Limited-Memory Quasi-Newton and Hessian-Free Newton Methods for Non-Smooth Optimization

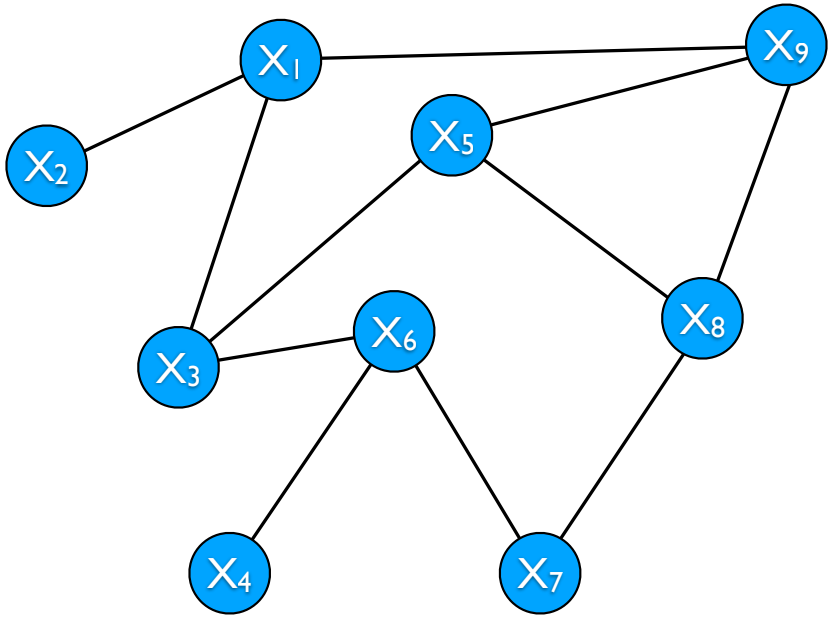
Mark Schmidt

Department of Computer Science
University of British Columbia

December 10, 2010

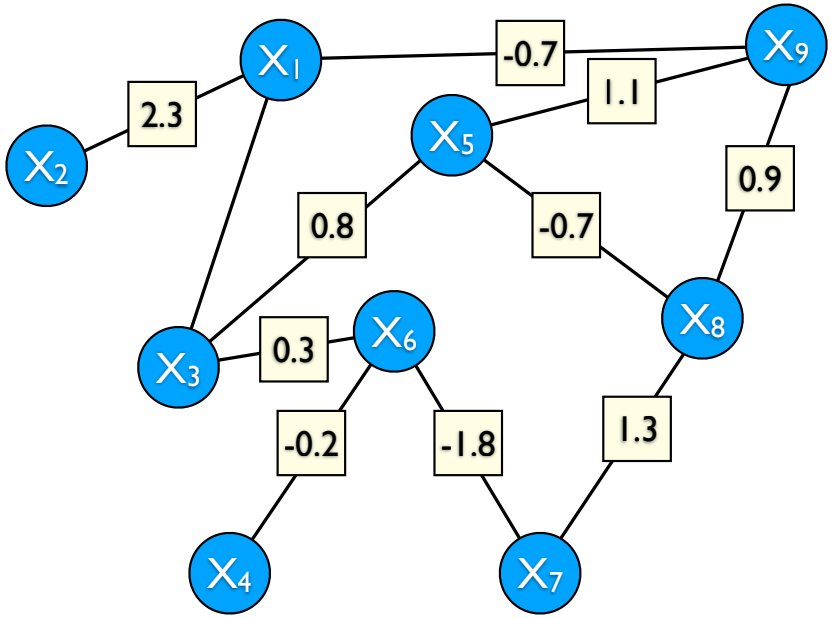
Outline

Motivating Problem: Structure Learning in Discrete MRFs



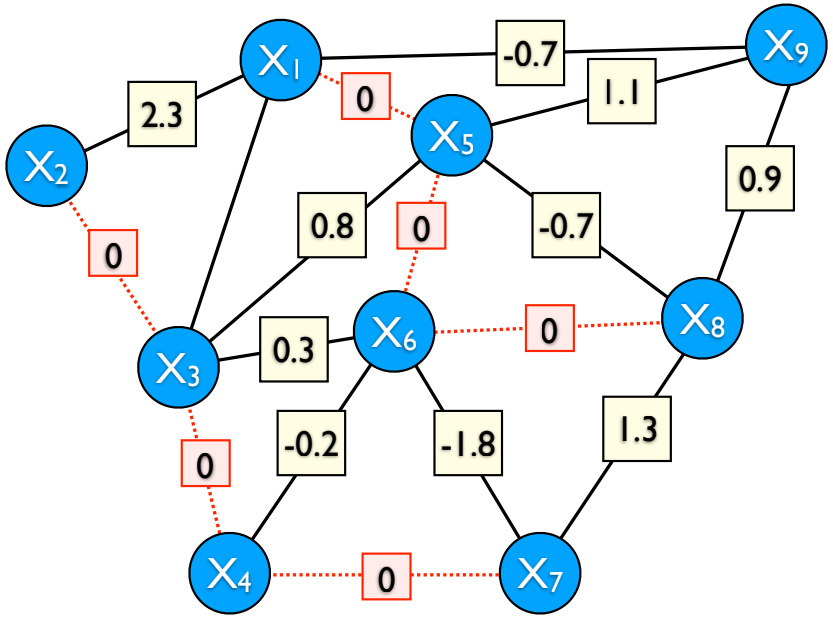
- We want to fit a Markov random field to discrete data, but don't know the graph structure.
- We can learn a sparse structure by using ℓ_1-regularization of the edge parameters [Lee et al. 2006, Wainwright et al. 2006].

Motivating Problem: Structure Learning in Discrete MRFs



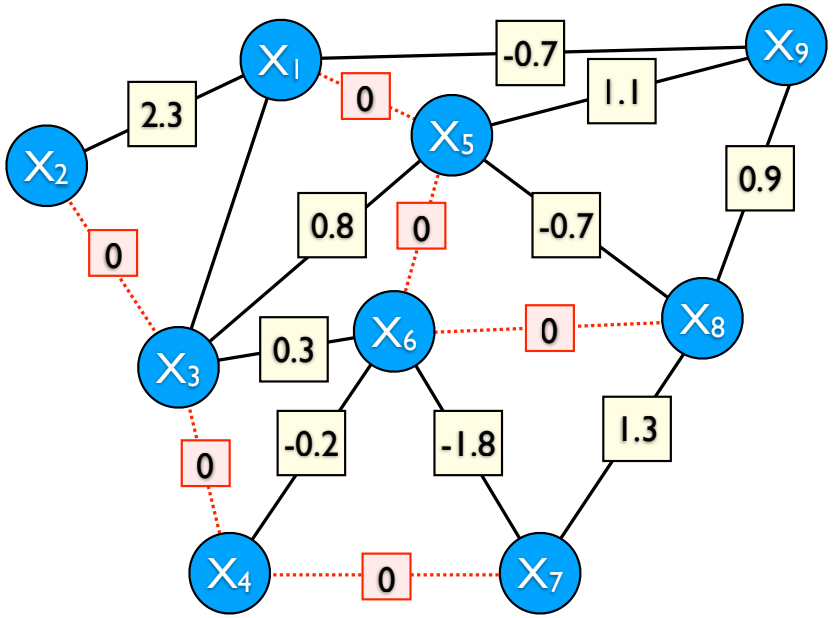
- We want to fit a Markov random field to discrete data, but don't know the graph structure.
- We can learn a sparse structure by using ℓ_1-regularization of the edge parameters [Lee et al. 2006, Wainwright et al. 2006].

Motivating Problem: Structure Learning in Discrete MRFs



- We want to fit a Markov random field to discrete data, but don't know the graph structure.
- We can learn a sparse structure by using ℓ_1-regularization of the edge parameters [Lee et al. 2006, Wainwright et al. 2006].

Motivating Problem: Structure Learning in Discrete MRFs



- We want to fit a Markov random field to discrete data, but don't know the graph structure.
- We can learn a sparse structure by using ℓ_1-regularization of the edge parameters [Lee et al. 2006, Wainwright et al. 2006].

Optimization with ℓ_1-Regularization

- This requires solving an optimization problem of the form

$$\min_{\mathbf{x}} f(\mathbf{x}) + \sum_{i=1}^{n} \lambda_i |x_i|_1$$

- Solving this optimization has 3 complicating factors:
  1. the number of parameters is large
  2. evaluating the objective is expensive
  3. the objective is non-smooth
- If the objective was smooth, we might consider Hessian-free Newton methods or limited-memory quasi-Newton methods.
- The non-smooth term is separable.

Optimization with ℓ_1-Regularization

- This requires solving an optimization problem of the form

$$\min_{\mathbf{x}} f(\mathbf{x}) + \sum_{i=1}^{n} \lambda_i |x_i|_1$$

- Solving this optimization has 3 complicating factors:
  1. the number of parameters is large
  2. evaluating the objective is expensive
  3. the objective is non-smooth
- If the objective was smooth, we might consider Hessian-free Newton methods or limited-memory quasi-Newton methods.
- The non-smooth term is separable.

Optimization with ℓ_1-Regularization

- This requires solving an optimization problem of the form

$$\min_{\mathbf{x}} f(\mathbf{x}) + \sum_{i=1}^{n} \lambda_i |x_i|_1$$

- Solving this optimization has 3 complicating factors:
  1. the number of parameters is large
  2. evaluating the objective is expensive
  3. the objective is non-smooth
- If the objective was smooth, we might consider Hessian-free Newton methods or limited-memory quasi-Newton methods.
- The non-smooth term is separable.

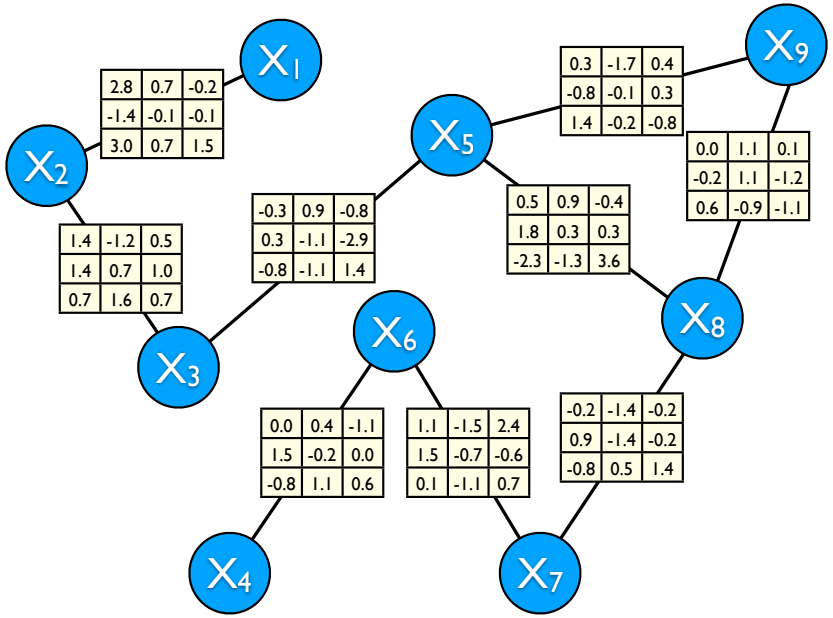
Optimization with ℓ_1-Regularization

- This requires solving an optimization problem of the form

$$\min_{\mathbf{x}} f(\mathbf{x}) + \sum_{i=1}^{n} \lambda_i |x_i|_1$$

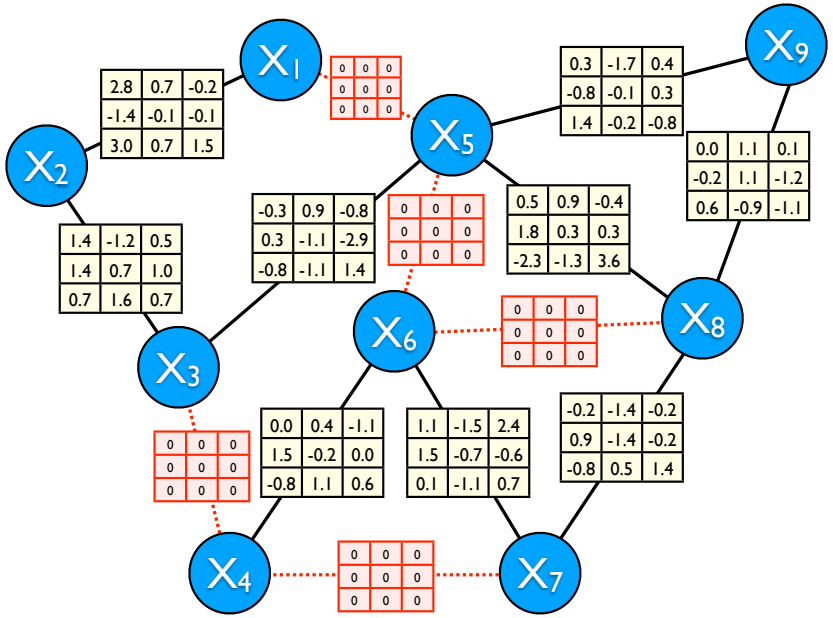
- Solving this optimization has 3 complicating factors:
  1. the number of parameters is large
  2. evaluating the objective is expensive
  3. the objective is non-smooth
- If the objective was smooth, we might consider Hessian-free Newton methods or limited-memory quasi-Newton methods.
- The non-smooth term is separable.

Structure Learning with Group ℓ_1-Regularization



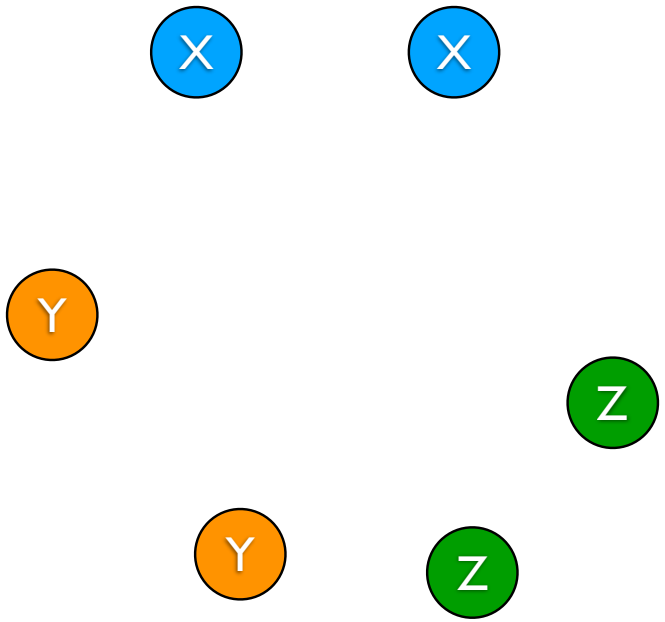
- In some cases, we want sparsity in groups of parameters:
  1. Multi-parameter edges [Lee et al., 2006].
  2. Blockwise-sparsity [Duchi et al., 2008].
  3. Conditional random fields [Schmidt et al., 2008]

Structure Learning with Group ℓ_1-Regularization



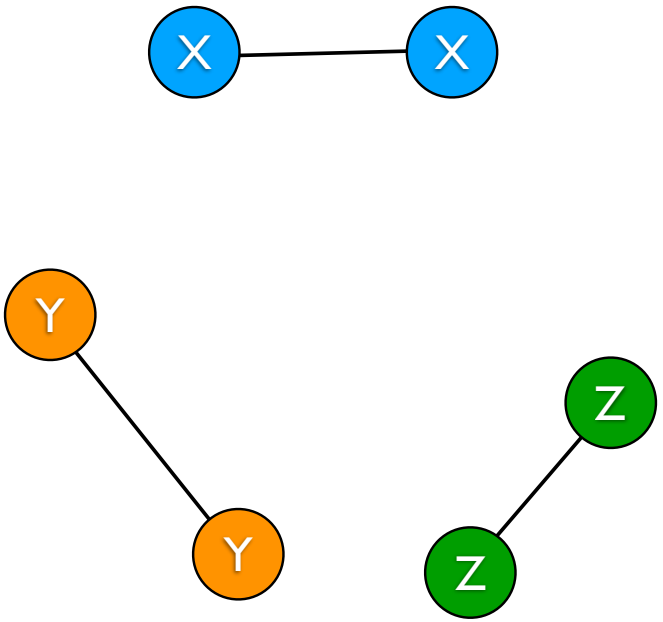
- In some cases, we want sparsity in groups of parameters:
  1. Multi-parameter edges [Lee et al., 2006].
  2. Blockwise-sparsity [Duchi et al., 2008].
  3. Conditional random fields [Schmidt et al., 2008]

Structure Learning with Group ℓ_1-Regularization



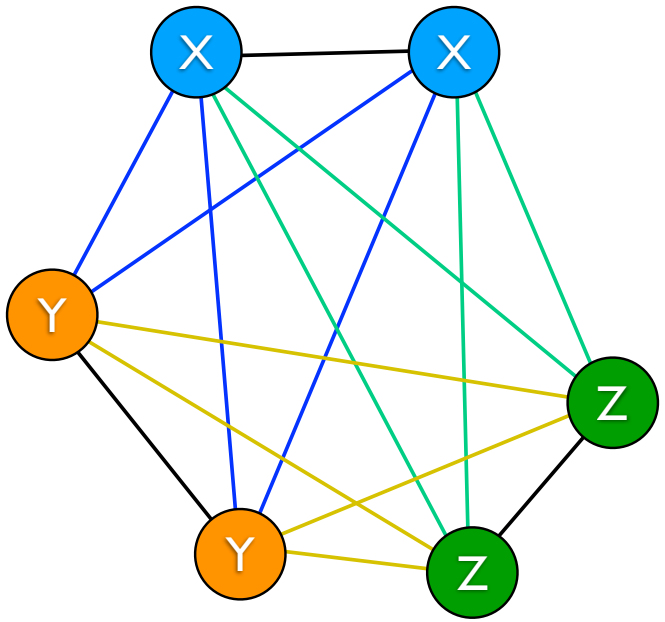
- In some cases, we want sparsity in groups of parameters:
  1. Multi-parameter edges [Lee et al., 2006].
  2. Blockwise-sparsity [Duchi et al., 2008].
  3. Conditional random fields [Schmidt et al., 2008]

Structure Learning with Group ℓ_1-Regularization



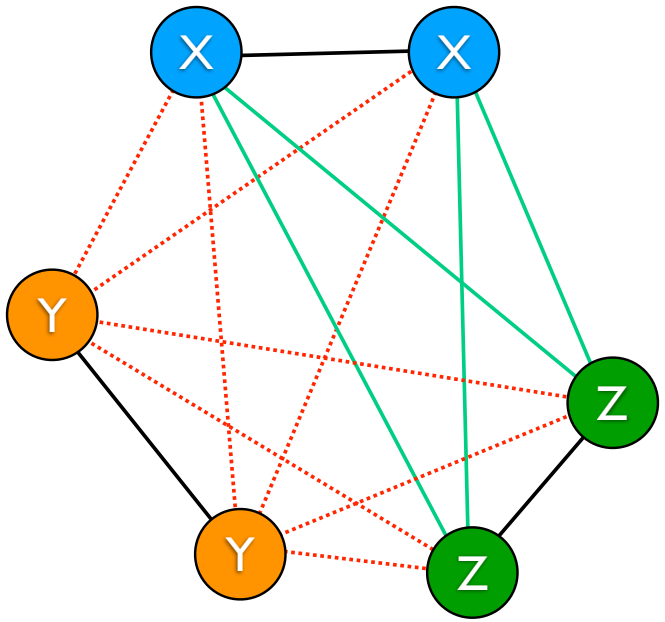
- In some cases, we want sparsity in groups of parameters:
  1. Multi-parameter edges [Lee et al., 2006].
  2. Blockwise-sparsity [Duchi et al., 2008].
  3. Conditional random fields [Schmidt et al., 2008]

Structure Learning with Group ℓ_1-Regularization



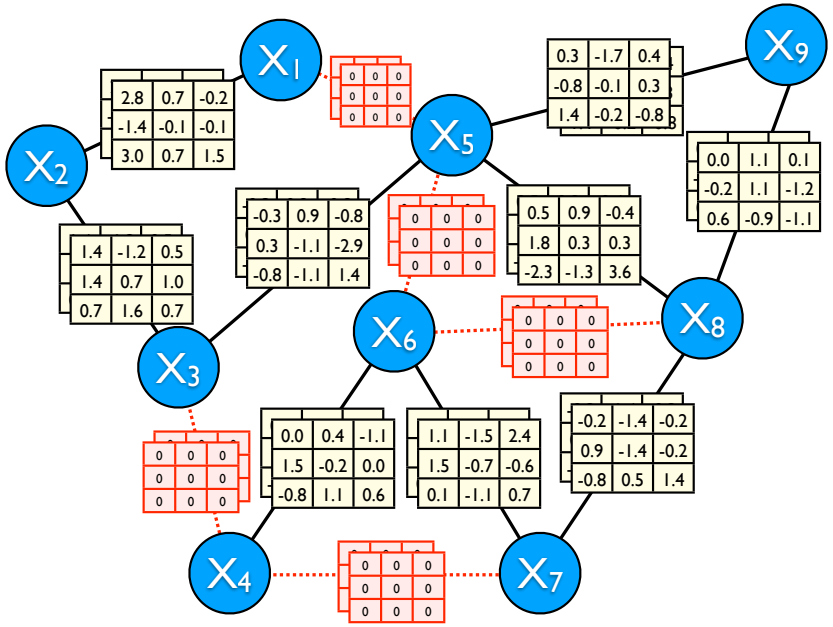
- In some cases, we want sparsity in groups of parameters:
  1. Multi-parameter edges [Lee et al., 2006].
  2. Blockwise-sparsity [Duchi et al., 2008].
  3. Conditional random fields [Schmidt et al., 2008]

Structure Learning with Group ℓ_1-Regularization



- In some cases, we want sparsity in groups of parameters:
  1. Multi-parameter edges [Lee et al., 2006].
  2. Blockwise-sparsity [Duchi et al., 2008].
  3. Conditional random fields [Schmidt et al., 2008]

Structure Learning with Group ℓ_1-Regularization



- In some cases, we want sparsity in groups of parameters:
  1. Multi-parameter edges [Lee et al., 2006].
  2. Blockwise-sparsity [Duchi et al., 2008].
  3. Conditional random fields [Schmidt et al., 2008]

Structure Learning with Group ℓ_1-Regularization

- In these cases we might consider group ℓ_1-regularization:

$$\min_{\mathbf{x}} f(\mathbf{x}) + \sum_g \lambda_g ||\mathbf{x}_g||_p$$

- Typically, we use the ℓ_2-norm, ℓ_∞-norm, or nuclear norm.
- Now, the non-smooth is term not even separable.
- However, the non-smooth term is still simple.

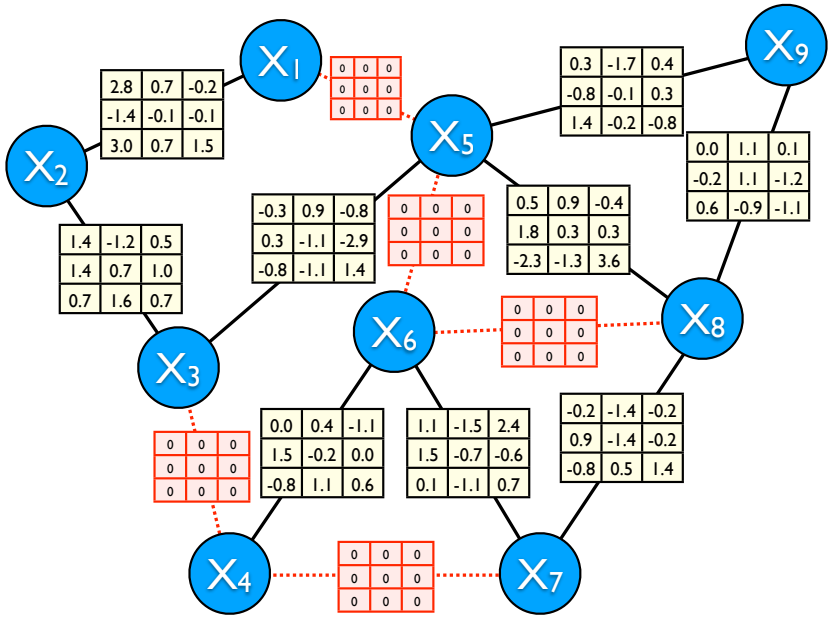
Structure Learning with Group ℓ_1-Regularization

- In these cases we might consider group ℓ_1-regularization:

$$\min_{\mathbf{x}} f(\mathbf{x}) + \sum_g \lambda_g ||\mathbf{x}_g||_p$$

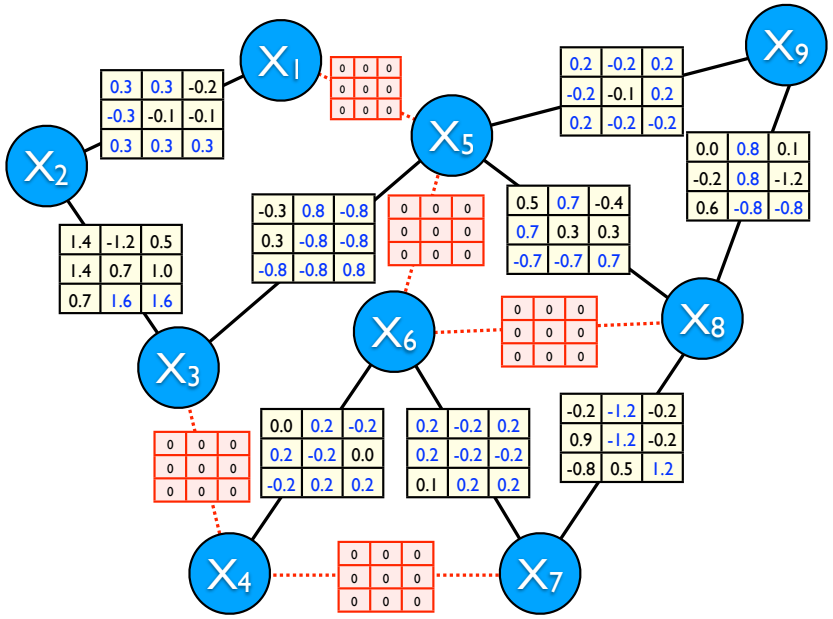
- Typically, we use the ℓ_2-norm, ℓ_∞-norm, or nuclear norm.
- Now, the non-smooth is term not even separable.
- However, the non-smooth term is still simple.

Structure Learning with Group ℓ_1-Regularization



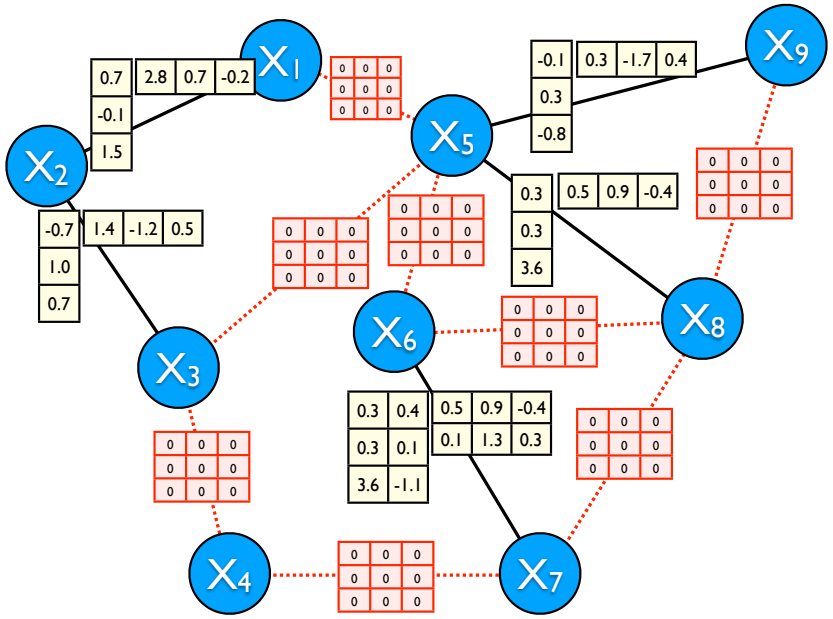
- Group ℓ_1-Regularization with the ℓ_2 group norm.
- Encourage group sparsity.

Structure Learning with Group ℓ_1-Regularization



- Group ℓ_1-Regularization with the ℓ_∞ group norm.
- Encourage group sparsity and parameter tieing.

Structure Learning with Group ℓ_1-Regularization



- Group ℓ_1-Regularization with the nuclear group norm.
- Encourage group sparsity and low-rank.

Structure Learning with Group ℓ_1-Regularization

- In these cases we might consider group ℓ_1-regularization:

$$\min_{\mathbf{x}} f(\mathbf{x}) + \sum_g \lambda_g ||\mathbf{x}_g||_p$$

- Typically, we use the ℓ_2-norm, ℓ_∞-norm, or nuclear norm.
- Now, the non-smooth is term not even separable.
- However, the non-smooth term is still simple.

Structure Learning with Group ℓ_1-Regularization

- In these cases we might consider group ℓ_1-regularization:

$$\min_{\mathbf{x}} f(\mathbf{x}) + \sum_g \lambda_g ||\mathbf{x}_g||_p$$

- Typically, we use the ℓ_2-norm, ℓ_∞-norm, or nuclear norm.
- Now, the non-smooth is term not even separable.
- However, the non-smooth term is still simple.

Structure Learning with Structured Sparsity

- Do we have to use pairwise models?
- We can use structured sparsity [Bach, 2008, Zhao et al., 2009] to learn sparse hierarchical models.
- In this case we use overlapping group ℓ_1-regularization:

$$\min_{\mathbf{x}} f(\mathbf{x}) + \sum_{A \subseteq \{1,\dots,p\}} \lambda_A \Big(\sum_{\{B | A \subseteq B\}} ||\mathbf{x}_B||_2^2 \Big)^{1/2}$$

- Now, the non-smooth term is not even simple.
- However, it is the sum of simple functions.

Structure Learning with Structured Sparsity

- Do we have to use pairwise models?
- We can use structured sparsity [Bach, 2008, Zhao et al., 2009] to learn sparse hierarchical models.
- In this case we use overlapping group ℓ_1-regularization:

$$\min_{\mathbf{x}} f(\mathbf{x}) + \sum_{A \subseteq \{1,\dots,p\}} \lambda_A \Big(\sum_{\{B | A \subseteq B\}} ||\mathbf{x}_B||_2^2 \Big)^{1/2}$$

- Now, the non-smooth term is not even simple.
- However, it is the sum of simple functions.

Structure Learning with Structured Sparsity

- Do we have to use pairwise models?
- We can use structured sparsity [Bach, 2008, Zhao et al., 2009] to learn sparse hierarchical models.
- In this case we use overlapping group ℓ_1-regularization:

$$\min_{\mathbf{x}} f(\mathbf{x}) + \sum_{A \subseteq \{1,\dots,p\}} \lambda_A \Big(\sum_{\{B | A \subseteq B\}} ||\mathbf{x}_B||_2^2 \Big)^{1/2}$$

- Now, the non-smooth term is not even simple.
- However, it is the sum of simple functions.

Structure Learning with Structured Sparsity

- Do we have to use pairwise models?
- We can use structured sparsity [Bach, 2008, Zhao et al., 2009] to learn sparse hierarchical models.
- In this case we use overlapping group ℓ_1-regularization:

$$\min_{\mathbf{x}} f(\mathbf{x}) + \sum_{A \subseteq \{1,\dots,p\}} \lambda_A \Big(\sum_{\{B | A \subseteq B\}} ||\mathbf{x}_B||_2^2 \Big)^{1/2}$$

- Now, the non-smooth term is not even simple.
- However, it is the sum of simple functions.

Structure Learning with Structured Sparsity

- Do we have to use pairwise models?
- We can use structured sparsity [Bach, 2008, Zhao et al., 2009] to learn sparse hierarchical models.
- In this case we use overlapping group ℓ_1-regularization:

$$\min_{\mathbf{x}} f(\mathbf{x}) + \sum_{A \subseteq \{1,\dots,p\}} \lambda_A \Big(\sum_{\{B | A \subseteq B\}} ||\mathbf{x}_B||_2^2 \Big)^{1/2}$$

- Now, the non-smooth term is not even simple.
- However, it is the sum of simple functions.

Outline

A Basic Newton-like Method

- We want first consider minimizing a twice-differentiable $f(\mathbf{x})$.
- Newton-like methods use a quadratic approximation of $f(\mathbf{x})$:

$$\mathcal{Q}^k(\mathbf{x}, \alpha) \triangleq f(\mathbf{x}^k) + (\mathbf{x} - \mathbf{x}^k)^T \nabla f(\mathbf{x}^k) + \frac{1}{2\alpha}(\mathbf{x} - \mathbf{x}^k)^T \mathbf{H}^k (\mathbf{x} - \mathbf{x}^k)$$

- $\mathbf{H}^k$ is a *positive-definite* approximation of the Hessian.
- The new iterate is set to the minimizer of the approximation

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \alpha \mathbf{d}^k,$$

where $\mathbf{d}^k$ is the solution to

$$\mathbf{H}^k \mathbf{d}^k = \nabla f(\mathbf{x}^k)$$

- Guarantees descent for small enough α.

A Basic Newton-like Method

- We want first consider minimizing a twice-differentiable $f(\mathbf{x})$.
- Newton-like methods use a quadratic approximation of $f(\mathbf{x})$:

$$\mathcal{Q}^k(\mathbf{x}, \alpha) \triangleq f(\mathbf{x}^k) + (\mathbf{x} - \mathbf{x}^k)^T \nabla f(\mathbf{x}^k) + \frac{1}{2\alpha}(\mathbf{x} - \mathbf{x}^k)^T \mathbf{H}^k (\mathbf{x} - \mathbf{x}^k)$$

- $\mathbf{H}^k$ is a *positive-definite* approximation of the Hessian.
- The new iterate is set to the minimizer of the approximation

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \alpha \mathbf{d}^k,$$

where $\mathbf{d}^k$ is the solution to

$$\mathbf{H}^k \mathbf{d}^k = \nabla f(\mathbf{x}^k)$$

- Guarantees descent for small enough α.

A Basic Newton-like Method

- We want first consider minimizing a twice-differentiable $f(\mathbf{x})$.
- Newton-like methods use a quadratic approximation of $f(\mathbf{x})$:

$$\mathcal{Q}^k(\mathbf{x}, \alpha) \triangleq f(\mathbf{x}^k) + (\mathbf{x} - \mathbf{x}^k)^T \nabla f(\mathbf{x}^k) + \frac{1}{2\alpha}(\mathbf{x} - \mathbf{x}^k)^T \mathbf{H}^k (\mathbf{x} - \mathbf{x}^k)$$

- $\mathbf{H}^k$ is a *positive-definite* approximation of the Hessian.
- The new iterate is set to the minimizer of the approximation

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \alpha \mathbf{d}^k,$$

where $\mathbf{d}^k$ is the solution to

$$\mathbf{H}^k \mathbf{d}^k = \nabla f(\mathbf{x}^k)$$

- Guarantees descent for small enough α.

A Basic Newton-like Method

- We want first consider minimizing a twice-differentiable $f(\mathbf{x})$.
- Newton-like methods use a quadratic approximation of $f(\mathbf{x})$:

$$\mathcal{Q}^k(\mathbf{x}, \alpha) \triangleq f(\mathbf{x}^k) + (\mathbf{x} - \mathbf{x}^k)^T \nabla f(\mathbf{x}^k) + \frac{1}{2\alpha}(\mathbf{x} - \mathbf{x}^k)^T \mathbf{H}^k (\mathbf{x} - \mathbf{x}^k)$$

- $\mathbf{H}^k$ is a *positive-definite* approximation of the Hessian.
- The new iterate is set to the minimizer of the approximation

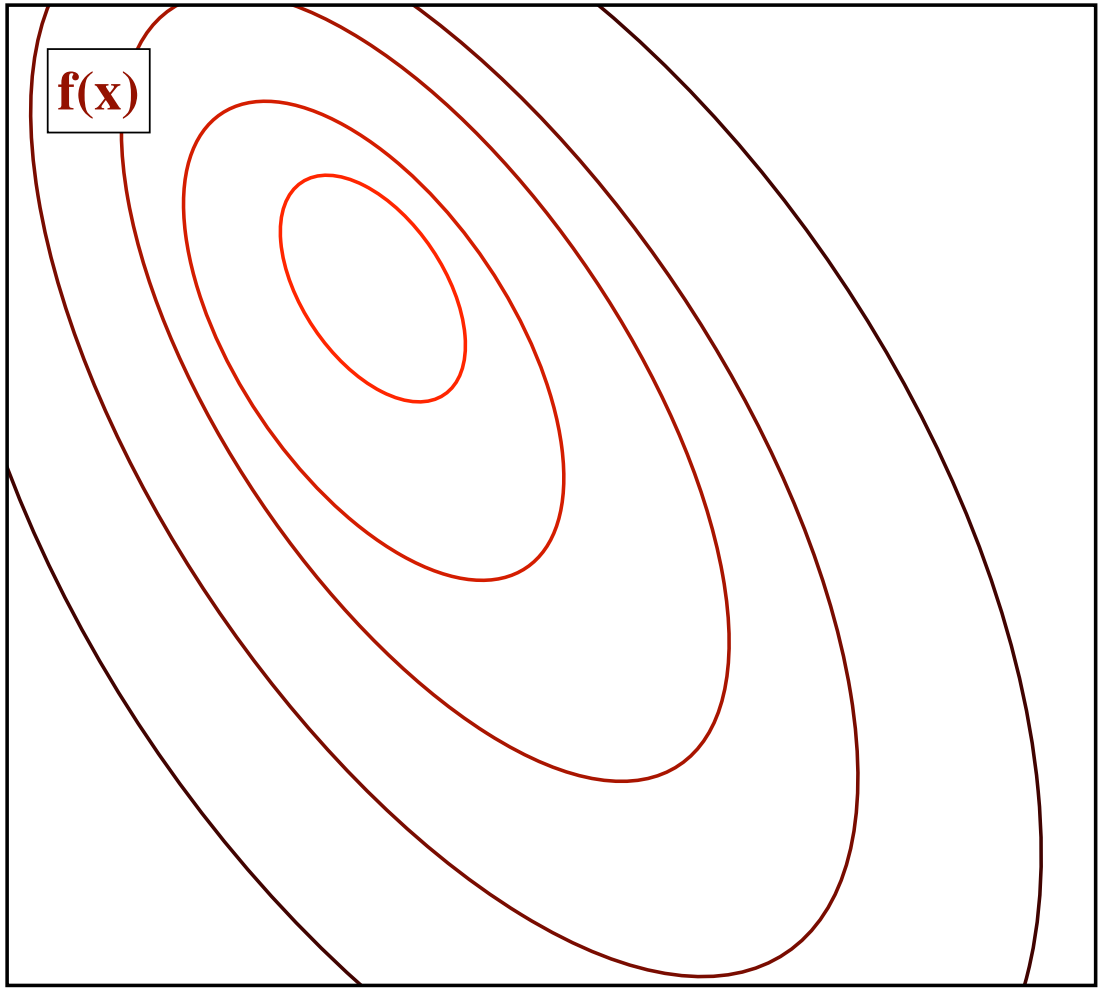
$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \alpha \mathbf{d}^k,$$

where $\mathbf{d}^k$ is the solution to

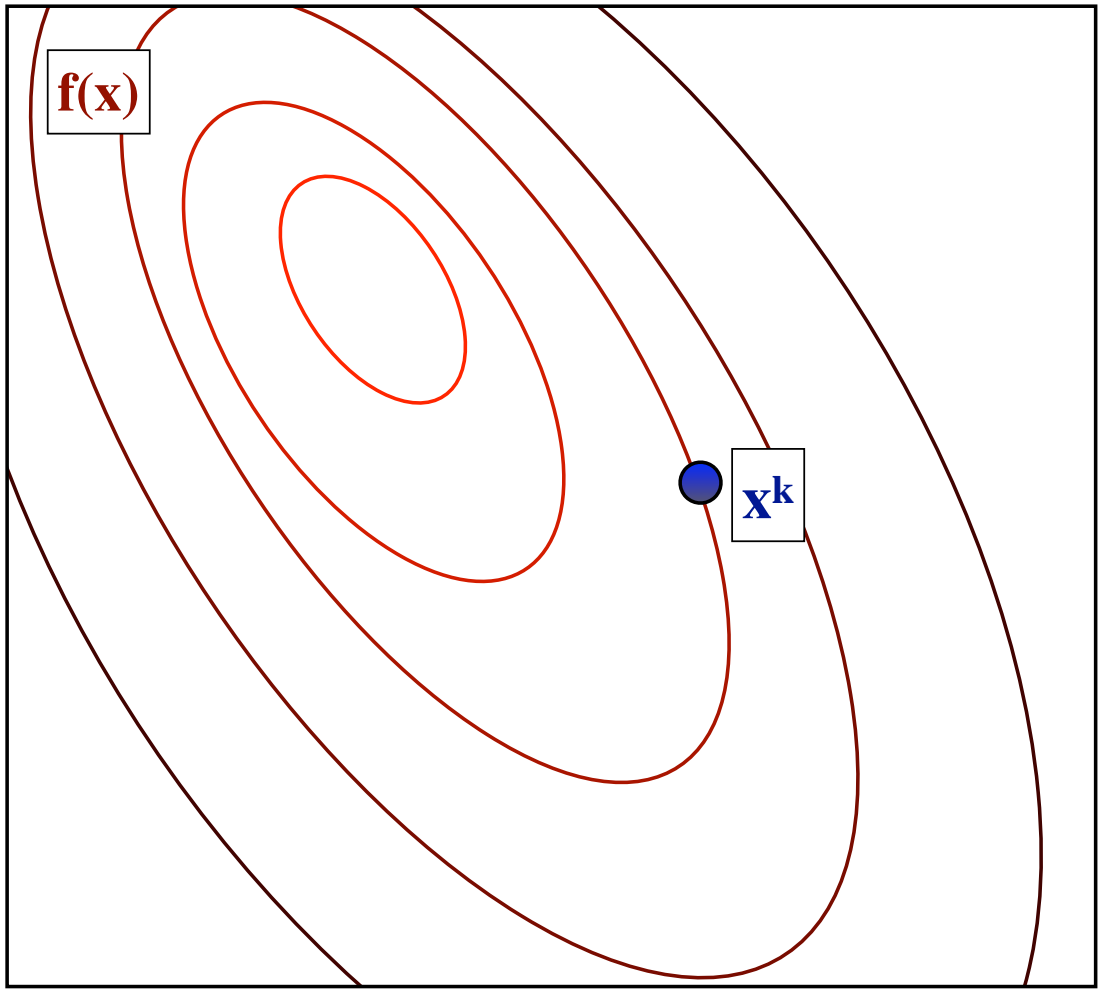
$$\mathbf{H}^k \mathbf{d}^k = \nabla f(\mathbf{x}^k)$$

- Guarantees descent for small enough α.

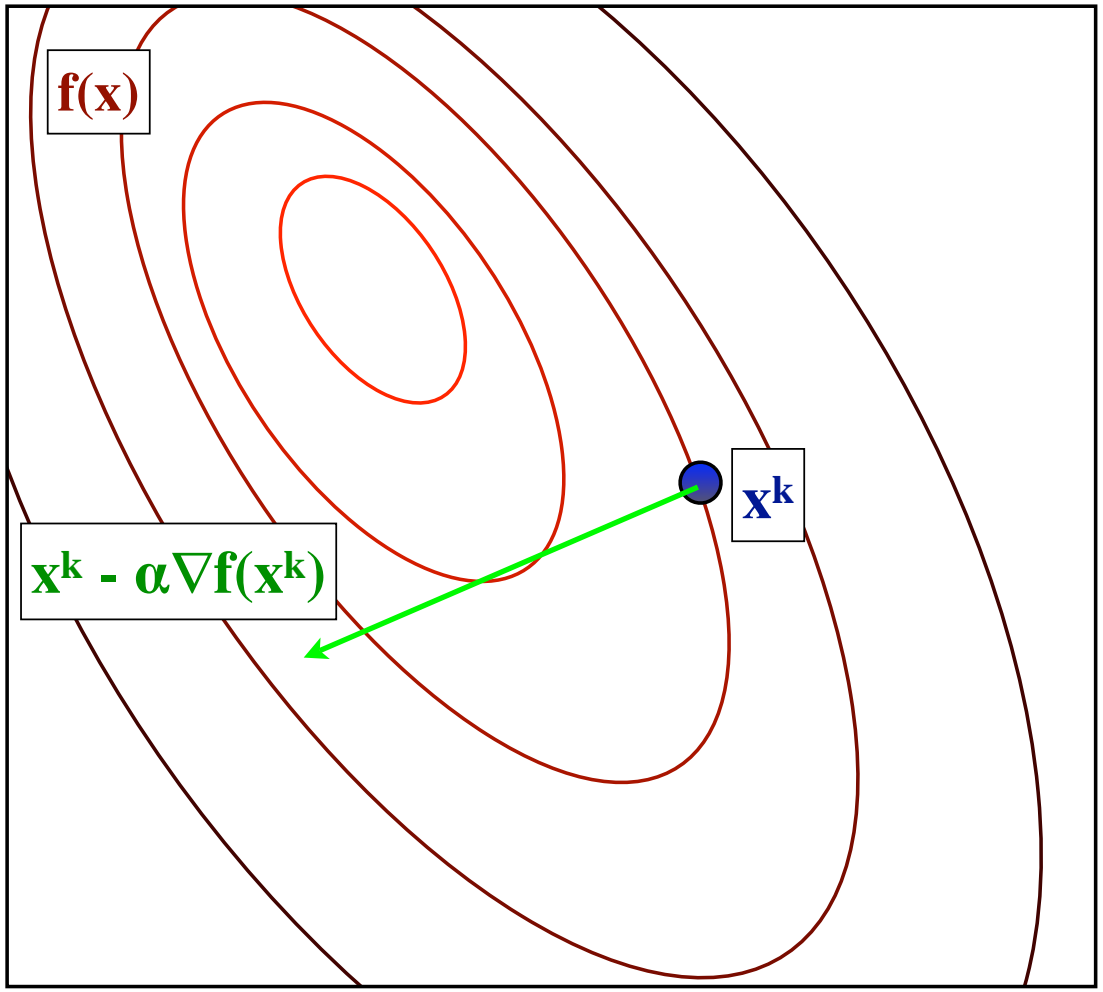
Gradient Method and Newton's Method



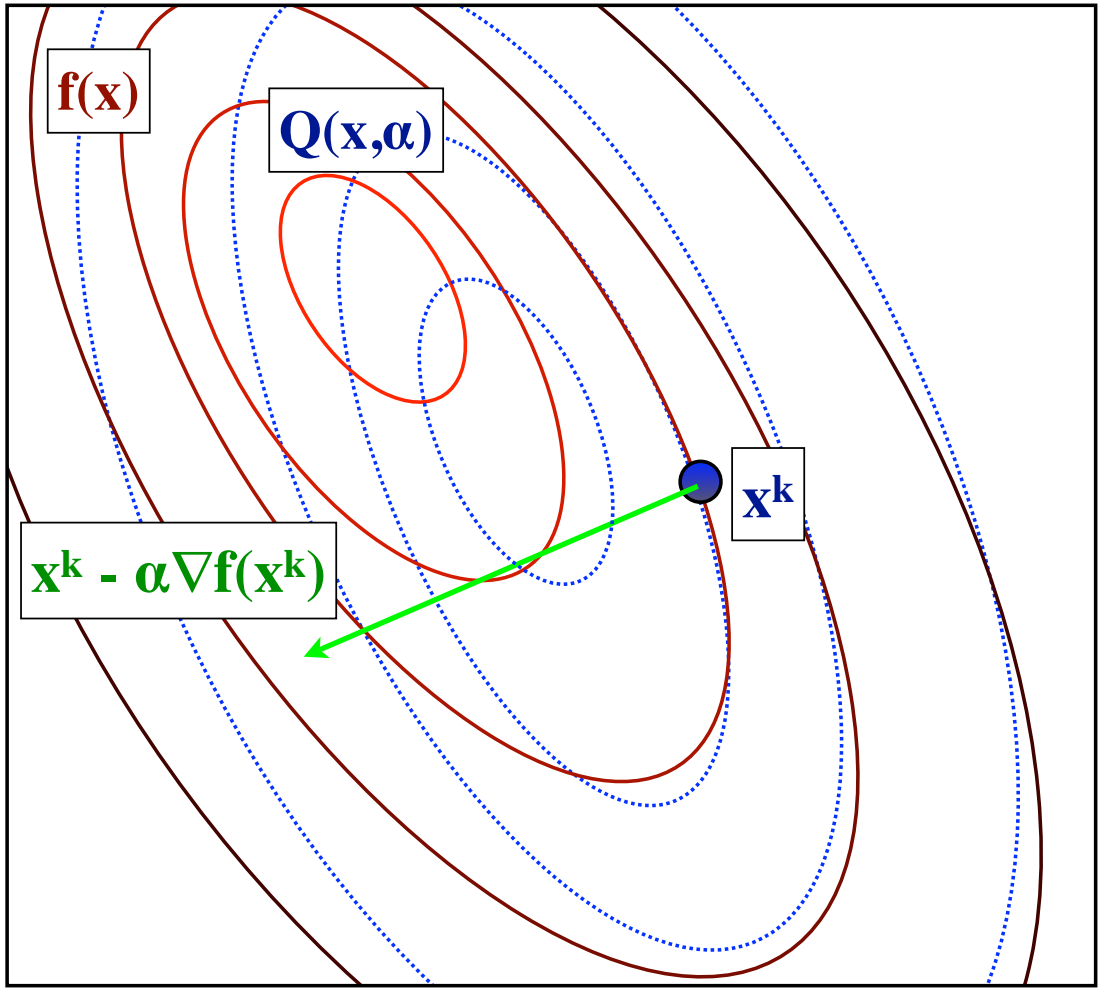
Gradient Method and Newton's Method



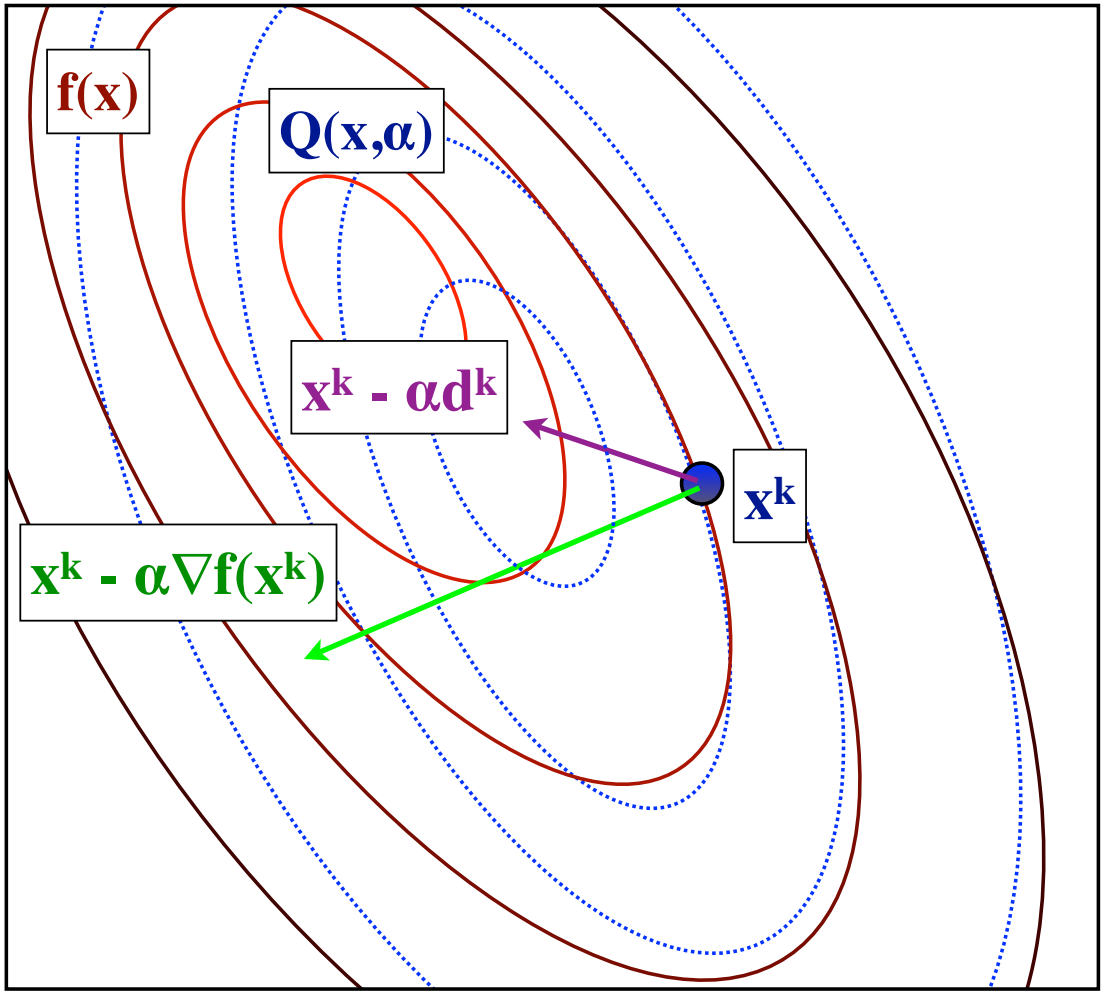
Gradient Method and Newton's Method



Gradient Method and Newton's Method



Gradient Method and Newton's Method



Armijo Inexact Line-Search

- We first try an initial step size α and then decrease it until we satisfy the Armijo sufficient decrease condition:

$$f(\mathbf{x}^{k+1}) \leq f(\mathbf{x}^k) + \eta \nabla f(\mathbf{x}^k)^T (\mathbf{x}^{k+1} - \mathbf{x}).$$

- With Hermite interpolation and control of the spectrum of $\mathbf{H}^k$, we typically accept the initial α or backtrack only once.

Armijo Inexact Line-Search

- We first try an initial step size α and then decrease it until we satisfy the Armijo sufficient decrease condition:

$$f(\mathbf{x}^{k+1}) \leq f(\mathbf{x}^k) + \eta \nabla f(\mathbf{x}^k)^T (\mathbf{x}^{k+1} - \mathbf{x}).$$

- With Hermite interpolation and control of the spectrum of $\mathbf{H}^k$, we typically accept the initial α or backtrack only once.

Rate of Convergence

- Under suitable smoothness and convexity assumptions, the method achieves a quadratic convergence rate:
  - It requires $\mathcal{O}(\log\log 1/\epsilon)$ iterations to for ϵ-accuracy.
- Any algorithm of the form

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \mathbf{B}^k \nabla f(\mathbf{x}^k)$$

has a superlinear local convergence rate around a strict minimizer if and only if $\mathbf{B}^k$ eventually behaves like the inverse Hessian [Dennis & Moré, 1974].

Rate of Convergence

- Under suitable smoothness and convexity assumptions, the method achieves a quadratic convergence rate:
  - It requires $\mathcal{O}(\log\log 1/\epsilon)$ iterations to for ϵ-accuracy.
- Any algorithm of the form

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \mathbf{B}^k \nabla f(\mathbf{x}^k)$$

has a superlinear local convergence rate around a strict minimizer if and only if $\mathbf{B}^k$ eventually behaves like the inverse Hessian [Dennis & Moré, 1974].

Rate of Convergence

- Under suitable smoothness and convexity assumptions, the method achieves a quadratic convergence rate:
  - It requires $\mathcal{O}(\log\log 1/\epsilon)$ iterations to for ϵ-accuracy.
- Any algorithm of the form

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \mathbf{B}^k \nabla f(\mathbf{x}^k)$$

has a superlinear local convergence rate around a strict minimizer if and only if $\mathbf{B}^k$ eventually behaves like the inverse Hessian [Dennis & Moré, 1974].

Disadvantages of Pure Newton Method

- For many problems, we can't afford to compute the Hessian, or even store an n by n matrix.
- There are several limited-memory alternatives available:
  1. Non-linear conjugate gradient.
  2. Nesterov's optimal gradient method.
  3. Diagonally-scaled steepest descent.
  4. Non-monotonic Barzilai-Borwein method.
  5. Hessian-free Newton methods.
  6. Limited-memory quasi-Newton methods.
- This talk mainly focuses on the last two.

Disadvantages of Pure Newton Method

- For many problems, we can't afford to compute the Hessian, or even store an n by n matrix.
- There are several limited-memory alternatives available:
  1. Non-linear conjugate gradient.
  2. Nesterov's optimal gradient method.
  3. Diagonally-scaled steepest descent.
  4. Non-monotonic Barzilai-Borwein method.
  5. Hessian-free Newton methods.
  6. Limited-memory quasi-Newton methods.
- This talk mainly focuses on the last two.

HFN: Hessian-Free Newton Methods

- We want to implement the step:

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \alpha \mathbf{d}^k.$$

where $\mathbf{d}^k$ is the solution of the linear system

$$\mathbf{H}^k \mathbf{d} = \nabla f(\mathbf{x}^k).$$

- Hessian-free Newton (HFN): find $\mathbf{d}^k$ with an iterative solver.
- We typically use conjugate gradient (but others are possible).
- Conjugate gradient only requires Hessian-vector products $\mathbf{H}^k \mathbf{y}$.

HFN: Hessian-Free Newton Methods

- We want to implement the step:

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \alpha \mathbf{d}^k.$$

where $\mathbf{d}^k$ is the solution of the linear system

$$\mathbf{H}^k \mathbf{d} = \nabla f(\mathbf{x}^k).$$

- Hessian-free Newton (HFN): find $\mathbf{d}^k$ with an iterative solver.
- We typically use conjugate gradient (but others are possible).
- Conjugate gradient only requires Hessian-vector products $\mathbf{H}^k \mathbf{y}$.

HFN: Hessian-Free Newton Methods

- We want to implement the step:

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \alpha \mathbf{d}^k.$$

where $\mathbf{d}^k$ is the solution of the linear system

$$\mathbf{H}^k \mathbf{d} = \nabla f(\mathbf{x}^k).$$

- Hessian-free Newton (HFN): find $\mathbf{d}^k$ with an iterative solver.
- We typically use conjugate gradient (but others are possible).
- Conjugate gradient only requires Hessian-vector products $\mathbf{H}^k \mathbf{y}$.

HFN: Hessian-Free Newton Methods

- We want to implement the step:

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \alpha \mathbf{d}^k.$$

where $\mathbf{d}^k$ is the solution of the linear system

$$\mathbf{H}^k \mathbf{d} = \nabla f(\mathbf{x}^k).$$

- Hessian-free Newton (HFN): find $\mathbf{d}^k$ with an iterative solver.
- We typically use conjugate gradient (but others are possible).
- Conjugate gradient only requires Hessian-vector products $\mathbf{H}^k \mathbf{y}$.

Compute Hessian-vector products

- We can compute Hessian-vector products without explicitly forming the Hessian.
- Sometimes this is due to the structure of the Hessian ($\nabla^2 f(\mathbf{x}) = A^T DA$ for logistic regression)
- Alternately, we can approximate the product numerically for one gradient evaluation:

$$\mathbf{H}^k \mathbf{y} \approx \frac{\nabla f(\mathbf{x}^k + \mu \mathbf{y}) - \nabla f(\mathbf{x}^k)}{\mu}.$$

- Under weak assumptions about $f(\mathbf{x}^k)$, we can approximate the Hessian-vector product without cancellation error with a very small complex μ [Squire and Trapp, 1988]:

$$\nabla f(\mathbf{x}^k + i\mu \mathbf{y}) = \nabla f(\mathbf{x}^k) + i\mu \mathbf{H}^k \mathbf{y} + \mathcal{O}(\mu^2).$$

Compute Hessian-vector products

- We can compute Hessian-vector products without explicitly forming the Hessian.
- Sometimes this is due to the structure of the Hessian ($\nabla^2 f(\mathbf{x}) = A^T DA$ for logistic regression)
- Alternately, we can approximate the product numerically for one gradient evaluation:

$$\mathbf{H}^k \mathbf{y} \approx \frac{\nabla f(\mathbf{x}^k + \mu \mathbf{y}) - \nabla f(\mathbf{x}^k)}{\mu}.$$

- Under weak assumptions about $f(\mathbf{x}^k)$, we can approximate the Hessian-vector product without cancellation error with a very small complex μ [Squire and Trapp, 1988]:

$$\nabla f(\mathbf{x}^k + i\mu \mathbf{y}) = \nabla f(\mathbf{x}^k) + i\mu \mathbf{H}^k \mathbf{y} + \mathcal{O}(\mu^2).$$

Compute Hessian-vector products

- We can compute Hessian-vector products without explicitly forming the Hessian.
- Sometimes this is due to the structure of the Hessian ($\nabla^2 f(\mathbf{x}) = A^T DA$ for logistic regression)
- Alternately, we can approximate the product numerically for one gradient evaluation:

$$\mathbf{H}^k\mathbf{y} \approx \frac{\nabla f(\mathbf{x}^k + \mu\mathbf{y}) - \nabla f(\mathbf{x}^k)}{\mu}.$$

- Under weak assumptions about $f(\mathbf{x}^k)$, we can approximate the Hessian-vector product without cancellation error with a very small complex μ [Squire and Trapp, 1988]:

$$\nabla f(\mathbf{x}^k + i\mu\mathbf{y}) = \nabla f(\mathbf{x}^k) + i\mu\mathbf{H}^k\mathbf{y} + \mathcal{O}(\mu^2).$$

Compute Hessian-vector products

- We can compute Hessian-vector products without explicitly forming the Hessian.
- Sometimes this is due to the structure of the Hessian ($\nabla^2 f(\mathbf{x}) = A^T DA$ for logistic regression)
- Alternately, we can approximate the product numerically for one gradient evaluation:

$$\mathbf{H}^k \mathbf{y} \approx \frac{\nabla f(\mathbf{x}^k + \mu \mathbf{y}) - \nabla f(\mathbf{x}^k)}{\mu}.$$

- Under weak assumptions about $f(\mathbf{x}^k)$, we can approximate the Hessian-vector product without cancellation error with a very small complex μ [Squire and Trapp, 1988]:

$$\nabla f(\mathbf{x}^k + i\mu \mathbf{y}) = \nabla f(\mathbf{x}^k) + i\mu \mathbf{H}^k \mathbf{y} + \mathcal{O}(\mu^2).$$

Rate of Convergence of Inexact Newton Methods

- There is no need to solve to full accuracy, leading to a residual:

$$\mathbf{H}^k\mathbf{d} = \nabla f(\mathbf{x}^k) + \mathbf{r}^k.$$

- Dembo, Eisenstat, Steihaug [1982] show fast convergence rates when the residuals are smaller than the gradient:
  1. Linear convergence: $||\mathbf{r}^k|| \leq \eta^k ||\nabla f(\mathbf{x}^k)||$ with $\eta^k < \eta < 1$.
  2. Superlinear convergence: $\lim_{k\to\infty} \eta^k = 0$.
  3. Quadratic convergence: $\eta^k = \mathcal{O}(||\nabla f(\mathbf{x}^k)||)$.
- For superlinear convergence, a typical forcing sequence is

$$\eta^k = \min\{.5, \sqrt{||\nabla f(\mathbf{x}^k)||}\}$$

Rate of Convergence of Inexact Newton Methods

- There is no need to solve to full accuracy, leading to a residual:

$$\mathbf{H}^k\mathbf{d} = \nabla f(\mathbf{x}^k) + \mathbf{r}^k.$$

- Dembo, Eisenstat, Steihaug [1982] show fast convergence rates when the residuals are smaller than the gradient:
  1. Linear convergence: $||\mathbf{r}^k|| \leq \eta^k ||\nabla f(\mathbf{x}^k)||$ with $\eta^k < \eta < 1$.
  2. Superlinear convergence: $\lim_{k\to\infty} \eta^k = 0$.
  3. Quadratic convergence: $\eta^k = \mathcal{O}(||\nabla f(\mathbf{x}^k)||)$.
- For superlinear convergence, a typical forcing sequence is

$$\eta^k = \min\{.5, \sqrt{||\nabla f(\mathbf{x}^k)||}\}$$

Rate of Convergence of Inexact Newton Methods

- There is no need to solve to full accuracy, leading to a residual:

$$\mathbf{H}^k\mathbf{d} = \nabla f(\mathbf{x}^k) + \mathbf{r}^k.$$

- Dembo, Eisenstat, Steihaug [1982] show fast convergence rates when the residuals are smaller than the gradient:
  1. Linear convergence: $||\mathbf{r}^k|| \leq \eta^k ||\nabla f(\mathbf{x}^k)||$ with $\eta^k < \eta < 1$.
  2. Superlinear convergence: $\lim_{k\to\infty} \eta^k = 0$.
  3. Quadratic convergence: $\eta^k = \mathcal{O}(||\nabla f(\mathbf{x}^k)||)$.
- For superlinear convergence, a typical forcing sequence is

$$\eta^k = \min\{.5, \sqrt{||\nabla f(\mathbf{x}^k)||}\}$$

Rate of Convergence of Inexact Newton Methods

- There is no need to solve to full accuracy, leading to a residual:

$$\mathbf{H}^k\mathbf{d} = \nabla f(\mathbf{x}^k) + \mathbf{r}^k.$$

- Dembo, Eisenstat, Steihaug [1982] show fast convergence rates when the residuals are smaller than the gradient:
  1. Linear convergence: $||\mathbf{r}^k|| \leq \eta^k ||\nabla f(\mathbf{x}^k)||$ with $\eta^k < \eta < 1$.
  2. Superlinear convergence: $\lim_{k\to\infty} \eta^k = 0$.
  3. Quadratic convergence: $\eta^k = \mathcal{O}(||\nabla f(\mathbf{x}^k)||)$.
- For superlinear convergence, a typical forcing sequence is

$$\eta^k = \min\{.5, \sqrt{||\nabla f(\mathbf{x}^k)||}\}$$

Discussion of HFN Methods

- Preconditioning often drastically reduces the number of Hessian-vector products.
- The conjugate gradient algorithm can be modified to give a descent direction even if the Hessian is not positive-definite.
- If the Hessian contains negative eigenvalues, the method may be able to find a direction of negative curvature.
- For details, Nocedal and Wright [2006, §7.1].

Discussion of HFN Methods

- Preconditioning often drastically reduces the number of Hessian-vector products.
- The conjugate gradient algorithm can be modified to give a descent direction even if the Hessian is not positive-definite.
- If the Hessian contains negative eigenvalues, the method may be able to find a direction of negative curvature.
- For details, Nocedal and Wright [2006, §7.1].

Discussion of HFN Methods

- Preconditioning often drastically reduces the number of Hessian-vector products.
- The conjugate gradient algorithm can be modified to give a descent direction even if the Hessian is not positive-definite.
- If the Hessian contains negative eigenvalues, the method may be able to find a direction of negative curvature.
- For details, Nocedal and Wright [2006, §7.1].

Discussion of HFN Methods

- Preconditioning often drastically reduces the number of Hessian-vector products.
- The conjugate gradient algorithm can be modified to give a descent direction even if the Hessian is not positive-definite.
- If the Hessian contains negative eigenvalues, the method may be able to find a direction of negative curvature.
- For details, Nocedal and Wright [2006, §7.1].

Hessian-Free Newton Methods vs. Quasi-Newton Methods

The main difference between HFN and quasi-Newton methods:

- Hessian-free methods approximately invert the Hessian.
- Quasi-Newton methods invert an approximate Hessian.

Hessian-Free Newton Methods vs. Quasi-Newton Methods

The main difference between HFN and quasi-Newton methods:

- Hessian-free methods approximately invert the Hessian.
- Quasi-Newton methods invert an approximate Hessian.

Hessian-Free Newton Methods vs. Quasi-Newton Methods

The main difference between HFN and quasi-Newton methods:

- Hessian-free methods approximately invert the Hessian.
- Quasi-Newton methods invert an approximate Hessian.

Broyden-Fletcher-Goldfarb-Shanno Quasi-Newton Method

- Quasi-Newton methods work with the parameter and gradient differences between successive iterations:

$$\mathbf{s}_k \triangleq \mathbf{x}^{k+1} - \mathbf{x}^k, \quad \mathbf{y}_k \triangleq \nabla f(\mathbf{x}^{k+1}) - \nabla f(\mathbf{x}^k).$$

- They start with an initial approximation $\mathbf{H}^0 \triangleq \sigma \mathbf{I}$, and choose $\mathbf{H}^{k+1}$ to interpolate the gradient difference:

$$\mathbf{H}^{k+1}\mathbf{s}_k = \mathbf{y}_k.$$

- Since $\mathbf{H}^{k+1}$ is not unique; the BFGS method chooses the symmetric matrix whose difference with $\mathbf{H}^k$ is minimal:

$$\mathbf{H}^{k+1} = \mathbf{H}^k - \frac{\mathbf{H}^k\mathbf{s}_k\mathbf{s}_k\mathbf{H}^k}{\mathbf{s}_k\mathbf{H}^k\mathbf{s}_k} + \frac{\mathbf{y}_k\mathbf{y}_k^T}{\mathbf{y}_k^T\mathbf{s}_k}.$$

Broyden-Fletcher-Goldfarb-Shanno Quasi-Newton Method

- Quasi-Newton methods work with the parameter and gradient differences between successive iterations:

$$\mathbf{s}_k \triangleq \mathbf{x}^{k+1} - \mathbf{x}^k, \quad \mathbf{y}_k \triangleq \nabla f(\mathbf{x}^{k+1}) - \nabla f(\mathbf{x}^k).$$

- They start with an initial approximation $\mathbf{H}^0 \triangleq \sigma \mathbf{I}$, and choose $\mathbf{H}^{k+1}$ to interpolate the gradient difference:

$$\mathbf{H}^{k+1}\mathbf{s}_k = \mathbf{y}_k.$$

- Since $\mathbf{H}^{k+1}$ is not unique; the BFGS method chooses the symmetric matrix whose difference with $\mathbf{H}^k$ is minimal:

$$\mathbf{H}^{k+1} = \mathbf{H}^k - \frac{\mathbf{H}^k \mathbf{s}_k \mathbf{s}_k \mathbf{H}^k}{\mathbf{s}_k \mathbf{H}^k \mathbf{s}_k} + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k}.$$

Broyden-Fletcher-Goldfarb-Shanno Quasi-Newton Method

- Quasi-Newton methods work with the parameter and gradient differences between successive iterations:

$$\mathbf{s}_k \triangleq \mathbf{x}^{k+1} - \mathbf{x}^k, \quad \mathbf{y}_k \triangleq \nabla f(\mathbf{x}^{k+1}) - \nabla f(\mathbf{x}^k).$$

- They start with an initial approximation $\mathbf{H}^0 \triangleq \sigma \mathbf{I}$, and choose $\mathbf{H}^{k+1}$ to interpolate the gradient difference:

$$\mathbf{H}^{k+1}\mathbf{s}_k = \mathbf{y}_k.$$

- Since $\mathbf{H}^{k+1}$ is not unique; the BFGS method chooses the symmetric matrix whose difference with $\mathbf{H}^k$ is minimal:

$$\mathbf{H}^{k+1} = \mathbf{H}^k - \frac{\mathbf{H}^k\mathbf{s}_k\mathbf{s}_k\mathbf{H}^k}{\mathbf{s}_k\mathbf{H}^k\mathbf{s}_k} + \frac{\mathbf{y}_k\mathbf{y}_k^T}{\mathbf{y}_k^T\mathbf{s}_k}.$$

Convergence and Limited-Memory BFGS (L-BFGS)

- Update skipping/damping or a more sophisticated line search (Wolfe conditions) can keep $\mathbf{H}^{k+1}$ positive-definite.
- The BFGS method has a superlinear convergence rate.
- But, it still uses a dense $\mathbf{H}^k$.
- Instead of storing $\mathbf{H}^k$, the limited-memory BFGS (L-BFGS) method stores the previous m differences $\mathbf{s}_k$ and $\mathbf{y}_k$.
- We can solve a linear system involving these updates applied to a diagonal $\mathbf{H}^0$ in $\mathcal{O}(mn)$ [Nocedal, 1980].

Convergence and Limited-Memory BFGS (L-BFGS)

- Update skipping/damping or a more sophisticated line search (Wolfe conditions) can keep $\mathbf{H}^{k+1}$ positive-definite.
- The BFGS method has a superlinear convergence rate.
- But, it still uses a dense $\mathbf{H}^k$.
- Instead of storing $\mathbf{H}^k$, the limited-memory BFGS (L-BFGS) method stores the previous m differences $\mathbf{s}_k$ and $\mathbf{y}_k$.
- We can solve a linear system involving these updates applied to a diagonal $\mathbf{H}^0$ in $\mathcal{O}(mn)$ [Nocedal, 1980].

Convergence and Limited-Memory BFGS (L-BFGS)

- Update skipping/damping or a more sophisticated line search (Wolfe conditions) can keep $\mathbf{H}^{k+1}$ positive-definite.
- The BFGS method has a superlinear convergence rate.
- But, it still uses a dense $\mathbf{H}^k$.
- Instead of storing $\mathbf{H}^k$, the limited-memory BFGS (L-BFGS) method stores the previous m differences $\mathbf{s}_k$ and $\mathbf{y}_k$.
- We can solve a linear system involving these updates applied to a diagonal $\mathbf{H}^0$ in $\mathcal{O}(mn)$ [Nocedal, 1980].

Convergence and Limited-Memory BFGS (L-BFGS)

- Update skipping/damping or a more sophisticated line search (Wolfe conditions) can keep $\mathbf{H}^{k+1}$ positive-definite.
- The BFGS method has a superlinear convergence rate.
- But, it still uses a dense $\mathbf{H}^k$.
- Instead of storing $\mathbf{H}^k$, the limited-memory BFGS (L-BFGS) method stores the previous m differences $\mathbf{s}_k$ and $\mathbf{y}_k$.
- We can solve a linear system involving these updates applied to a diagonal $\mathbf{H}^0$ in $\mathcal{O}(mn)$ [Nocedal, 1980].

Convergence and Limited-Memory BFGS (L-BFGS)

- Update skipping/damping or a more sophisticated line search (Wolfe conditions) can keep $\mathbf{H}^{k+1}$ positive-definite.
- The BFGS method has a superlinear convergence rate.
- But, it still uses a dense $\mathbf{H}^k$.
- Instead of storing $\mathbf{H}^k$, the limited-memory BFGS (L-BFGS) method stores the previous m differences $\mathbf{s}_k$ and $\mathbf{y}_k$.
- We can solve a linear system involving these updates applied to a diagonal $\mathbf{H}^0$ in $\mathcal{O}(mn)$ [Nocedal, 1980].

L-BFGS Recursive Formula

Solving $\mathbf{H}^k\mathbf{d}^k = \mathbf{g}$ given $\{\mathbf{H}^0, \mathbf{s}_k, \mathbf{y}_k\}$ [Nocedal, 1980]:

```
q(:,k+1) = g;
for i = k:-1:1
    al(i) = ro(i)*s(:,i)'*q(:,i+1);
    q(:,i) = q(:,i+1)-al(i)*y(:,i);
end
r(:,1) = H0\q(:,1);
for i = 1:k
    be(i) = ro(i)*y(:,i)'*r(:,i);
    r(:,i+1) = r(:,i) + s(:,i)*(al(i)-be(i));
end
d=r(:,k+1);
```

Scaling L-BFGS

- The choice of $\mathbf{H}^0$ on each iteration is crucial to the performance of L-BFGS methods.
- A common choice is $\mathbf{H}^0 = \alpha_{bb}^{-1}\mathbf{I}$ [Shanno & Phua, 1978]:

$$\alpha_{bb} = \underset{\alpha}{\operatorname{argmin}} ||\mathbf{s}_k - \alpha \mathbf{I}\mathbf{y}_k|| = (\mathbf{s}_k^T \mathbf{y}_k)/(\mathbf{y}_k^T \mathbf{y}_k)$$

- Convergence theory is not as nice for L-BFGS, but often outperforms HFN and other competing approaches.

Scaling L-BFGS

- The choice of $\mathbf{H}^0$ on each iteration is crucial to the performance of L-BFGS methods.
- A common choice is $\mathbf{H}^0 = \alpha_{bb}^{-1}\mathbf{I}$ [Shanno & Phua, 1978]:

$$\alpha_{bb} = \underset{\alpha}{\operatorname{argmin}} \, ||\mathbf{s}_k - \alpha \mathbf{I} \mathbf{y}_k|| = (\mathbf{s}_k^T \mathbf{y}_k)/(\mathbf{y}_k^T \mathbf{y}_k)$$

- Convergence theory is not as nice for L-BFGS, but often outperforms HFN and other competing approaches.

Scaling L-BFGS

- The choice of $\mathbf{H}^0$ on each iteration is crucial to the performance of L-BFGS methods.
- A common choice is $\mathbf{H}^0 = \alpha_{bb}^{-1}\mathbf{I}$ [Shanno & Phua, 1978]:

$$\alpha_{bb} = \underset{\alpha}{\operatorname{argmin}} \, ||\mathbf{s}_k - \alpha \mathbf{I} \mathbf{y}_k|| = (\mathbf{s}_k^T \mathbf{y}_k)/(\mathbf{y}_k^T \mathbf{y}_k)$$

- Convergence theory is not as nice for L-BFGS, but often outperforms HFN and other competing approaches.

Barzilai-Borwein Method

- We can also consider the approximate quasi-Newton method:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_{bb}\nabla f(\mathbf{x}^k).$$

- This is the Barzilai & Borwein [1988] method.
- The step size is typically used with a non-monotomic Armijo condition [Grippo et al., 1986, Raydan et al., 1997]:

$$f(\mathbf{x}^{k+1}) \leq \max_{j\in\{k-m:k\}}\{f(\mathbf{x}^j)\} + \eta\nabla f(\mathbf{x}^k)^T(\mathbf{x}^{k+1} - \mathbf{x}).$$

- This simple method performs surprisingly well in a variety of problems [Raydan et al., 1997, Birgin et al., 2000, Dai & Fletcher, 2005, Figuereido et al., 2007, van den Berg and Friedlander, 2008, Wright et al., 2010].

Barzilai-Borwein Method

- We can also consider the approximate quasi-Newton method:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_{bb}\nabla f(\mathbf{x}^k).$$

- This is the Barzilai & Borwein [1988] method.
- The step size is typically used with a non-monotomic Armijo condition [Grippo et al., 1986, Raydan et al., 1997]:

$$f(\mathbf{x}^{k+1}) \leq \max_{j\in\{k-m:k\}}\{f(\mathbf{x}^j)\} + \eta\nabla f(\mathbf{x}^k)^T(\mathbf{x}^{k+1} - \mathbf{x}).$$

- This simple method performs surprisingly well in a variety of problems [Raydan et al., 1997, Birgin et al., 2000, Dai & Fletcher, 2005, Figuereido et al., 2007, van den Berg and Friedlander, 2008, Wright et al., 2010].

Barzilai-Borwein Method

- We can also consider the approximate quasi-Newton method:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_{bb}\nabla f(\mathbf{x}^k).$$

- This is the Barzilai & Borwein [1988] method.
- The step size is typically used with a non-monotomic Armijo condition [Grippo et al., 1986, Raydan et al., 1997]:

$$f(\mathbf{x}^{k+1}) \leq \max_{j\in\{k-m:k\}}\{f(\mathbf{x}^j)\} + \eta\nabla f(\mathbf{x}^k)^T(\mathbf{x}^{k+1} - \mathbf{x}).$$

- This simple method performs surprisingly well in a variety of problems [Raydan et al., 1997, Birgin et al., 2000, Dai & Fletcher, 2005, Figuereido et al., 2007, van den Berg and Friedlander, 2008, Wright et al., 2010].

Barzilai-Borwein Method

- We can also consider the approximate quasi-Newton method:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_{bb}\nabla f(\mathbf{x}^k).$$

- This is the Barzilai & Borwein [1988] method.
- The step size is typically used with a non-monotomic Armijo condition [Grippo et al., 1986, Raydan et al., 1997]:

$$f(\mathbf{x}^{k+1}) \leq \max_{j\in\{k-m:k\}}\{f(\mathbf{x}^j)\} + \eta\nabla f(\mathbf{x}^k)^T(\mathbf{x}^{k+1} - \mathbf{x}).$$

- This simple method performs surprisingly well in a variety of problems [Raydan et al., 1997, Birgin et al., 2000, Dai & Fletcher, 2005, Figuereido et al., 2007, van den Berg and Friedlander, 2008, Wright et al., 2010].

Hybrid L-BFGS and Hessian-Free Methods

L-BFGS and HFN methods can be combined:

- Use an L-BFGS approximation to precondition the conjugate gradient iterations [Morales & Nocedal, 2000].
- Use conjugate gradient iterations to to improve $\mathbf{H}^0$ in the L-BFGS approximation [Morales & Nocedal, 2002] .

Hybrid L-BFGS and Hessian-Free Methods

L-BFGS and HFN methods can be combined:

- Use an L-BFGS approximation to precondition the conjugate gradient iterations [Morales & Nocedal, 2000].
- Use conjugate gradient iterations to to improve $\mathbf{H}^0$ in the L-BFGS approximation [Morales & Nocedal, 2002] .

Hybrid L-BFGS and Hessian-Free Methods

L-BFGS and HFN methods can be combined:

- Use an L-BFGS approximation to precondition the conjugate gradient iterations [Morales & Nocedal, 2000].
- Use conjugate gradient iterations to to improve $\mathbf{H}^0$ in the L-BFGS approximation [Morales & Nocedal, 2002] .

Outline

Optimization with ℓ_1-Regularization

- We want to optimize a smooth function with ℓ_1-regularization:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \ell(\mathbf{x}) + \sum_{i=1}^{n} \lambda_i |x_i|$$

- The non-smooth regularizer breaks quasi-Newton and Hessian-free Newton methods.
- But the regularizer is separable.
- We consider two methods that take advantage of this:
  1. Two-metric projection on an equivalent problem.
  2. Two-metric sub-gradient projection applied directly.

Optimization with ℓ_1-Regularization

- We want to optimize a smooth function with ℓ_1-regularization:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \ell(\mathbf{x}) + \sum_{i=1}^{n} \lambda_i |x_i|$$

- The non-smooth regularizer breaks quasi-Newton and Hessian-free Newton methods.
- But the regularizer is separable.
- We consider two methods that take advantage of this:
  1. Two-metric projection on an equivalent problem.
  2. Two-metric sub-gradient projection applied directly.

Optimization with ℓ_1-Regularization

- We want to optimize a smooth function with ℓ_1-regularization:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \ell(\mathbf{x}) + \sum_{i=1}^{n} \lambda_i |x_i|$$

- The non-smooth regularizer breaks quasi-Newton and Hessian-free Newton methods.
- But the regularizer is separable.
- We consider two methods that take advantage of this:
  1. Two-metric projection on an equivalent problem.
  2. Two-metric sub-gradient projection applied directly.

Optimization with ℓ_1-Regularization

- We want to optimize a smooth function with ℓ_1-regularization:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \ell(\mathbf{x}) + \sum_{i=1}^{n} \lambda_i |x_i|$$

- The non-smooth regularizer breaks quasi-Newton and Hessian-free Newton methods.
- But the regularizer is separable.
- We consider two methods that take advantage of this:
  1. Two-metric projection on an equivalent problem.
  2. Two-metric sub-gradient projection applied directly.

Converting to a Bound-Constrained Problem

- We can re-write the non-smooth objective

$$\min_{\mathbf{x}} \ell(\mathbf{x}) + \sum_{i=1}^{n} \lambda_i |x_i|,$$

as a smooth objective with non-negative constraints:

$$\min_{\mathbf{x}} \ell(\mathbf{x}^+ - \mathbf{x}^-) + \sum_{i=1}^{n} \lambda_i (x_i^+ + x_i^-), \text{ subject to } \mathbf{x}^+ \geq \mathbf{0}, \mathbf{x}^- \geq \mathbf{0}.$$

- We can now use methods for bound-constrained optimization of smooth objectives.

Converting to a Bound-Constrained Problem

- We can re-write the non-smooth objective

$$\min_{\mathbf{x}} \ell(\mathbf{x}) + \sum_{i=1}^{n} \lambda_i |x_i|,$$

as a smooth objective with non-negative constraints:

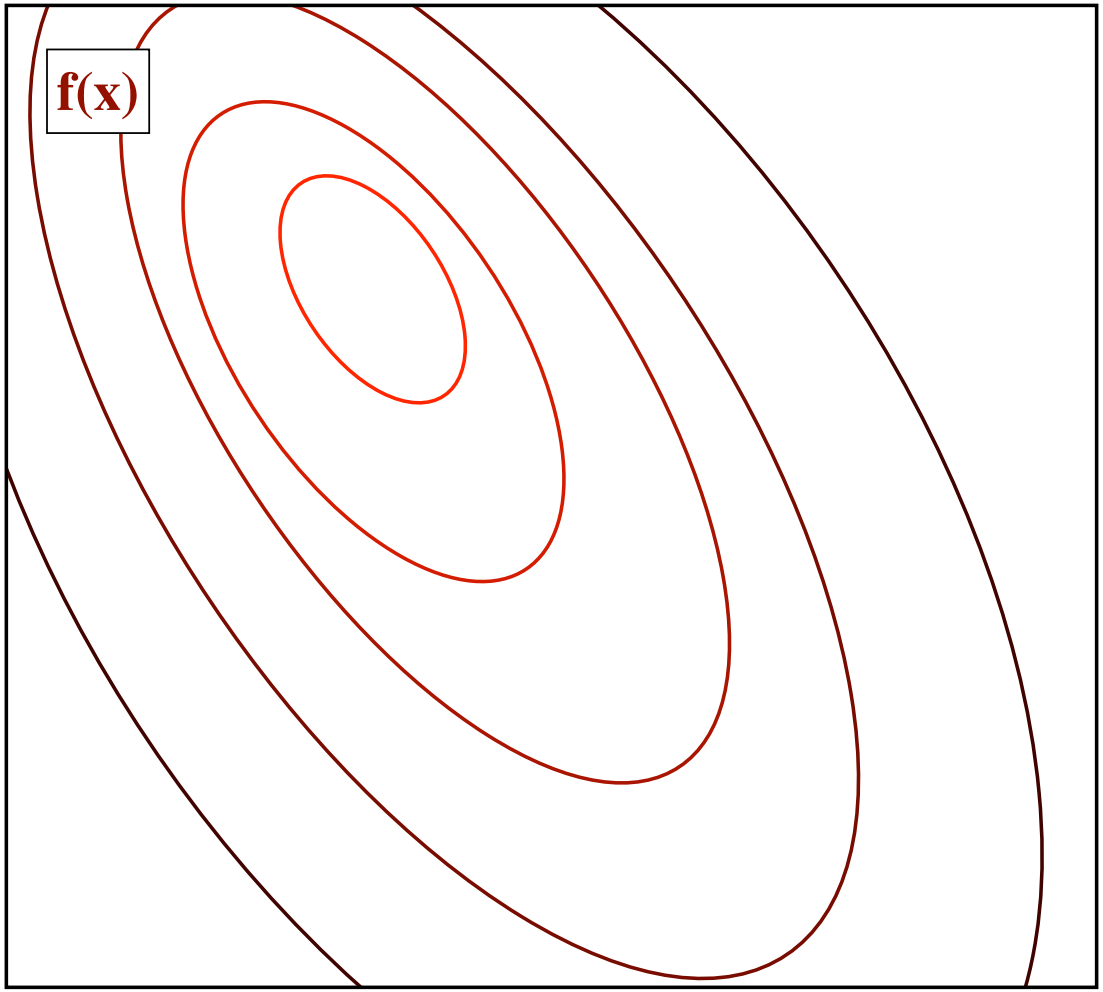
$$\min_{\mathbf{x}} \ell(\mathbf{x}^+ - \mathbf{x}^-) + \sum_{i=1}^{n} \lambda_i (x_i^+ + x_i^-), \text{ subject to } \mathbf{x}^+ \geq \mathbf{0}, \mathbf{x}^- \geq \mathbf{0}.$$

- We can now use methods for bound-constrained optimization of smooth objectives.

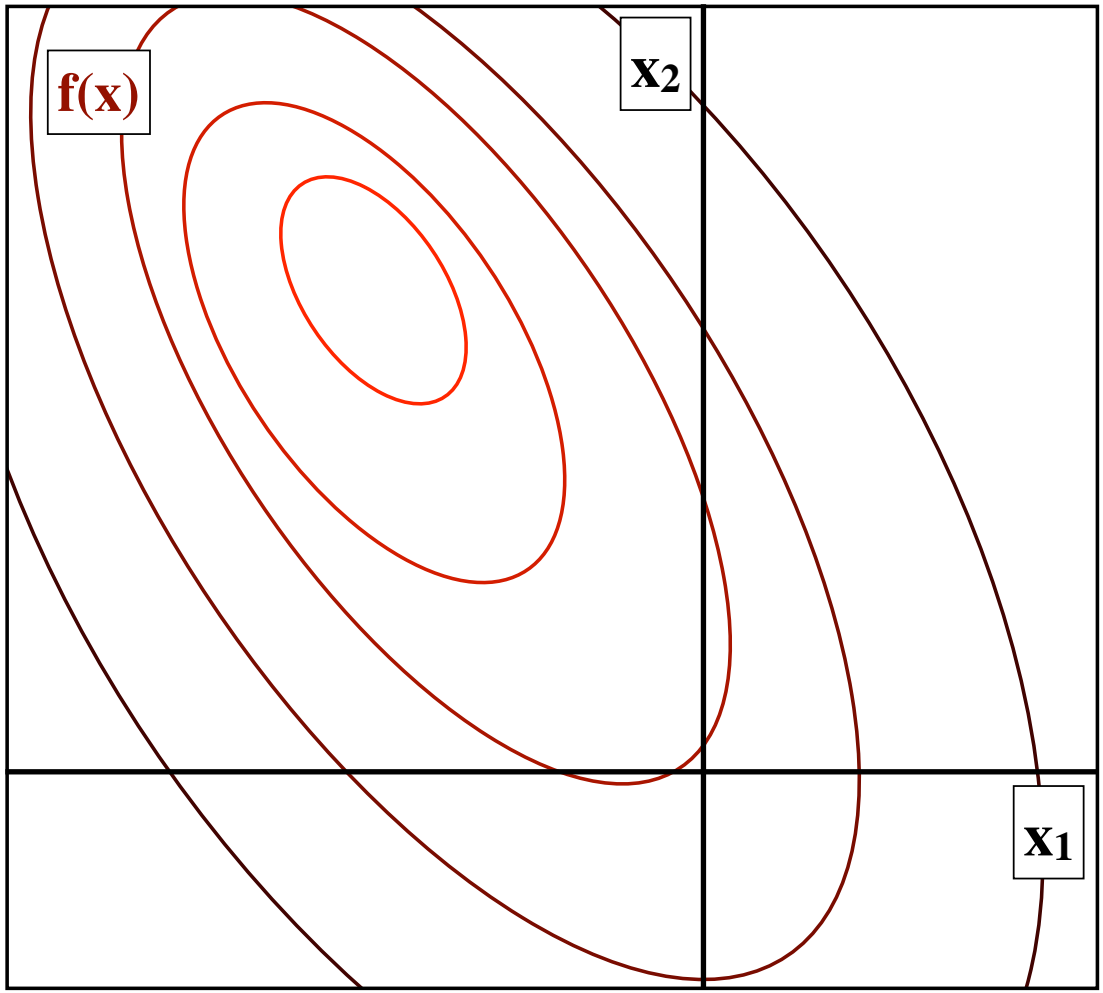
Gradient Projection

- A classic algorithm for bound-constrained problems is gradient projection:
$$\mathbf{x}^{k+1} \leftarrow [\mathbf{x}^k - \alpha \nabla f(\mathbf{x}^k)]^+.$$
- The Armijo condition guarantees sufficient decrease and global convergence.
- However, the convergence rate may be vey slow.

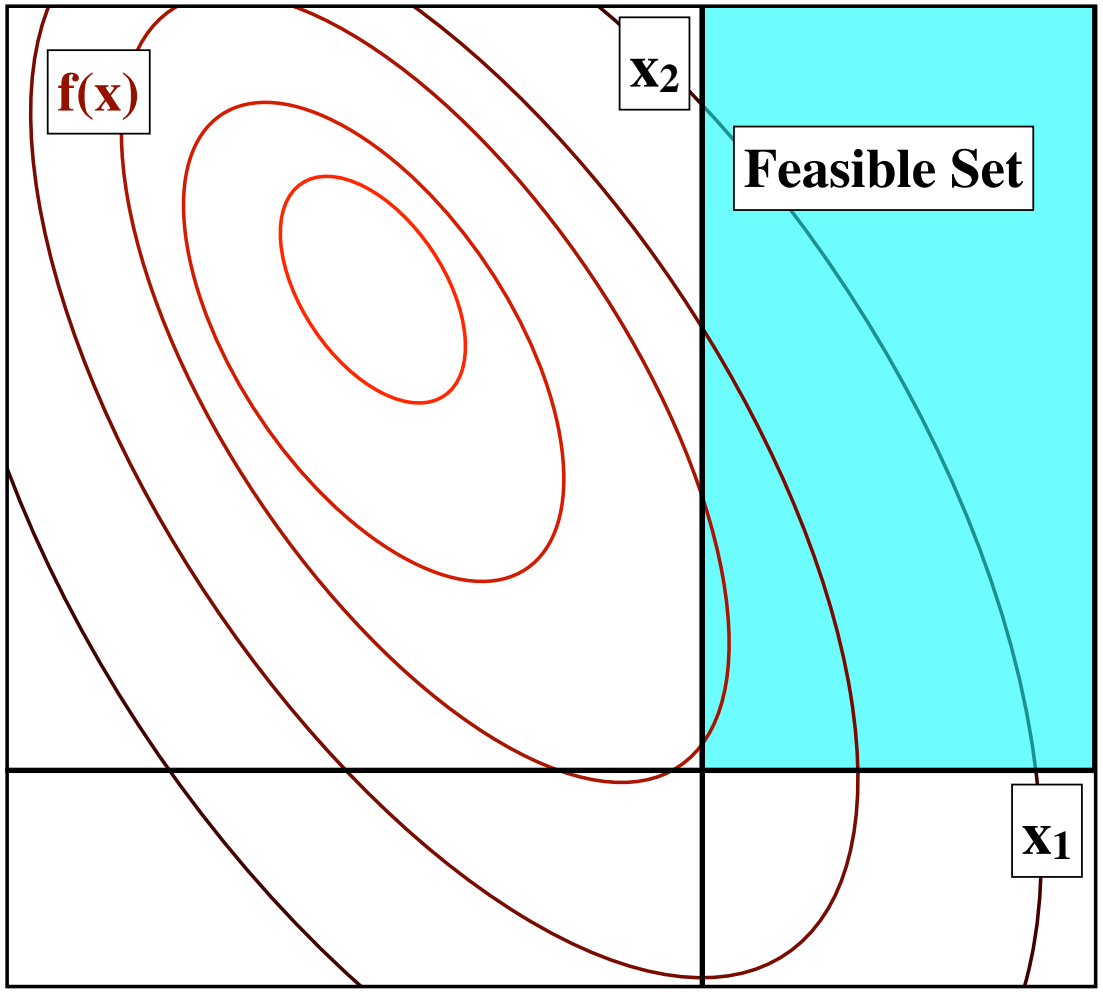
Gradient Projection for Non-Negative Constraints



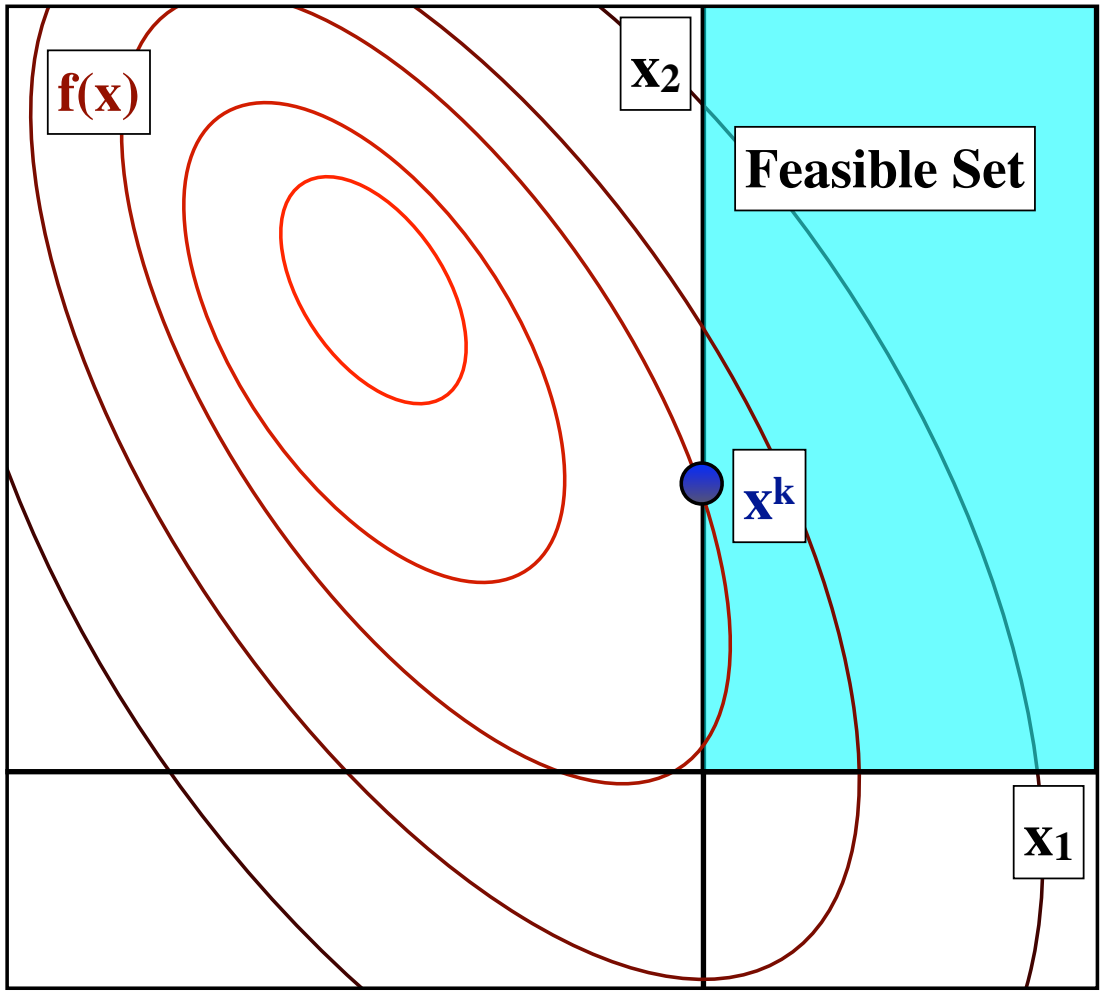
Gradient Projection for Non-Negative Constraints



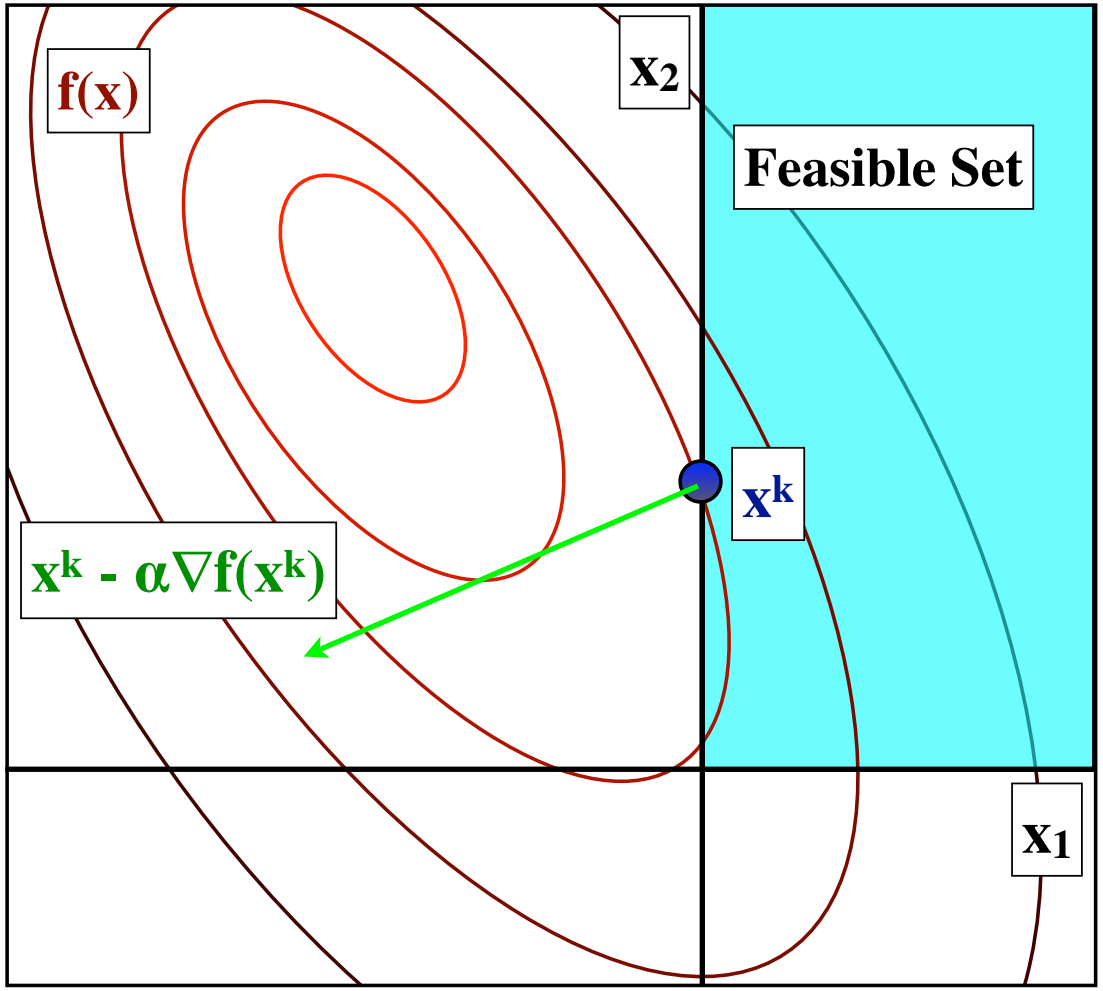
Gradient Projection for Non-Negative Constraints



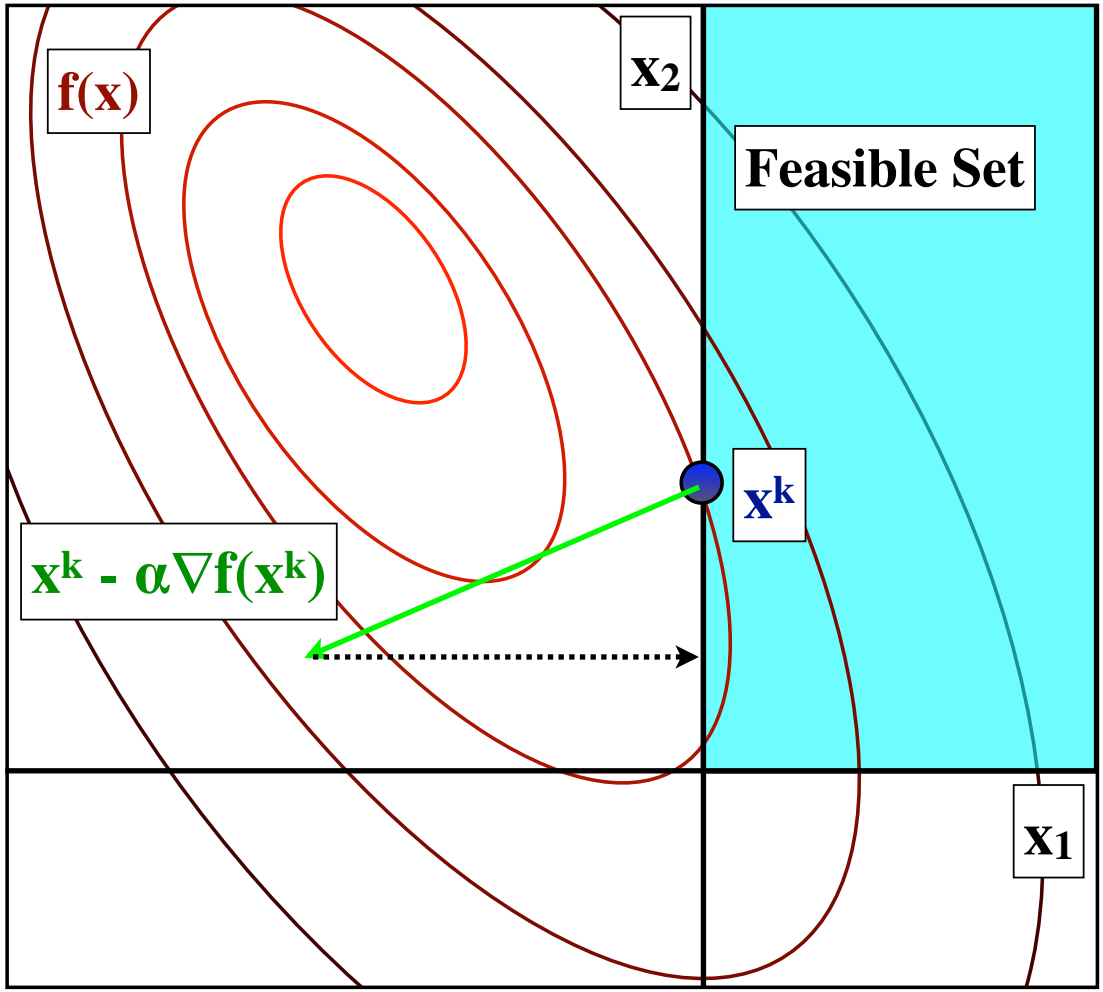
Gradient Projection for Non-Negative Constraints



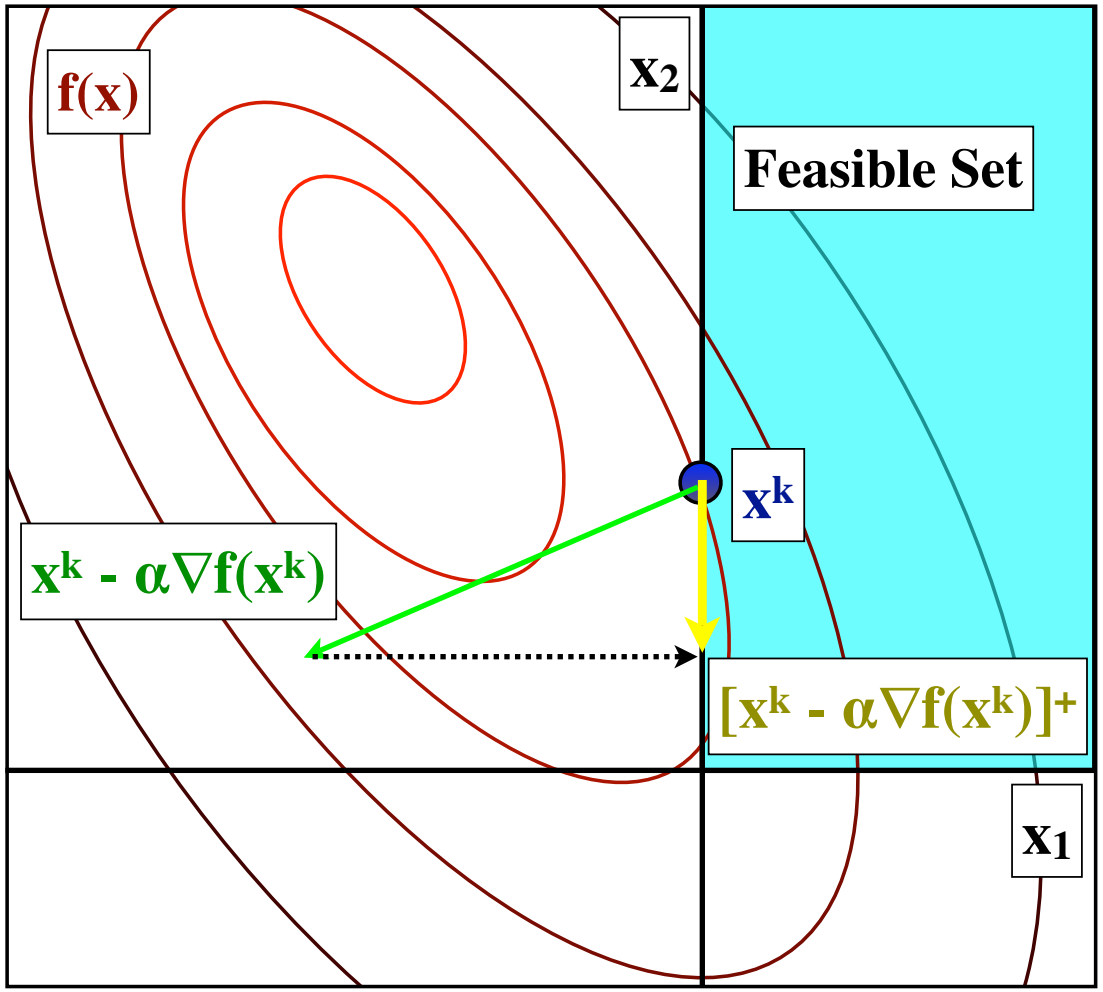
Gradient Projection for Non-Negative Constraints



Gradient Projection for Non-Negative Constraints



Gradient Projection for Non-Negative Constraints



Naive Two-Metric Projection

- To speed convergence, we might consider projecting a Newton-like step:

$$\mathbf{x}^{k+1} \leftarrow [\mathbf{x}^k - \alpha \mathbf{d}^k]^+,$$

where $\mathbf{d}^k$ is the solution of

$$\mathbf{H}^k \mathbf{d}^k = \nabla f(\mathbf{x}^k).$$

- This is known as a two-metric projection algorithm (the gradient and projection norm are different).
- This method does not work. It may not be possible to guarantee descent even if $\mathbf{H}^k$ is positive-definite.

Naive Two-Metric Projection

- To speed convergence, we might consider projecting a Newton-like step:

$$\mathbf{x}^{k+1} \leftarrow [\mathbf{x}^k - \alpha \mathbf{d}^k]^+,$$

where $\mathbf{d}^k$ is the solution of

$$\mathbf{H}^k \mathbf{d}^k = \nabla f(\mathbf{x}^k).$$

- This is known as a two-metric projection algorithm (the gradient and projection norm are different).
- This method does not work. It may not be possible to guarantee descent even if $\mathbf{H}^k$ is positive-definite.

Naive Two-Metric Projection

- To speed convergence, we might consider projecting a Newton-like step:

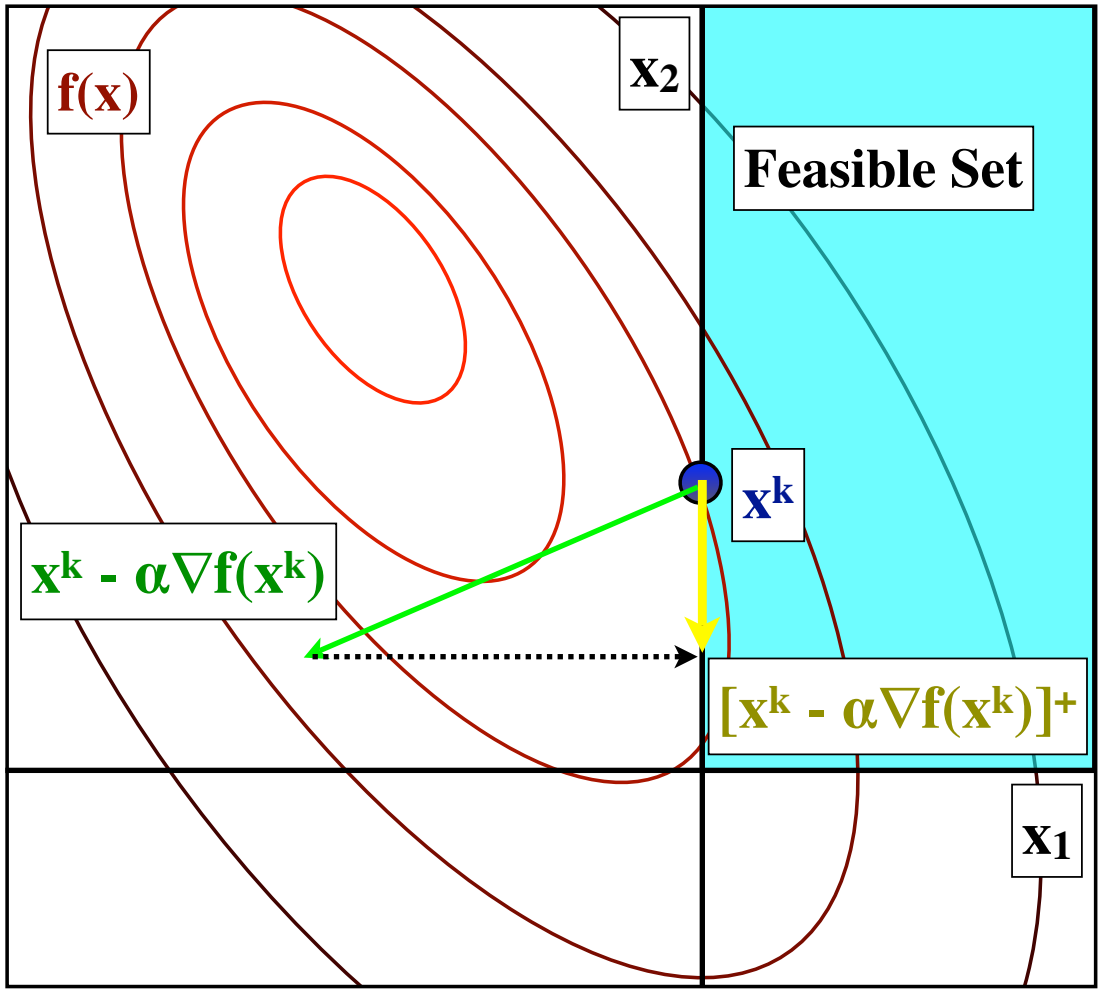
$$\mathbf{x}^{k+1} \leftarrow [\mathbf{x}^k - \alpha \mathbf{d}^k]^+,$$

where $\mathbf{d}^k$ is the solution of

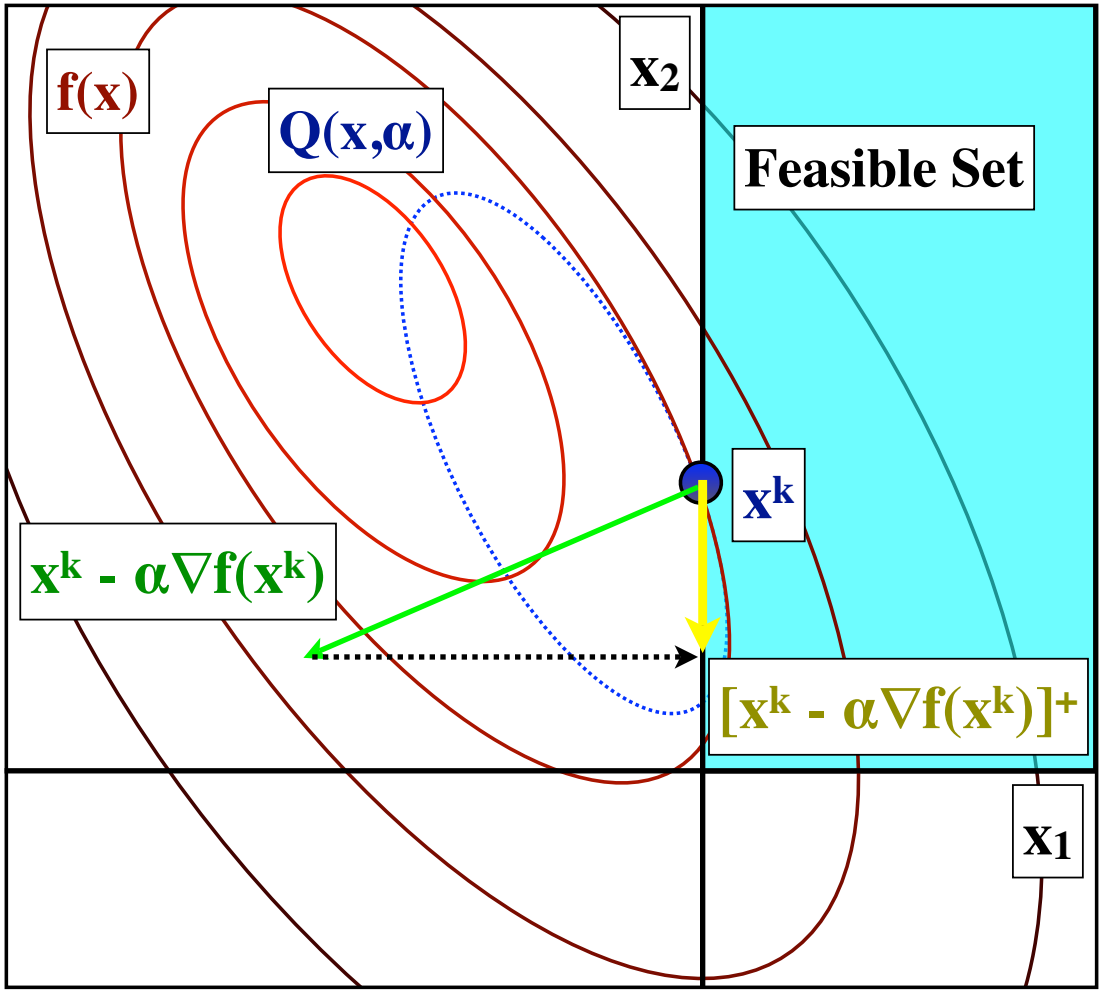
$$\mathbf{H}^k \mathbf{d}^k = \nabla f(\mathbf{x}^k).$$

- This is known as a two-metric projection algorithm (the gradient and projection norm are different).
- This method does not work. It may not be possible to guarantee descent even if $\mathbf{H}^k$ is positive-definite.

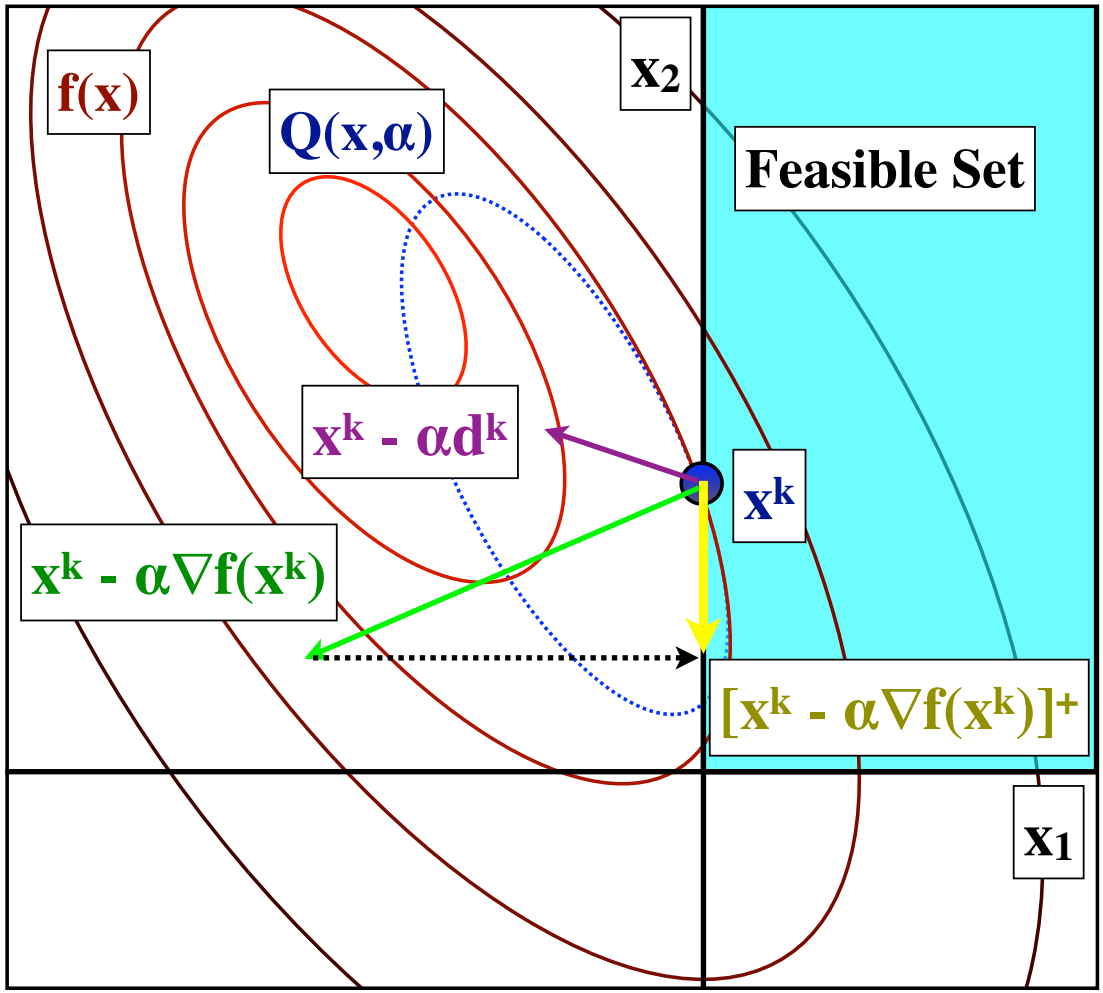
Naive Two-Metric Projection



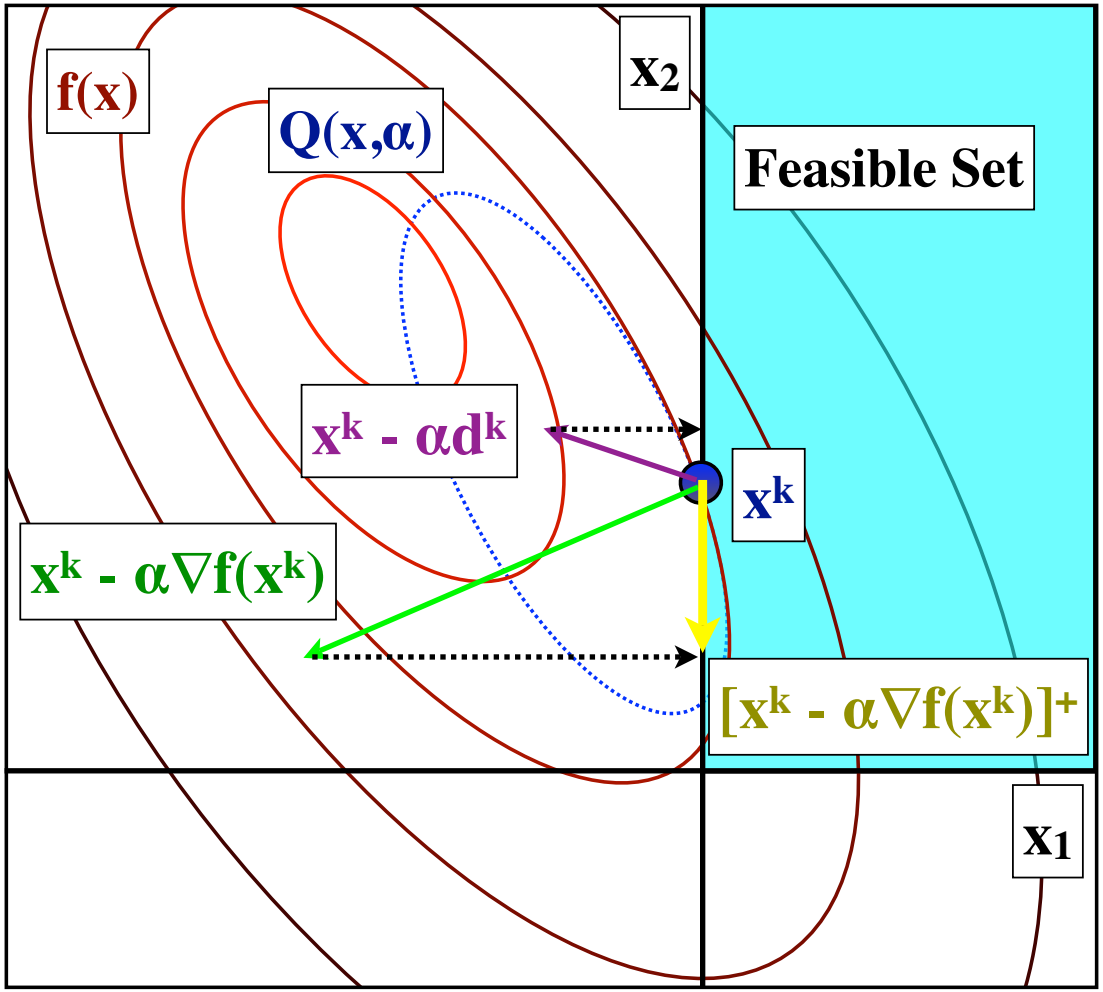
Naive Two-Metric Projection



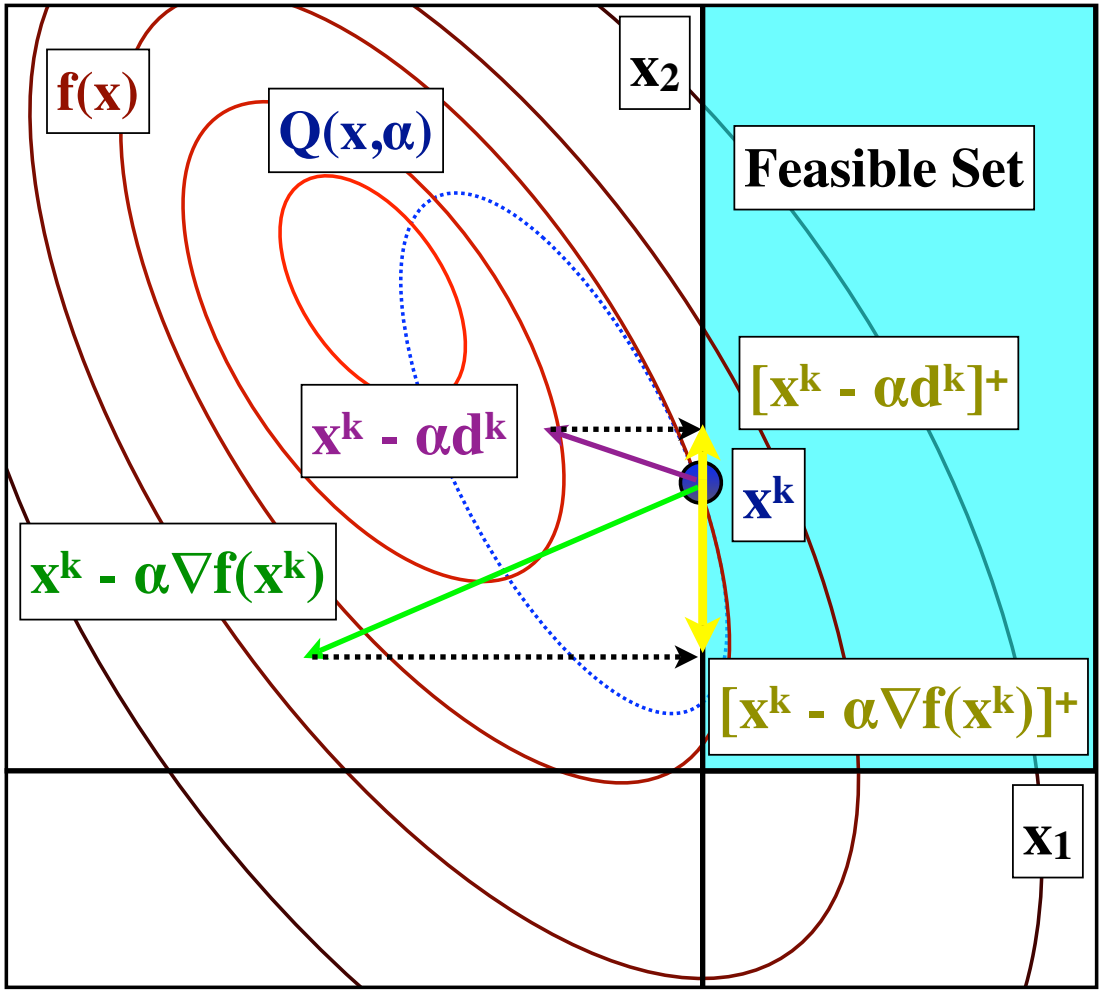
Naive Two-Metric Projection



Naive Two-Metric Projection



Naive Two-Metric Projection



Diagonal-Scaling and Spectral Projected Gradient (SPG)

- We can guarantee descent with further restrictions on $\mathbf{H}^k$.
- For example, we can make $\mathbf{H}^k$ diagonal:

$$\mathbf{x}^{k+1} \leftarrow [\mathbf{x}^k - \alpha \mathbf{D}^k \nabla f(\mathbf{x}^k)]^+$$

- In the spectral projected gradient (SPG) method, we use the Barzilai-Borwein step and non-monotonic Armijo condition:

$$\mathbf{x}^{k+1} \leftarrow [\mathbf{x}^k - \alpha_{bb} \nabla f(\mathbf{x}^k)]^+$$

[Birgin et al., 2000, Figueiredo et al., 2007]

Diagonal-Scaling and Spectral Projected Gradient (SPG)

- We can guarantee descent with further restrictions on $\mathbf{H}^k$.
- For example, we can make $\mathbf{H}^k$ diagonal:

$$\mathbf{x}^{k+1} \leftarrow [\mathbf{x}^k - \alpha \mathbf{D}^k \nabla f(\mathbf{x}^k)]^+$$

- In the spectral projected gradient (SPG) method, we use the Barzilai-Borwein step and non-monotonic Armijo condition:

$$\mathbf{x}^{k+1} \leftarrow [\mathbf{x}^k - \alpha_{bb} \nabla f(\mathbf{x}^k)]^+$$

[Birgin et al., 2000, Figueiredo et al., 2007]

Diagonal-Scaling and Spectral Projected Gradient (SPG)

- We can guarantee descent with further restrictions on $\mathbf{H}^k$.
- For example, we can make $\mathbf{H}^k$ diagonal:

$$\mathbf{x}^{k+1} \leftarrow [\mathbf{x}^k - \alpha \mathbf{D}^k \nabla f(\mathbf{x}^k)]^+$$

- In the spectral projected gradient (SPG) method, we use the Barzilai-Borwein step and non-monotonic Armijo condition:

$$\mathbf{x}^{k+1} \leftarrow [\mathbf{x}^k - \alpha_{bb} \nabla f(\mathbf{x}^k)]^+$$

[Birgin et al., 2000, Figueiredo et al., 2007]

Two-Metric Projection

- Do we need $\mathbf{H}^k$ to be diagonal?
- To guarantee descent, it is sufficient that $\mathbf{H}^k$ is diagonal with respect to a subset of the variables [Gafni & Bertsekas, 1984]:

$$\mathcal{A} \triangleq \{i | x_i^k \leq \epsilon \text{ and } \nabla_i f(\mathbf{x}^k) > 0\}$$

- Re-arranging variables, this leads to a scaling of the form

$$\mathbf{H}^k = \begin{bmatrix} \mathbf{D}^k & \mathbf{0} \\ \mathbf{0} & \bar{\mathbf{H}}^k \end{bmatrix}$$

- We want $\bar{\mathbf{H}}^k$ to approximate the sub-Hessian $\nabla^2_{\mathcal{F}} f(\mathbf{x}^k)$.

Two-Metric Projection

- Do we need $\mathbf{H}^k$ to be diagonal?
- To guarantee descent, it is sufficient that $\mathbf{H}^k$ is diagonal with respect to a subset of the variables [Gafni & Bertsekas, 1984]:

$$\mathcal{A} \triangleq \{i | x_i^k \leq \epsilon \text{ and } \nabla_i f(\mathbf{x}^k) > 0\}$$

- Re-arranging variables, this leads to a scaling of the form

$$\mathbf{H}^k = \begin{bmatrix} \mathbf{D}^k & \mathbf{0} \\ \mathbf{0} & \bar{\mathbf{H}}^k \end{bmatrix}$$

- We want $\bar{\mathbf{H}}^k$ to approximate the sub-Hessian $\nabla^2_{\mathcal{F}} f(\mathbf{x}^k)$.

Two-Metric Projection

- Do we need $\mathbf{H}^k$ to be diagonal?
- To guarantee descent, it is sufficient that $\mathbf{H}^k$ is diagonal with respect to a subset of the variables [Gafni & Bertsekas, 1984]:

$$\mathcal{A} \triangleq \{i | x_i^k \leq \epsilon \text{ and } \nabla_i f(\mathbf{x}^k) > 0\}$$

- Re-arranging variables, this leads to a scaling of the form

$$\mathbf{H}^k = \begin{bmatrix} \mathbf{D}^k & \mathbf{0} \\ \mathbf{0} & \bar{\mathbf{H}}^k \end{bmatrix}$$

- We want $\bar{\mathbf{H}}^k$ to approximate the sub-Hessian $\nabla^2_{\mathcal{F}} f(\mathbf{x}^k)$.

Two-Metric Projection

- Do we need $\mathbf{H}^k$ to be diagonal?
- To guarantee descent, it is sufficient that $\mathbf{H}^k$ is diagonal with respect to a subset of the variables [Gafni & Bertsekas, 1984]:

$$\mathcal{A} \triangleq \{i | x_i^k \leq \epsilon \text{ and } \nabla_i f(\mathbf{x}^k) > 0\}$$

- Re-arranging variables, this leads to a scaling of the form

$$\mathbf{H}^k = \begin{bmatrix} \mathbf{D}^k & \mathbf{0} \\ \mathbf{0} & \bar{\mathbf{H}}^k \end{bmatrix}$$

- We want $\bar{\mathbf{H}}^k$ to approximate the sub-Hessian $\nabla^2_{\mathcal{F}} f(\mathbf{x}^k)$.

Two-Metric Projection

- We can thus write the two-metric projection step as:

$$\mathbf{x}_{\mathcal{A}}^{k+1} \leftarrow [\mathbf{x}_{\mathcal{A}}^{k} - \alpha \mathbf{D}^k \nabla_{\mathcal{A}} f(\mathbf{x}^k)]^+$$

$$\mathbf{x}_{\mathcal{F}}^{k+1} \leftarrow [\mathbf{x}_{\mathcal{F}}^{k} - \alpha \mathbf{d}^k]^+$$

where $\mathbf{d}^k$ is the solution of

$$\bar{\mathbf{H}}^k \mathbf{d}^k = \nabla_{\mathcal{F}} f(\mathbf{x}^k).$$

- We can implement an HFN method by using conjugate gradient to solve this system (very fast if solution is sparse).
- We can implement an L-BFGS method by setting $\bar{\mathbf{H}}^k$ to the L-BFGS approximation.

Two-Metric Projection

- We can thus write the two-metric projection step as:

$$\mathbf{x}_{\mathcal{A}}^{k+1} \leftarrow [\mathbf{x}_{\mathcal{A}}^{k} - \alpha \mathbf{D}^k \nabla_{\mathcal{A}} f(\mathbf{x}^k)]^+$$

$$\mathbf{x}_{\mathcal{F}}^{k+1} \leftarrow [\mathbf{x}_{\mathcal{F}}^{k} - \alpha \mathbf{d}^k]^+$$

  where $\mathbf{d}^k$ is the solution of

$$\bar{\mathbf{H}}^k \mathbf{d}^k = \nabla_{\mathcal{F}} f(\mathbf{x}^k).$$

- We can implement an HFN method by using conjugate gradient to solve this system (very fast if solution is sparse).
- We can implement an L-BFGS method by setting $\bar{\mathbf{H}}^k$ to the L-BFGS approximation.

Two-Metric Projection

- We can thus write the two-metric projection step as:

$$\mathbf{x}_{\mathcal{A}}^{k+1} \leftarrow [\mathbf{x}_{\mathcal{A}}^{k} - \alpha \mathbf{D}^k \nabla_{\mathcal{A}} f(\mathbf{x}^k)]^+$$

$$\mathbf{x}_{\mathcal{F}}^{k+1} \leftarrow [\mathbf{x}_{\mathcal{F}}^{k} - \alpha \mathbf{d}^k]^+$$

where $\mathbf{d}^k$ is the solution of

$$\bar{\mathbf{H}}^k \mathbf{d}^k = \nabla_{\mathcal{F}} f(\mathbf{x}^k).$$

- We can implement an HFN method by using conjugate gradient to solve this system (very fast if solution is sparse).
- We can implement an L-BFGS method by setting $\bar{\mathbf{H}}^k$ to the L-BFGS approximation.

Discussion of Two-Metric Projection

- If the algorithm identifies the optimal manifold, it is equivalent to the unconstrained method on the non-zero variables.
- But should we convert to a bound-constrained problem in the first place?
  - The number of variables is doubled.
  - The transformed problem might be harder
    (the transformed problem is never strongly convex)
- Can we apply tricks from bound-constrained optimization to directly solve to ℓ_1-regularization problems?

Discussion of Two-Metric Projection

- If the algorithm identifies the optimal manifold, it is equivalent to the unconstrained method on the non-zero variables.
- But should we convert to a bound-constrained problem in the first place?
  - The number of variables is doubled.
  - The transformed problem might be harder
    (the transformed problem is never strongly convex)
- Can we apply tricks from bound-constrained optimization to directly solve to ℓ_1-regularization problems?

Discussion of Two-Metric Projection

- If the algorithm identifies the optimal manifold, it is equivalent to the unconstrained method on the non-zero variables.
- But should we convert to a bound-constrained problem in the first place?
  - The number of variables is doubled.
  - The transformed problem might be harder
    (the transformed problem is never strongly convex)
- Can we apply tricks from bound-constrained optimization to directly solve to ℓ_1-regularization problems?

Non-Smooth Steepest Descent

- Recall the motivating problem:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \ell(\mathbf{x}) + \sum_{i=1}^{n} \lambda_i |x_i|$$

- If $f(\mathbf{x})$ is convex, the objective has sub-gradients and directional derivatives everywhere.
- We use $\mathbf{z}^k$ to denote the minimum-norm sub-gradient:

$$\mathbf{z}^k = \underset{\mathbf{z} \in \partial f(\mathbf{x}^k)}{\operatorname{argmin}} ||\mathbf{z}||$$

- The direction that minimizes the directional derivative is $-\mathbf{z}^k$.
- This is the steepest descent direction for non-smooth convex optimization problems.

Non-Smooth Steepest Descent

- Recall the motivating problem:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \ell(\mathbf{x}) + \sum_{i=1}^{n} \lambda_i |x_i|$$

- If $f(\mathbf{x})$ is convex, the objective has sub-gradients and directional derivatives everywhere.
- We use $\mathbf{z}^k$ to denote the minimum-norm sub-gradient:

$$\mathbf{z}^k = \underset{\mathbf{z} \in \partial f(\mathbf{x}^k)}{\operatorname{argmin}} ||\mathbf{z}||$$

- The direction that minimizes the directional derivative is $-\mathbf{z}^k$.
- This is the steepest descent direction for non-smooth convex optimization problems.

Non-Smooth Steepest Descent

- Recall the motivating problem:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \ell(\mathbf{x}) + \sum_{i=1}^{n} \lambda_i |x_i|$$

- If $f(\mathbf{x})$ is convex, the objective has sub-gradients and directional derivatives everywhere.
- We use $\mathbf{z}^k$ to denote the minimum-norm sub-gradient:

$$\mathbf{z}^k = \underset{\mathbf{z} \in \partial f(\mathbf{x}^k)}{\operatorname{argmin}} ||\mathbf{z}||$$

- The direction that minimizes the directional derivative is $-\mathbf{z}^k$.
- This is the steepest descent direction for non-smooth convex optimization problems.

Non-Smooth Steepest Descent

- Recall the motivating problem:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \ell(\mathbf{x}) + \sum_{i=1}^{n} \lambda_i |x_i|$$

- If $f(\mathbf{x})$ is convex, the objective has sub-gradients and directional derivatives everywhere.
- We use $\mathbf{z}^k$ to denote the minimum-norm sub-gradient:

$$\mathbf{z}^k = \underset{\mathbf{z} \in \partial f(\mathbf{x}^k)}{\operatorname{argmin}} ||\mathbf{z}||$$

- The direction that minimizes the directional derivative is $-\mathbf{z}^k$.
- This is the steepest descent direction for non-smooth convex optimization problems.

Non-Smooth Steepest Descent

- Recall the motivating problem:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \ell(\mathbf{x}) + \sum_{i=1}^{n} \lambda_i |x_i|$$

- If $f(\mathbf{x})$ is convex, the objective has sub-gradients and directional derivatives everywhere.
- We use $\mathbf{z}^k$ to denote the minimum-norm sub-gradient:

$$\mathbf{z}^k = \underset{\mathbf{z} \in \partial f(\mathbf{x}^k)}{\operatorname{argmin}} ||\mathbf{z}||$$

- The direction that minimizes the directional derivative is $-\mathbf{z}^k$.
- This is the steepest descent direction for non-smooth convex optimization problems.

Non-Smooth Steepest Descent

- For our problem:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \ell(\mathbf{x}) + \sum_{i=1}^{n} \lambda_i |x_i|,$$

we can compute the minimum-norm sub-gradient coordinate-wise because the ℓ_1-norm is separable:

$$z_i^k \triangleq \begin{cases} \nabla_i \ell(\mathbf{x}) + \lambda_i \operatorname{sign}(x_i), & |x_i| > 0 \\ \nabla_i \ell(\mathbf{x}) - \lambda_i \operatorname{sign}(\nabla_i \ell(\mathbf{x})), & x_i = 0, |\nabla_i \ell(\mathbf{x})| > \lambda_i \\ 0, & x_i = 0, |\nabla_i \ell(\mathbf{x})| \leq \lambda_i \end{cases}$$

- This is the steepest descent direction for ℓ_1-regularization problems.

Non-Smooth Steepest Descent

- For our problem:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \ell(\mathbf{x}) + \sum_{i=1}^{n} \lambda_i |x_i|,$$

we can compute the minimum-norm sub-gradient coordinate-wise because the ℓ_1-norm is separable:

$$z_i^k \triangleq \begin{cases} \nabla_i \ell(\mathbf{x}) + \lambda_i \operatorname{sign}(x_i), & |x_i| > 0 \\ \nabla_i \ell(\mathbf{x}) - \lambda_i \operatorname{sign}(\nabla_i \ell(\mathbf{x})), & x_i = 0, |\nabla_i \ell(\mathbf{x})| > \lambda_i \\ 0, & x_i = 0, |\nabla_i \ell(\mathbf{x})| \leq \lambda_i \end{cases}$$

- This is the steepest descent direction for ℓ_1-regularization problems.

Scaled Non-Smooth Steepest Descent

- We can consider a non-smooth steepest descent step:

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \alpha \mathbf{z}^k.$$

- We can use $\mathbf{z}^k$ in the Armijo condition to guarantee a sufficient decrease.*
- We can even try a Newton-like version:

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \alpha \mathbf{d}^k,$$

where $\mathbf{d}^k$ solves $\mathbf{H}^k \mathbf{d}^k = \mathbf{z}^k$.

- However, there are two problems with this step:
  1. The iterations are unlikely to be sparse.
  2. It doesn't guarantee descent, even if $\mathbf{H}^k$ is positive-definite.

Scaled Non-Smooth Steepest Descent

- We can consider a non-smooth steepest descent step:

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \alpha \mathbf{z}^k.$$

- We can use $\mathbf{z}^k$ in the Armijo condition to guarantee a sufficient decrease.*
- We can even try a Newton-like version:

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \alpha \mathbf{d}^k,$$

where $\mathbf{d}^k$ solves $\mathbf{H}^k \mathbf{d}^k = \mathbf{z}^k$.

- However, there are two problems with this step:
  1. The iterations are unlikely to be sparse.
  2. It doesn't guarantee descent, even if $\mathbf{H}^k$ is positive-definite.

Scaled Non-Smooth Steepest Descent

- We can consider a non-smooth steepest descent step:

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \alpha \mathbf{z}^k.$$

- We can use $\mathbf{z}^k$ in the Armijo condition to guarantee a sufficient decrease.*
- We can even try a Newton-like version:

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k - \alpha \mathbf{d}^k,$$

  where $\mathbf{d}^k$ solves $\mathbf{H}^k \mathbf{d}^k = \mathbf{z}^k$.
- However, there are two problems with this step:
  1. The iterations are unlikely to be sparse.
  2. It doesn't guarantee descent, even if $\mathbf{H}^k$ is positive-definite.

Scaled Non-Smooth Steepest Descent

- To get sparse iterates, many authors use orthant projection:

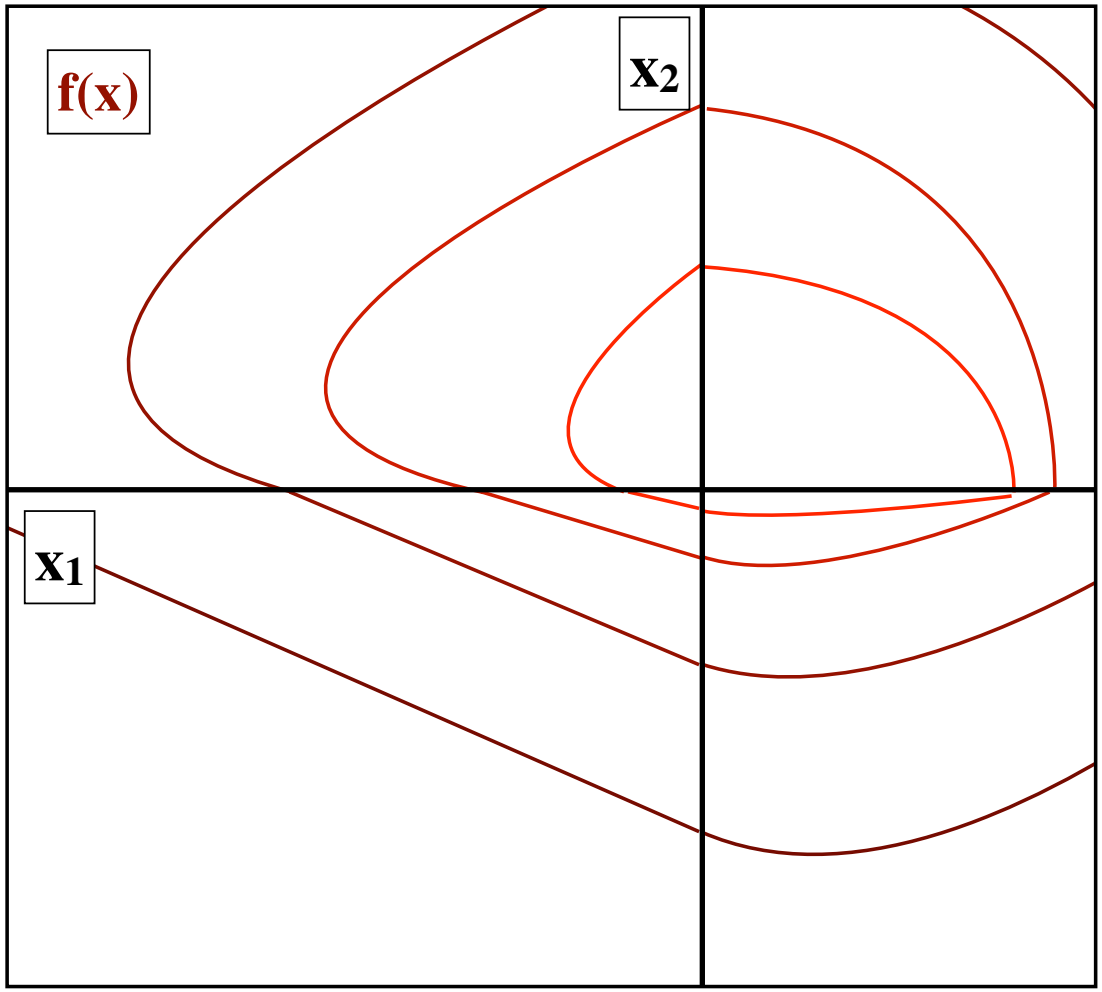
$$\mathbf{x}^{k+1} \leftarrow \mathcal{P}_{\mathcal{O}}[\mathbf{x}^k - \alpha \mathbf{d}^k, \mathbf{x}^k],$$

where [Osborne et al., 2000, Andrew & Gao, 2007]

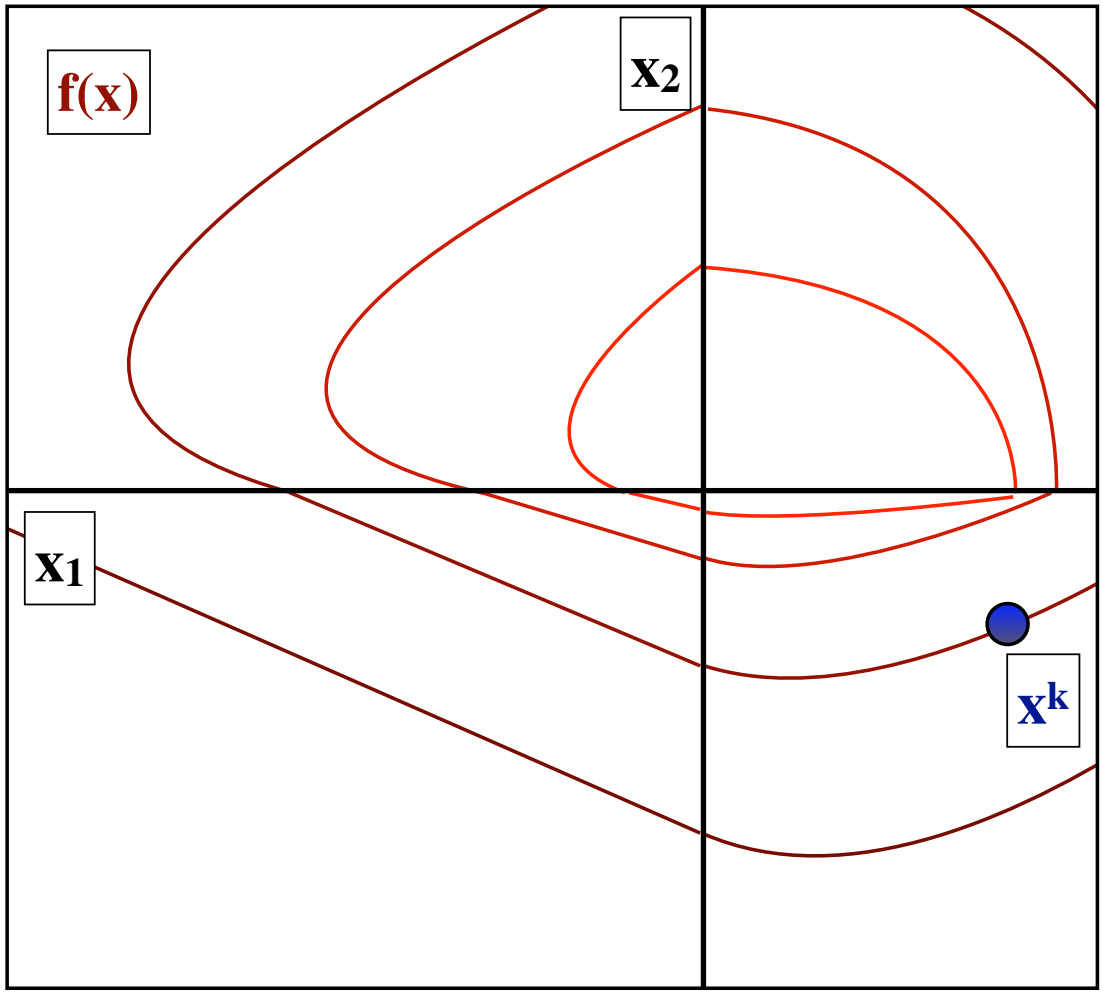
$$\mathcal{P}_{\mathcal{O}}(\mathbf{y}, \mathbf{x})_i \triangleq \begin{cases} 0 & \text{if } x_i y_i < 0 \\ y_i & \text{otherwise} \end{cases}$$

- Sets variables that change signs to exactly zero:
  1. Effecitve at sparsifying the solution.
  2. Restricts quadratic approximation to valid region.

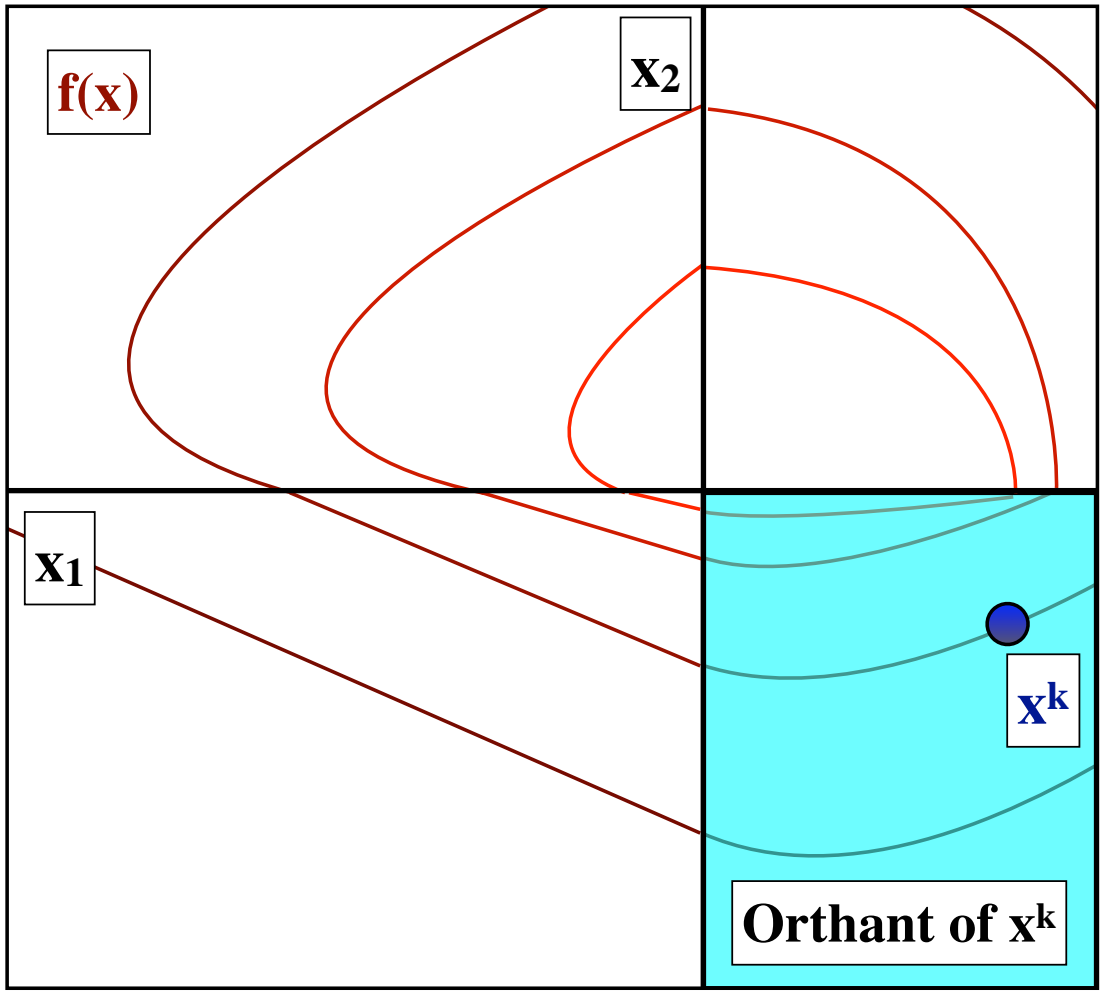
Orthant Projection



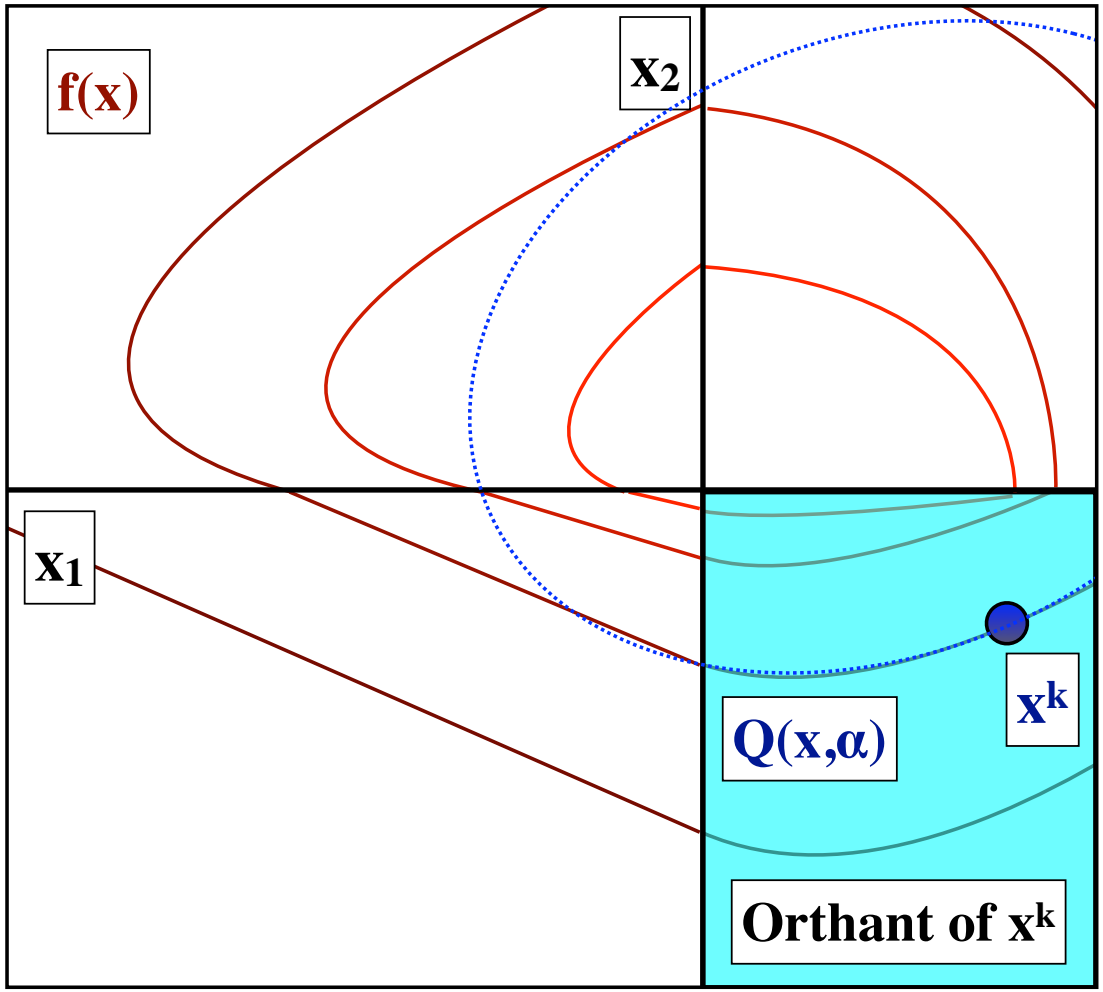
Orthant Projection



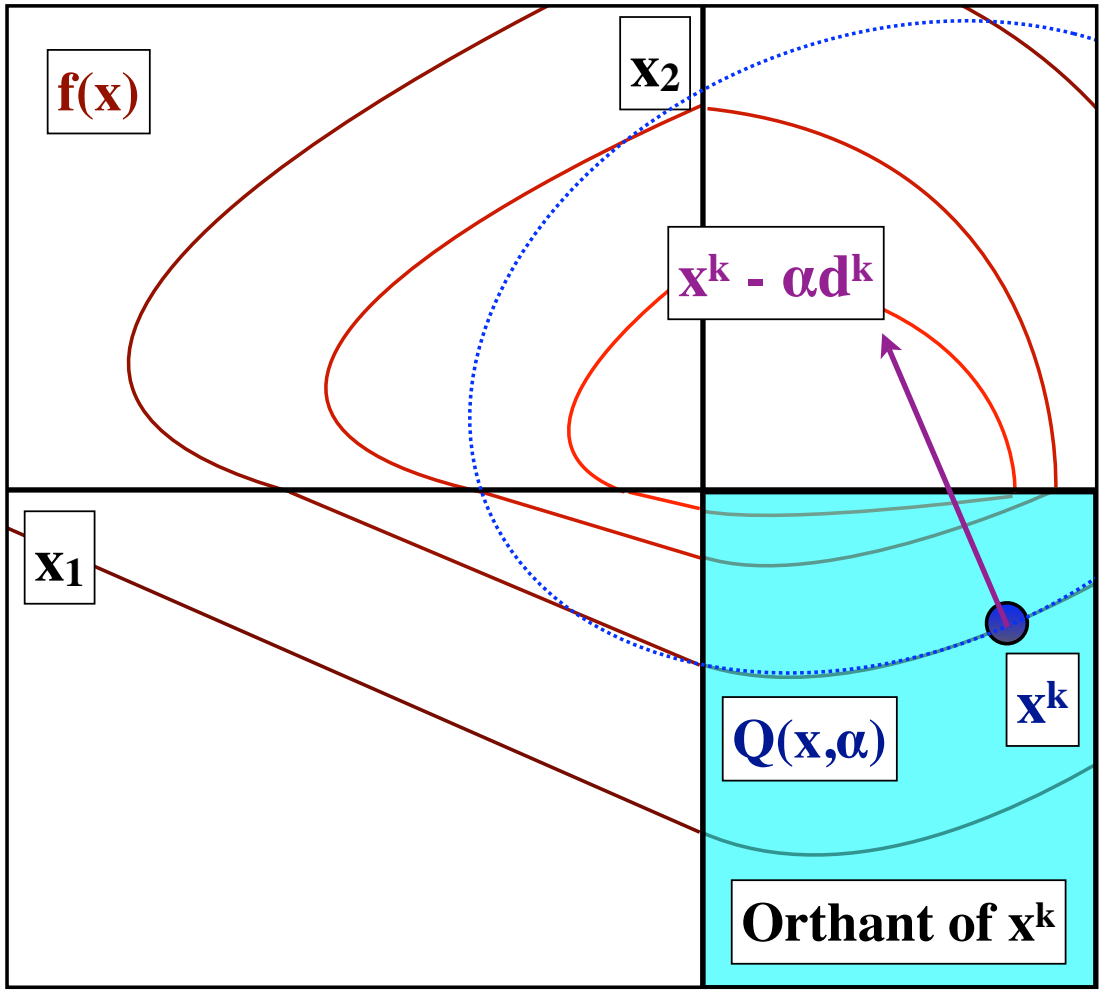
Orthant Projection



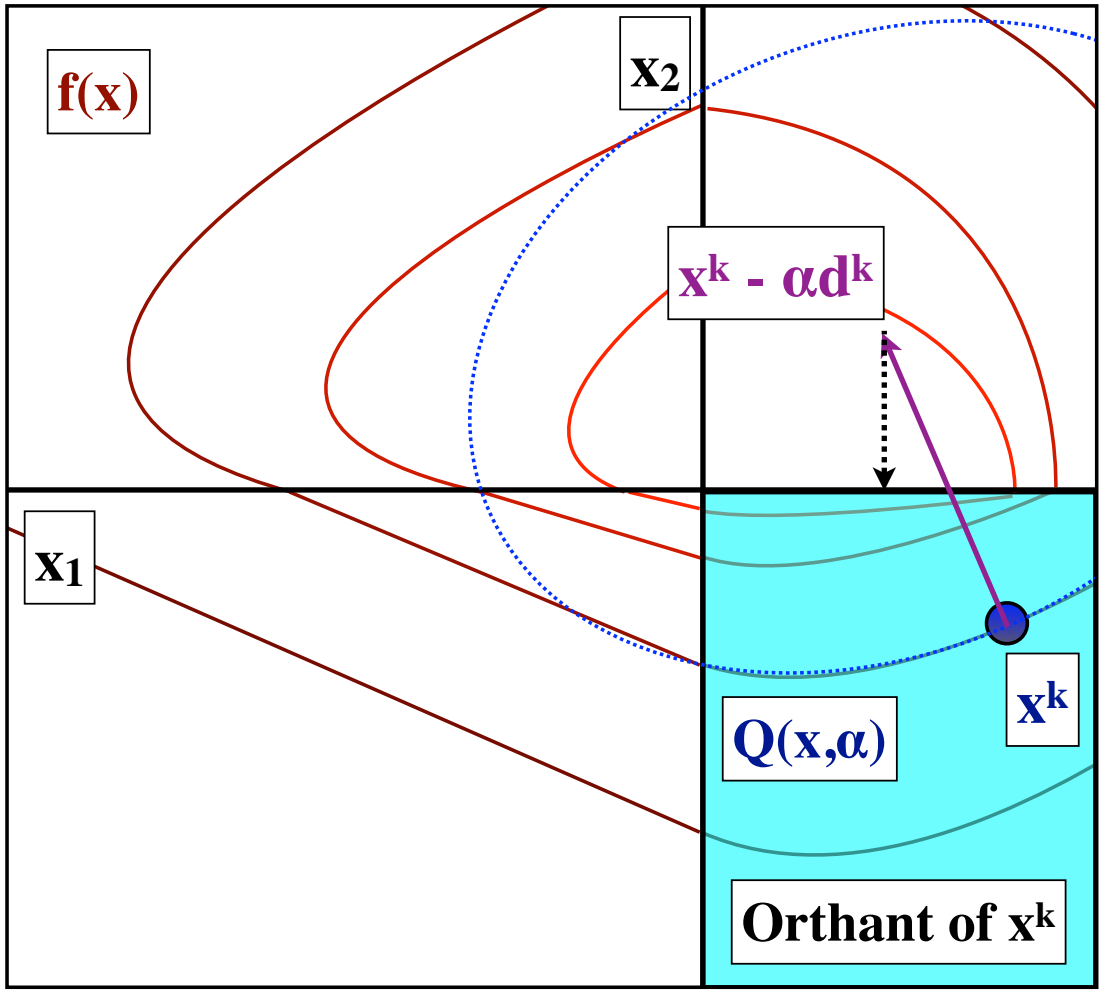
Orthant Projection



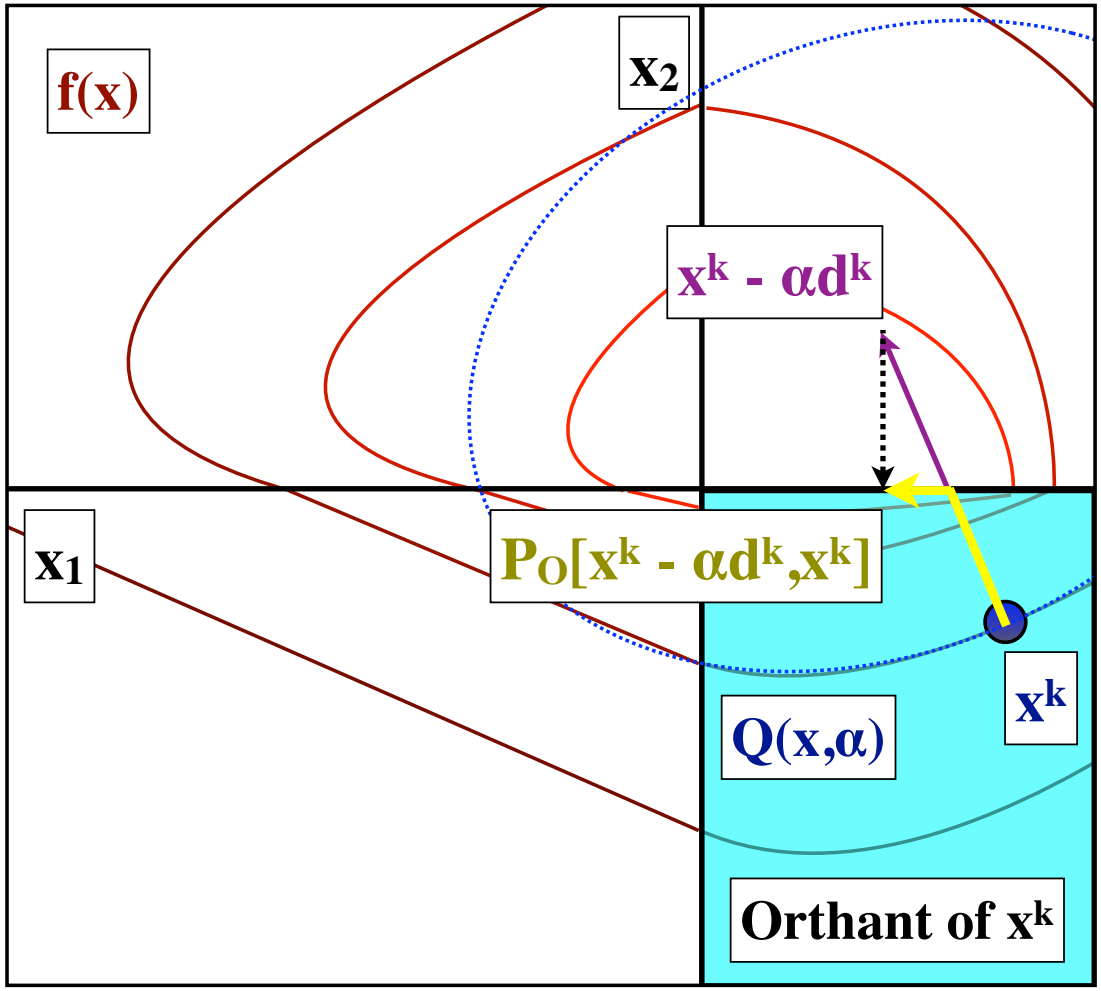
Orthant Projection



Orthant Projection



Orthant Projection



Two-Metric Sub-Gradient Projection

- There are several ways to guarantee descent.
- We could use a diagonal scaling $\mathbf{D}^k$:

$$\mathbf{x}^{k+1} \leftarrow \mathcal{P}_{\mathcal{O}}[\mathbf{x}^k - \alpha \mathbf{D}^k \mathbf{z}^k, \mathbf{x}^k].$$

- We could use the Barzilai-Borwein step with non-monotonic line searches:

$$\mathbf{x}^{k+1} \leftarrow \mathcal{P}_{\mathcal{O}}[\mathbf{x}^k - \alpha_{bb} \mathbf{z}^k, \mathbf{x}^k].$$

- We could make the scaling diagonal with respect to an appropriate subset of the variables:

$$\mathcal{A} \triangleq \{i \,|\, |x_i^k| \leq \epsilon\}$$

Two-Metric Sub-Gradient Projection

- There are several ways to guarantee descent.
- We could use a diagonal scaling $\mathbf{D}^k$:

$$\mathbf{x}^{k+1} \leftarrow \mathcal{P}_{\mathcal{O}}[\mathbf{x}^k - \alpha \mathbf{D}^k \mathbf{z}^k, \mathbf{x}^k].$$

- We could use the Barzilai-Borwein step with non-monotonic line searches:

$$\mathbf{x}^{k+1} \leftarrow \mathcal{P}_{\mathcal{O}}[\mathbf{x}^k - \alpha_{bb} \mathbf{z}^k, \mathbf{x}^k].$$

- We could make the scaling diagonal with respect to an appropriate subset of the variables:

$$\mathcal{A} \triangleq \{i \,|\, |x_i^k| \leq \epsilon\}$$

Two-Metric Sub-Gradient Projection

- There are several ways to guarantee descent.
- We could use a diagonal scaling $\mathbf{D}^k$:

$$\mathbf{x}^{k+1} \leftarrow \mathcal{P}_{\mathcal{O}}[\mathbf{x}^k - \alpha \mathbf{D}^k \mathbf{z}^k, \mathbf{x}^k].$$

- We could use the Barzilai-Borwein step with non-monotonic line searches:

$$\mathbf{x}^{k+1} \leftarrow \mathcal{P}_{\mathcal{O}}[\mathbf{x}^k - \alpha_{bb} \mathbf{z}^k, \mathbf{x}^k].$$

- We could make the scaling diagonal with respect to an appropriate subset of the variables:

$$\mathcal{A} \triangleq \{i \,|\, |x_i^k| \leq \epsilon\}$$

Two-Metric Sub-Gradient Projection

- There are several ways to guarantee descent.
- We could use a diagonal scaling $\mathbf{D}^k$:

$$\mathbf{x}^{k+1} \leftarrow \mathcal{P}_{\mathcal{O}}[\mathbf{x}^k - \alpha \mathbf{D}^k \mathbf{z}^k, \mathbf{x}^k].$$

- We could use the Barzilai-Borwein step with non-monotonic line searches:

$$\mathbf{x}^{k+1} \leftarrow \mathcal{P}_{\mathcal{O}}[\mathbf{x}^k - \alpha_{bb} \mathbf{z}^k, \mathbf{x}^k].$$

- We could make the scaling diagonal with respect to an appropriate subset of the variables:

$$\mathcal{A} \triangleq \{i \| x_i^k| \leq \epsilon\}$$

Two-Metric Sub-Gradient Projection

- The latter leads to a two-metric sub-gradient projection method:

$$\mathbf{x}_{\mathcal{A}}^{k+1} \leftarrow \mathcal{P}_{\mathcal{O}}[\mathbf{x}_{\mathcal{A}}^{k} - \alpha \mathbf{D}^{k} \mathbf{z}_{\mathcal{A}}^{k}, \mathbf{x}_{\mathcal{A}}^{k}],$$

$$\mathbf{x}_{\mathcal{F}}^{k+1} \leftarrow \mathcal{P}_{\mathcal{O}}[\mathbf{x}_{\mathcal{F}}^{k} - \alpha \mathbf{d}^{k}, \mathbf{x}_{\mathcal{F}}^{k}],$$

where $\mathbf{d}^k$ solves

$$\bar{\mathbf{H}}^k \mathbf{d}^k = \nabla_{\mathcal{F}} f(\mathbf{x}^k).$$

- We can derive HFN and L-BFGS methods as before.
- One choice of $\mathbf{D}^k$ might be the Barzilai-Borwein step $\alpha_{bb}\mathbf{I}$.

Two-Metric Sub-Gradient Projection

- The latter leads to a two-metric sub-gradient projection method:

$$\mathbf{x}_{\mathcal{A}}^{k+1} \leftarrow \mathcal{P}_{\mathcal{O}}[\mathbf{x}_{\mathcal{A}}^{k} - \alpha \mathbf{D}^{k} \mathbf{z}_{\mathcal{A}}^{k}, \mathbf{x}_{\mathcal{A}}^{k}],$$

$$\mathbf{x}_{\mathcal{F}}^{k+1} \leftarrow \mathcal{P}_{\mathcal{O}}[\mathbf{x}_{\mathcal{F}}^{k} - \alpha \mathbf{d}^{k}, \mathbf{x}_{\mathcal{F}}^{k}],$$

  where $\mathbf{d}^k$ solves

$$\bar{\mathbf{H}}^{k} \mathbf{d}^{k} = \nabla_{\mathcal{F}} f(\mathbf{x}^{k}).$$

- We can derive HFN and L-BFGS methods as before.
- One choice of $\mathbf{D}^k$ might be the Barzilai-Borwein step $\alpha_{bb}\mathbf{I}$.

Two-Metric Sub-Gradient Projection

- The latter leads to a two-metric sub-gradient projection method:

$$\mathbf{x}_{\mathcal{A}}^{k+1} \leftarrow \mathcal{P}_{\mathcal{O}}[\mathbf{x}_{\mathcal{A}}^{k} - \alpha \mathbf{D}^{k} \mathbf{z}_{\mathcal{A}}^{k}, \mathbf{x}_{\mathcal{A}}^{k}],$$

$$\mathbf{x}_{\mathcal{F}}^{k+1} \leftarrow \mathcal{P}_{\mathcal{O}}[\mathbf{x}_{\mathcal{F}}^{k} - \alpha \mathbf{d}^{k}, \mathbf{x}_{\mathcal{F}}^{k}],$$

  where $\mathbf{d}^k$ solves

$$\bar{\mathbf{H}}^{k} \mathbf{d}^{k} = \nabla_{\mathcal{F}} f(\mathbf{x}^{k}).$$

- We can derive HFN and L-BFGS methods as before.
- One choice of $\mathbf{D}^k$ might be the Barzilai-Borwein step $\alpha_{bb}\mathbf{I}$.

Advantages

The method has several appealing properties:

- No variable doubling.
- No losing strong convexity.
- Gives sparse iterations.
- Allows warm-starting.
- Many variables can be set to zero at once.
- Many variables can move away from zero at once.
- If it identifies the optimal sparsity pattern, it is equivalent Newton's method on the non-zero variables.

Practical Issues

For very large-scale problems with very spare solutions, practical methods seem to need to consider two more issues:

- **Continuation**: Start with a large value of λ and progressively decrease it.
- **Sub-Optimization**: Ignore variables that are unlikely to move away from zero (temporarily or permanently).

Extensions

- The algorithm extends to problems of the form

$$\min_{\mathbf{l} \preceq \mathbf{x} \preceq \mathbf{u}} \ell(\mathbf{x}) + r(\mathbf{x}),$$

  where $r(\mathbf{x})$ is separable and differentiable almost everywhere.
- Two-metric projection algorithms also exist for other constraints [Gafni & Bertsekas, 1984].

Discussion

- There are several closely related methods for using curvature in ℓ_1-regularization algorithms, including:
  - Perkins et al. [2003].
  - Andrew & Gao [2007].
  - Shi et al. [2007].
  - Kim & Park [2010].
- While descent is guaranteed, convergence theory is not fully developed:
  - Global convergence.
  - Active set identification.

Outline

Problems with non-separable case

- We now turn to the group ℓ_1-regularization problem:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \ell(\mathbf{x}) + \sum_g \lambda_g ||\mathbf{x}_g||_2$$

- The regularizer is not separable.
- But the regularizer is simple.
- We consider two methods that take advantage of this:
  - Inexact projected Newton on an equivalent problem.
  - Inexact proximal Newton applied directly.

Problems with non-separable case

- We now turn to the group ℓ_1-regularization problem:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \ell(\mathbf{x}) + \sum_g \lambda_g ||\mathbf{x}_g||_2$$

- The regularizer is not separable.
- But the regularizer is simple.
- We consider two methods that take advantage of this:
  - Inexact projected Newton on an equivalent problem.
  - Inexact proximal Newton applied directly.

Problems with non-separable case

- We now turn to the group ℓ_1-regularization problem:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \ell(\mathbf{x}) + \sum_g \lambda_g ||\mathbf{x}_g||_2$$

- The regularizer is not separable.
- But the regularizer is simple.
- We consider two methods that take advantage of this:
  - Inexact projected Newton on an equivalent problem.
  - Inexact proximal Newton applied directly.

Problems with non-separable case

- We now turn to the group ℓ_1-regularization problem:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \ell(\mathbf{x}) + \sum_g \lambda_g ||\mathbf{x}_g||_2$$

- The regularizer is not separable.
- But the regularizer is simple.
- We consider two methods that take advantage of this:
  - Inexact projected Newton on an equivalent problem.
  - Inexact proximal Newton applied directly.

Converting to a Constrained Problem

- We can introduce extra variables $\mathbf{t}$ to convert the problem into a smooth optimization with cone constraints:

$$\min_{\mathbf{x},\mathbf{t}} \ell(\mathbf{x}) + \sum_g \lambda_g t_g, \text{ subject to } ||\mathbf{x}_g||_2 \leq t_g, \forall_g.$$

- Alternately, we can the optimize over the norm ball:

$$\min_{\mathbf{x}} \ell(\mathbf{x}), \text{ subject to } \sum_g \lambda_g ||\mathbf{x}_g||_2 \leq \tau$$

- In both cases the constraints are simple; we can compute the projection in linear time [Boyd & Vandenberghe, 2004, Exercise 8.3(c), van den Berg et al., 2008].

Converting to a Constrained Problem

- We can introduce extra variables $\mathbf{t}$ to convert the problem into a smooth optimization with cone constraints:

$$\min_{\mathbf{x},\mathbf{t}} \ell(\mathbf{x}) + \sum_g \lambda_g t_g, \text{ subject to } ||\mathbf{x}_g||_2 \leq t_g, \forall_g.$$

- Alternately, we can the optimize over the norm ball:

$$\min_{\mathbf{x}} \ell(\mathbf{x}), \text{ subject to } \sum_g \lambda_g ||\mathbf{x}_g||_2 \leq \tau$$

- In both cases the constraints are simple; we can compute the projection in linear time [Boyd & Vandenberghe, 2004, Exercise 8.3(c), van den Berg et al., 2008].

Projected Gradient

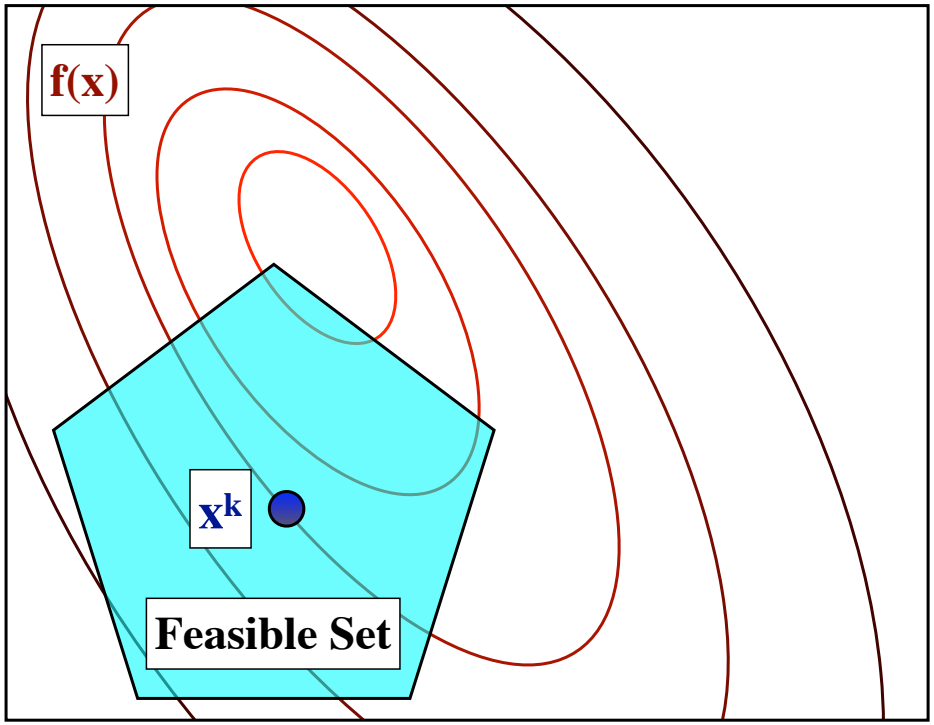
Recall the basic projected gradient step:

$$\mathbf{x}^{k+1} \leftarrow \underset{\mathbf{x} \in \mathcal{C}}{\operatorname{argmin}} ||\mathbf{x} - (\mathbf{x}^k - \alpha \nabla f(\mathbf{x}^k))||_2^2$$

Projected Gradient

Recall the basic projected gradient step:

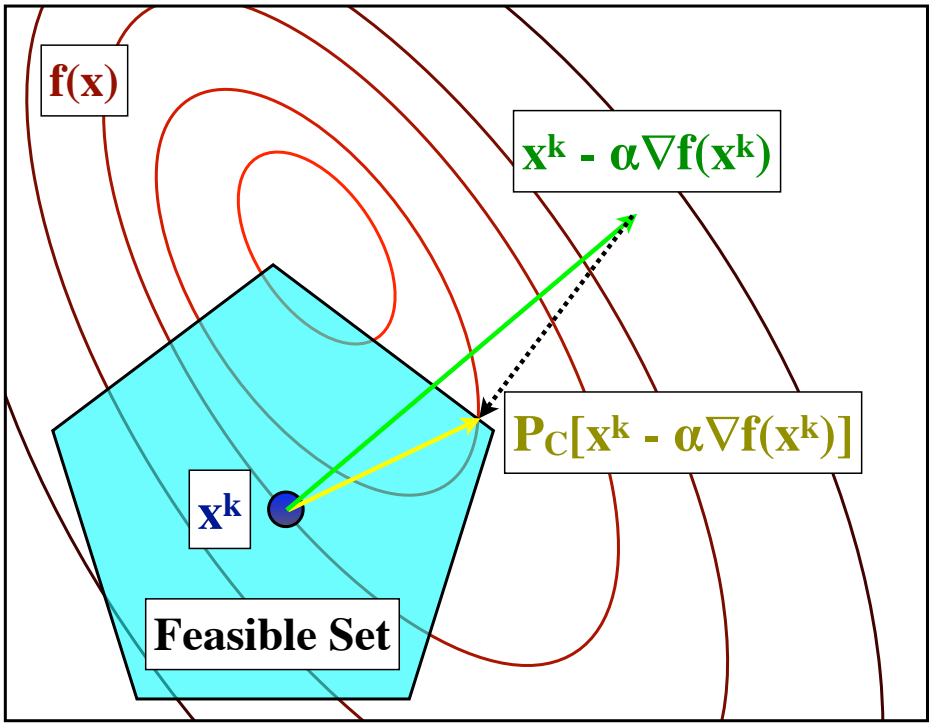
$$\mathbf{x}^{k+1} \leftarrow \underset{\mathbf{x}\in\mathcal{C}}{\operatorname{argmin}} ||\mathbf{x} - (\mathbf{x}^k - \alpha\nabla f(\mathbf{x}^k))||_2^2$$



Projected Gradient

Recall the basic projected gradient step:

$$\mathbf{x}^{k+1} \leftarrow \underset{\mathbf{x}\in\mathcal{C}}{\operatorname{argmin}} ||\mathbf{x} - (\mathbf{x}^k - \alpha\nabla f(\mathbf{x}^k))||_2^2$$



Projected Gradient and Projected Newton

- To speed the convergence, we might consider a scaled step:

$$\mathbf{x}^{k+1} \leftarrow \underset{\mathbf{x}\in\mathcal{C}}{\operatorname{argmin}} ||\mathbf{x} - (\mathbf{x}^k - \alpha\mathbf{d}^k)||_2^2,$$

where $\mathbf{d}^k$ solves $\mathbf{H}^k\mathbf{d}^k = \nabla f(\mathbf{x}^k)$.

- As we saw, in general this does not work.
- To guarantee descent, projected Newton methods project under a quadratic norm:

$$\mathbf{x}^{k+1} \leftarrow \underset{\mathbf{x}\in\mathcal{C}}{\operatorname{argmin}} ||\mathbf{x} - (\mathbf{x}^k - \alpha\mathbf{d}^k)||_{\mathbf{H}^k}^2,$$

where $||\mathbf{y}||_{\mathbf{H}^k} = \sqrt{\mathbf{y}^T\mathbf{H}^k\mathbf{y}}$ [Levitin & Polyak, 1966].

Projected Gradient and Projected Newton

- To speed the convergence, we might consider a scaled step:

$$\mathbf{x}^{k+1} \leftarrow \underset{\mathbf{x}\in\mathcal{C}}{\operatorname{argmin}} ||\mathbf{x} - (\mathbf{x}^k - \alpha\mathbf{d}^k)||_2^2,$$

  where $\mathbf{d}^k$ solves $\mathbf{H}^k\mathbf{d}^k = \nabla f(\mathbf{x}^k)$.
- As we saw, in general this does not work.
- To guarantee descent, projected Newton methods project under a quadratic norm:

$$\mathbf{x}^{k+1} \leftarrow \underset{\mathbf{x}\in\mathcal{C}}{\operatorname{argmin}} ||\mathbf{x} - (\mathbf{x}^k - \alpha\mathbf{d}^k)||_{\mathbf{H}^k}^2,$$

  where $||\mathbf{y}||_{\mathbf{H}^k} = \sqrt{\mathbf{y}^T\mathbf{H}^k\mathbf{y}}$ [Levitin & Polyak, 1966].

Projected Gradient and Projected Newton

- To speed the convergence, we might consider a scaled step:

$$\mathbf{x}^{k+1} \leftarrow \underset{\mathbf{x}\in\mathcal{C}}{\operatorname{argmin}} ||\mathbf{x} - (\mathbf{x}^k - \alpha\mathbf{d}^k)||_2^2,$$

  where $\mathbf{d}^k$ solves $\mathbf{H}^k\mathbf{d}^k = \nabla f(\mathbf{x}^k)$.
- As we saw, in general this does not work.
- To guarantee descent, projected Newton methods project under a quadratic norm:

$$\mathbf{x}^{k+1} \leftarrow \underset{\mathbf{x}\in\mathcal{C}}{\operatorname{argmin}} ||\mathbf{x} - (\mathbf{x}^k - \alpha\mathbf{d}^k)||_{\mathbf{H}^k}^2,$$

  where $||\mathbf{y}||_{\mathbf{H}^k} = \sqrt{\mathbf{y}^T\mathbf{H}^k\mathbf{y}}$ [Levitin & Polyak, 1966].

Projected Newton

- Projecting under the $\mathbf{H}^k$ norm is equivalent to minimizing the quadratic approximation over the convex set:

$$\mathbf{x}^{k+1} \leftarrow \underset{\mathbf{x}\in\mathcal{C}}{\operatorname{argmin}}\, \mathcal{Q}^k(\mathbf{x}, \alpha)$$

where

$$\mathcal{Q}^k(\mathbf{x}, \alpha) = f(\mathbf{x}^k) + (\mathbf{x}-\mathbf{x}^k)^T \nabla f(\mathbf{x}^k) + \frac{1}{2\alpha}(\mathbf{x}-\mathbf{x}^k)^T \mathbf{H}^k (\mathbf{x}-\mathbf{x}^k)$$

- In general, this projection will be expensive even if the constraints are simple.
- May be inexpensive if $\mathbf{H}^k$ is diagonal.

Projected Newton

- Projecting under the $\mathbf{H}^k$ norm is equivalent to minimizing the quadratic approximation over the convex set:

$$\mathbf{x}^{k+1} \leftarrow \underset{\mathbf{x} \in \mathcal{C}}{\operatorname{argmin}}\, \mathcal{Q}^k(\mathbf{x}, \alpha)$$

where

$$\mathcal{Q}^k(\mathbf{x}, \alpha) = f(\mathbf{x}^k) + (\mathbf{x} - \mathbf{x}^k)^T \nabla f(\mathbf{x}^k) + \frac{1}{2\alpha}(\mathbf{x} - \mathbf{x}^k)^T \mathbf{H}^k (\mathbf{x} - \mathbf{x}^k)$$

- In general, this projection will be expensive even if the constraints are simple.
- May be inexpensive if $\mathbf{H}^k$ is diagonal.

Inexact Projected Newton

- If we want to use a non-diagonal $\mathbf{H}^k$, we can consider an inexact projected Newton method.
- Analogous to the unconstrained HFN methods, we compute the step using a constrained iterative solver.
- For example, we can minimize $\mathcal{Q}^k(\mathbf{y}, \alpha)$ use SPG iterations:

$$\mathbf{y}^{k+1} \leftarrow \underset{\mathbf{y} \in \mathcal{C}}{\operatorname{argmin}} ||\mathbf{y} - (\mathbf{y}^k - \alpha_{bb} \nabla_{\mathbf{y}} \mathcal{Q}^k(\mathbf{y}, \alpha))||_2^2.$$

- These iterations are dominated by the cost of:
  - Euclidean projections and Hessian-vector products.

Inexact Projected Newton

- If we want to use a non-diagonal $\mathbf{H}^k$, we can consider an inexact projected Newton method.
- Analogous to the unconstrained HFN methods, we compute the step using a constrained iterative solver.
- For example, we can minimize $\mathcal{Q}^k(\mathbf{y}, \alpha)$ use SPG iterations:

$$\mathbf{y}^{k+1} \leftarrow \underset{\mathbf{y} \in \mathcal{C}}{\operatorname{argmin}} ||\mathbf{y} - (\mathbf{y}^k - \alpha_{bb} \nabla_{\mathbf{y}} \mathcal{Q}^k(\mathbf{y}, \alpha))||_2^2.$$

- These iterations are dominated by the cost of:
  - Euclidean projections and Hessian-vector products.

Inexact Projected Newton

- If we want to use a non-diagonal $\mathbf{H}^k$, we can consider an inexact projected Newton method.
- Analogous to the unconstrained HFN methods, we compute the step using a constrained iterative solver.
- For example, we can minimize $\mathcal{Q}^k(\mathbf{y}, \alpha)$ use SPG iterations:

$$\mathbf{y}^{k+1} \leftarrow \underset{\mathbf{y} \in \mathcal{C}}{\operatorname{argmin}} ||\mathbf{y} - (\mathbf{y}^k - \alpha_{bb} \nabla_{\mathbf{y}} \mathcal{Q}^k(\mathbf{y}, \alpha))||_2^2.$$

- These iterations are dominated by the cost of:
  - Euclidean projections and Hessian-vector products.

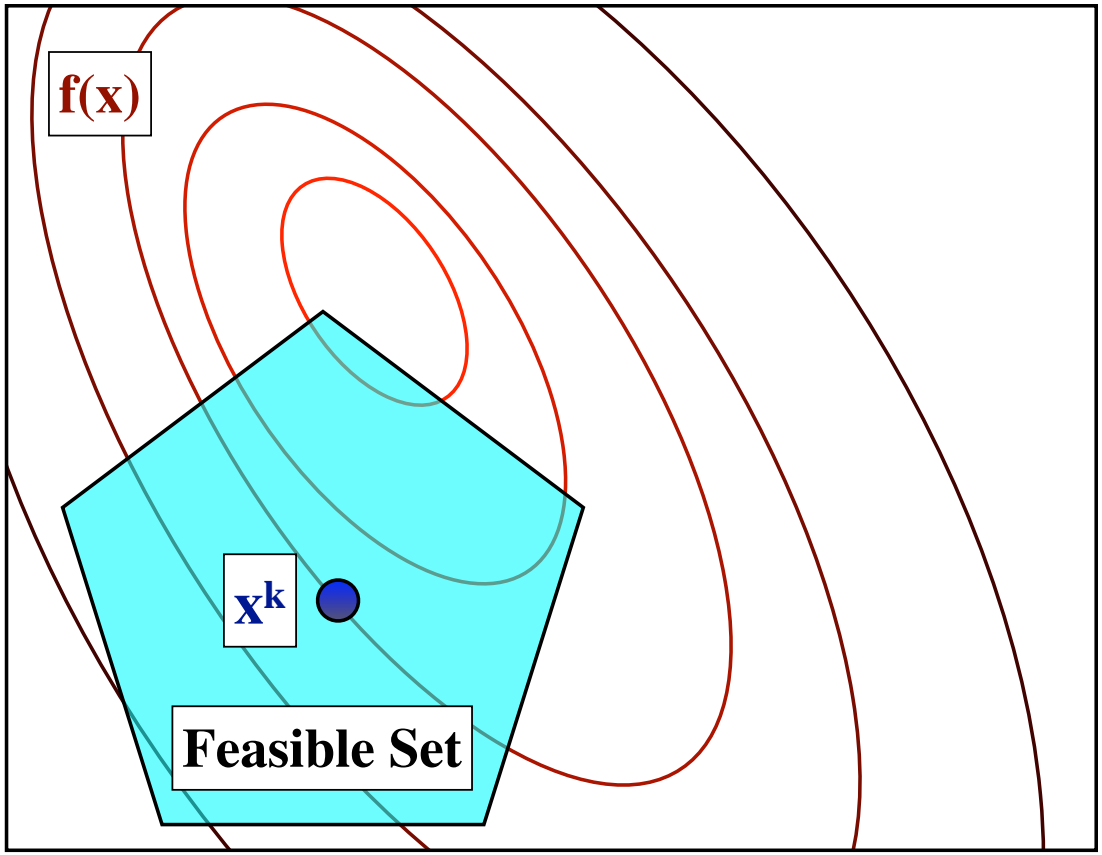
Inexact Projected Newton

- If we want to use a non-diagonal $\mathbf{H}^k$, we can consider an inexact projected Newton method.
- Analogous to the unconstrained HFN methods, we compute the step using a constrained iterative solver.
- For example, we can minimize $\mathcal{Q}^k(\mathbf{y}, \alpha)$ use SPG iterations:

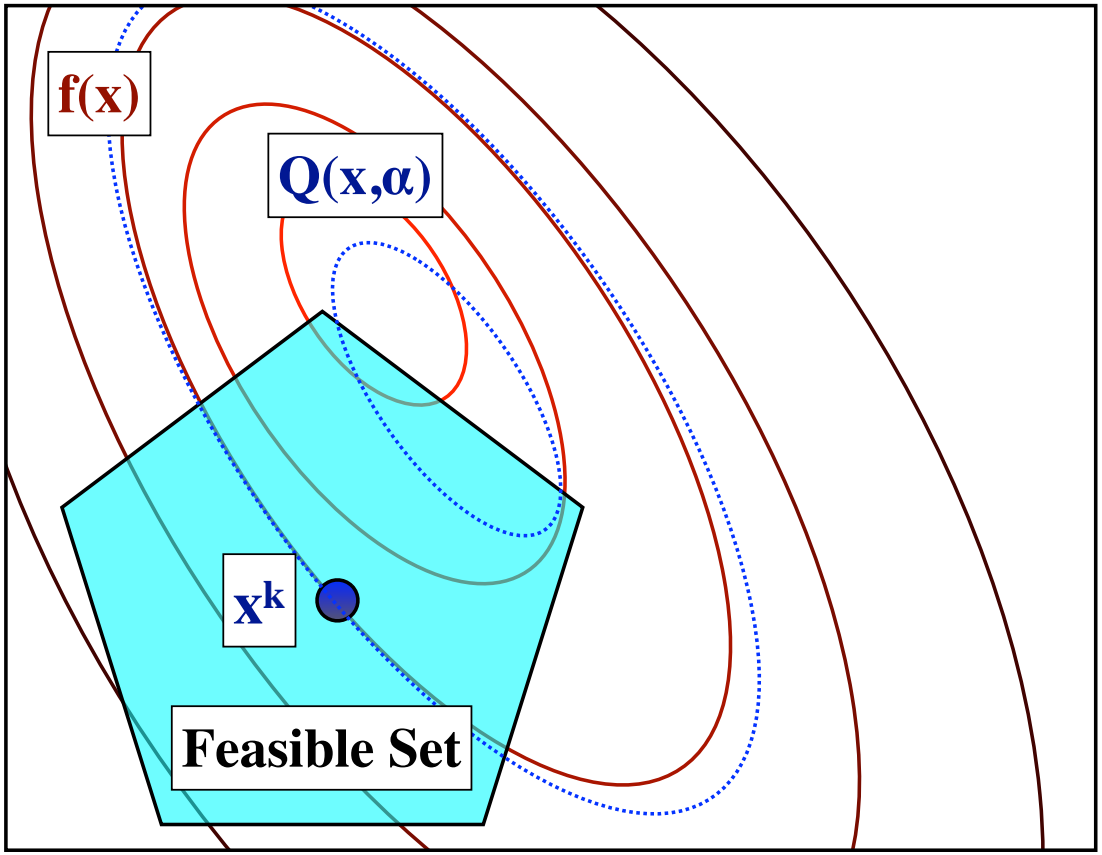
$$\mathbf{y}^{k+1} \leftarrow \underset{\mathbf{y} \in \mathcal{C}}{\operatorname{argmin}} ||\mathbf{y} - (\mathbf{y}^k - \alpha_{bb} \nabla_{\mathbf{y}} \mathcal{Q}^k(\mathbf{y}, \alpha))||_2^2.$$

- These iterations are dominated by the cost of:
  - Euclidean projections and Hessian-vector products.

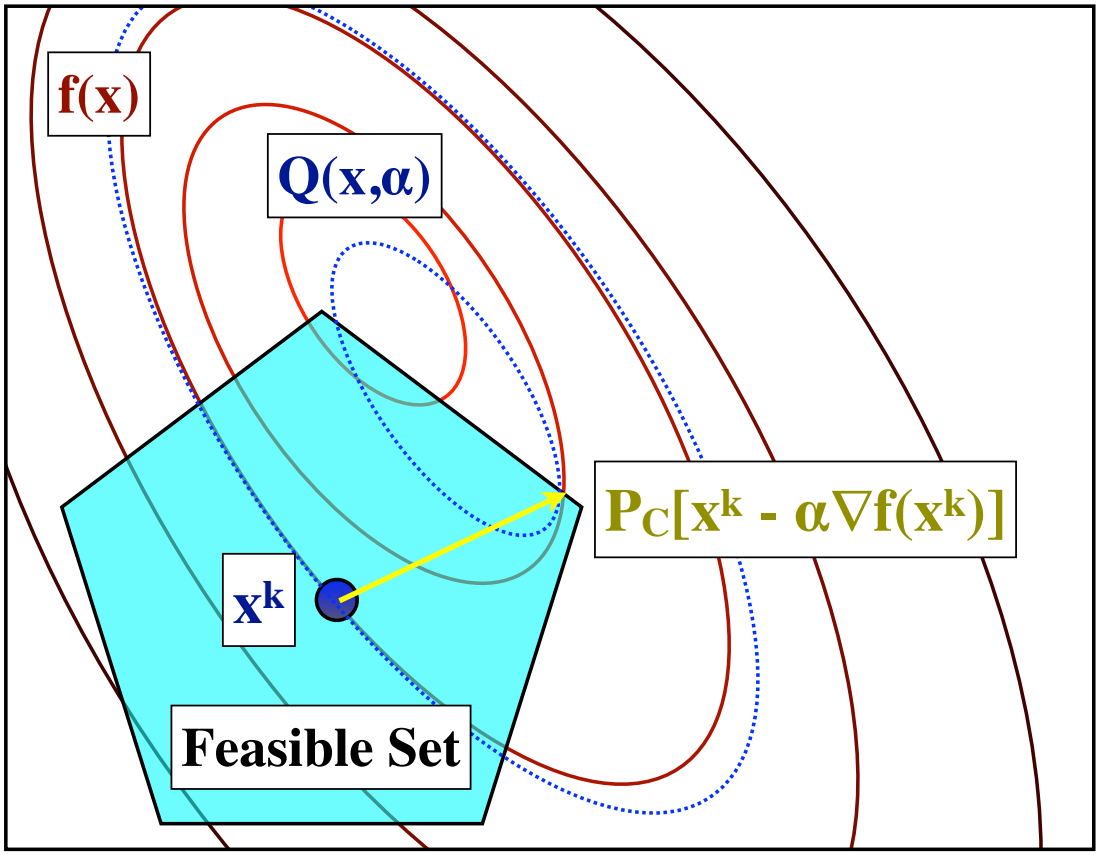
Inexact Projected Newton



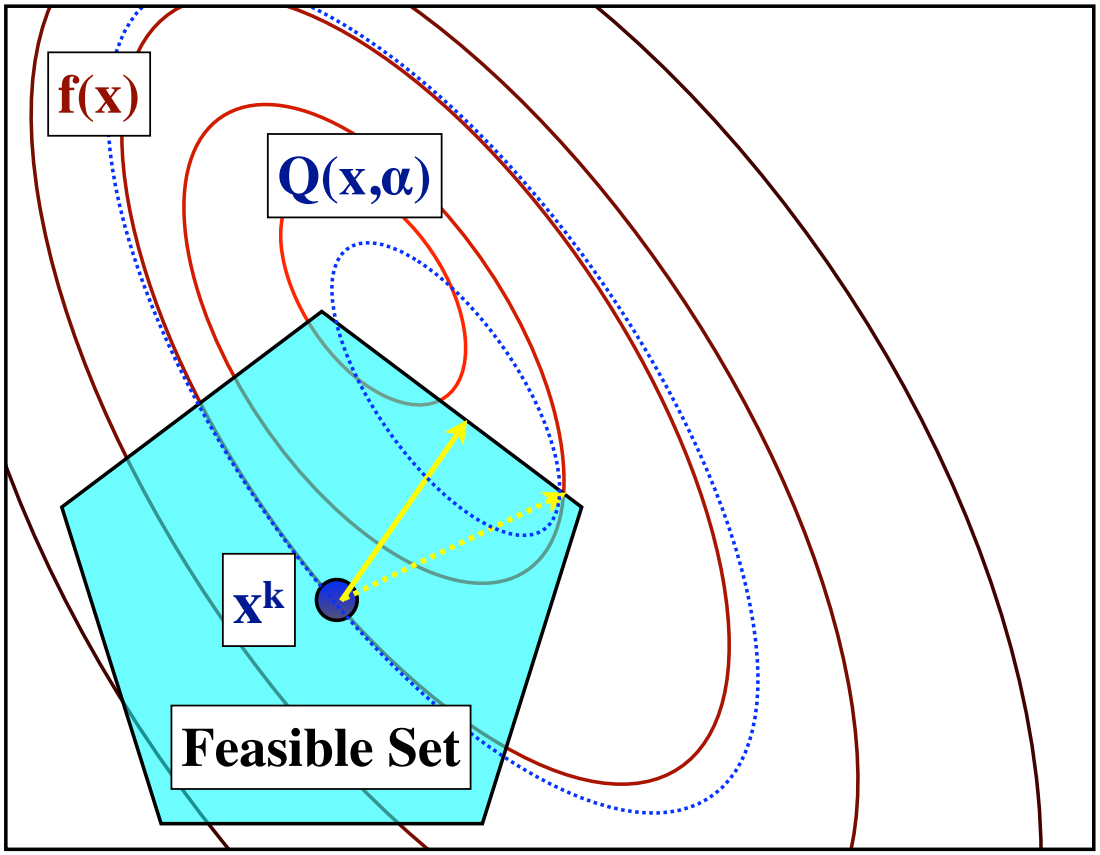
Inexact Projected Newton



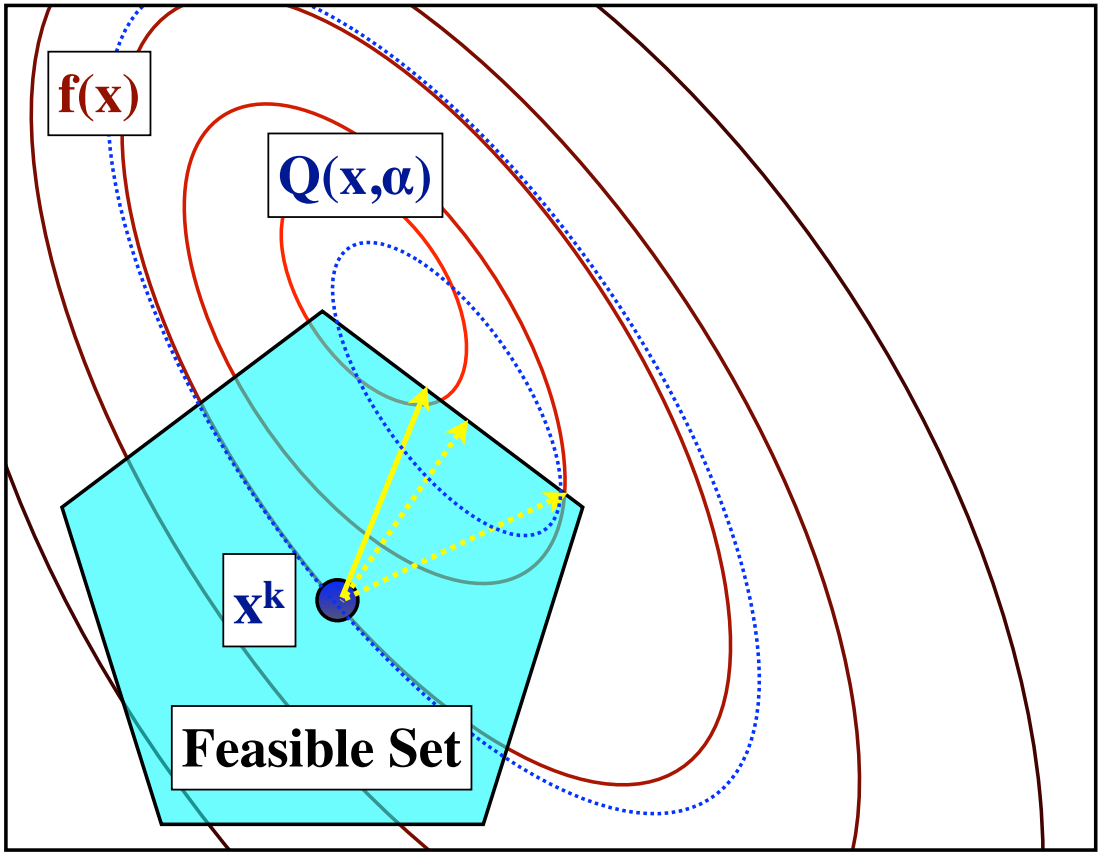
Inexact Projected Newton



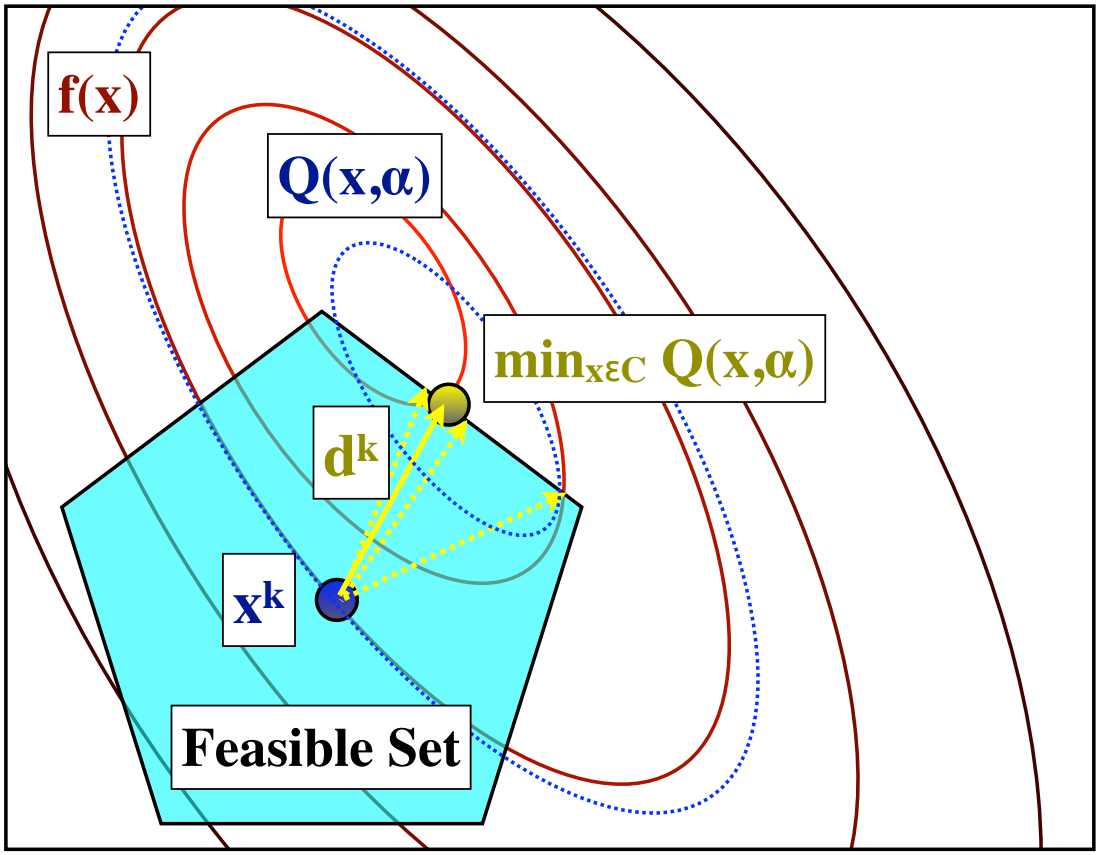
Inexact Projected Newton



Inexact Projected Newton



Inexact Projected Newton



Inexact Projected Newton

Can we terminate this early?

- If we set $\mathbf{y}^0 = \mathbf{x}^k$, then $f(\mathbf{y}^k) < f(\mathbf{x}^k)$ for $k \geq 1$ (for α small).
- Alternately, we can set $\alpha = 1$ and show that $\mathbf{d}^k = \mathbf{y}^k - \mathbf{x}^k$ is a feasible descent direction for $k \geq 1$.

In Schmidt et al. [2009], we used an L-BFGS approximation in an inexact projected Newton method:

- The (approximate-)Hessian-vector products take $\mathcal{O}(mn)$.
- The SPG iterations use projections but not the objective.
- Efficient for optimizing costly functions with simple constraints.

Inexact Projected Newton

Can we terminate this early?

- If we set $\mathbf{y}^0 = \mathbf{x}^k$, then $f(\mathbf{y}^k) < f(\mathbf{x}^k)$ for $k \geq 1$ (for α small).
- Alternately, we can set $\alpha = 1$ and show that $\mathbf{d}^k = \mathbf{y}^k - \mathbf{x}^k$ is a feasible descent direction for $k \geq 1$.

In Schmidt et al. [2009], we used an L-BFGS approximation in an inexact projected Newton method:

- The (approximate-)Hessian-vector products take $\mathcal{O}(mn)$.
- The SPG iterations use projections but not the objective.
- Efficient for optimizing costly functions with simple constraints.

Inexact Projected Newton

Can we terminate this early?

- If we set $\mathbf{y}^0 = \mathbf{x}^k$, then $f(\mathbf{y}^k) < f(\mathbf{x}^k)$ for $k \geq 1$ (for α small).
- Alternately, we can set $\alpha = 1$ and show that $\mathbf{d}^k = \mathbf{y}^k - \mathbf{x}^k$ is a feasible descent direction for $k \geq 1$.

In Schmidt et al. [2009], we used an L-BFGS approximation in an inexact projected Newton method:

- The (approximate-)Hessian-vector products take $\mathcal{O}(mn)$.
- The SPG iterations use projections but not the objective.
- Efficient for optimizing costly functions with simple constraints.

Inexact Projected Newton

Can we terminate this early?

- If we set $\mathbf{y}^0 = \mathbf{x}^k$, then $f(\mathbf{y}^k) < f(\mathbf{x}^k)$ for $k \geq 1$ (for α small).
- Alternately, we can set $\alpha = 1$ and show that $\mathbf{d}^k = \mathbf{y}^k - \mathbf{x}^k$ is a feasible descent direction for $k \geq 1$.

In Schmidt et al. [2009], we used an L-BFGS approximation in an inexact projected Newton method:

- The (approximate-)Hessian-vector products take $\mathcal{O}(mn)$.
- The SPG iterations use projections but not the objective.
- Efficient for optimizing costly functions with simple constraints.

Proximal Operators

- Can we avoid introducing constraints and directly solve the original non-smooth group ℓ_1-regularization problem?
- We can generalize projections to proximal operators:

$$\text{prox}_r(\mathbf{x}^k) = \underset{\mathbf{x}}{\text{argmin}} \frac{1}{2}||\mathbf{x} - \mathbf{x}^k||_2^2 + r(\mathbf{x}).$$

- The group ℓ_1-regularizer is simple; we can efficiently compute the proximal operator in linear time [Wright et al., 2009].

$$\begin{aligned}\text{prox}_{\ell_{1,2}}(\mathbf{x}^k)_g &= \underset{\mathbf{x}}{\text{argmin}} \frac{1}{2}||\mathbf{x}_g - \mathbf{x}_g^k||_2^2 + \lambda_g||\mathbf{x}_g||_2 \\ &= \text{sgn}(\mathbf{x}_g)\max\{0, ||\mathbf{x}_g||_2 - \lambda_g\}\end{aligned}$$

Proximal Operators

- Can we avoid introducing constraints and directly solve the original non-smooth group ℓ_1-regularization problem?
- We can generalize projections to proximal operators:

$$\mathrm{prox}_r(\mathbf{x}^k) = \underset{\mathbf{x}}{\mathrm{argmin}}\, \frac{1}{2}||\mathbf{x} - \mathbf{x}^k||_2^2 + r(\mathbf{x}).$$

- The group ℓ_1-regularizer is simple; we can efficiently compute the proximal operator in linear time [Wright et al., 2009].

$$\begin{aligned}\mathrm{prox}_{\ell_{1,2}}(\mathbf{x}^k)_g &= \underset{\mathbf{x}}{\mathrm{argmin}}\, \frac{1}{2}||\mathbf{x}_g - \mathbf{x}_g^k||_2^2 + \lambda_g||\mathbf{x}_g||_2 \\ &= \mathrm{sgn}(\mathbf{x}_g)\max\{0, ||\mathbf{x}_g||_2 - \lambda_g\}\end{aligned}$$

Proximal Operators

- Can we avoid introducing constraints and directly solve the original non-smooth group ℓ_1-regularization problem?
- We can generalize projections to proximal operators:

$$\mathrm{prox}_r(\mathbf{x}^k) = \underset{\mathbf{x}}{\mathrm{argmin}}\, \frac{1}{2}||\mathbf{x} - \mathbf{x}^k||_2^2 + r(\mathbf{x}).$$

- The group ℓ_1-regularizer is simple; we can efficiently compute the proximal operator in linear time [Wright et al., 2009].

$$\begin{aligned}\mathrm{prox}_{\ell_{1,2}}(\mathbf{x}^k)_g &= \underset{\mathbf{x}}{\mathrm{argmin}}\, \frac{1}{2}||\mathbf{x}_g - \mathbf{x}_g^k||_2^2 + \lambda_g||\mathbf{x}_g||_2 \\ &= \mathrm{sgn}(\mathbf{x}_g)\max\{0, ||\mathbf{x}_g||_2 - \lambda_g\}\end{aligned}$$

Proximal Gradient and Proximal Newton

- The basic proximal gradient step:

$$\mathbf{x}^{k+1} \leftarrow \underset{\mathbf{x}}{\operatorname{argmin}} \frac{1}{2}||\mathbf{x} - (\mathbf{x}^k - \alpha \nabla f(\mathbf{x}^k))||_2^2 + r(\mathbf{x})$$

- To speed the convergence, we might consider a scaled step:

$$\mathbf{x}^{k+1} \leftarrow \underset{\mathbf{x}}{\operatorname{argmin}} \frac{1}{2}||\mathbf{x} - (\mathbf{x}^k - \alpha \mathbf{d}^k)||_2^2 + r(\mathbf{x}),$$

where $\mathbf{d}^k$ solves $\mathbf{H}^k \mathbf{d}^k = \nabla f(\mathbf{x}^k)$.

- But to ensure descent, we need to match the norms:

$$\mathbf{x}^{k+1} \leftarrow \underset{\mathbf{x}}{\operatorname{argmin}} \frac{1}{2}||\mathbf{x} - (\mathbf{x}^k - \alpha \mathbf{d}^k)||_{\mathbf{H}^k}^2 + r(\mathbf{x})$$

Proximal Gradient and Proximal Newton

- The basic proximal gradient step:

$$\mathbf{x}^{k+1} \leftarrow \underset{\mathbf{x}}{\operatorname{argmin}} \frac{1}{2}||\mathbf{x} - (\mathbf{x}^k - \alpha \nabla f(\mathbf{x}^k))||_2^2 + r(\mathbf{x})$$

- To speed the convergence, we might consider a scaled step:

$$\mathbf{x}^{k+1} \leftarrow \underset{\mathbf{x}}{\operatorname{argmin}} \frac{1}{2}||\mathbf{x} - (\mathbf{x}^k - \alpha \mathbf{d}^k)||_2^2 + r(\mathbf{x}),$$

  where $\mathbf{d}^k$ solves $\mathbf{H}^k \mathbf{d}^k = \nabla f(\mathbf{x}^k)$.

- But to ensure descent, we need to match the norms:

$$\mathbf{x}^{k+1} \leftarrow \underset{\mathbf{x}}{\operatorname{argmin}} \frac{1}{2}||\mathbf{x} - (\mathbf{x}^k - \alpha \mathbf{d}^k)||_{\mathbf{H}^k}^2 + r(\mathbf{x})$$

Proximal Gradient and Proximal Newton

- The basic proximal gradient step:

$$\mathbf{x}^{k+1} \leftarrow \underset{\mathbf{x}}{\operatorname{argmin}} \frac{1}{2}||\mathbf{x} - (\mathbf{x}^k - \alpha \nabla f(\mathbf{x}^k))||_2^2 + r(\mathbf{x})$$

- To speed the convergence, we might consider a scaled step:

$$\mathbf{x}^{k+1} \leftarrow \underset{\mathbf{x}}{\operatorname{argmin}} \frac{1}{2}||\mathbf{x} - (\mathbf{x}^k - \alpha \mathbf{d}^k)||_2^2 + r(\mathbf{x}),$$

  where $\mathbf{d}^k$ solves $\mathbf{H}^k \mathbf{d}^k = \nabla f(\mathbf{x}^k)$.

- But to ensure descent, we need to match the norms:

$$\mathbf{x}^{k+1} \leftarrow \underset{\mathbf{x}}{\operatorname{argmin}} \frac{1}{2}||\mathbf{x} - (\mathbf{x}^k - \alpha \mathbf{d}^k)||_{\mathbf{H}^k}^2 + r(\mathbf{x})$$

Inexact Proximal Newton

- The proximal step under the $\mathbf{H}^k$ norm is equivalent to minimizing $\alpha r(\mathbf{x})$ and the quadratic approximation:

$$\mathbf{x}^{k+1} \leftarrow \arg\min_{\mathbf{x}} \mathcal{Q}^k(\mathbf{x}, \alpha) + \alpha r(\mathbf{x})$$

- This problem will be expensive even if $r(\mathbf{x})$ is simple.
- We could use a diagonal scaling or Barzilai-Borwein steps [Hofling & Tibshirani, 2009, Wright et al., 2009].
- To use a non-diagonal scaling, we can use an iterative solver:
  - Use Euclidean proximal operators and Hessian-vector products.
  - Guarantees descent after first iteration.
  - With an L-BFGS approximation, suitable for optimizing costly objectives with simple regularizers.

Inexact Proximal Newton

- The proximal step under the $\mathbf{H}^k$ norm is equivalent to minimizing $\alpha r(\mathbf{x})$ and the quadratic approximation:

$$\mathbf{x}^{k+1} \leftarrow \arg\min_{\mathbf{x}} \mathcal{Q}^k(\mathbf{x}, \alpha) + \alpha r(\mathbf{x})$$

- This problem will be expensive even if $r(\mathbf{x})$ is simple.
- We could use a diagonal scaling or Barzilai-Borwein steps [Hofling & Tibshirani, 2009, Wright et al., 2009].
- To use a non-diagonal scaling, we can use an iterative solver:
  - Use Euclidean proximal operators and Hessian-vector products.
  - Guarantees descent after first iteration.
  - With an L-BFGS approximation, suitable for optimizing costly objectives with simple regularizers.

Inexact Proximal Newton

- The proximal step under the $\mathbf{H}^k$ norm is equivalent to minimizing $\alpha r(\mathbf{x})$ and the quadratic approximation:

$$\mathbf{x}^{k+1} \leftarrow \arg\min_{\mathbf{x}} \mathcal{Q}^k(\mathbf{x}, \alpha) + \alpha r(\mathbf{x})$$

- This problem will be expensive even if $r(\mathbf{x})$ is simple.
- We could use a diagonal scaling or Barzilai-Borwein steps [Hofling & Tibshirani, 2009, Wright et al., 2009].
- To use a non-diagonal scaling, we can use an iterative solver:
  - Use Euclidean proximal operators and Hessian-vector products.
  - Guarantees descent after first iteration.
  - With an L-BFGS approximation, suitable for optimizing costly objectives with simple regularizers.

Inexact Proximal Newton

- The proximal step under the $\mathbf{H}^k$ norm is equivalent to minimizing $\alpha r(\mathbf{x})$ and the quadratic approximation:

$$\mathbf{x}^{k+1} \leftarrow \arg\min_{\mathbf{x}} \mathcal{Q}^k(\mathbf{x}, \alpha) + \alpha r(\mathbf{x})$$

- This problem will be expensive even if $r(\mathbf{x})$ is simple.
- We could use a diagonal scaling or Barzilai-Borwein steps [Hofling & Tibshirani, 2009, Wright et al., 2009].
- To use a non-diagonal scaling, we can use an iterative solver:
  - Use Euclidean proximal operators and Hessian-vector products.
  - Guarantees descent after first iteration.
  - With an L-BFGS approximation, suitable for optimizing costly objectives with simple regularizers.

Discussion

Easily extends to other group norms:

- ℓ_∞ norm: the projection/proximal operators can be computed in $\mathcal{O}(n \log n)/\mathcal{O}(n)$ [Duchi et al., 2008, Quattoni et al., 2009, Wright et al, 2009].
- Nuclear norm: the projection/proximal operators can be computed in $\mathcal{O}(n^{3/2})$ [Cai et al., 2010].

Convergence theory of inexact projected/proximal Newton is not fully developed:

- Proof of global convergence.
- Accuracy needed for convergence rates.

Outline

Sums of Simple Regularizers

- Recall the overlapping group ℓ_1-regularization problem:

$$\min_{\mathbf{x}} f(\mathbf{x}) + \sum_{A \subseteq \{1,\ldots,p\}} \lambda_A \Big(\sum_{\{B | A \subseteq B\}} ||\mathbf{x}_B||_2^2 \Big)^{1/2}$$

- This regularizer is not simple.
- But, it is the sum of simple regularizers.
- After converting to a constrained problem, we can use Dykstra's [1983] algorithm to compute the projection.
- Bauschke & Combettes [2008] generalize Dykstra's algorithm to compute proximal operators for sums of simple functions.

Sums of Simple Regularizers

- Recall the overlapping group ℓ_1-regularization problem:

$$\min_{\mathbf{x}} f(\mathbf{x}) + \sum_{A \subseteq \{1,\dots,p\}} \lambda_A \Big(\sum_{\{B | A \subseteq B\}} ||\mathbf{x}_B||_2^2 \Big)^{1/2}$$

- This regularizer is not simple.
- But, it is the sum of simple regularizers.
- After converting to a constrained problem, we can use Dykstra's [1983] algorithm to compute the projection.
- Bauschke & Combettes [2008] generalize Dykstra's algorithm to compute proximal operators for sums of simple functions.

Sums of Simple Regularizers

- Recall the overlapping group ℓ_1-regularization problem:

$$\min_{\mathbf{x}} f(\mathbf{x}) + \sum_{A \subseteq \{1,\dots,p\}} \lambda_A \Big(\sum_{\{B | A \subseteq B\}} ||\mathbf{x}_B||_2^2 \Big)^{1/2}$$

- This regularizer is not simple.
- But, it is the sum of simple regularizers.
- After converting to a constrained problem, we can use Dykstra's [1983] algorithm to compute the projection.
- Bauschke & Combettes [2008] generalize Dykstra's algorithm to compute proximal operators for sums of simple functions.

Sums of Simple Regularizers

- Recall the overlapping group ℓ_1-regularization problem:

$$\min_{\mathbf{x}} f(\mathbf{x}) + \sum_{A \subseteq \{1,\ldots,p\}} \lambda_A \Big(\sum_{\{B | A \subseteq B\}} ||\mathbf{x}_B||_2^2 \Big)^{1/2}$$

- This regularizer is not simple.
- But, it is the sum of simple regularizers.
- After converting to a constrained problem, we can use Dykstra's [1983] algorithm to compute the projection.
- Bauschke & Combettes [2008] generalize Dykstra's algorithm to compute proximal operators for sums of simple functions.

Sums of Simple Regularizers

- Recall the overlapping group ℓ_1-regularization problem:

$$\min_{\mathbf{x}} f(\mathbf{x}) + \sum_{A \subseteq \{1,\ldots,p\}} \lambda_A \Big(\sum_{\{B | A \subseteq B\}} ||\mathbf{x}_B||_2^2 \Big)^{1/2}$$

- This regularizer is not simple.
- But, it is the sum of simple regularizers.
- After converting to a constrained problem, we can use Dykstra's [1983] algorithm to compute the projection.
- Bauschke & Combettes [2008] generalize Dykstra's algorithm to compute proximal operators for sums of simple functions.

Other Non-Smooth Newton-like Methods

- There is also recent work on other methods for computing the proximal operator for overlapping group ℓ_1-regularization [Jenatton et al., 2010, Kim & Xing, 2010, Liu & Ye, 2010, Mairal et al., 2010].
- Several other Newton-like methods for general non-smooth optimization are available:
  - Incorporating the sub-differential into the quadratic approximation [Yu et al., 2008].
  - Applying the basic method to a smoothed version of the problem [Chen et al., 2010].
  - Augmented Lagrangian and dual augmented Lagrangian methods [Tomioka et al., 2011].

Other Non-Smooth Newton-like Methods

- There is also recent work on other methods for computing the proximal operator for overlapping group ℓ_1-regularization [Jenatton et al., 2010, Kim & Xing, 2010, Liu & Ye, 2010, Mairal et al., 2010].
- Several other Newton-like methods for general non-smooth optimization are available:
  - Incorporating the sub-differential into the quadratic approximation [Yu et al., 2008].
  - Applying the basic method to a smoothed version of the problem [Chen et al., 2010].
  - Augmented Lagrangian and dual augmented Lagrangian methods [Tomioka et al., 2011].

Inexact Objective Information

In some scenarios we may have a stochastic objective:

- We may be using a Monte Carlo approximation.
- We may be using a mini-batch.

There is work on stochastic variants:

- Limited-memory quasi-Newton [Sunehag et al., 2009].
- Hessian-free Newton [Martens, 2010].
- Optimal Barzilai-Borwein [Swersky, unpublished].

However, they don't share the fast convergence rates of deterministic variants.

Co-Authors

Thanks to my great co-authors:

- Ewout van den Berg
- Michael Friedlander
- Glenn Fung
- Dongmin Kim
- Kevin Murphy
- Rómer Rosales
- Suvrit Sra

Summary

- Hessian-Free Newton and limited-memory BFGS methods are two workhorses of unconstrained optimization.
- Two-metric (sub-)gradient projection methods let us apply these methods to problems with bound constraints or non-differentiable but separable regularizers.
- Inexact projected/proximal Newton methods are an appealing approach to optimizing costly objective functions with simple constraints or simple non-differentiable regularizers.

Summary

- Hessian-Free Newton and limited-memory BFGS methods are two workhorses of unconstrained optimization.
- Two-metric (sub-)gradient projection methods let us apply these methods to problems with bound constraints or non-differentiable but separable regularizers.
- Inexact projected/proximal Newton methods are an appealing approach to optimizing costly objective functions with simple constraints or simple non-differentiable regularizers.

Summary

- Hessian-Free Newton and limited-memory BFGS methods are two workhorses of unconstrained optimization.
- Two-metric (sub-)gradient projection methods let us apply these methods to problems with bound constraints or non-differentiable but separable regularizers.
- Inexact projected/proximal Newton methods are an appealing approach to optimizing costly objective functions with simple constraints or simple non-differentiable regularizers.