned} U &= MA \\ M^{-1}U &= A \\ LU &= A \\ L &= M^{-1} = (M_{n-1}..M_1)^{-1} \\ L &= M_1^{-1}..M_{n-1}^{-1} = L_1..L_{n-1} \end{aligned}$$

With partial pivoting in rows:

$$\begin{aligned} PAx &= Pb \\ LU &= PA \\ LUx &= Pb \\ y &= Ux \ //solve\ for\ y \\ Ly &= Pb \\ Ux &= y \ //Solve\ for\ x \end{aligned}$$

Algorithm 12: LU with Partial Pivot

```
P ← [0 .. n−1]//permute rows
S ← 0//row swaps
for col in 0 .. n do
    let rowmax ←col
    for row in col + 1 .. n do
        if A_{row,col} > A_{rowmax,col} then
            rowmax ← row
    swap(A_{rowmax,:}, A_{col,:})
    swap(P_{rowmax}, P_{col})
    S ← S + rowmax != col ? 1 : 0
    if A_{col,col} < ε_{tol} then
        raise degenerate matrix error
    for row in col + 1 .. n do
        r ← A_{row,col}/A_{col,col}
        A_{row,col} ← r //save value for L
        //reduction for U
        for c in col + 1 .. n do
            A_{row,c} ← A_{row,c} − rA_{col,c}
return A, P, S
```

L stored in lower left triangle of A with implicit diagonal ones.

U stored in upper right triangle of A.

Determinant:

$LU = PA$
$det(A) = det(P^{-1}LU)$
$det(A) = det(P^{-1})det(L)det(U)$
$det(P^{-1}) = (-1)^S$
where S is the number of row exchanges from the permutation matrix P
$det(A) = (-1)^S \prod_i L_{i,i} \prod_i U_{i,i}$
With implicit diagonal ones of L in Algo 12 and storage of U in returned A:
$det(A) = (-1)^S \prod_i A_{i,i}$

14.2 QR

$A = QR$
$A = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix}$
Q: m x m, orthogonal
R: n x n, upper triangular
m $\geq n$

Reduced QR factorization of A:
$A = Q_1 R$
Q_1: m x n
R: n x n, upper triangular

$Ax = b$
$Q_1 Rx = b$
$Rx = Q_1^{-1} b = Q_1^T b$
solve via subsubstitution for x

14.3 Least Squares Problem

Orthogonal matrix is norm preserving for $L2$:
$\|Qx\|_2^2 = x^T Q^T Q x = x^T x = \|x\|_2^2$

$$
\begin{aligned}
&Ax = b \\
&r = b - Ax \\
&\|r\|_2^2 = \|b - Ax\|_2^2 \\
&\|r\|_2^2 = \|b - Q \begin{bmatrix} R \\ 0 \end{bmatrix} x\|_2^2 \\
&\|r\|_2^2 = \|Q^T b - Q^T Q \begin{bmatrix} R \\ 0 \end{bmatrix} x\|_2^2 \\
&\|r\|_2^2 = \|Q^T b - I \begin{bmatrix} R \\ 0 \end{bmatrix} x\|_2^2 \\
&Q^T b = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \\
&\|r\|_2^2 = \|c_1 - Rx\|_2^2 + \|c_2\|_2^2
\end{aligned}
$$

Solve $c_1 - Rx = 0 \implies \|r\|_2^2 = \|c_2\|^2$

14.3.1 Householder Reflection Method

Idea: relfect vector so all coordinates but one disappear.
let $|\alpha| = \|x\|_2$
$u \leftarrow x - \alpha e_1$
$v = \frac{u}{\|u\|}$
$Q = I - 2vv^T$

$$
\begin{aligned}
&(I - 2\frac{vv^t}{v^T v})(I - 2\frac{vv^t}{v^T v})^T \\
&= (I - 2\frac{vv^t}{v^T v})(I - 2\frac{vv^t}{v^T v}) \\
&= I - 4\frac{vv^t}{v^T v} + 4\frac{vv^t vv^T}{v^T vv^T v} \\
&= I - 4\frac{vv^t}{v^T v} + 4\frac{vv^t}{v^T v} \\
&= I \\
&\implies (I - 2\frac{vv^t}{v^T v}) \text{ orthongal, symmetric}
\end{aligned}
$$

$Q^T = Q^{-1} = Q$

$$Qx = \begin{bmatrix} \alpha \\ 0 \\ .. \\ 0 \end{bmatrix}$$

A: m x n, $m \geq n$

$$Q_1 A = \begin{bmatrix} \alpha_1 & .. \\ 0 & A' \end{bmatrix}$$

$$Q_x = \begin{bmatrix} I_{k-1} & 0 \\ 0 & Q'_k \end{bmatrix}$$

Transform to upper triangular matrix:
$R = Q_t .. Q_1 A$
$(Q_t .. Q_1)^T R = (Q_t .. Q_1)^T (Q_t .. Q_1) A$
let $Q = (Q_t .. Q_1)^T = (Q_1 .. Q_t)$
$QR = QQ^T A$
$QR = (Q_1 .. Q_t) R = A$

Solve $Ax = b$
$QRx = b$
$Rx = Q^{-1} b$
$Rx = Q^T b$
$Rx = (Q_1 .. Q_t)^T b$
$Rx = (Q_t^T .. Q_1^T) b$
$Rx = (Q_t .. Q_1) b$
Solve with backsubstitution

15 Eigenvalues Eigenvectors

15.1 Power Iteration

$x_k = Ax_{k-1} = A^k x_0$
$x_k = A^k \sum_j \alpha_j v_j$
$x_k = \sum_j \alpha_j A^k v_j$
$x_k = \sum_j \alpha_j \lambda_j^k v_j$
$x_k = \alpha_1 \lambda_1^k v_1 + \sum_{j \neq 1} \alpha_j \lambda_j^k v_j$
$x_k = \lambda_1^k (\alpha_1 v_1 + \sum_{j \neq 1} \alpha_j \frac{\lambda_j^k}{\lambda_1^k} v_j$
$\lim_{k \to \infty} |(\frac{\lambda_j}{\lambda_1})^k| \to 0$
$\lim_{k \to \infty} x_k = \lambda_1^k \alpha_1 v_1$

Algorithm 13: Power Iteration Algo

```
let x_0 ← ...
for k = 0.. do
    y_{k+1} = A x_k
    x_{k+1} = y_{k+1} / ||y_{k+1}||_∞ //normalize
```

Issue: failure to converge to linear comb. of eigenvectors corresponding to same eigenvalue.

15.2 Inverse Iteration

Inverse iteration: use factorization and $eig(A^{-1}) = \frac{1}{eig(A)}$

For computing smallest eigenvalue:

Algorithm 14: Inverse Iteration

```
let x_0 ← ...
for k = 0.. do
    Solve for y_{k+1} with factorization:
    A y_{k+1} = x_k
    x_{k+1} = y_{k+1} / ||y_{k+1}||_∞ //normalize
```

converge to eigenvector corresponding to smallest eigenvalue of A = largest eigenvalue of A^{-1}

15.3 Shifting

Rayleigh Quotient for better eigenvalue estimation:
$\lambda = \frac{x^T A x}{x^T x}$
$\sigma_{smallest}(A - \sigma I) = \lambda - \sigma$, λ: eigenvalue of A closest to σ
use eigenvalue estimate from Rayleigh Quotient in inverse iteration to get better eigenvector

Algorithm 15: Inverse Iteration with Shifting

```
let x_0 ← guess of eigenvector
for k = 0.. do
    σ_{k+1} = (x_k^T A x_k) / (x_k^T x_k) //shift
    solve y_{k+1}: (A − σ_{k+1} I) y_{k+1} = x_k //next vector iterate
    x_{k+1} = y_{k+1} / ||y_{k+1}||_∞ //normalize
```

15.4 Simultaneous Iteration

Algorithm 16: Simultaneous Iteration Algo

```
let X_0: n × p matrix of rank p (columns of linearly independent vectors)
for k = 1.. do
    X_k = A X_{k-1}
```

let $S_0 = span(X_0)$
$S :=$ invariant subspace spanned by p eigenvectors associated with largest eigenvalues of A.
$S_k = A^k S_0$
columns of X_k form a basis for S_k
$S_k \to S$

Issue: X_k beomes ill-conditioned bassi for subspace S_k.
One solution: apply normalization such as QR factorization of columns of X_k:

Algorithm 17: Orthogonal Iteration Algo

```
let X_0: n × p matrix of rank p (columns of linearly independent vectors)
for k = 1.. do
    Q̂_k R_k = X_{k-1} //normalize
    X_k = A Q̂_k //gen next iterate
```

$\hat{Q}_k : n \times p$ orthogonal
$R_k : p \times p$ upper triangle

15.5 QR Iteration

Reference: section 4.5.6 from Heath

$A\hat{Q} = \hat{Q}B$
B triangular

Define $A_k = \hat{Q}_k^H A \hat{Q}_k$
$\hat{Q}_k$ orthogonal $\implies \sigma(A) = \sigma(\hat{Q}_k^H A \hat{Q}_k)$
$\hat{Q}_k A_k = A\hat{Q}_k$
$A\hat{Q}_k = \hat{Q}_k A_k$
Shur Form of A:
A_k triangular / block triangular when it converges

Avoiding explicit computation of X_k and its factorization $\hat{Q}_k R_k = X_k$:
let $X_0 = I$
factorize: $\hat{Q}_0 R_0 = X_0$
$\hat{Q}_0 = R_0 = I$
$X_1 = A\hat{Q}_0 = AI = A$
factorize: $\hat{Q}_1 R_1 = X_1 = A$
using $A_k = \hat{Q}_k^H A \hat{Q}_k$:
$A_1 = \hat{Q}_1^H A \hat{Q}_1$
$A_1 = \hat{Q}_1^H \hat{Q}_1 R_1 \hat{Q}_1$, where $\hat{Q}_1 R_1 = A$
$A_1 = R_1 \hat{Q}_1$ (notice reverse of $\hat{Q}_1 R_1 = A$)
$X_2 = A\hat{Q}_1$

$X_2 = \hat{Q}_1 \hat{Q}_1^H A \hat{Q}_1$
$X_2 = \hat{Q}_1 A_1$

instead of factorizing: $\hat{Q}_2 R_2 = X_2 = A\hat{Q}_1$
factorize this: $Q_2 R_2 = A_1$:
$X_2 = \hat{Q}_1 Q_2 R_2$
let $\hat{Q}_2 = \hat{Q}_1 Q_2$
$X_2 = \hat{Q}_2 R_2$

$A_2 = \hat{Q}_2^H A \hat{Q}_2$
$A_2 = \hat{Q}_2^H \hat{Q}_1 R_1 \hat{Q}_2$, where $A = \hat{Q}_1 R_1$
since $\hat{Q}_2 = \hat{Q}_1 Q_2$
$A_2 = Q_2^H \hat{Q}_1^H \hat{Q}_1 R_1 \hat{Q}_1 Q_2$
$A_2 = Q_2^H R_1 \hat{Q}_1 Q_2$
since $R_1 \hat{Q}_1 = A_1$
$A_2 = Q_2^H A_1 Q_2$
since $Q_2 R_2 = A_1$
$A_2 = Q_2^H Q_2 R_2 Q_2$
$A_2 = R_2 Q_2$ (notice reverse of $Q_2 R_2 = A_1$)

$A_3 = \hat{Q}_3^H A \hat{Q}_3$
since $\hat{Q}_1 R_1 = A$
$A_3 = \hat{Q}_3^H \hat{Q}_1 R_1 \hat{Q}_3$
since $\hat{Q}_3 = \hat{Q}_1 Q_2 Q_3$
$A_3 = Q_3^H Q_2^H \hat{Q}_1^H \hat{Q}_1 R_1 \hat{Q}_1 Q_2 Q_3$
$A_3 = Q_3^H Q_2^H R_1 \hat{Q}_1 Q_2 Q_3$
since $A_1 = R_1 \hat{Q}_1$
$A_3 = Q_3^H Q_2^H A_1 Q_2 Q_3$
since $Q_2 R_2 = A_1$
$A_3 = Q_3^H Q_2^H Q_2 R_2 Q_2 Q_3$
$A_3 = Q_3^H R_2 Q_2 Q_3$
since $R_2 Q_2 = A_2$
$A_3 = Q_3^H A_2 Q_3$
factorize $Q_3 R_3 = A_2$
$A_3 = Q_3^H Q_3 R_3 Q_3$
$A_3 = R_3 Q_3$ (notice reverse of $Q_3 R_3 = A_2$)
...

Thus X_k, and factoriaztion $\hat{Q}_k R_k = X_k$ is not calculated.
Convergence is checked on A_k for (block) triangular form, in which the eigenvalues can be read off.

Algorithm 18: QR Iteration Algo

```
1 let A_0 = A
2 for k = 1.. do
3 |   Q_k R_k = A_{k-1} //normalize
4 |   A_k = R_k Q_k //gen next iterate
```

TODO 2