10-725/36-725: Convex Optimization **Spring 2015**

Lecture 24: April 13

Lecturer: Ryan Tibshirani *Scribes: Calvin McCarter, Anurag Kumar, Logan Brooks*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

24.1 Administrative Notes

Today: finish up Frank-Wolfe.

Next week, have option of doing one/two advanced topics.

Options:

- exact path-following methods (scattered votes)
- nonconvex (many votes)
  - not going to give general algorithms
- fast stochastic algorithms (a few votes)

24.2 Finishing up Frank-Wolfe / conditional gradient method

We thought of Frank-Wolfe as a method like projected gradient descent; it addresses problems where we optimize some smooth convex function $f(x)$ subject to x is in some constraint set C.

- Projected gradient descent: forms quadratic approximation to f, minimizes, projects onto C;
- Frank-Wolfe: uses linear approximation, optimizes within C rather than projecting.

For each step of Frank-Wolfe:

1. Consider a first-order approximation of f at $x^{(k-1)}$: $g(s) = f(x^{(k-1)}) + \nabla f(x^{(k-1)})^T(s - x^{(k-1)})$.
2. Find a constrained minimizer $s^{(k-1)} \in \arg\min_{s \in C} g(s) = \arg\min_{s \in C} \nabla f(x^{(k-1)})^T s$ (the equality holds because $f(x^{(k-1)})$ and $\nabla f(x^{(k-1)})$ are constant wrt s).
3. Let $x^{(k)} = (1 - \gamma_k)x^{(k-1)} + \gamma_k s^{(k-1)}$, a convex combination of $x^{(k-1)}$ and $s^{(k-1)}$, or, equivalently, let $x^{(k)} = x^{(k-1)} + \gamma_k(s^{(k-1)} - x^{(k-1)})$, a step towards $s^{(k-1)}$ from $x^{(k-1)}$.
4. Let γ_k above be a step size; the default choice for fixed γ_k's is $\gamma_k = \frac{2}{(k+1)}$; we can also choose them with backtracking line search.

How do we find s above when C is a norm ball?

- Find a subgradient of the dual norm, scale: $\arg\min_{\|s\| \leq t} \nabla f(x^{(k-1)})^T s = -t \cdot \partial \left\| \nabla f(x^{(k-1)}) \right\|_*$
- Finding this subgradient is often cheaper than projecting onto the norm ball.

Some examples of Frank-Wolfe update for regularization

- L^1 regularization: Problems of the form $\max_x f(x)$ subject to $\|x\|_1 \leq t$. The update rules in this case are simply

$$i_{k-1} \in \arg\max_{i=1,..p} \left| \nabla_i f(x^{k-1}) \right|$$
$$x^{(k)} = (1-\gamma_k) x^{(k-1)} - \gamma_k t \cdot \text{sign}(\nabla_{i_{k-1}} f(x^(k-1))) \cdot e_{i_{k-1}}$$

- L^p regularization: Problems of form $\max_x f(x)$ subject to $\|x\|_p \leq t$ for $1 \leq p \leq \infty$, we have. In this case we can choose

$$s_i^{(k-1)} = -\alpha \cdot \text{sign}(\nabla f_i(x^{(k-1)})) \cdot \left| \nabla f_i(x^{(k-1)}) \right|^{p/q}, \quad i = 1, ...n$$

  where α is a constant such that $\left\| s^(k-1) \right\|_q = t$, after which usual Frank-Wolfe update follows. This is in general much simpler than projection onto l_p ball.

- Trace norm regularization: Problems of form $\min_X f(X)$ subject to $\|X\|_{\text{tr}} \leq t$. We can choose

$$S^{(k-1)} = -t \cdot uv^T$$

  where u, v are leading left, right singular vectors of $\nabla f(X^{k-1})$. Projection onto the trace norm ball requires SVD and prox also requires SVD. Frank-Wolfe updates are subgradient of operator norm and are much cheaper to do. Calculating the SVD gives us the leading left and right singular vectors of $\nabla f(X^{(k-1)})$, which is what Frank-Wolfe needs, but also all of the rest; it is much harder to do.

24.2.1 Constrained and Lagrange forms

We are often solving for many values of the regularization tuning parameter:

- t in constrained form, or
- λ in penalized form.

The problem are equivalent and often we just choose the one which is easier to solve. So We should compare Frank-Wolfe with

- projected gradient descent, projecting onto C, using the constrained form, or
- proximal gradient descent, using the penalized form,

and choose the easiest method.

Summary:

- L^1 norm: Frank-Wolfe and projection both $O(n)$; don't really gain much with Frank-Wolfe
- L^p norm ($p \geq 1$): Frank-Wolfe is still $O(n)$, and we have an explicit formula for its updates; projection and prox are not generally directly computable
- Trace norm: projection and prox are much more expensive than Frank-Wolfe updates
- Many other regularizers, e.g., special polyhedra or cone constraints, sum-of-norms (group-based) regularization, and atomic norms: yield efficient Frank-Wolfe updates; see [Jaggi, 2013].

Good question from Sami: quadratic programming: fairly easy and closed form with Frank-Wolfe, that requires inverse of Q; efficiency depends on if Q is structured.

24.2.2 The limitations of Frank-Wolfe

Frank-Wolfe appears to have the same convergence rate as projected gradient ($O(1/\epsilon)$ rate) in theory; however, in practice, even in cases where each iteration is much cheaper computationally, it can be slower than first-order methods to converge to high accuracy.

Two things to note:

- The Frank-Wolfe method is not a descent method.
- Frank-Wolfe has a hard time getting to the high-accuracy regime.

24.2.3 Frank-Wolfe Duality Gap

The Frank-Wolfe iterates admits a very natural duality gap; this fact was featured very prominently in Martin Jaggi's PhD thesis [Jaggi, 2011], although it may have been known earlier.

Recall that each Frank-Wolfe step finds a minimizer

$$s^{(k-1)} \in \arg\min_{s \in C} f(x^{(k-1)}) + \nabla f(x^{(k-1)})^T (s - x^{(k-1)}) = -\arg\max_{s \in C} \nabla f(x^{(k-1)})^T (x^{(k-1)} - s);$$

just the max here is a (bound on the) duality gap, as well as a (bound on the) suboptimality gap:

$$f(x^{(k-1)}) - f^* \leq \max_{s \in C} \nabla f(x^{(k-1)})^T (x^{(k-1)} - s),$$

where f^* is the optimal value for the primal problem; this is an upper bound on the primal suboptimality.

Proof:

$$\begin{aligned}
f(s) &\geq f(x^{(k-1)}) + \nabla f(x^{(k-1)})^T (s - x^{(k-1)}) \quad \text{(applying first-order condition for convexity)} \\
f^* = \min_{s \in C} f(s) &\geq f(x^{(k-1)}) + \min_{s \in C} \nabla f(x^{(k-1)})^T (s - x^{(k-1)}) \quad \text{(minimizing both sides over } s \in C) \\
f(x^{(k-1)}) - f^* &\leq \max_{s \in C} \nabla f(x^{(k-1)})^T (x^{(k-1)} - s) \quad \text{(rearranging).}
\end{aligned}$$

■

Why is this called a duality gap? Let f^* now be the convex conjugate of f, p^* the primal optimal value, and d^* the dual optimal value. Rewrite the original problem in the unconstrained form

$$\min_x f(x) + I_C(x),$$

where

$$I_C(x) = \begin{cases} 0, & x \in C \\ \infty, & x \notin C \end{cases}$$

is the convex $0/\infty$ indicator function of C; the dual (immediately via Fenchel's duality theorem) is

$$\max_u -f^*(u) - I_C^*(-u),$$

where the conjugate of the indicator function is the support function of C,

$$I_C^*(u) = \max_{s \in C} s^T u.$$

The "iterate" duality gap for any primal feasible x and any u is

$$\begin{aligned} \mathsf{idg}(x,u) &= [f(x) + \underbrace{I_C(x)}_{=0}] - [-f^*(u) - I_C^*(-u)] = f(x) + f^*(u) + I_C^*(-u) \\ &\geq x^T u + I_C^*(-u) \quad \text{(Fenchel's inequality)}. \end{aligned}$$

Substituting $x = x^{(k-1)}$ (which is primal feasible), $u = \nabla f(x^{(k-1)})$, this gives

$$\begin{aligned} \mathsf{idg}(x^{(k-1)}, \nabla f(x^{(k-1)})) &\geq (x^{(k-1)})^T \nabla f(x^{(k-1)}) + \max_{s \in C} s^T(-\nabla f(x^{(k-1)})) \quad \text{(sub. in } x, u, I_C^*) \\ &= \max_{s \in C} \nabla f(x^{(k-1)})^T (x^{(k-1)} - s); \end{aligned}$$

this is a lower bound on the iterate duality gap.

Recall the relationship

$$\mathsf{idg}(x,u) = f(x) - f^*(u) = (f(x) - p^*) + (p^* - d^*) + (d^* - f^*(u)) \geq f(x) - p^* = \text{primal suboptimality}.$$

We can use the bounds above on the primal suboptimality gap and the duality gap to devise a stopping rule (stop when the primal suboptimality gap is small).

24.2.4 Convergence analysis

Part of the proof of the result is in the slides; for the rest, see the referenced papers. The Bregman divergence of a continuously differentiable, convex function f on a closed (convex) domain associated with points y and x is the difference between the function's value at y and the first-order Taylor approximation at y when expanding at x [Wikipedia, 2015a]:

$$D_f(y,x) = f(y) - f(x) - \nabla f(x)^T (y - x) \geq 0;$$

where the inequality comes from the first-order conditions for convexity. How far a function departs from its first-order approximations can be interpreted as a measure of its curvature, which motivates the following definition.

The curvature constant of a convex function f is

$$M = \max_{\substack{x,s,y \in C \\ y = (1-\gamma)x + \gamma s \\ \gamma \in [0,1]}} \frac{2}{\gamma^2} \underbrace{\big(f(y) - f(x) - \nabla f(x)^T (y - x)\big)}_{D_f(y,x)};$$

the $\frac{2}{\gamma^2}$ factor is designed to make things work out in the proof.

Theorem 24.1 *Conditional gradient method using step sizes $\gamma_k = 2/(k+1), k = 1, 2, 3...$ satisfies*

$$f(x^{(k)}) - f^* \leq \frac{2M}{k+2} \tag{24.1}$$

Clearly, the convergence rate matches that of projected gradient descent with Lipschitz gradient. But its worth checking how the assumption of bounded curvature compare with the Lipschitz gradient assumption.

Claim: If ∇f is Lipschitz with constant L then $M \leq \text{diam}^2(C) \cdot L$ where $\text{diam}(C) = \max_{x,s \in C} \|x - s\|_2$. This is easily seen through

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

Maximizing over all $y = (1-\gamma)x + \gamma s$ and multiplying by $2/\gamma^2$, we get

$$M \leq \max_{\substack{x,s,y \in C \\ y=(1-\gamma)x+\gamma s}} \frac{2}{\gamma^2} \frac{L}{2} \|y - x\|_2^2 = \max_{x,s \in C} L \|x - s\|_2^2$$

This shows that the assumptions are no stronger than for proximal gradient method. The paper gives an even strong result regarding the suboptimality/duality gap and termination within a certain number of steps.

Please see these scribe notes, which cover the rest of the Frank-Wolfe-related content of this lecture (as well as what is included above).

24.3 Proximal and Projected Newton

24.3.1 Motivation: proximal gradient descent

The proximal gradient descent operates on problems of the form

$$\min_x f(x) = \min_x g(x) + h(x)$$

where g is convex and smooth, while h is convex and non-smooth. The key to this approach is that h is "simple", so that the proximal operator for h

$$\text{prox}_t(\cdot) = \arg\min_z \frac{1}{2t} \|x - z\|_2^2 + h(z)$$

can be applied efficiently. For proximal gradient descent, the updates take the form

$$x^{(k)} = \text{prox}_{t_k}(x^{(k-1)} - t_k \nabla g(x^{(k-1)})),$$

so ∇g must be computable, but the proximal operator only depends on h. Proximal gradient descent also has the same convergence rate as gradient descent has for smooth objectives, so it is applicable for optimization problems when prox is efficient (e.g. when h is separable).

Proximal gradient is motivated by the idea that we can iteratively take the quadratic expansion of g, plus original h, rather than the quadratic expansion of $g + h$. Each update takes the form:

$$x^+ = \arg\min_z \frac{1}{2t} \|x - t\nabla g(x) - z\|_2^2 + h(z) \tag{24.2}$$

$$= \arg\min_z \nabla g(x)^T (z - x) + \frac{1}{2t} \|z - x\|_2^2 + h(z), \tag{24.3}$$

where the quadratic approximation uses the scaled identity $\frac{1}{t}I$ matrix as the Hessian.

Proximal Newton applies a similar technique to Newton's method. In Newton's method the quadratic approximation uses the local Hessian of the objective, rather than $\frac{1}{t}I$ as in gradient descent. Thus, the proximal Newton method replaces $\frac{1}{t}I$ in Equation 24.3 with $\nabla^2 g(x)$.

24.3.2 Proximal Newton method

To derive the proximal Newton method, we define the "scaled proximal mapping" $\text{prox}_H(x)$ as follows:

$$\text{prox}_H(x) = \arg\min_z \frac{1}{2}\|x - z\|_H^2 + h(z)$$

where $\|x\|_H^2 = x^T H x$ defines a norm, given a matrix $H \succ 0$. The scaled proximal mapping coincides with the previous definition when $H = \frac{1}{t}I$.

Then, with initial $x^{(0)}$, we repeat the following updates for $k = 1, 2, 3, \ldots$:

$$\begin{aligned} y^{(k)} &= \text{prox}_{H_{k-1}}\left(x^{(k-1)} - H_{k-1}^{-1}\nabla g(x^{(k-1)})\right) \\ x^{(k)} &= x^{(k-1)} + t_k(y^{(k)} - x^{(k-1)}). \end{aligned}$$

Here we use $H_{k-1} = \nabla^2 g(x^{(k-1)})$, the local Hessian of the smooth portion of the objective. As in the usual Newton's method, t_k is a step size, chosen via backtracking line search.

We can verify that y minimizes a quadratic approximation of g, plus h:

$$\begin{aligned} y &= \arg\min_z \frac{1}{2}\|x - H^{-1}\nabla g(x) - z\|_H^2 + h(z) && (24.4) \\ &= \arg\min_z \nabla g(x)^T(z - x)\frac{1}{2}(z - x)^T H(z - x) + h(z). && (24.5) \end{aligned}$$

The usual Newton update rule is a special case of proximal Newton with $h(z) = 0$. For $H \succ 0$, the scaled proximal map $\text{prox}_H(\cdot)$ still has many of the properties of proximal map [Lee et al., 2012]. For example, the minimizer of 24.5 is unique, so the operator is well-defined.

There is one major difference between proximal Newton and proximal gradient. With proximal gradient, the gradient of g must be computable, but it does not affect the difficulty of computing the prox. With proximal Newton, the difficulty of computing the prox in Equation 24.5 does depend on the smooth component g through its local Hessian $H = \nabla^2 g(x)$. Thus, the efficiency of proximal Newton depends on whether the Hessian H contains structure that can be exploited.

24.3.3 Backtracking line search

Just as Newton's method for smooth problems is not guaranteed to converge if we use full step sizes $t_k = 1, k = 1, 2, 3, \ldots$, proximal Newton requires us to compute step sizes to ensure convergence. Typically, backtracking line search is used. As usual, we have parameters $0 < \alpha \leq 1/2$ for sufficient decrease condition and backtracking parameter $0 < \beta < 1$. We define

$$v = \text{prox}_H\left(x - H^{-1}\nabla g(x)\right) - x$$

as the proximal Newton direction at a given iteration. Starting with full step size $t = 1$, we shrink $t \leftarrow \beta t$ until we have sufficient decrease

$$f(x + tv) \leq f(x) + \alpha t \nabla g(x)^T v + \alpha \left(h(x + tv) - h(x) \right).$$

This scheme is actually different than the backtracking scheme for proximal gradient descent. Recall that backtracking line search had sufficient decrease condition

$$g(x - tG_t(x)) \leq g(x) - t\nabla g(x)^T G_t(x) + \frac{t}{2} \|G_t(x)\|_2^2$$

where for each step size t we must re-compute

$$G_t(x) = \frac{x - \text{prox}_t \left(x - t\nabla g(x) \right)}{t}.$$

The backtracking scheme for proximal Newton avoids computing the prox at each inner backtracking iteration, because doing so is typically much more expensive.

24.3.4 Does this even make sense?

We can write down the proximal Newton, and think that it has analogous properties to proximal gradient, but does it make sense to use? Our motivation for proximal gradient is that we took a problem of the form

$$\min_x g(x) + h(x)$$

and replaced it with a sequence of problems where g is replaced by the quadratic approximation $g(x^{(k-1)}) + \frac{1}{2t_k} \left\| x - (x^{(k-1)} - t_k \nabla g(x^{(k-1)})) \right\|_2^2$, which has the same minimizers as

$$\min_x \frac{1}{2t_k} \|x - b_k\|_2^2 + h(x) = \text{prox}_{h,t_k}(b_k)$$

(where $b_k = x^{(k-1)} - t_k \nabla g(x^{(k-1)})$). Presuming that the prox operator is cheap, we're happy to do it over and over again, iteratively, and will get the minimizer for the original problem.

Proximal Newton is doing the same thing, but we're replacing $g(x)$ with

$$g(x^{(k-1)}) + b_k^T (x - x^{(k-1)}) + (x - x^{(k-1)})^T A_k (x - x^{(k-1)}),$$

so it's not a simple least-squares quadratic, but a general quadratic; why should we think that solving this general-quadratic prox problem is any easier than solving the original problem? It may be the case that it isn't, and using proximal Newton doesn't make any sense. In general, this prox problem can be very hard to solve, which seemingly defeats the spirit of proximal gradient, because the subproblem is not easy to solve.

This is all true; nothing was misleading in the above discussion. For example, if suppose $h(x) = \|x\|_1$; then each subproblem is solving a full lasso problem with the design matrix A_k, which requires another algorithm (e.g., proximal gradient, coordinate descent) to solve. What we're going to hope for is that we'll get a convergence rate like Newton, so that even though we have to solve this hard prox, we only have to solve it a few times before getting to a highly accurate solution. If proximal Newton had the same convergence properties as proximal gradient, we'd be in a lot of trouble, because we've made the prox problem super hard, and need a lot of them. Under broad conditions, though, we do get that proximal Newton has the same convergence rate as Newton in terms of the number of iterations, as we would hope. So while proximal Newton has a lot of similarities with proximal gradient, it is in a different regime: we solve a much harder prox problem, but only a few times.

We rely on having a good algorithm for the inner prox problem; we would never use it if this was not the case. Proximal Newton should not be applied without care. Some well-known examples using prox Newton include `glmnet` and `QUIC` (see subsection 24.3.7).

24.3.5 Convergence analysis

Proximal Newton attains a $O(\log\log(1/\epsilon))$ (quadratic) local convergence rate in terms of the number of outer iterations, which is the same as Newton's method, under the same assumptions as were taken for Newton's method (the original analysis, not the self-concordant analysis) [Lee et al., 2012]:

- $f = g + h$, g, h convex, g twice smooth
- $mI \preceq \nabla g \preceq LI$, $\nabla^2 g$ Lipschitz with parameter M
- $\text{prox}_H(\cdot)$ is exactly evaluable.

Note that each outer iteration involves a scaled prox evaluation!

Theorem 24.2 ***Proximal Newton method convergence****: Proximal Newton with backtracking line search converges globally, and for all $k \geq k_0$:*

$$\left\|x^{(k)} - x^*\right\|_2 \leq \frac{M}{2m}\left\|x^{(k-1)} - x^*\right\|_2^2$$

(local quadratic convergence).

Proof sketch:

- Global convergence: the backtracking condition is satisfied by $t \leq \min\left\{1, \frac{2m}{L}(1-\alpha)\right\}$, so updates converge to zero, which can only happen at the global minimum.
- Local quadratic convergence rate: the pure step $t = 1$ eventually satisfies the backtracking exit condition; show $\|x^+ - x^*\| \leq \frac{M}{2m}\|x - x^*\|_2^2$ by relating to the quadratic norm associated with H and expanding x^+.

24.3.6 Inexact prox evaluations

Unlike proximal gradient, prox evaluations in proximal Newton will almost always be inexact, because the subproblem we're solving generally doesn't have a direct form for the solution, and requires running an optimization algorithm (producing approximate results).

See [Lee et al., 2012] for an analysis of an adaptive stopping criteria for the graphical lasso problem, which carefully quits the inner optimizer when a sufficient level of accuracy is achieved; the practical convergence rate of the adaptive stopping rule and the "stop-when-extremely-accurate rule" are essentially the same in terms of outer iterations, but the adaptive rule is much faster in terms of actual time taken. (See the slides for some figures showing performance comparisons of different methods for logistic lasso regression.)

The adaptive criterion above stops when $\left\|G_{\tilde{f}_{k-1}/M}(z)\right\|_2 \leq \eta_k \left\|G_{f/M}(x^{(k-1)})\right\|_2$ for a specific sequence $(\eta_k)_{k\in\mathbb{N}_1}$ which is proven to give a local superlinear rate. Here, G denotes generalized gradients and $\tilde{f}$ is a quadratic approximation of f; note the relation to the non-prox Newton method inner-problem stopping rule $\|\nabla \tilde{g}_{k-1}(z)\|_2 \leq \eta_k \left\|\nabla g(x^{(k-1)})\right\|_2$.

24.3.7 Usage

Proximal Newton methods are very commonly used in practice when we have very good algorithms for solving the inner prox problems. For example, coordinate descent is a very good algorithm for solving the lasso problem with Gaussian regression loss. However, with logistic regression loss, we cannot write down the coordinate descent updates exactly (we cannot minimize along each coordinate exactly), and are stuck, even though we love all the tricks about coordinate descent.

What we can do, though, and what's done in `glmnet` and other implementations, is make a quadratic approximation to the logistic regression loss that uses the Hessian; each inner subproblem is then a fully dense lasso problem. We solve each inner subproblem with coordinate descent, because coordinate descent is our hammer, and it works well. Even though this inner solver is pretty expensive, it's a good algorithm for that problem, and produces high-accuracy solutions. If done properly, the whole algorithm will converge within 5–10 iterations, if we're lucky, and has a Newton-like convergence once the iterates are in the right region.

The above approach is used by the R package `glmnet` [Friedman et al., 2010] for L^1-penalized generalized linear models. In fact, the authors of the package are so confident in proximal Newton that they don't even use backtracking line search, instead relying on unit steps, which is pretty surprising, but works pretty well.

The quadratic approximation method above is also used in the R package `QUIC` (QUadratic Inverse Covariance) [Hsieh et al., 2011] for estimating large graphical models (graphical lasso).

24.3.8 Proximal quasi-Newton methods

Proximal quasi-Newton methods avoid exactly forming the Hessian $H_{k-1} = \nabla g(x^{(k-1)})$ at each step; this can be helpful not only when computing the Hessian is expensive (in general, computing the Hessian in large problems is prohibitively expensive), but also when it is ill-conditioned (singular or close to singular). See [Lee et al., 2012] (BFGS-type [Wikipedia, 2015b] updates) and [Tseng and Yun, 2009] (blockwise Hessian approximations for smooth + block separable problems) for examples.

24.3.9 Projected Newton

We know that projected gradient method is a special case of proximal methods. Specifically, the problem is $\underset{x}{min}\ g(x)$ subject to $x \in C$ where C is convex set. To solve this using proximal methods we simply define the indicator function $I_C(x)$ for the given convex set and then use this functions as the prox function. The proximal gradient method simply becomes projected gradient method where in each iteration the point is projected on the set C. So do we have a simple projected Newton's method as well ? The updates in this case looks like

$$\begin{aligned} y &= \arg\min_{z \in C} \frac{1}{2} \left\| x - H^{-}1\nabla g(x) - z \right\|_H^2 \\ &= \arg\min_{z \in C} \nabla g(x)^T (z - x) + \frac{1}{2}(z - x)^T H (z - x) \end{aligned}$$

For $H = I$ this is simply a projection of $x - \nabla g(x)$ onto C but not in general. So projected newton does not follow directly from proximal newton method. However, there are special cases where it is easy to perform projected newton. One such case is box constraints problems. Box constraints appear in several problems

such as Nonnegative least squares, Nonnegative KL divergence minimization, support vector machine dual, dual of l_1 penalized problems.

A box constraint problem is simply $\min_x g(x)$ subject to $l \leq x \leq u$. The projected newton method for this problem takes the following approach

- Define the sets

  $$B_{k-1} = \{i : x_i^{(k-1)} \leq l_i + \epsilon \text{ and } \nabla_i g(x^(k-1)) > 0\} \cup \{i : x_i^{(k-1)} \geq u_i - \epsilon \text{ and } \nabla_i g(x^{(k-1)}) < 0\}$$
  $$F_{k-1} = \{1, ...n\} \setminus B_{k-1}$$

  Here B_{k-1} is called the binding set and F_{k-1} is called free set.

- Now consider the submatrix of inverse Hessian along the free variables

  $$S_{k-1} = [(\nabla^2 g(x^{(k-1)}))^-1]_{F_{k-1}}$$

- Newton step along free variables only

  $$x^{(k)} = P_{[l,u]}\left(x^{k-1} - t_k \begin{bmatrix} S_{k-1} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \nabla_{F_{k-1}} g(x^{(k-1)} \\ \nabla_{B_{k-1}} g(x^{(k-1)}) \end{bmatrix}\right)$$

  $P_{[l,u]}$ is the projection onto $[l, u] = [l_1, u_1] \times ....[l_n, u_n]$

Quasi-Newton can also be done here. Thus instead of actual hessian $(\nabla^2 g(x^{(k-1)}))^-1$ in S_{k-1} one can use an iterative approximation. Some references for the projected newton method are given the last slide.

References

[Friedman et al., 2010] Friedman, J., Hastie, T., and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1.

[Hsieh et al., 2011] Hsieh, C.-J., Sustik, M. A., Dhillon, I. S., and Ravikumar, P. K. (2011). Sparse inverse covariance matrix estimation using quadratic approximation. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K., editors, *Advances in Neural Information Processing Systems 24*, pages 2330–2338.

[Jaggi, 2011] Jaggi, M. (2011). *Sparse convex optimization methods for machine learning*. PhD thesis, ETH Zürich.

[Jaggi, 2013] Jaggi, M. (2013). Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 427–435.

[Lee et al., 2012] Lee, J., Sun, Y., and Saunders, M. (2012). Proximal newton-type methods for convex optimization. In *Advances in Neural Information Processing Systems*, pages 836–844.

[Tseng and Yun, 2009] Tseng, P. and Yun, S. (2009). A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 117(1-2):387–423.

[Wikipedia, 2015a] Wikipedia (2015a). Bregman divergence — wikipedia, the free encyclopedia. [Online; accessed 1-May-2015].

[Wikipedia, 2015b] Wikipedia (2015b). Broydenfletchergoldfarbshanno algorithm — wikipedia, the free encyclopedia. [Online; accessed 1-May-2015].