Lecture 13: Parallel Optimization Algorithms

Prof. Krishna R. Pattipati

Dept. of Electrical and Computer Engineering
University of Connecticut
Contact: krishna@engr.uconn.edu (860) 486-2890

ECE 6437
Computational Methods for Optimization

Fall 2009
December 1 , 2009

Outline of Lecture 13

- Key Ideas of Parallel Algorithms
- Jacobi Algorithm
- Parallel implementation of Jacobi method and its variants
- Convergence Analysis
- Non-linear Jacobi and Gauss-Seidel Algorithms
- Constrained Optimization
- Decomposition Methods

Key Ideas of Parallel Algorithms

- Almost all of the parallel optimization algorithms are based on one of the two ideas :
  1. Jacobi and Gauss-Seidel type relaxation schemes
  2. Dual or decomposition methods…. mainly for convex constrained optimization methods.

Let us illustrate these ideas by means of a series of examples :

Consider $\min\limits_{\underline{x}\in R^n} \underline{x}^T Q \underline{x} - \underline{b}^T x$

$\Rightarrow$ Solve $Q\underline{x} = \underline{b}$

i^{th} equation is: $\sum\limits_{j=1}^{n} q_{ij}\, x_j = b_i \;;\quad i = 1, 2, \ldots, n$

If $q_{ii} \neq 0$ (which it will be if $Q > 0$)

$$q_{ii} x_i = b_i - \sum_{j \neq i} q_{ij}\, x_j$$

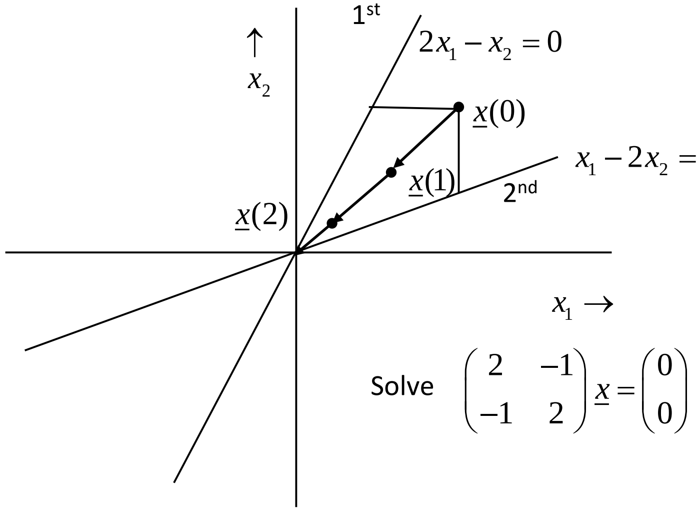
$$\boxed{x_i = \frac{-1}{q_{ii}} \left[\sum_{j \neq i} q_{ij}\, x_j - b_i \right]}$$

Jacobi Algorithm

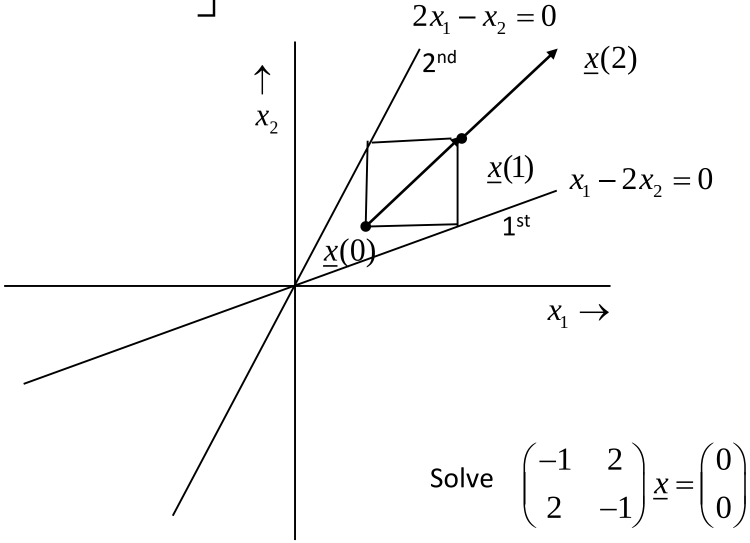
- Jacobi Algorithm:

Starting with an initial vector $\underline{x}(0) \in R^n$, evaluate $\underline{x}(k),\ k = 0, 1, ..$ using the iteration:

$$x_i(k+1) = \frac{-1}{q_{ii}}\left[\sum_{j \neq i} q_{ij}\ x_j(k) - b_i\right]$$



CONVERGENCE



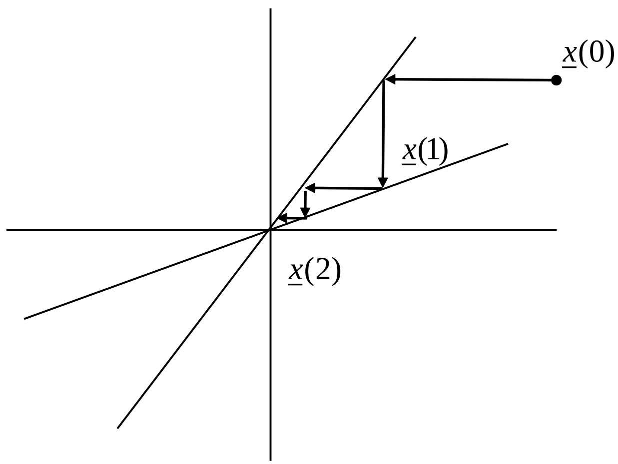
DIVERGENCE

It is well known that convergence is enhanced if i^{th} component is solved by an equation with $|q_{ii}| > |q_{ij}| \quad \forall\ j \neq i$

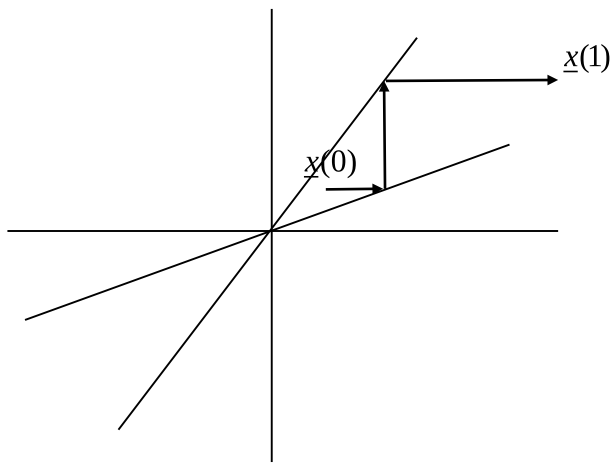
Gauss-Seidel Algorithm

- **Gauss-Seidel Algorithm**: On a serial computer, we can enhance convergence if we use new estimates of x_j , $j< i$ when updating x_i

$$x_i(k+1) = \frac{-1}{q_{ii}}\left[\sum_{j<i} q_{ij}x_j(k+1) + \sum_{j>i} q_{ij}x_j(k) - b_i\right]$$



CONVERGENCE



DIVERGENCE

Richardson-Gauss-Seidel Algorithm

- Richardson –Gauss-Seidel Algorithm:

  Solution of $Q\underline{x}=\underline{b} \Rightarrow$ solve $\underline{x} \leftarrow \underline{x}-\alpha[A\underline{x}-\underline{b}]$

  $$x_i(k+1)=x_i(k)-\alpha\left[\sum_{j<i} q_{ij}x_j(k+1)+\sum_{j\geq i} q_{ij}x_j(k)-b_i\right]$$

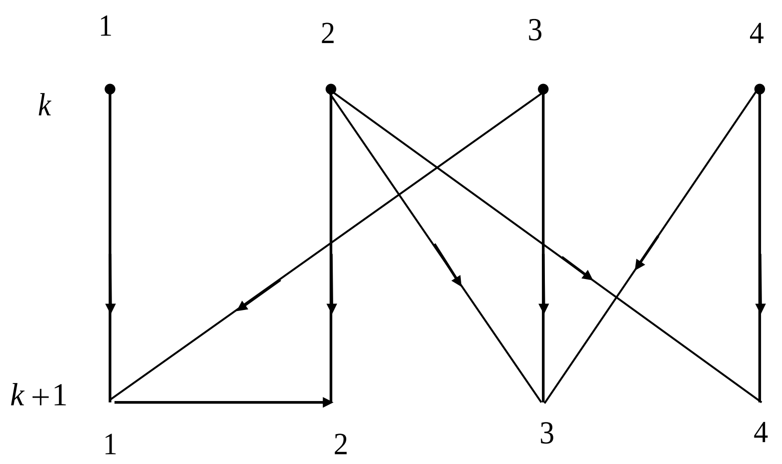
  Note the similarity to steepest descent with a constant step-size α

- Parallel implementation of methods:
  - Jacobi and Jacobi version of Richardson's algorithms are straight forward to implement in parallel. $\exists$ two ways :
    - If we have n processors, processor i knows row i of matrix Q. Each processor broadcasts x_i (k) to all other (n-1) processors. In a hypercube, this can be done in O(n/$\log_2 n$) time, since $\exists$ $\log_2 n$ distinct paths between any two processors.
    - If we have n processors, processor i knows column i of Q. Each processor computes $q_{ji}\, x_i \ \forall\ j=1,2,\ldots,n$. Then for each , the quantities $q_{ji}\, x_i$ are sent to processor j with partial sums accumulated along the way.

Parallel Implementation Methods

If number of available processors $p<n$, assign $\left\lceil \frac{n}{p} \right\rceil$ components to each processor.

- Gauss-Siedel is, unfortunately, not well-suited for parallel implementation. When matrix A is sparse, it is sometimes possible to exploit the non-dependencies.



Updating Order

$$x_1(k+1) = f_1(x_1(k), x_3(k))$$
$$x_3(k+1) = f_3(x_2(k), x_3(k), x_4(k))$$
$$x_4(k+1) = f_4(x_2(k), x_4(k))$$
$$x_2(k+1) = f_2(x_1(k+1), x_2(k))$$

If we have two processors, then it can be done in two steps.

Parallel Unconstrained Minimization

- Parallel Algorithms for unconstrained minimization

Jacobi:

$$x_i\,(k+1) = x_i(k) - \alpha\, h_i(\underline{x}_k)\,\frac{\partial f(\underline{x}_k)}{\partial x_i}$$

$$h_i\,(\underline{x}(k)) = \left[\frac{\partial^2 f\,(\underline{x}_k)}{\partial x_i^{\,2}}\right]^{-1}$$

$h_i = 1 \;\Rightarrow$ Gradient or Richardson's Algorithm

Gauss-Seidel:

$$x_i\,(k+1) = x_i(k) - \frac{\alpha}{\dfrac{\partial^2 f[\underline{z}_i(k)]}{\partial x_i^{\,2}}}\,\frac{\partial f}{\partial x_i}[\underline{z}_i(k)]$$

where $\underline{z}_i(k) = \left[x_1(k+1), \ldots, x_{i-1}(k+1),\; x_i(k), \ldots, x_n(k)\right]$

These are Coordinate descent Algorithms.

Convergence Analysis

- Convergence Analysis:

Assumptions: $f(\underline{x})$ is bounded below $\Rightarrow$ existence of a minimum

$\nabla \underline{f}(\underline{x})$ is Lipschitz continuous $\Rightarrow f(\underline{x})$ is continously differentiable with bounded derivatives

$$\left\|\nabla \underline{f}(\underline{x})-\nabla \underline{f}(\underline{y})\right\| \leq K \left\|\underline{x}-\underline{y}\right\|_2, \quad \forall\ \underline{x},\underline{y} \in R^n$$

Key to proving convergence is the following **DESCENT LEMMA**

Under the above assumption

$$f(\underline{x}+\underline{y}) \leq f(\underline{x}) + \underline{y}^T \nabla \underline{f}(\underline{x}) + \frac{K}{2}\underline{y}^T\underline{y} \quad \forall\ \underline{x},\underline{y} \in R^n$$

Proof: Let $g(\alpha) = f(\underline{x}+\alpha\underline{y})$, $\dfrac{dg(\alpha)}{d\alpha} = \nabla \underline{f}^T(\underline{x}+\alpha\underline{y})\,\underline{y}$

$$f(\underline{x}+\underline{y}) - f(\underline{x}) = g(1)-g(0) = \int_0^1 \frac{dg}{d\alpha}\,d\alpha$$

$$= \int_0^1 \underline{y}^T \nabla \underline{f}(\underline{x}+\alpha\underline{y})\,d\alpha \leq \int_0^1 \underline{y}^T \nabla \underline{f}(\underline{x})\,d\alpha + \left|\int_0^1 \underline{y}^T\left[\nabla \underline{f}(\underline{x}+\alpha\underline{y}) - \nabla \underline{f}(\underline{x})\right]d\alpha\right|$$

$$\leq \int_0^1 \underline{y}^T \nabla \underline{f}(\underline{x})\,d\alpha + \int_0^1 \left\|\underline{y}\right\|_2 K\,\alpha \left\|\underline{y}\right\|_2 d\alpha = \underline{y}^T \nabla \underline{f}(\underline{x}) + \frac{K}{2}\left\|\underline{y}\right\|_2^2$$

Convergence Theorem - 1

- Convergence Theorem:

Suppose $\underline{x}(k+1) = \underline{x}(k) + \alpha\, \underline{d}(k)$

where $\|\underline{d}(k)\|_2 \geq K_1 \|\nabla \underline{f}(\underline{x}(k))\|_2 \quad \forall k$

$$\underline{d}^T(k)\nabla \underline{f}(\underline{x}_k) \leq -K_2 \|\underline{d}(k)\|^2$$

Then, gradient and Jacobi algorithms converge if $0 < \alpha < \dfrac{2K_2}{K}$

Gradient: $K_1 = 1$ and $K_2 = 1$

Jacobi: $K_2 \leq \min\limits_i \dfrac{\partial^2 f}{\partial x_i^2}, \qquad \dfrac{1}{K_1} \geq \max\limits_i \dfrac{\partial^2 f}{\partial x_i^2}$

From Descent Lemma

$$f(\underline{x}(k+1)) \leq f(\underline{x}(k)) + \alpha\, \underline{d}^T(k)\nabla \underline{f}(\underline{x}(k)) + \frac{K}{2}\alpha^2 \underline{d}^T(k)\,\underline{d}(k)$$

$$\leq f(\underline{x}(k)) - \alpha\left(K_2 - \frac{K\alpha}{2}\right)\|\underline{d}(k)\|_2^2 \Rightarrow f(\underline{x}(k+1)) \leq f(\underline{x}(k))$$

Convergence Theorem - 2

Since $f(\underline{x})$ is bounded from below, we obtain optimum (local) if $0 < \alpha < \dfrac{2K_2}{K}$

Gradient: $0 < \alpha < \dfrac{2}{K}$

Jacobi: $0 < \alpha < \dfrac{2 \min_i \partial^2 f / \partial x_i^2}{K}$

For Gauss – Seidel, one obtains a stronger result. If $0 < \gamma_i \le \dfrac{\partial^2 f}{\partial x_i^2} \le \Gamma_i$ for all

$\underline{x} \in R^n$, then convergence occurs if $0 < \alpha < \dfrac{2\gamma_i}{\Gamma_i}$ for all i

In the quadratic case, $\frac{1}{2}\underline{x}^T Q \underline{x} - \underline{b}^T \underline{x}$

$$\frac{\partial^2 f}{\partial x_i^2} = q_{ii} = \gamma_i = \Gamma_i \Rightarrow \text{Convergence occurs if } 0 < \alpha < 2.$$

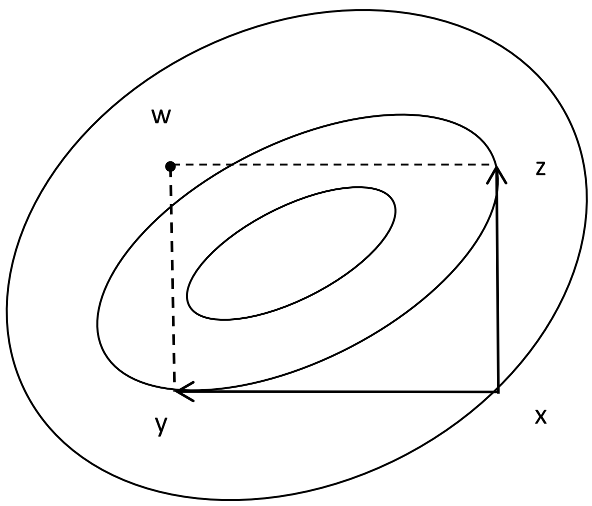
Jacobi requires α small enough for convergence.

Nonlinear Jacobi and Gauss-Seidel -1

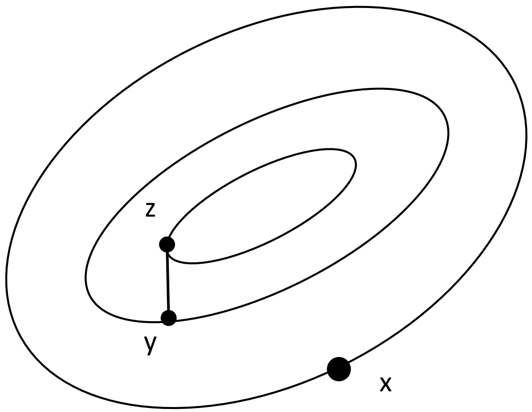
- Non-linear Jacobi and Gauss-Seidel Algorithm

Jacobi: $x_i(k+1) = \arg\min_{x_i} f(x_1(k), \ldots., x_{i-1}(k), x_i, x_{i+1}(k), \ldots, x_n(k))$

$$\text{(or) solve } \frac{\partial f(x_1(k), \ldots., x_{i-1}(k), x_i, x_{i+1}(k), \ldots, x_n(k))}{\partial x_i} = 0$$



JACOBI



Gauss-Seidel

Nonlinear Jacobi and Gauss-Seidel - 2

Gauss-Seidel:

$$x_i(k+1) = \arg\min_{x_i} f(x_1(k+1), \ldots., x_{i-1}(k+1), x_i, x_{i+1}(k), \ldots, x_n(k))$$

Gauss – Seidel algorithms are guaranteed to converge for convex functions. Note that the function value is strictly decreasing at each iteration. The Jacobi algorithm requires more stringent restrictions (e.g., $\underline{x}_k - \gamma \nabla \underline{f}(\underline{x}_k)$ must be a contraction map $\Rightarrow \left\| \underline{x}_k - \alpha \nabla \underline{f}(\underline{x}_k) \right\| < 1$ for $\alpha > 0) \Rightarrow \nabla^2 f(\underline{x})$ must be diagonally dominant.

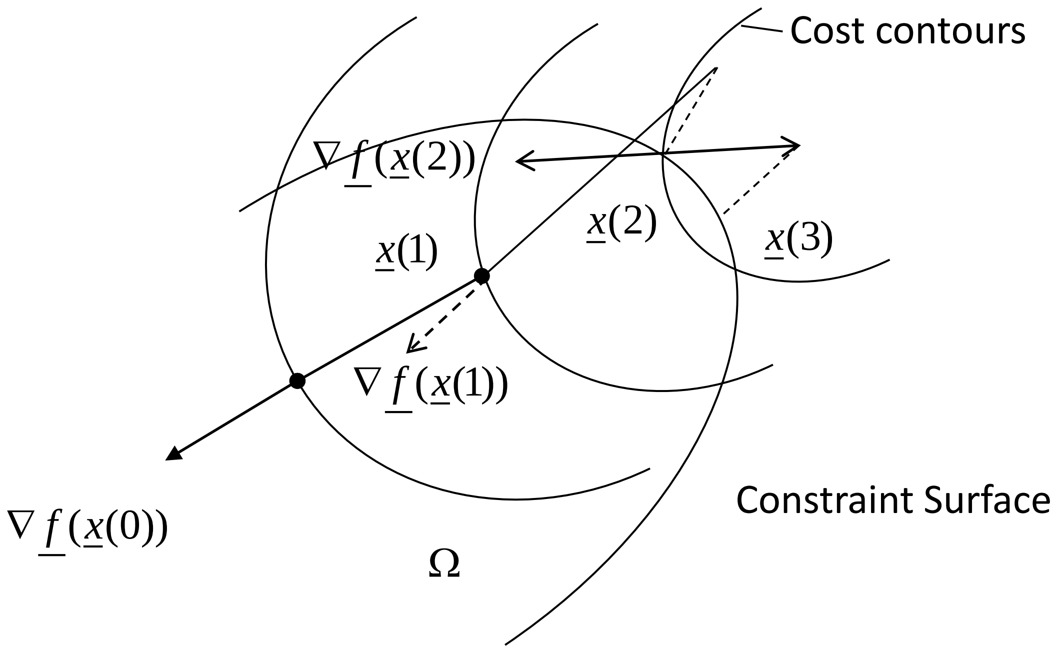
- Constrained Optimization:

Recall the gradient projection algorithm

$$\underline{x}(k+1) \;=\; \underbrace{\left[\underline{x}(k) - \alpha H_k \nabla \underline{f}(\underline{x}_k)\right]^+}_{\text{Projection on to the feasible set } \Omega}$$

Parallel Constrained Optimization - 1

- Constrained Optimization (continued):



Unfortunately the gradient projection algorithm is not amenable to parallel implementation unless the constraints are of the “box type”, i.e., upper and lower bound constraints on each variable

Parallel Constrained Optimization - 2

Recall that projection of $[\underline{x}-\alpha\nabla f(\underline{x})]$ into Ω is equivalent to

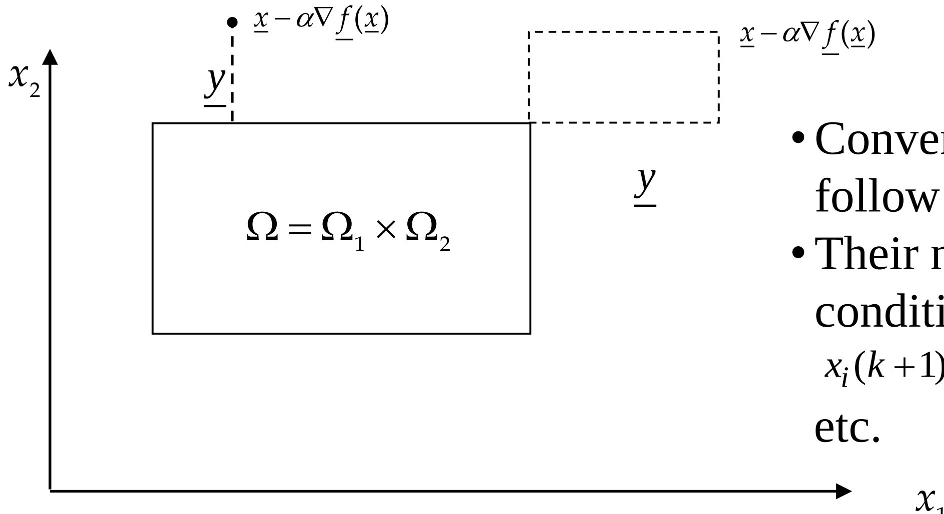
$$\min_{\underline{y}} \left\| \underline{y}-\underline{x}+\alpha\nabla \underline{f}(\underline{x}) \right\|_2^2 \qquad \text{s.t. } \underline{y}\in\Omega$$

$$\text{(or)}\quad \min_{\underline{y}} (\underline{y}-\underline{x})^T \nabla \underline{f}(\underline{x}) + \frac{1}{2\alpha}(\underline{y}-\underline{x})^T(\underline{y}-\underline{x}) \quad \text{s.t. } \underline{y}\in\Omega$$

If $\Omega=\prod_{i=1}^{n}\left[x_{i\min}\,,\ x_{i\max}\right]$, then the problem is equivalent to

$$\sum_{i=1}^{n} \min_{x_{i\min}\le x\le x_{i\max}} \frac{1}{2\alpha}(y_i-x_i)^2+(y_i-x_i)\frac{\partial f}{\partial x_i}$$

$\Rightarrow$ Minimizations can be carried out independently.



- Convergence of Gauss-Seidel and Jacobi follow for small enough α
- Their non-linear versions converge under conditions similar to unconstrained case

$$x_i(k+1)=\arg\min_{x_i\in\Omega_i} f(x_1(k),\ x_2(k),\dots,x_i,\dots,x_n(k)),$$

etc.

Decomposition Methods - 1

- Decomposition Methods: By far the best for constrained minimization. We will present ideas via a series of examples.
  - Example 1:

$$\min \frac{1}{2}\underline{x}^T\underline{x} \text{ “Project the Origin on the Constraint Set”}$$

s.t. $A\underline{x} \le \underline{c}$ $\quad A$ is an m by n matrix

$$q(\underline{\mu}) = \min_{\underline{x}} \frac{1}{2}\underline{x}^T x + \underline{\mu}^T(A\underline{x} - \underline{c})$$

Optimal $\underline{x}^*$ for a given $\underline{\mu}$: $\quad \underline{x} + A^T\underline{\mu} = 0 \Rightarrow \underline{x} = -A^T\underline{\mu}$

$$\therefore q(\underline{\mu}) = -\frac{1}{2}\underline{\mu}^T AA^T\underline{\mu} - \underline{\mu}^T\underline{c}$$

Dual: $\max\limits_{\underline{\mu} \ge \underline{0}} q(\underline{\mu})$ or $\min\limits_{\underline{\mu} \ge \underline{0}} \frac{1}{2}\underline{\mu}^T P\underline{\mu} + \underline{\mu}^T\underline{c}$, $\; P = AA^T$

Unlike the primal problem, the dual is parallelizable

$$\Rightarrow \text{Unconstrained min } \tilde{\mu}_j = \frac{-1}{p_{jj}}\left[c_j + \sum_{k \ne j} p_{jk}\,\mu_k\right] = \mu_j - \frac{1}{p_{jj}}\left[c_j + \sum_{k=1}^{m} p_{jk}\,\mu_k\right]$$

Decomposition Methods - 2

The iteration is: $\mu_j = \max(0, \tilde{\mu}_j) = \max\left(0, \mu_j - \frac{1}{p_{jj}}\left[c_j + \sum_{k=1}^{m} p_{jk}\mu_k\right]\right) \quad (*)$

The matrix A has a sparse structure in practice and we must exploit it

(even if A is sparse, $P = AA^T$ need not be!!!). Define $\underline{y} = -A^T\underline{\mu} \Rightarrow P\underline{\mu} = -A\underline{y}$

So, $\sum_{k=1}^{m} p_{jk}\underline{\mu}_k = \underline{p}_j^T\underline{\mu} = -\underline{a}_j^T\underline{y}$, $\quad \underline{a}_j^T = j^{th}$ row of A or j^{th} column of $A^T = \underline{a}_j$

Also, $p_{jj} = (AA^T)_{jj} = \underline{a}_j^T\underline{a}_j$; So, iteration $(*)$ can be rewritten as two iterations:

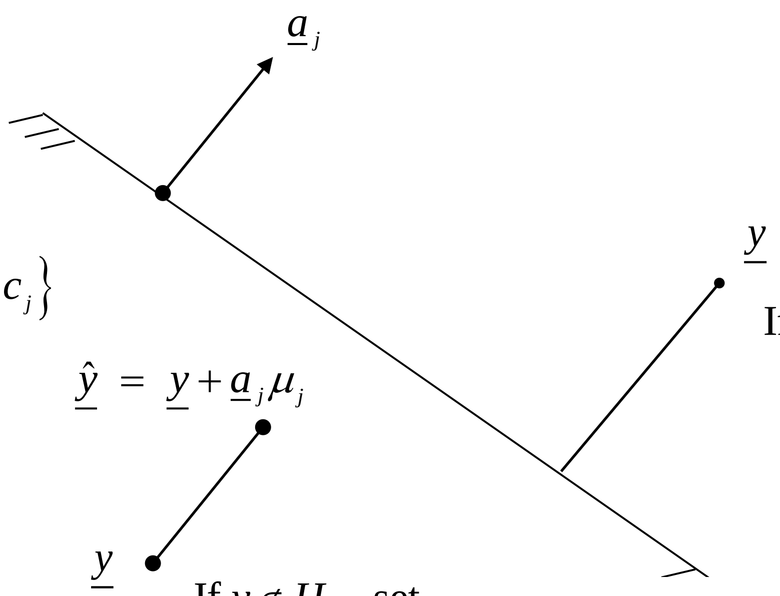
$$\mu_j = \max(0, \mu_j - \frac{1}{\underline{a}_j^T\underline{a}_j}(c_j - \underline{a}_j^T\underline{y})) = \mu_j - \min\left(\mu_j, \frac{1}{\underline{a}_j^T\underline{a}_j}(c_j - \underline{a}_j^T\underline{y})\right)$$

A Gauss – Seidel algorithm is of the form: $\underline{\mu} = \underline{\mu} - \min\left(\mu_j, \frac{1}{\underline{a}_j^T\underline{a}_j}(c_j - \underline{a}_j^T\underline{y})\right)\underline{e}_j$

Corresponding iteration for $\underline{y}$ is $\underline{y} = -A^T\underline{\mu} \Rightarrow \underline{y} = \underline{y} + \min\left(\mu_j, \frac{1}{\underline{a}_j^T\underline{a}_j}(c_j - \underline{a}_j^T\underline{y})\right)\underline{a}_j$

Geometric Interpretation

- Geometric Interpretation:



Half Space

$$H_j = \{\underline{a}_j^T \underline{x} \le c_j\}$$

$$\underline{\hat{y}} = \underline{y} + \underline{a}_j \mu_j$$

If $\underline{y} \notin H_j$, set it to

$$\underline{\hat{y}} = \underline{y} + \frac{1}{\underline{a}_j^T \underline{a}_j}\left[c_j - \underline{a}_j^T \underline{y}\right]\underline{a}_j$$

If $\underline{y} \notin H_j$, set

$$\underline{\hat{y}} = \begin{cases} \underline{y} + \underline{a}_j \mu_j & \text{if} \quad \mu_j < \dfrac{c_j - \underline{a}_j^T \underline{y}}{\underline{a}_j^T \underline{a}_j} \\ \underline{y} + \dfrac{1}{\underline{a}_j^T \underline{a}_j}\left[c_j - \underline{a}_j^T \underline{y}\right]\underline{a}_j, & \text{otherwise} \end{cases}$$

Extension to General QPP

For general quadratic programming problems, the procedure extends easily.

$$\min \frac{1}{2}\underline{x}^T Q\underline{x} - \underline{b}^T \underline{x} \quad \text{s.t. } A\underline{x} \le \underline{c} \quad \Leftrightarrow \quad \min_{\underline{\mu}\ge 0} \frac{1}{2}\underline{\mu}^T P\underline{\mu} + \underline{r}^T \underline{\mu}; \quad P = AQ^{-1}A^T; \underline{r} = c - AQ^{-1}\underline{b}$$

Define $\underline{y} = -A^T\underline{\mu}$ and $P\underline{\mu} = -AQ^{-1}\underline{y} \Rightarrow \underline{p}_j^T\underline{\mu} = -\underline{w}_j^T\underline{y};\ \underline{w}_j^T = j^{th}$ row of AQ^{-1}

The iterations are:

$$\underline{\mu} = \underline{\mu} + \min\left(\mu_j, \frac{1}{\underline{w}_j^T\underline{a}_j}\left[r_j - \underline{w}_j^T\underline{y}\right]\right)\underline{e}_j$$

$$\underline{y} = \underline{y} + \min\left(\mu_j, \frac{1}{\underline{w}_j^T\underline{a}_j}\left[r_j - \underline{w}_j^T\underline{y}\right]\right)\underline{a}_j$$

- Example 2: Finding a point in a set intersection by parallel projections

$$\min \frac{1}{2}\sum_{i=1}^{m}\|x_i - \underline{x}\|_2^2$$

s.t. $\underline{x} \in R^n$

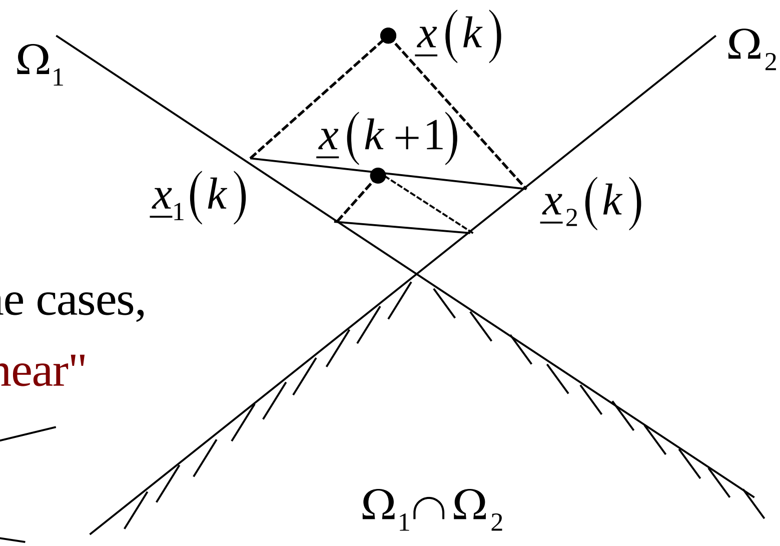
and $\underline{x}_i \in \Omega_i \ , i = 1, 2, \ldots, m$

This is a simple seperable problem. Start with any $\underline{x}(k)$ and solve for

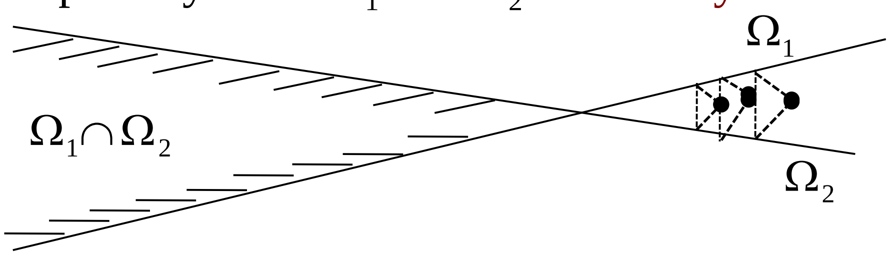
Set Intersection Problem

$$\underline{x}_i(k) = \arg\min_{\underline{x}_i \in \Omega_i} \left\| \underline{x}_i - \underline{x}(k) \right\|^2$$

Then, $\underline{x}(k+1) = \dfrac{1}{m}\sum_{i=1}^{m} \underline{x}_i(k)$



Convergence can be extremely poor in some cases, especially when Ω_1 and Ω_2 are "nearly colinear"



- Example 3: Linear programming problems

$$\min \underline{c}^T \underline{x}$$

$$\text{s.t. } A\underline{x} = \underline{b}$$

$$0 \le \underline{x} \le \underline{d}$$

Using the method of multipliers: at step k

Primal: $\underline{x}_{k+1} = \arg\min_{0 \le \underline{x} \le \underline{d}} \left[\underline{c}^T \underline{x} + \frac{h_k}{2}\left(A\underline{x} - \underline{b}\right)^T \left(A\underline{x} - \underline{b}\right) + \underline{\lambda}_k^T \left(A\underline{x} - \underline{b}\right) \right]$

Dual: $\underline{\lambda}_{k+1} = \underline{\lambda}_k + h_k \left(A\underline{x}_{k+1} - \underline{b}\right)$

Parallel LP - 1

– Approach1: The primal is equivalent to:

$$\min_{\underline{x}} \frac{1}{2}\underline{x}^T A^T A\underline{x} + \frac{1}{h_k}\underbrace{\left[\underline{c} + A^T\left(\underline{\lambda}_k - h_k\underline{b}\right)\right]^T}_{r_k}\underline{x}; P = A^T A$$

This is a quadratic programming problem , similar to the one we studied earlier. All we need to do is to project the unconstrained minimum along each direction i onto $[0,d_i]$. The unconstrained minimum $\tilde{x}_i$ is

$$\tilde{x}_i = \frac{-1}{\left(\underline{a}_i^T\underline{a}_i\right)h_k}\left[c_i + \underline{a}_i^T\left[\underline{\lambda}_k - h_k\underline{b} + h_k\sum_{j\neq i}\underline{a}_j x_j\right]\right]$$

$$\text{or} \quad x_i = \left\{x_i - \frac{1}{\left(\underline{a}_i^T\underline{a}_i\right)h_i}\left[c_i + \underline{a}_i^T\underline{\lambda}_k - h_k\left(\underline{b} + \underline{y}\right)\right]\right\}^{\#}$$

where $\underline{y} = -A\underline{x}$ #Projection onto $\left[0, d_i\right]$

Parallel LP - 2

The update for $\underline{y}$ is :

$$\underline{y} : \underline{y} + \left[x_i - \left\{ x_i - \frac{1}{\left(\underline{a}_i^T \underline{a}_i\right) h_i} \left[c_i + \underline{a}_i^T \underline{\lambda}_k - h_k \left(\underline{b} + \underline{y} \right) \right] \right\}^+ \right] \underline{a}_i$$

– Approach2: Let $I(i) = \{ j \,|\, a_{ij} \neq 0 \}; i = 1,2,\ldots.,m$

$$z_{ij} = a_{ij} x_j \quad i = 1,2,\ldots,m; \quad j = 1,2,\ldots..,n \qquad (1)$$

$$\Rightarrow \sum_{j \in I(i)} z_{ij} = b_i \quad i = 1,2,\ldots,m$$

Let λ_{ij} be the set of Lagrange multipliers associated with (1)

The problem is: $\min \underline{c}^T \underline{x} = \sum_{j=1}^{n} c_j x_j$

s.t. $\quad a_{ij} x_j = z_{ij}$

and $\quad \sum_{j \in I(i)} z_{ij} = b_i$

Parallel LP - 3

Form augmented Lagrangian

$$\sum_{j=1}^{n} c_i x_i + \sum_{i=1}^{m} \sum_{j\in I(i)} \lambda_{ijk}\left(a_{ij}x_j - z_{ij}\right) + \frac{h_k}{2}\sum_{i=1}^{m}\sum_{j\in I(i)}\left(a_{ij}x_j - z_{ij}\right)^2$$

$$\text{s.t.} \quad \sum_{j\in I(i)} z_{ij} = b_i;\ i = 1,2,\ldots\ldots,m; \quad 0 \le x_i \le d_i$$

so,

$$x_j = \arg\min_{0\le x_j \le d_j}\left\{\left[c_j x_j + \sum_{\{i \mid j\in I(i)\}} \lambda_{ijk} a_{ij} x_j + \frac{h_k}{2}\left(a_{ij}x_j - z_{ij}\right)^2\right]\right\}$$

and

$$\left\{z_{ij} \middle| j \in I(i)\right\} = \underset{z_{ij} \ni \sum_{j\in I(i)} z_{ij} = b_i}{\arg\min}\left\{-\sum_{j\in I(i)} \lambda_{ijk} z_{ij} \frac{h_k}{2}\sum_{j\in I(i)}\left(a_{ij}x_j - z_{ij}\right)^2\right\}$$

The optimization w.r.t z_{ij} is a quadratic programming problem with a single equality constraint. We can write optimal solution directly.

Parallel LP - 4

If p_i is the Lagrange multiplier, it is given by:

$$p_{i,k+1} = \frac{1}{m_i} \sum_{j \in I(i)} \lambda_{ijk} + \frac{h_k}{m_i} \left[\sum_{j \in I(i)} \left(a_{ij} x_j - b_i \right) \right]; i = 1, 2, \ldots, m$$

where $m_i = |I(i)|$ the number of non zero elements in row i of A

$$\therefore z_{ijk+1} = a_{ij} x_{jk+1} + \frac{\lambda_{ijk} - p_{i,k+1}}{h_k}$$

Note that

$$\lambda_{ijk+1} = \lambda_{ijk} + h_k \left[a_{ij} x_{jk+1} - z_{ijk+1} \right]$$

$$= p_{i,k+1} \Rightarrow \text{Don't actually need } \lambda_{ij}, \text{ need only } p_i$$

$$\text{so, } p_{i,k+1} = p_{i,k} + \frac{h_k}{m_i} \sum_{j \in I(i)} \left[a_{ij} x_{j,k+1} - b_i \right] \;;$$

$$\text{so, } z_{ij,k+1} = a_{ij} x_{jk+1} + \frac{p_{i,k} - p_{i,k+1}}{h_k} = a_{ij} x_{j,k+1} - \frac{1}{m_i} \sum_{j \in I(i)} \left[a_{ij} x_{jk+1} - b_i \right]$$

$$= a_{ij} x_{j,k+1} - \frac{1}{m_i} \left[\underline{a}_i^T \underline{x}_{k+1} - b_i \right]$$

Parallel LP - 5

$\Rightarrow$ Can eliminate z_{ij} altogether and get a Gauss-seidel iteration.

$$x_j(k+1) = \arg\min_{0 \le x_j \le d_j} \left\{ \left(c_j + \sum_{\{i \mid j \in I(i)\}} p_{i,k} a_{ij} \right) x_j + \frac{h_k}{2} \sum_{\{i \mid j \in I(i)\}} \left[a_{ij}\left(x_j - x_{j,k}\right) + w_i \right]^2 \right\}$$

where $w_i = \frac{1}{m_i}\left(\underline{a}_i^T \underline{x}_k - \underline{b}_i\right)$ $\Rightarrow$ one dimensional minimization

Since quadratic, can find unconstrained minimum and project onto $\left(0, d_j\right)$

The solution is:

$$x_j(k+1) = \left\{ x_{j,k} - \frac{1}{h_k\left(\underline{a}_j^T \underline{a}_j\right)} \left[c_j + \underline{a}_j^T \underline{b} + h_k \underline{a}_j^T \underline{w} \right] \right\}^{\#}$$

The methods can be extremely slow. May need to use Diagonal scaling for faster convergence. The method extends easily to separable nonlinear functions

$$f\left(\underline{x}\right) = \sum_{i=1}^{n} f_i\left(x_i\right)$$

Summary

- Key Ideas of Parallel Algorithms
- Jacobi Algorithm
- Parallel implementation of methods
- Convergence Analysis
- Non-linear Jacobi and Gauss-Seidel Algorithms
- Constrained Optimization
- Decomposition Methods