Optimization Algorithms

Daniel P. Palomar (2025). *Portfolio Optimization: Theory and Application.* Cambridge University Press.

portfoliooptimizationbook.com

Latest update: 2025-09-15

Outline

Executive Summary

- Over the past century, convex optimization has evolved through key algorithmic breakthroughs, beginning with Dantzig's 1947 simplex method for linear programming.
- Despite exponential worst-case complexity, the simplex method became widely adopted and laid the foundation for modern optimization.
- Karmarkar's 1984 interior-point method revolutionized linear programming by achieving polynomial time complexity, spurring extensive research into interior-point approaches.
- In 1994, Nesterov and Nemirovskii's theory of self-concordant functions enabled log-barrier algorithms to tackle broader convex problems including semidefinite programming and second-order cone programming.
- Specialized techniques such as block-coordinate descent, majorization-minimization, and successive convex approximation have emerged to create customized algorithms for specific problem structures.
- These specialized methods often provide enhanced complexity guarantees and improved convergence rates compared to general-purpose solvers.
- These slides explore a comprehensive range of practical algorithms developed through this rich algorithmic evolution (Palomar 2025, Appendix B).

Outline

Solvers

- A solver, or optimizer, is an engine designed to solve specific types of mathematical problems.

- Available in various programming languages: R, Python, Matlab, Julia, Rust, C, C++.

- Each solver typically handles specific problem categories: LP, QP, QCQP, SOCP, SDP.

Popular Solvers

GLPK (GNU Linear Programming Kit):

- Intended for large-scale LP including mixed-integer variables.
- Written in C.

Quadprog:

- Popular open-source QP solver.
- Originally written in Fortran by Berwin Turlach in the late 1980s.
- Accessible from most programming languages.

MOSEK:

- Proprietary solver for LP, QP, SOCP, SDP including mixed-integer variables.
- Established in 1997 by Erling Andersen.
- Specialized in large-scale problems; very fast, robust, and reliable.
- Free license available for academia.

SeDuMi:

- Open-source solver for LP, QP, SOCP, SDP.
- Originally developed by Sturm in 1999 for Matlab.

Popular Solvers

SDPT3:

- Open-source solver for LP, QP, SOCP, SDP.
- Originally developed in 1999 for Matlab.

Gurobi:

- Proprietary solver for LP, QP, and SOCP including mixed-integer variables.
- Free license available for academia.

Embedded Conic Solver (ECOS):

- SOCP solver originally written in C.

CPLEX:

- Proprietary solver for LP and QP, also handles mixed-integer variables.
- Free license available for academia.

Complexity of Interior-Point Methods

General Complexity

Complexity for LP, QP, QCQP, SOCP, and SDP is $O(n^3L)$, where

- n: number of variables.
- L: number of accuracy digits of the solution.

Specific Complexities

- **LP:** $O((m+n)^{3/2}n^2L)$.
- **QCQP:** $O(\sqrt{m}(m+n)n^2L)$.
- **SOCP:** $O(\sqrt{m+1}\; n(n^2+m+(m+1)k^2)L)$ with k the cone dimension.
- **SDP:** $O(\sqrt{1+mk}\; n(n^2+nmk^2+mk^3)L)$, with $k \times k$ the matrix dimension.

Example Analysis

- For SOCP with $m = O(n)$ and $k = O(n)$, complexity is $O(n^{4.5}L)$.
- For SDP with $k = O(n)$, complexity is $O(n^4)$.
- If $m = O(n)$ for SDP, complexity becomes $O(n^6L)$.
- Complexity for solving SOCP is higher than LP, QP, and QCQP; even higher for SDP.

Interface with Solvers

Solvers and Standard Form:

- Problems must be expressed in a standard form for solvers.
- Transformation to standard form is time-consuming and error-prone.

General Norm Approximation Problem:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|$$

Solution depends on the choice of the norm.

Norm Approximation with Euclidean or ℓ_2-norm:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2$$

Least squares (LS) problem with analytic solution: $\boldsymbol{x}^\star = (\boldsymbol{A}^\mathsf{T}\boldsymbol{A})^{-1}\boldsymbol{A}^\mathsf{T}\boldsymbol{b}$.

Norm Approximation with Chebyshev or ℓ_∞-norm:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_\infty$$

- Rewritten as LP:

$$\begin{array}{ll} \underset{\boldsymbol{x},t}{\text{minimize}} & t \\ \text{subject to} & -t\mathbf{1} \leq \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \leq t\mathbf{1} \end{array}$$

- Equivalent form:

$$\begin{array}{ll} \underset{\boldsymbol{x},t}{\text{minimize}} & \begin{bmatrix} \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ t \end{bmatrix} \\ \text{subject to} & \begin{bmatrix} \boldsymbol{A} & -\mathbf{1} \\ -\boldsymbol{A} & -\mathbf{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ t \end{bmatrix} \leq \begin{bmatrix} \boldsymbol{b} \\ -\boldsymbol{b} \end{bmatrix} \end{array}$$

- Matlab code:

```
xt = linprog( [zeros(n,1); 1],
              [A,-ones(m,1); -A,-ones(m,1)],
              [b; -b] )
x = xt(1:n)
```

Norm Approximation Problem with Manhattan or ℓ_1-Norm:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_1$$

- Rewritten as LP:

$$\begin{aligned} \underset{\boldsymbol{x},\boldsymbol{t}}{\text{minimize}} \quad & \mathbf{1}^\mathsf{T}\boldsymbol{t} \\ \text{subject to} \quad & -\boldsymbol{t} \le \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \le \boldsymbol{t} \end{aligned}$$

- Equivalent form:

$$\begin{aligned} \underset{\boldsymbol{x},\boldsymbol{t}}{\text{minimize}} \quad & \begin{bmatrix} \mathbf{0}^\mathsf{T} & \mathbf{1}^\mathsf{T} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{t} \end{bmatrix} \\ \text{subject to} \quad & \begin{bmatrix} \boldsymbol{A} & -\boldsymbol{I} \\ -\boldsymbol{A} & -\boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{t} \end{bmatrix} \le \begin{bmatrix} \boldsymbol{b} \\ -\boldsymbol{b} \end{bmatrix} \end{aligned}$$

- Matlab code:

```
xt = linprog( [zeros(n,1); ones(n,1)],
              [A,-eye(m,1); -A,-eye(m,1)],
              [b; -b] )
x = xt(1:n)
```

Euclidean Norm Approximation Problem with Linear Constraints:

$$\begin{array}{ll} \underset{\boldsymbol{x}}{\text{minimize}} & \|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2 \\ \text{subject to} & \boldsymbol{C}\boldsymbol{x}=\boldsymbol{d} \\ & \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}. \end{array}$$

- Equivalent form:

$$\begin{array}{ll} \underset{\boldsymbol{x},\boldsymbol{y},t,\boldsymbol{s}_l,\boldsymbol{s}_u}{\text{minimize}} & t \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}=\boldsymbol{y} \\ & \boldsymbol{C}\boldsymbol{x}=\boldsymbol{d} \\ & \boldsymbol{x}-\boldsymbol{s}_l=\boldsymbol{l} \\ & \boldsymbol{x}+\boldsymbol{s}_u=\boldsymbol{u} \\ & \boldsymbol{s}_l,\boldsymbol{s}_u \geq \boldsymbol{0} \\ & \|\boldsymbol{y}\|_2 \leq t \end{array}$$

Interface with Solvers

- Recall the previous equivalent form:

$$\begin{array}{ll} \underset{\boldsymbol{x},\boldsymbol{y},t,\boldsymbol{s}_l,\boldsymbol{s}_u}{\text{minimize}} & t \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = \boldsymbol{y} \\ & \boldsymbol{C}\boldsymbol{x} = \boldsymbol{d} \\ & \boldsymbol{x} - \boldsymbol{s}_l = \boldsymbol{l} \\ & \boldsymbol{x} + \boldsymbol{s}_u = \boldsymbol{u} \\ & \boldsymbol{s}_l, \boldsymbol{s}_u \geq \boldsymbol{0} \\ & \|\boldsymbol{y}\|_2 \leq t \end{array}$$

- Equivalent form in matrix notation:

$$\begin{array}{ll} \underset{\boldsymbol{x},\boldsymbol{y},t,\boldsymbol{s}_l,\boldsymbol{s}_u}{\text{minimize}} & \begin{bmatrix} \boldsymbol{0}^\mathsf{T} & \boldsymbol{0}^\mathsf{T} & \boldsymbol{0}^\mathsf{T} & \boldsymbol{0}^\mathsf{T} & 1 \end{bmatrix} \bar{\boldsymbol{x}} \\ \text{subject to} & \begin{bmatrix} \boldsymbol{A} & & & -\boldsymbol{I} \\ \boldsymbol{C} & & & \\ \boldsymbol{I} & -\boldsymbol{I} & & \\ \boldsymbol{I} & & \boldsymbol{I} & \end{bmatrix} \bar{\boldsymbol{x}} \leq \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{d} \\ \boldsymbol{l} \\ \boldsymbol{u} \end{bmatrix} \\ & \bar{\boldsymbol{x}} \in \boldsymbol{R}^n \times \boldsymbol{R}_+^n \times \boldsymbol{R}_+^n \times \boldsymbol{Q}^m \end{array}$$

Interface with solvers

- Matlab code:

```
AA = [ A, zeros(m,n), zeros(m,n),   -eye(m),     0;
       C, zeros(p,n), zeros(p,n),   zeros(p,n),  0;
       eye(n), -eye(n), zeros(n,n), zeros(n,n),  0;
       eye(n), zeros(n,n), eye(n),  zeros(n,n),  0 ]
bb = [ b; d; l; u ]
cc = [ zeros(3*n + m, 1); 1 ]
K.f = n; K.l = 2*n; K.q = m + 1
xsyz = sedumi( AA, bb, cc, K )
x = xsyz(1:n)
```

Modeling Frameworks

Modeling Framework:

- Simplifies the use of solvers by handling solver argument formatting.
- Acts as an interface between the user and the solver.
- Can interface with various solvers, allowing user choice based on problem type.
- Useful for rapid prototyping and avoiding transcription errors.
- Direct solver calls may be preferred for high-speed requirements.

Successful Examples:

- YALMIP: For Matlab (Löfberg 2004).
- CVX: Initially released in 2005 for Matlab. Now available in Python, R, and Julia. (Grant and Boyd 2008, 2014; Fu, Narasimhan, and Boyd 2020).

CVX (Convex Disciplined Programming):

- Tool for rapid prototyping of models and algorithms with convex optimization.
- Interfaces with solvers like SeDuMi, SDPT3, Gurobi, and MOSEK.
- Recognizes elementary convex and concave functions and composition rules.
- Determines problem convexity.
- Simple and convenient for prototyping.

Example: Constrained Euclidean Norm Approximation in CVX:

- Problem statement:

$$\begin{array}{ll} \underset{\boldsymbol{x}}{\text{minimize}} & \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2 \\ \text{subject to} & \boldsymbol{C}\boldsymbol{x} = \boldsymbol{d} \\ & \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} \end{array}$$

- Matlab code:

```
cvx_begin
    variable x(n)
    minimize(norm(A * x - b, 2))
    subject to
        C * x == d
        l <= x
        x <= u
cvx_end
```

Example: Constrained Euclidean Norm Approximation in CVX: (cont'd)

- R code:

```
x <- Variable(n)
prob <- Problem(Minimize(cvxr_norm(A %*% x - b, 2)),
                list(C %*% x == d,
                     l <= x,
                     x <= u))
solve(prob)
```

- Python code:

```
x = cvxpy.Variable(n)
prob = cvxpy.Problem(cvxpy.Minimize(cvxpy.norm(A @ x - b, 2)),
                     [C @ x == d,
                      l <= x,
                      x <= u])
prob.solve()
```

Outline

Gradient Methods

Unconstrained Optimization Problem:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad f(\boldsymbol{x})$$

where f is the objective function, assumed to be continuously differentiable.

Iterative Methods:

- Produce a sequence of iterates $\boldsymbol{x}^0, \boldsymbol{x}^1, \boldsymbol{x}^2, \ldots$.
- Sequence may or may not converge to an optimal solution $\boldsymbol{x}^\star$.

Ideal Case with Convex f: As iterations proceed ($k \to \infty$),

- objective function converges to the optimal value:

$$f\left(\boldsymbol{x}^k\right) \to p^\star$$

- gradient tends to zero:

$$\nabla f\left(\boldsymbol{x}^k\right) \to \mathbf{0}$$

References: (Bertsekas 1999), (S. P. Boyd and Vandenberghe 2004), (Nocedal and Wright 2006), (Beck 2017).

Descent Methods

Descent Methods (Gradient Methods):

- Satisfy the property: $f\left(\boldsymbol{x}^{k+1}\right) < f\left(\boldsymbol{x}^{k}\right)$.
- Iterates are obtained as:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^{k} + \alpha^{k}\boldsymbol{d}^{k},$$

  - $\boldsymbol{d}^{k}$: *search direction*.
  - α^{k}: *stepsize* at iteration k.

Descent Property:

- For a sufficiently small step, $\boldsymbol{d}$ must satisfy:

$$\nabla f\left(\boldsymbol{x}\right)^{\mathsf{T}}\boldsymbol{d} < 0$$

- α must be properly chosen (if too large, the descent property may be violated even with a descent direction).

Line Search: Procedure to choose the stepsize α.

- **Exact Line Search:** Solves the univariate optimization problem

$$\alpha = \arg\min_{\alpha > 0} f(\boldsymbol{x} + \alpha \boldsymbol{d}).$$

- **Backtracking Line Search (Armijo Rule):** Starting at $\alpha = 1$, repeat $\alpha \leftarrow \beta\alpha$ until

$$f(\boldsymbol{x} + \alpha \boldsymbol{d}) < f(\boldsymbol{x}) + \sigma\alpha \nabla f(\boldsymbol{x})^{\mathsf{T}} \boldsymbol{d},$$

where $\sigma \in (0, 1/2)$ and $\beta \in (0, 1)$ are given parameters.

Gradient Descent Method (Steepest Descent Method)
A descent method where the search direction is the opposite of the gradient:

$$\boldsymbol{d} = -\nabla f(\boldsymbol{x}),$$

which is a descent direction since $\nabla f(\boldsymbol{x})^{\mathsf{T}} \boldsymbol{d} < 0$.

Gradient Descent Update:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^{k} - \alpha^{k} \nabla f\left(\boldsymbol{x}^{k}\right).$$

Stopping Criterion: Common heuristic: $\|\nabla f(\boldsymbol{x})\|_2 \leq \epsilon$.

Convergence:

- Often slow, making it rarely used in practice.
- Useful in high-dimensional problems or when distributed implementation is required.

Algorithm

Gradient descent method

Initialization:

- Choose initial point $\boldsymbol{x}^0$.
- Set $k \leftarrow 0$.

Repeat (kth iteration):

1. Compute the negative gradient as descent direction: $\boldsymbol{d}^k = -\nabla f\left(\boldsymbol{x}^k\right)$.
2. Line search: Choose a stepsize $\alpha^k > 0$ via exact or backtracking line search.
3. Obtain next iterate:
$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \alpha^k \nabla f\left(\boldsymbol{x}^k\right).$$
4. $k \leftarrow k + 1$

Until: convergence

Newton's Method

Newton's Method

- A descent method using both the gradient and the Hessian of f.
- **Search direction:**

$$\boldsymbol{d} = -\nabla^2 f(\boldsymbol{x})^{-1} \nabla f(\boldsymbol{x}),$$

- Assumes f is convex, twice continuously differentiable, and the Hessian matrix is positive definite for all $\boldsymbol{x}$.

Second-Order Approximation: $\boldsymbol{x} + \boldsymbol{d}$ minimizes the second-order approximation of $f(\boldsymbol{x})$ around $\boldsymbol{x}$:

$$\hat{f}(\boldsymbol{x} + \boldsymbol{v}) = f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathsf{T}} \boldsymbol{v} + \frac{1}{2} \boldsymbol{v}^{\mathsf{T}} \nabla^2 f(\boldsymbol{x}) \boldsymbol{v}.$$

Newton's Method Update:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \alpha^k \nabla^2 f\left(\boldsymbol{x}^k\right)^{-1} \nabla f\left(\boldsymbol{x}^k\right).$$

Newton's Method

Newton Decrement

- Measures the proximity of $\boldsymbol{x}$ to an optimal point:

$$\lambda(\boldsymbol{x}) = (\nabla f(\boldsymbol{x})^{\mathsf{T}} \nabla^2 f(\boldsymbol{x})^{-1} \nabla f(\boldsymbol{x}))^{1/2}$$

- Estimates $f(\boldsymbol{x}) - p^{\star}$:

$$f(\boldsymbol{x}) - \inf_{\boldsymbol{y}} \hat{f}(\boldsymbol{y}) = \frac{1}{2}\lambda(\boldsymbol{x})^2$$

- Computational cost of the Newton decrement is negligible since $\lambda(\boldsymbol{x})^2 = -\nabla f(\boldsymbol{x})^{\mathsf{T}} \boldsymbol{d}$.

Advantages and Limitations

- Fast convergence.
- Central to most modern solvers.
- Impractical for very large dimensional problems due to computation and storage of the Hessian.
- For large problems, quasi-Newton methods are used (Nocedal and Wright 2006).

Algorithm

Newton's method

Initialization:

- Choose initial point $\boldsymbol{x}^0$ and tolerance $\epsilon > 0$. Set $k \leftarrow 0$.

Repeat (kth iteration):

1. Compute Newton direction and decrement:

$$\boldsymbol{d}^k = -\nabla^2 f(\boldsymbol{x}^k)^{-1} \nabla f(\boldsymbol{x}^k) \quad \text{and} \quad \lambda(\boldsymbol{x}^k)^2 = -\nabla f(\boldsymbol{x}^k)^\mathsf{T} \boldsymbol{d}^k.$$

2. Line search: Choose a stepsize $\alpha^k > 0$ via exact or backtracking line search.
3. Obtain next iterate:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \alpha^k \nabla^2 f\left(\boldsymbol{x}^k\right)^{-1} \nabla f\left(\boldsymbol{x}^k\right).$$

4. $k \leftarrow k+1$

Until: convergence (i.e., $\lambda(\boldsymbol{x}^k)^2/2 \leq \epsilon$)

Convergence of Descent Methods

- Ideally, the sequence $\{\boldsymbol{x}^k\}$ should converge to a global minimum.
- For non-convex f, convergence to a global minimum is unlikely due to local minima.
- Descent methods typically converge to a stationary point.
- For convex f, a stationary point is a global minimum.

Theoretical Convergence

- Descent methods have nice theoretical convergence properties (Bertsekas 1999).
- **Theorem: Convergence of descent methods**
  - Suppose $\{\boldsymbol{x}^k\}$ is a sequence generated by a descent method (e.g., gradient descent or Newton's method).
  - Stepsize α^k chosen by exact line search or backtracking line search.
  - Every limit point of $\{\boldsymbol{x}^k\}$ is a stationary point of the problem.

Simpler Stepsize Rules with Theoretical Convergence (Bertsekas 1999)

- Constant stepsize: $\alpha^k = \alpha$ for sufficiently small α.
- Diminishing stepsize rule: $\alpha^k \to 0$ with $\sum_{k=0}^{\infty} \alpha^k = \infty$.

Convergence

Newton's Method Convergence Phases

- Damped Newton phase: Slow convergence.
- Quadratically convergent phase: Extremely fast convergence.

Practical Considerations

- Gradient descent method converges slowly.
- Newton's method converges much faster but requires computing the Hessian.
- Newton's method is preferred if problem dimensionality is manageable.
- For extremely large dimensional problems (e.g., deep learning), computing and storing the Hessian is not feasible.

Projected Gradient Methods

Constrained Optimization Problem

$$\begin{array}{ll} \underset{\boldsymbol{x}}{\text{minimize}} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in \mathcal{X}, \end{array}$$

where f is the objective function (continuously differentiable) and $\mathcal{X}$ is a convex set.

Descent Method

- Iterative update:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha^k \boldsymbol{d}^k$$

  where $\boldsymbol{d}^k$ is a descent direction.
- Potential issue: $\boldsymbol{x}^{k+1}$ may be infeasible.

Projected Gradient Methods (Gradient Projection Methods)

Address infeasibility by projecting onto the feasible set after taking the step (Bertsekas 1999; Beck 2017):

$$\boldsymbol{x}^{k+1} = \left[\boldsymbol{x}^k + \alpha^k \boldsymbol{d}^k\right]_{\mathcal{X}}$$

where $[\boldsymbol{x}]_{\mathcal{X}}$ denotes projection onto the set $\mathcal{X}$: $\min_{\boldsymbol{y}} \|\boldsymbol{y} - \boldsymbol{x}\| \quad \text{subject to} \quad \boldsymbol{y} \in \mathcal{X}$.

Projected Gradient Methods

Generalized Gradient Projection Method

- Iterative update:

$$\bar{\boldsymbol{x}}^k = \left[\boldsymbol{x}^k + s^k \boldsymbol{d}^k\right]_{\mathcal{X}}$$
$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha^k \left(\bar{\boldsymbol{x}}^k - \boldsymbol{x}^k\right),$$

  - $\boldsymbol{d}^k = \bar{\boldsymbol{x}}^k - \boldsymbol{x}^k$ is a feasible direction.
  - α^k is the stepsize.
  - s^k is a positive scalar (Bertsekas 1999).
- Special case: $\alpha^k = 1$:

$$\boldsymbol{x}^{k+1} = \left[\boldsymbol{x}^k + s^k \boldsymbol{d}^k\right]_{\mathcal{X}}$$

  - s^k can be viewed as a stepsize.
  - If $\boldsymbol{x}^k + s^k \boldsymbol{d}^k$ is already feasible, the method reduces to the regular gradient method.

Practical Consideration

Gradient projection method is practical only if the projection is easy to compute.

Projected Gradient Descent Method

- Uses the negative gradient as the search direction.
- Iterative update:

$$\bar{\boldsymbol{x}}^k = \left[\boldsymbol{x}^k - s^k \nabla f\left(\boldsymbol{x}^k\right)\right]_{\mathcal{X}}$$

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha^k \left(\bar{\boldsymbol{x}}^k - \boldsymbol{x}^k\right),$$

  - $\bar{\boldsymbol{x}}^k$: Projection of $\boldsymbol{x}^k - s^k \nabla f\left(\boldsymbol{x}^k\right)$ onto the set $\mathcal{X}$.
  - α^k: Stepsize.
  - s^k: Positive scalar stepsize for the gradient step.

Constrained Newton's Method

- Assumptions:
  - f is twice continuously differentiable.
  - The Hessian matrix is positive definite for all $\boldsymbol{x} \in \mathcal{X}$.
- Iterative update:

$$\bar{\boldsymbol{x}}^k = \underset{\boldsymbol{x} \in \mathcal{X}}{\arg\min} \left\{ \nabla f\left(\boldsymbol{x}^k\right)^\mathsf{T} \left(\boldsymbol{x} - \boldsymbol{x}^k\right) + \frac{1}{2s^k} \left(\boldsymbol{x} - \boldsymbol{x}^k\right)^\mathsf{T} \nabla^2 f\left(\boldsymbol{x}^k\right) \left(\boldsymbol{x} - \boldsymbol{x}^k\right) \right\}$$

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha^k \left(\bar{\boldsymbol{x}}^k - \boldsymbol{x}^k\right).$$

  - $\bar{\boldsymbol{x}}^k$: Solution to the quadratic subproblem.
  - α^k: Stepsize.
  - s^k: Positive scalar.
- Observations:
  - If $s^k = 1$, the quadratic cost is the second-order Taylor series expansion of f around $\boldsymbol{x}^k$.
  - The main difficulty is solving the quadratic subproblem to find $\bar{\boldsymbol{x}}^k$.
    - This may not be simple even when the constraint set $\mathcal{X}$ has a simple structure.
    - The method typically makes practical sense only for problems of small dimension.

Convergence

Convergence of Gradient Projection Methods

- Detailed in (Bertsekas 1999).
- **Theorem: Convergence of gradient projection methods.**
    - Suppose $\{\boldsymbol{x}^k\}$ is a sequence generated by a gradient projection method (e.g., projected gradient descent method or constrained Newton's method).
    - Stepsize α^k chosen by exact line search or backtracking line search.
    - Every limit point of $\{\boldsymbol{x}^k\}$ is a stationary point of the problem.

Simpler Stepsize Rules with Theoretical Convergence

- Constant stepsize: $\alpha^k = 1$ and $s^k = s$ for sufficiently small s (Bertsekas 1999).

Outline

Interior-Point Methods (IPM)

Traditional Optimization Algorithms

- Based on gradient projection methods.
- May suffer from:
  - Slow convergence.
  - Sensitivity to algorithm initialization.
  - Sensitivity to stepsize selection.

Interior-Point Methods (IPM)

- Modern approach for convex problems.
- Enjoy excellent convergence properties (polynomial convergence).
- Do not suffer from the usual problems of traditional methods.

Convex Optimization Problem

$$\begin{array}{lll} \underset{\boldsymbol{x}}{\text{minimize}} & f_0(\boldsymbol{x}) & \\ \text{subject to} & f_i(\boldsymbol{x}) \leq 0, & i = 1, \ldots, m \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} & \end{array}$$

References: (Nesterov and Nemirovskii 1994; Bertsekas 1999; Nemirovski 2001; S. P. Boyd and Vandenberghe 2004; Nocedal and Wright 2006; Nesterov 2018).

Eliminating Equality Constraints

Dealing with Equality Constraints

- Can be handled via Lagrange duality (S. P. Boyd and Vandenberghe 2004).
- Alternatively, can be eliminated in a pre-processing stage.

Representation of Solutions to $\boldsymbol{Ax} = \boldsymbol{b}$:

$$\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{Ax} = \boldsymbol{b}\} = \{\boldsymbol{Fz} + \boldsymbol{x}_0 \mid \boldsymbol{z} \in \mathbb{R}^{n-p}\},$$

- $\boldsymbol{x}_0$ is any particular solution to $\boldsymbol{Ax} = \boldsymbol{b}$.
- $\boldsymbol{F} \in \mathbb{R}^{n \times (n-p)}$ spans the nullspace of $\boldsymbol{A}$, i.e., $\boldsymbol{AF} = \boldsymbol{0}$.

Reduced or Eliminated Problem

- Equivalent to the original problem:

$$\begin{array}{ll} \underset{\boldsymbol{z}}{\text{minimize}} & \tilde{f}_0(\boldsymbol{z}) \triangleq f_0(\boldsymbol{Fz} + \boldsymbol{x}_0) \\ \text{subject to} & \tilde{f}_i(\boldsymbol{z}) \triangleq f_i(\boldsymbol{Fz} + \boldsymbol{x}_0) \leq 0, \qquad i = 1, \ldots, m, \end{array}$$

- Gradients and Hessians:

$$\nabla \tilde{f}_i(\boldsymbol{z}) = \boldsymbol{F}^T \nabla f_i(\boldsymbol{x})$$
$$\nabla^2 \tilde{f}_i(\boldsymbol{z}) = \boldsymbol{F}^T \nabla^2 f_i(\boldsymbol{x}) \boldsymbol{F}.$$

Logarithmic Barrier Method

Reformulation via Indicator Function:

$$\begin{array}{ll} \underset{\boldsymbol{x}}{\text{minimize}} & f_0(\boldsymbol{x}) + \sum_{i=1}^{m} I_{-}\left(f_i(\boldsymbol{x})\right) \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \end{array}$$

using the indicator function:

$$I_{-}(u) = \begin{cases} 0 & \text{if } u \leq 0 \\ \infty & \text{otherwise.} \end{cases}$$

Characteristics of the Reformulated Problem

- The inequality constraints are eliminated.
- The indicator function is included in the objective.
- The main drawbacks are that the indicator function is noncontinuous and nondifferentiable, making the approach not practical for optimization.

Logarithmic Barrier Method

Logarithmic Barrier

A popular smooth approximation of the indicator function is the **logarithmic barrier**:

$$I_{-}(u) \approx -\frac{1}{t}\log(-u),$$

- The parameter $t > 0$ controls the approximation.
- The approximation improves as $t \to \infty$.

Approximate Problem Using the Logarithmic Barrier

The reformulated problem is:

$$\begin{array}{ll} \underset{\boldsymbol{x}}{\text{minimize}} & f_0(\boldsymbol{x}) - \frac{1}{t}\sum_{i=1}^{m} \log\left(-f_i(\boldsymbol{x})\right) \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}. \end{array}$$

Logarithmic Barrier Function

- The overall barrier function (excluding the $1/t$ factor) is:

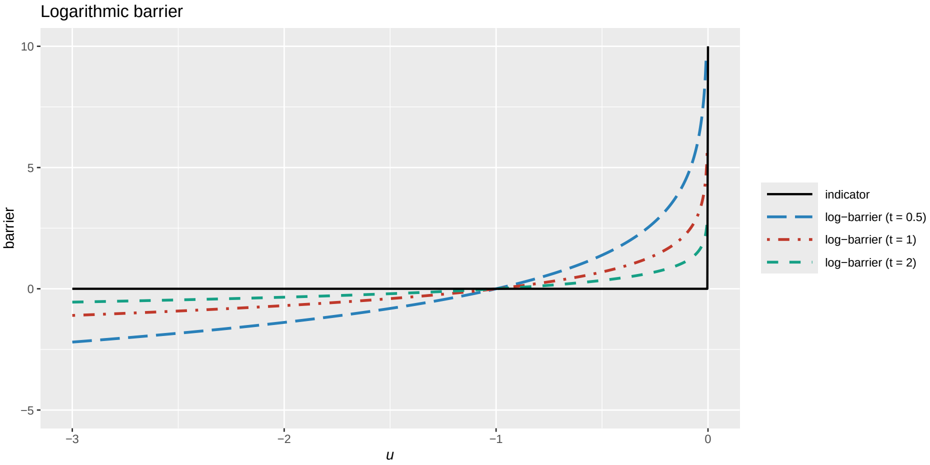
$$\phi(\boldsymbol{x}) = -\sum_{i=1}^{m} \log\left(-f_i(\boldsymbol{x})\right),$$

  which is convex (from composition rules).
- The gradient and Hessian are:

$$\nabla \phi(\boldsymbol{x}) = \sum_{i=1}^{m} \frac{1}{-f_i(\boldsymbol{x})} \nabla f_i(\boldsymbol{x})$$

$$\nabla^2 \phi(\boldsymbol{x}) = \sum_{i=1}^{m} \frac{1}{f_i(\boldsymbol{x})^2} \nabla f_i(\boldsymbol{x}) \nabla f_i(\boldsymbol{x})^\mathsf{T} + \sum_{i=1}^{m} \frac{1}{-f_i(\boldsymbol{x})} \nabla^2 f_i(\boldsymbol{x}).$$

Logarithmic barrier for several values of the parameter t:



Central Path

Defined as the curve $\{\boldsymbol{x}^\star(t) \mid t > 0\}$, where $\boldsymbol{x}^\star(t)$ is the solution to

$$\begin{array}{ll} \underset{\boldsymbol{x}}{\text{minimize}} & t f_0(\boldsymbol{x}) + \phi(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \end{array}$$

which can be solved via Newton's method.

Solution to the Central Path Problem

- Ignoring equality constraints for simplicity:

$$t\nabla f_0(\boldsymbol{x}) + \sum_{i=1}^{m} \frac{1}{-f_i(\boldsymbol{x})} \nabla f_i(\boldsymbol{x}) = \boldsymbol{0}.$$

- Define $\lambda_i^\star(t) = 1/(-t f_i(\boldsymbol{x}^\star(t)))$.
- $\boldsymbol{x}^\star(t)$ minimizes the Lagrangian:

$$L(\boldsymbol{x}; \boldsymbol{\lambda}^\star(t)) = f_0(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i^\star(t) f_i(\boldsymbol{x}).$$

Central Path

Convergence to Optimal Value

As $t \to \infty$, $f_0(\boldsymbol{x}^\star(t)) \to p^\star$. From Lagrange duality theory:

$$\begin{aligned} p^\star &\geq g\left(\boldsymbol{\lambda}^\star(t)\right) \\ &= L\left(\boldsymbol{x}^\star(t); \boldsymbol{\lambda}^\star(t)\right) \\ &= f_0\left(\boldsymbol{x}^\star(t)\right) - m/t. \end{aligned}$$

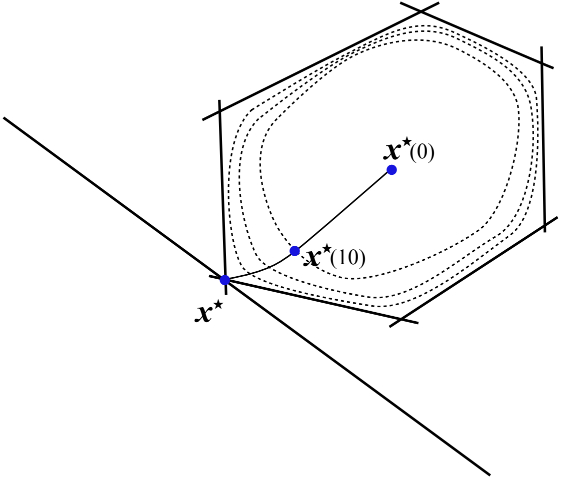
Connection with KKT Conditions

- The pair $\boldsymbol{x}^\star(t)$ and $\boldsymbol{\lambda}^\star(t)$ satisfies:

$$\begin{array}{rll} f_i(\boldsymbol{x}) \leq 0, & i = 1, \ldots, m & \text{(primal feasibility)} \\ \lambda_i \geq 0, & i = 1, \ldots, m & \text{(dual feasibility)} \\ \lambda_i f_i(\boldsymbol{x}) = -\frac{1}{t}, & i = 1, \ldots, m & \text{(approximate complementary slackness)} \\ \nabla f_0(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i \nabla f_i(\boldsymbol{x}) = \mathbf{0}. & & \text{(zero Lagrangian gradient)} \end{array}$$

- The difference with the original KKT conditions is:
  - Complementary slackness is approximately satisfied.
  - The approximation improves as $t \to \infty$.

Example of central path of an LP:



Logarithmic Barrier Method

Smooth Approximation with Logarithmic Barrier

- The log-barrier problem is a smooth approximation of the original problem.
- The approximation improves as $t \to \infty$.

Challenges with Choosing t

- Large t:
  - Leads to slow convergence.
  - Gradients and Hessians vary greatly near the boundary of the feasible set.
  - Newton's method fails to reach quadratic convergence.
- Small t:
  - Facilitates better convergence.
  - The approximation is not close to the original problem.

Adaptive t Approach

- Change t over iterations to balance fast convergence and accurate approximation.
- At each outer iteration, update t and compute $\boldsymbol{x}^\star(t)$ using Newton's method.
- Interior-point methods (IPM):
  - Achieve this trade-off.
  - For each $t > 0$, $\boldsymbol{x}^\star(t)$ is strictly feasible and lies in the interior of the feasible set.

Barrier Method

- A type of primal-based IPM.
- Update rule for t:
  - $t^{k+1} \leftarrow \mu t^k$, where $\mu > 1$.
  - Typically, $t^0 = 1$.
- Choice of μ:
  - Large μ means fewer outer iterations but more inner (Newton) iterations.
  - Typical values: $\mu = 10 \sim 20$.
- Termination criterion:
  - $m/t < \epsilon$, guaranteeing $f_0(\boldsymbol{x}) - p^\star \leq \epsilon$.
- Refer to (S. P. Boyd and Vandenberghe 2004) for practical details.

Algorithm

Logarithmic barrier method for constrained optimization

Initialization:

- Choose initial point $\boldsymbol{x}^0 \in \mathcal{X}$ stricly feasible, $t^0 > 0$, $\mu > 1$, and tolerance $\epsilon > 0$.
- Set $k \leftarrow 0$.

Repeat (kth iteration):

1. Centering step: compute next iterate $\boldsymbol{x}^{k+1}$ by solving the central path problem with $t = t^k$ and initial point $\boldsymbol{x}^k$.
2. Increase t: $\quad t^{k+1} \leftarrow \mu t^k$.
3. $k \leftarrow k + 1$

Until: convergence (i.e., $m/t < \epsilon$)

Example: Barrier Method for LP

Consider the LP:

$$\begin{array}{ll} \underset{\boldsymbol{x}}{\text{minimize}} & \boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}. \end{array}$$

Convergence Analysis

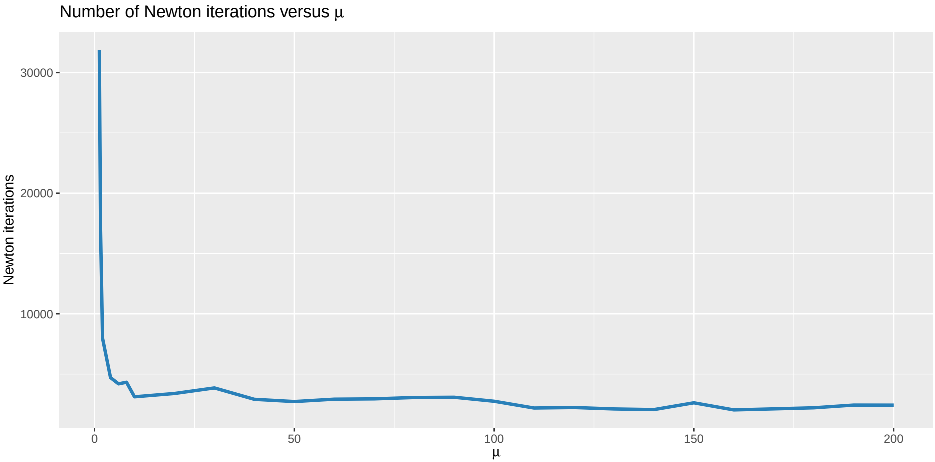
- Use the barrier method with different μ values.
- The case is $m = 100$ inequalities and $n = 50$ variables.
- The duality gap is $\epsilon = 10^{-6}$.
- The centering problem is solved via Newton's method.

Observation

The total number of Newton iterations is not very sensitive to μ as long as $\mu \geq 10$.

Logarithmic Barrier Method

Convergence of barrier method for an LP for different values of μ:



Convergence

Termination Criterion

- Number of outer iterations (centering steps) required:

$$\frac{m}{\mu^k t^0} \leq \epsilon$$

- Solving for k:

$$\left\lceil \frac{\log\left(m/\left(\epsilon t^0\right)\right)}{\log(\mu)} \right\rceil$$

where $\lceil \cdot \rceil$ is the ceiling operator.

Convergence of Centering Steps

- Characterized via the convergence for Newton's method.
- Specific updates for μ and good initialization points for each centering step are not considered in this simple analysis.

References

For detailed convergence analysis: (Nesterov and Nemirovskii 1994; Nemirovski 2001; S. P. Boyd and Vandenberghe 2004; Nocedal and Wright 2006; Nesterov 2018).

Feasibility and Phase I Methods

Barrier Method and Strictly Feasible Initial Point

- The barrier method requires a strictly feasible initial point $\boldsymbol{x}^0$ (such that $f_i\left(\boldsymbol{x}^0\right) < 0$).
- If such a point is not known, a preliminary stage (*Phase I*) is used to find it.
- The barrier method itself is then called *Phase II*.

Phase I Methods

- Aim to find a feasible point for the original problem by solving the feasibility problem:

$$\begin{array}{lll} \underset{\boldsymbol{x}}{\text{find}} & \boldsymbol{x} & \\ \text{subject to} & f_i(\boldsymbol{x}) \leq 0, & i = 1, \ldots, m \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}. & \end{array}$$

- The barrier method cannot be used directly for the feasibility problem as it requires a feasible starting point.

Feasibility and Phase I Methods

Formulating Phase I Methods

A simple example involves solving the convex optimization problem:

$$\begin{array}{lll} \underset{\boldsymbol{x},s}{\text{minimize}} & s & \\ \text{subject to} & f_i(\boldsymbol{x}) \leq s, & i = 1, \dots, m \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}. & \end{array}$$

- To construct a strictly feasible point, choose any $\boldsymbol{x}$ that satisfies the equality constraints.
- Then, choose s such that $s > f_i(\boldsymbol{x})$, e.g., $s = 1.1 \times \max_i\{f_i(\boldsymbol{x})\}$.
- This provides an initial strictly feasible point for the Phase I problem.

Solving the Phase I Problem

- After obtaining $(\boldsymbol{x}^\star, s^\star)$, the value of $s^\star$ is checked.
- If $s^\star < 0$, then $\boldsymbol{x}^\star$ is a strictly feasible point and can be used in the barrier method to solve the original problem.
- If $s^\star > 0$, then no feasible point exists, and there is no need to attempt solving the original problem as it is infeasible.

Primal-Dual Interior-Point Methods

Primal Barrier Method

- Requires a strictly feasible initial point.
- Involves distinct inner and outer iterations.

Primal-Dual IPMs

- More efficient, especially for high accuracy.
- Exhibit superlinear asymptotic convergence.
- Key features:
  - Update both primal and dual variables at each iteration.
  - No distinction between inner and outer iterations.
  - Can start at infeasible points, eliminating the need for Phase I methods.

Advantages

- Efficiency: Better for high accuracy.
- Convergence: Superlinear asymptotic convergence.
- Initialization: Can start from infeasible points, simplifying the process.

Summary: Primal-dual IPMs offer significant advantages over the primal barrier method in terms of efficiency, convergence, and ease of initialization.

Outline

Fractional Programming (FP) Methods

Concave-Convex Fractional Program (FP):

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{maximize}} \quad & \frac{f(\boldsymbol{x})}{g(\boldsymbol{x})} \\ \text{subject to} \quad & \boldsymbol{x} \in \mathcal{X}, \end{aligned}$$

- $f(\boldsymbol{x})$ is a concave function
- $g(\boldsymbol{x}) > 0$ is a convex function
- $\mathcal{X}$ is a convex feasible set

Nature of Fractional Programs

- Nonconvex problems, generally difficult to solve.
- Concave-convex FP is a quasiconvex optimization problem, making it more tractable.

Methods to Solve Concave-Convex Fractional Programs:

- Iterative bisection method
- Dinkelbach method
- Schaible transform

FP: Bisection Method

Problem Reformulation

- Solve a sequence of convex feasibility problems:

$$\begin{array}{ll} \underset{\boldsymbol{x}}{\text{find}} & \boldsymbol{x} \\ \text{subject to} & t g(\boldsymbol{x}) \leq f(\boldsymbol{x}) \\ & \boldsymbol{x} \in \mathcal{X} \end{array}$$

- $t > 0$ is a fixed parameter, not an optimization variable.

Goal: Find the optimal value of t for the original problem.

Procedure:

- If the feasibility problem is infeasible, t is too large and must be decreased.
- If feasible, t is too small and can be increased.

Algorithm

Key points:

- Starts with an interval $[l, u]$ known to contain the optimal value $p^\star$ and sequentially halves the interval.
- The length of the interval after k iterations is $2^{-k}(u - l)$.
- Number of iterations required to achieve a tolerance of ϵ is $\lceil \log_2((u - l)/\epsilon) \rceil$.

Bisection method (aka "sandwich technique") for concave-convex FP

Initialization:

- Initialize l and u such that $p^\star \in [l, u]$.

Repeat while $(u - l) > \epsilon$:

- Compute midpoint of interval: $t = (l + u)/2$.
- Solve the convex feasibility problem for t.
- If feasible, set $u = t$; otherwise set $l = t$.

FP: Dinkelbach Method

Dinkelbach Transform

- Objective: reformulate the original concave-convex FP into a sequence of simpler convex problems.
- Reformulated problem:

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{maximize}} \quad & f(\boldsymbol{x}) - y^k g(\boldsymbol{x}) \\ \text{subject to} \quad & \boldsymbol{x} \in \mathcal{X} \end{aligned}$$

- Parameter update: $y^k = \frac{f(\boldsymbol{x}^k)}{g(\boldsymbol{x}^k)}$ with k as the iteration index.

Convergence

- The Dinkelbach method converges to global optimum of the original concave-convex FP.
- Key properties:
  - Increasing sequence $\{y^k\}$.
  - Function $F(y) = \arg\max_{\boldsymbol{x}}\{f(\boldsymbol{x}) - yg(\boldsymbol{x})\}$.

Algorithm

Key points:

- Transforms a nonconvex problem into a sequence of convex problems.
- Ensures global optimality through iterative updates.

Dinkelback method for concave-convex FP

Initialization:

- Choose initial point $\boldsymbol{x}^0$.
- Set $k \leftarrow 0$.

Repeat (kth iteration):

1. Set $y^k = f(\boldsymbol{x}^k)/g(\boldsymbol{x}^k)$.
2. Solve the reformulated convex problem and keep current solution as $\boldsymbol{x}^{k+1}$.
3. $k \leftarrow k + 1$

Until: convergence

FP: Charnes-Cooper Transform

Linear Fractional Program (LFP)

$$\begin{array}{ll} \underset{\boldsymbol{x}}{\text{minimize}} & \dfrac{\boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} + d}{\boldsymbol{e}^{\mathsf{T}}\boldsymbol{x} + f} \\ \text{subject to} & \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h} \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \end{array}$$

with $\operatorname{dom} f_0 = \left\{\boldsymbol{x} \mid \boldsymbol{e}^{\mathsf{T}}\boldsymbol{x} + f > 0\right\}$.

Charnes-Cooper Transform: Transforms the original LFP into a linear program (LP):

$$\begin{array}{ll} \underset{\boldsymbol{y},t}{\text{minimize}} & \boldsymbol{c}^{\mathsf{T}}\boldsymbol{y} + dt \\ \text{subject to} & \boldsymbol{G}\boldsymbol{y} \leq \boldsymbol{h}t \\ & \boldsymbol{A}\boldsymbol{y} = \boldsymbol{b}t \\ & \boldsymbol{e}^{\mathsf{T}}\boldsymbol{y} + ft = 1 \\ & t \geq 0 \end{array}$$

where $\boldsymbol{y} = \frac{\boldsymbol{x}}{\boldsymbol{e}^{\mathsf{T}}\boldsymbol{x}+f}$ and $t = \frac{1}{\boldsymbol{e}^{\mathsf{T}}\boldsymbol{x}+f}$.

FP: Charnes-Cooper Transform

Proof:

- Any feasible point $\boldsymbol{x}$ in the original LFP leads to a feasible point $(\boldsymbol{y}, t)$ in the LP with the same objective value.
- Conversely, any feasible point $(\boldsymbol{y}, t)$ in the LP leads to a feasible point $\boldsymbol{x}$ in the original LFP via $\boldsymbol{x} = \boldsymbol{y}/t$, also with the same objective value:

$$\frac{\boldsymbol{c}^{\mathsf{T}}\boldsymbol{y} + dt}{1} = \frac{\boldsymbol{c}^{\mathsf{T}}\boldsymbol{y} + dt}{\boldsymbol{e}^{\mathsf{T}}\boldsymbol{y} + ft} = \frac{\boldsymbol{c}^{\mathsf{T}}\boldsymbol{y}/t + d}{\boldsymbol{e}^{\mathsf{T}}\boldsymbol{y}/t + f} = \frac{\boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} + d}{\boldsymbol{e}^{\mathsf{T}}\boldsymbol{x} + f}.$$

Concave-Convex Fractional Program (FP)

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{maximize}} & \dfrac{f(\mathbf{x})}{g(\mathbf{x})} \\ \text{subject to} & \mathbf{x} \in \mathcal{X} \end{array}$$

Schaible Transform: Rewrites the original concave-convex FP into a convex problem:

$$\begin{array}{ll} \underset{\mathbf{y},t}{\text{maximize}} & tf\left(\dfrac{\mathbf{y}}{t}\right) \\ \text{subject to} & tg\left(\dfrac{\mathbf{y}}{t}\right) \leq 1 \\ & t \geq 0 \\ & \mathbf{y}/t \in \mathcal{X} \end{array}$$

where $\mathbf{y} = \frac{\mathbf{x}}{g(\mathbf{x})}$ and $t = \frac{1}{g(\mathbf{x})}$.

FP: Schaible Transform

Proof:

- Any feasible point $\boldsymbol{x}$ in the original FP leads to a feasible point $(\boldsymbol{y}, t)$ in the convex problem with the same objective value.
- Conversely, any feasible point $(\boldsymbol{y}, t)$ in the convex problem leads to a feasible point $\boldsymbol{x}$ in the original FP via $\boldsymbol{x} = \boldsymbol{y}/t$, also with the same objective value:

$$tf\left(\frac{\boldsymbol{y}}{t}\right) = \frac{f(\boldsymbol{x})}{g(\boldsymbol{x})}.$$

Outline

Block-Coordinate Descent (BCD) Method

- Also known as: Gauss-Seidel method, alternate minimization method.
- Objective: solve a difficult optimization problem by solving a sequence of simpler subproblems.

Problem Formulation:

$$\begin{array}{ll} \underset{\boldsymbol{x}}{\text{minimize}} & f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \\ \text{subject to} & \boldsymbol{x}_i \in \mathcal{X}_i, \qquad i = 1, \ldots, n, \end{array}$$

where f is the (possibly nonconvex) objective function, each $\mathcal{X}_i$ is a convex set, and the variable $\boldsymbol{x}$ is partitioned into n blocks $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$.

Method Description

- The iterative process produces a sequence of iterates $\boldsymbol{x}^0, \boldsymbol{x}^1, \boldsymbol{x}^2, \ldots$ that converge to $\boldsymbol{x}^\star$.
- The update rule optimizes the problem with respect to each block $\boldsymbol{x}_i$ sequentially.
- At each outer iteration k, the method executes n inner iterations sequentially:

$$\boldsymbol{x}_i^{k+1} = \arg\min_{\boldsymbol{x}_i \in \mathcal{X}_i} f\left(\boldsymbol{x}_1^{k+1}, \ldots, \boldsymbol{x}_{i-1}^{k+1}, \boldsymbol{x}_i, \boldsymbol{x}_{i+1}^{k}, \ldots, \boldsymbol{x}_n^{k}\right), \quad i = 1, \ldots, n.$$

Algorithm

Key points:

- Usefulness: Derive simple and practical algorithms.
- References: (Bertsekas 1999; Bertsekas and Tsitsiklis 1997; Beck 2017).

BCD for separable problems

Initialization:

- Choose initial point $\boldsymbol{x}^0 = (\boldsymbol{x}_1^0, \ldots, \boldsymbol{x}_n^0) \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$.
- Set $k \leftarrow 0$.

Repeat (kth iteration):

1. Execute n inner iterations sequentially:

$$\boldsymbol{x}_i^{k+1} = \arg\min_{\boldsymbol{x}_i \in \mathcal{X}_i} f\left(\boldsymbol{x}_1^{k+1}, \ldots, \boldsymbol{x}_{i-1}^{k+1}, \boldsymbol{x}_i, \boldsymbol{x}_{i+1}^k, \ldots, \boldsymbol{x}_n^k\right), \qquad i = 1, \ldots, n.$$

2. $k \leftarrow k + 1$

Until: convergence

Monotonicity:

$$f\left(\boldsymbol{x}^{k+1}\right) \leq f\left(\boldsymbol{x}^{k}\right).$$

Assumptions

- f is continuously differentiable over the convex closed set $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$.
- f is blockwise strictly convex in each block variable $\boldsymbol{x}_i$.

Convergence

Every limit point of the sequence $\{\boldsymbol{x}^k\}$ is a stationary point of the original problem.

References

(Bertsekas 1999; Bertsekas and Tsitsiklis 1997; Grippo and Sciandrone 2000).

BCD: Parallel Updates

Parallel Update (Jacobi Method)

- The method consists in executing n inner iterations in parallel instead of sequentially.
- The update rule is:

$$\boldsymbol{x}_i^{k+1} = \arg\min_{\boldsymbol{x}_i \in \mathcal{X}_i} f\left(\boldsymbol{x}_1^k, \ldots, \boldsymbol{x}_{i-1}^k, \boldsymbol{x}_i, \boldsymbol{x}_{i+1}^k, \ldots, \boldsymbol{x}_n^k\right), \quad i = 1, \ldots, n.$$

Jacobi Method

- The Jacobi method consists in a parallel update of the block variables.
- It is algorithmically attractive due to its potential for faster execution.

Convergence Properties

- The Jacobi method does not enjoy nice convergence properties.
- Convergence is guaranteed if the mapping defined by $T(\boldsymbol{x}) = \boldsymbol{x} - \gamma \nabla f(\boldsymbol{x})$ is a contraction for some γ.
- Reference: (Bertsekas 1999).

BCD Example: Soft-Thresholding Operator

Univariate Convex Optimization Problem

$$\underset{x}{\text{minimize}} \quad \frac{1}{2}\|\boldsymbol{a}x - \boldsymbol{b}\|_2^2 + \lambda|x|$$

Solution:

$$x = \frac{1}{\|\boldsymbol{a}\|_2^2}\text{sign}\left(\boldsymbol{a}^{\mathsf{T}}\boldsymbol{b}\right)\left(|\boldsymbol{a}^{\mathsf{T}}\boldsymbol{b}| - \lambda\right)^{+}$$

- Sign function:

$$\text{sign}(u) = \begin{cases} +1 & u > 0 \\ 0 & u = 0 \\ -1 & u < 0 \end{cases}$$

- Positive part function: $(\cdot)^{+} = \max(0, \cdot)$

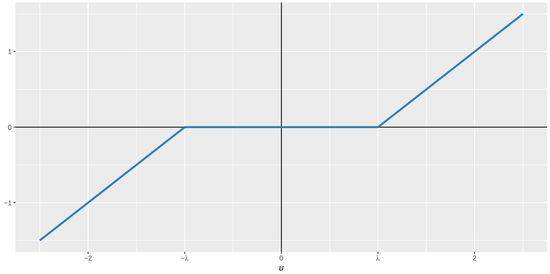
BCD Example: Soft-Thresholding Operator

Solution in compact form:

$$x = \frac{1}{\|\boldsymbol{a}\|_2^2} \mathcal{S}_\lambda \left(\boldsymbol{a}^\mathsf{T} \boldsymbol{b} \right)$$

Soft-thresholding operator:

$$\mathcal{S}_\lambda(u) = \text{sign}(u)(|u| - \lambda)^+$$



BCD Example: $\ell_2 - \ell_1$-Norm Minimization

Problem Formulation:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda\|\boldsymbol{x}\|_1$$

Solution Approach

- Can be solved with a QP solver.
- Iterative algorithm via BCD with soft-thresholding operator (Zibulevsky and Elad 2010).

BCD Method

- Variable partitioning: divide the variable into each constituent element $\boldsymbol{x} = (x_1, \ldots, x_n)$.
- Sequence of problems at each iteration $k = 0, 1, 2, \ldots$ for each element $i = 1, \ldots, n$:

$$\underset{x_i}{\text{minimize}} \quad \frac{1}{2}\left\|\boldsymbol{a}_i x_i - \tilde{\boldsymbol{b}}_i^k\right\|_2^2 + \lambda|x_i|$$

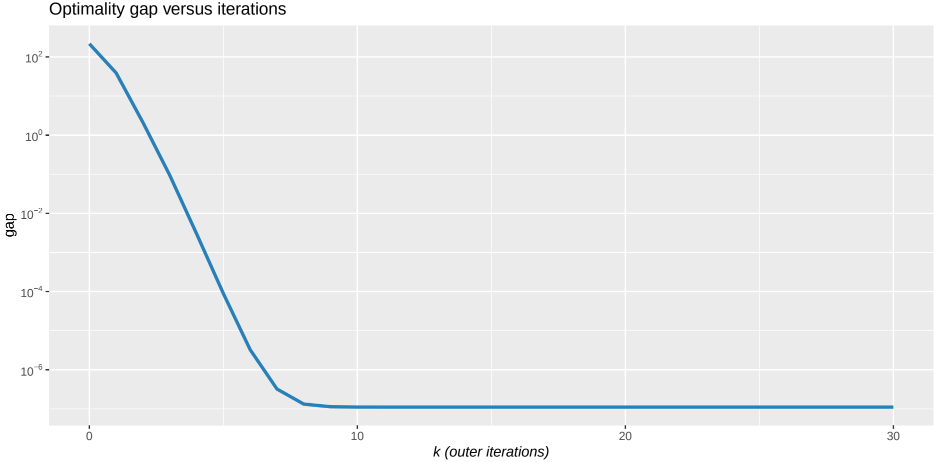
  where $\tilde{\boldsymbol{b}}_i^k \triangleq \boldsymbol{b} - \sum_{j<i} \boldsymbol{a}_j x_j^{k+1} - \sum_{j>i} \boldsymbol{a}_j x_j^k$.

Iterative Algorithm: For $k = 0, 1, 2, \ldots$

$$x_i^{k+1} = \frac{1}{\|\boldsymbol{a}_i\|_2^2}\mathcal{S}_\lambda\left(\boldsymbol{a}_i^\mathsf{T}\tilde{\boldsymbol{b}}_i^k\right), \quad i = 1, \ldots, n.$$

BCD Example: $\ell_2 - \ell_1$-Norm Minimization

Convergence of BCD for the $\ell_2 - \ell_1$-norm minimization:



Outline

Original Difficult Problem

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & f(\mathbf{x}) \\ \text{subject to} & \mathbf{x} \in \mathcal{X} \end{array}$$

where the objective function f and the feasible set $\mathcal{X}$ are possibly nonconvex.

Majorization-Minimization (MM) Method

- Approximates a difficult optimization problem by a sequence of simpler problems:

$$\mathbf{x}^{k+1} = \underset{\mathbf{x} \in \mathcal{X}}{\arg\min}\; u\left(\mathbf{x}; \mathbf{x}^k\right) \qquad k = 0, 1, 2, \ldots$$

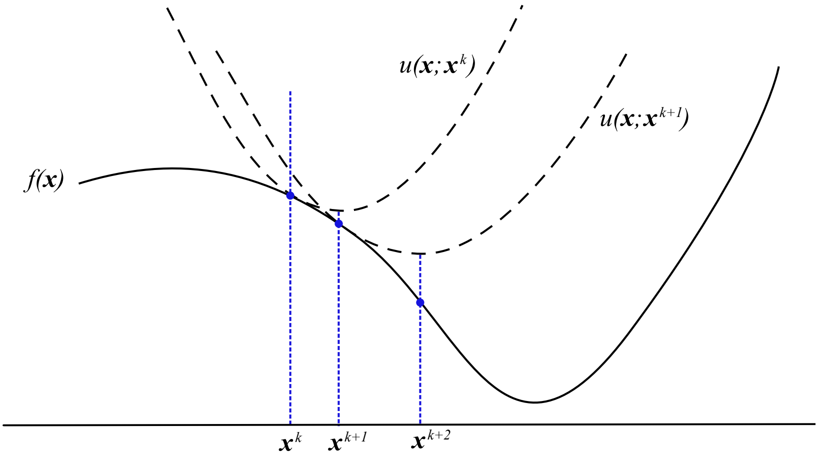
  where $u\left(\mathbf{x}; \mathbf{x}^k\right)$ is a surrogate function at iteration k that approximates $f(\mathbf{x})$ around the current point $\mathbf{x}^k$.
- Produces a sequence of iterates $\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2, \ldots$ that converge to $\mathbf{x}^\star$.

References

- Concise tutorial: (Hunter and Lange 2004).
- Long tutorial with applications: (Sun, Babu, and Palomar 2017).
- Convergence analysis: (Razaviyayn, Hong, and Luo 2013).

MM

Illustration of sequence of surrogate problems in MM:



MM algorithm

Initialization:

- Choose initial point $\boldsymbol{x}^0 \in \mathcal{X}$.
- Set $k \leftarrow 0$.

Repeat (kth iteration):

1. Construct majorizer of $f(\boldsymbol{x})$ around current point $\boldsymbol{x}^k$ as $u\left(\boldsymbol{x}; \boldsymbol{x}^k\right)$.
2. Obtain next iterate by solving the majorized problem:

$$\boldsymbol{x}^{k+1} = \underset{\boldsymbol{x} \in \mathcal{X}}{\arg\min}\; u\left(\boldsymbol{x}; \boldsymbol{x}^k\right).$$

3. $k \leftarrow k + 1$

Until: convergence

MM: Convergence

Conditions for the Surrogate Function:

- Upper-bound property: $u(\boldsymbol{x};\boldsymbol{x}^k) \geq f(\boldsymbol{x})$.
- Touching property: $u(\boldsymbol{x}^k;\boldsymbol{x}^k) = f(\boldsymbol{x}^k)$.
- Tangent property: $u(\boldsymbol{x};\boldsymbol{x}^k)$ must be differentiable with $\nabla u(\boldsymbol{x};\boldsymbol{x}^k) = \nabla f(\boldsymbol{x})$.

Properties

- Monotonicity: $f(\boldsymbol{x}^{k+1}) \leq f(\boldsymbol{x}^k)$.
- Convergence: If $\mathcal{X}$ is convex, every limit point of the sequence $\{\boldsymbol{x}^k\}$ is a stationary point of the original problem.
- If $\mathcal{X}$ is nonconvex, then convergence must be studied on a case-by-case basis, e.g., (Song, Babu, and Palomar 2015; Sun, Babu, and Palomar 2017; Kumar et al. 2019, 2020).

Majorizer Construction

- Objective: find an appropriate majorizer $u(\boldsymbol{x};\boldsymbol{x}^k)$ that satisfies the technical conditions and leads to a simpler surrogate problem.
- Techniques and examples: refer to (Sun, Babu, and Palomar 2017).

MM: Acceleration Techniques

MM Convergence Speed

- Issue: MM may require many iterations to converge if the surrogate function $u(\boldsymbol{x}; \boldsymbol{x}^k)$ is not tight enough.
- Reason: Strict global upper-bound requirement.

Acceleration Techniques

- Objective: Improve convergence speed.
- Popular technique: SQUAREM (Squared Iterative Methods for Accelerating EM-like Monotone Algorithms) (Varadhan and Roland 2008).

MM Example: Nonnegative LS

Problem Formulation

$$\underset{\boldsymbol{x}\geq\boldsymbol{0}}{\text{minimize}} \quad \frac{1}{2}\left\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\right\|_2^2$$

where the parameters are

- $\boldsymbol{b} \in \mathbb{R}_+^m$ (nonnegative elements)
- $\boldsymbol{A} \in \mathbb{R}_{++}^{m\times n}$ (positive elements)

Conventional LS Solution

Not applicable due to nonnegativity constraints: $\boldsymbol{x}^\star \neq (\boldsymbol{A}^\mathsf{T}\boldsymbol{A})^{-1}\boldsymbol{A}^\mathsf{T}\boldsymbol{b}$.

Alternative Approach

- Use a QP solver: Standard method.
- Develop an iterative algorithm based on MM: More interesting approach.

MM Example: Nonnegative LS

Objective Function:

$$f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 .$$

Majorizer

$$u\left(\boldsymbol{x}; \boldsymbol{x}^k\right) = f\left(\boldsymbol{x}^k\right) + \nabla f\left(\boldsymbol{x}^k\right)^\mathsf{T}\left(\boldsymbol{x} - \boldsymbol{x}^k\right) + \frac{1}{2}\left(\boldsymbol{x} - \boldsymbol{x}^k\right)^\mathsf{T}\boldsymbol{\Phi}\left(\boldsymbol{x}^k\right)\left(\boldsymbol{x} - \boldsymbol{x}^k\right)$$

- Gradient: $\nabla f\left(\boldsymbol{x}^k\right) = \boldsymbol{A}^\mathsf{T}\boldsymbol{A}\boldsymbol{x}^k - \boldsymbol{A}^\mathsf{T}\boldsymbol{b}$.
- Matrix $\boldsymbol{\Phi}$: $\boldsymbol{\Phi}\left(\boldsymbol{x}^k\right) = \mathrm{Diag}\left(\frac{\left[\boldsymbol{A}^\mathsf{T}\boldsymbol{A}\boldsymbol{x}^k\right]_1}{x_1^k}, \ldots, \frac{\left[\boldsymbol{A}^\mathsf{T}\boldsymbol{A}\boldsymbol{x}^k\right]_n}{x_n^k}\right)$.

Verification of Majorizer Properties

- Upper-bound property: $u(\boldsymbol{x}; \boldsymbol{x}^k) \geq f(\boldsymbol{x})$ (proved using Jensen's inequality)
- Touching property: $u(\boldsymbol{x}^k; \boldsymbol{x}^k) = f(\boldsymbol{x}^k)$
- Tangent property: $\nabla u(\boldsymbol{x}^k; \boldsymbol{x}^k) = \nabla f(\boldsymbol{x}^k)$

MM Example: Nonnegative LS

Sequence of Majorized Problems

$$\underset{\boldsymbol{x} \geq \boldsymbol{0}}{\text{minimize}} \quad \nabla f\left(\boldsymbol{x}^k\right)^{\mathsf{T}} \boldsymbol{x} + \tfrac{1}{2}\left(\boldsymbol{x} - \boldsymbol{x}^k\right)^{\mathsf{T}} \boldsymbol{\Phi}\left(\boldsymbol{x}^k\right)\left(\boldsymbol{x} - \boldsymbol{x}^k\right)$$

with solution

$$\boldsymbol{x} = \boldsymbol{x}^k - \boldsymbol{\Phi}\left(\boldsymbol{x}^k\right)^{-1} \nabla f\left(\boldsymbol{x}^k\right).$$

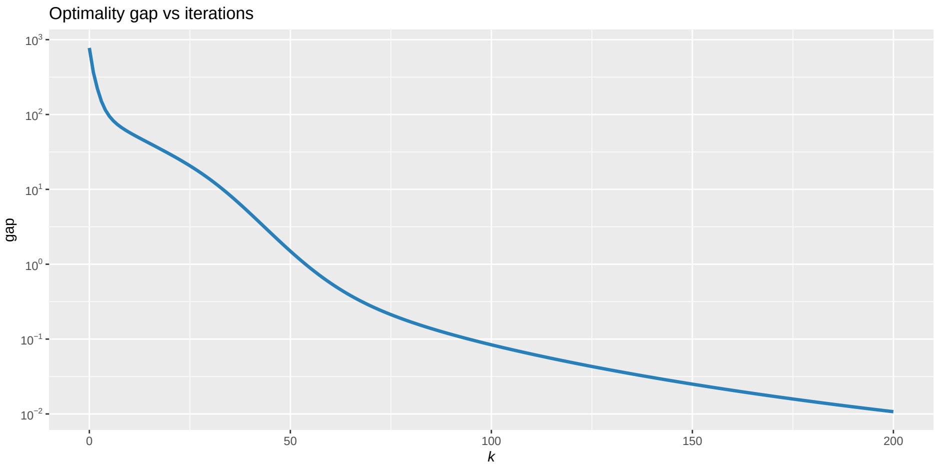
Iterative Update:

$$\boldsymbol{x}^{k+1} = \boldsymbol{c}^k \odot \boldsymbol{x}^k, \quad k = 0, 1, 2, \ldots$$

where $c_i^k = \frac{[\boldsymbol{A}^{\mathsf{T}}\boldsymbol{b}]_i}{[\boldsymbol{A}^{\mathsf{T}}\boldsymbol{A}\boldsymbol{x}^k]_i}$ and $\odot$ denotes the elementwise product.

MM Example: Nonnegative LS

Convergence of MM for nonnegative LS:



MM Example: $\ell_2 - \ell_1$-Norm Minimization

$\ell_2 - \ell_1$-Norm Minimization Problem:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda\|\boldsymbol{x}\|_1.$$

Solution Methods

- Can be solved via BCD or a QP solver.
- We will develop an iterative algorithm based on MM.

MM for $\ell_2 - \ell_1$-Norm Minimization

- The variable $\boldsymbol{x}$ is partitioned into elements $(x_1, \ldots, x_n)$.
- The majorizer function is:

$$u\left(\boldsymbol{x}; \boldsymbol{x}^k\right) = \frac{\kappa}{2}\|\boldsymbol{x} - \bar{\boldsymbol{x}}^k\|_2^2 + \lambda\|\boldsymbol{x}\|_1 + \text{constant},$$

  where $\bar{\boldsymbol{x}}^k = \boldsymbol{x}^k - \frac{1}{\kappa}\boldsymbol{A}^\mathsf{T}\left(\boldsymbol{A}\boldsymbol{x}^k - \boldsymbol{b}\right)$.
- Verification as a majorizer is given by:

$$u\left(\boldsymbol{x}; \boldsymbol{x}^k\right) = f(\boldsymbol{x}) + \text{dist}\left(\boldsymbol{x}, \boldsymbol{x}^k\right),$$

  where $\text{dist}\left(\boldsymbol{x}, \boldsymbol{x}^k\right) = \frac{\kappa}{2}\|\boldsymbol{x} - \boldsymbol{x}^k\|_2^2 - \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}\boldsymbol{x}^k\|_2^2$ and $\kappa > \lambda_{\max}\left(\boldsymbol{A}^\mathsf{T}\boldsymbol{A}\right)$.

MM Example: $\ell_2 - \ell_1$-Norm Minimization

Sequence of Majorized Problems: For $k = 0, 1, 2, \ldots$

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \tfrac{\kappa}{2}\|\boldsymbol{x} - \bar{\boldsymbol{x}}^k\|_2^2 + \lambda\|\boldsymbol{x}\|_1.$$

MM Iterative Algorithm:

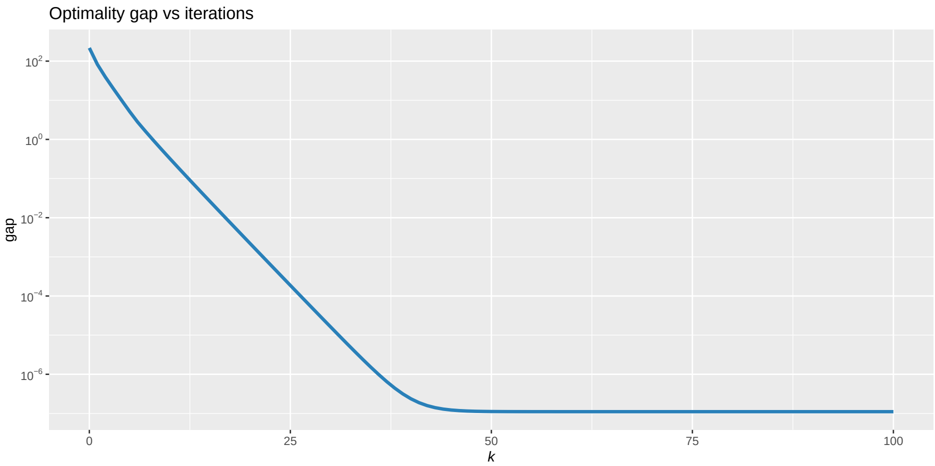
$$\boldsymbol{x}^{k+1} = \mathcal{S}_{\lambda/\kappa}\left(\bar{\boldsymbol{x}}^k\right), \qquad k = 0, 1, 2, \ldots$$

where $\mathcal{S}_{\lambda/\kappa}(\cdot)$ is the soft-thresholding operator:

$$\mathcal{S}_{\lambda/\kappa}(z) = \text{sign}(z)\max(|z| - \lambda/\kappa, 0).$$

MM Example: $\ell_2 - \ell_1$-Norm Minimization

Convergence of MM for $\ell_2 - \ell_1$-norm minimization:



Block MM: Combining BCD and MM

Objective

- Address situations where both the original problem and direct application of MM are too difficult to solve.

Approach

- Combine Block-Coordinate Descent (BCD) and Majorization-Minimization (MM).

Original Problem

$$\begin{array}{ll} \underset{\boldsymbol{x}}{\text{minimize}} & f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \\ \text{subject to} & \boldsymbol{x}_i \in \mathcal{X}_i, \quad i = 1, \ldots, n \end{array}$$

- Partitioning: Variables are partitioned into n blocks $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$.
- Constraints: Each block $\boldsymbol{x}_i$ is separately constrained.

Idea

- Solve the problem block by block as in BCD, but majorize each block $f(\boldsymbol{x}_i)$ with a surrogate function $u(\boldsymbol{x}_i; \boldsymbol{x}^k)$.

Block MM: Combining BCD and MM

Procedure

Initialization: Start with an initial guess $\boldsymbol{x}^0 = (\boldsymbol{x}_1^0, \ldots, \boldsymbol{x}_n^0)$.

Iterative process: For each outer iteration $k = 0, 1, 2, \ldots$:

- For each block $i = 1, \ldots, n$:
    - Majorize: Construct a surrogate function $u\left(\boldsymbol{x}_i; \boldsymbol{x}^k\right)$ for the block $f(\boldsymbol{x}_i)$.
    - Update: Solve the majorized problem for the block:

$$\boldsymbol{x}_i^{k+1} = \underset{\boldsymbol{x}_i \in \mathcal{X}_i}{\arg\min}\ u\left(\boldsymbol{x}_i; \boldsymbol{x}^k\right).$$

- Update the full variable: $\boldsymbol{x}^{k+1} = (\boldsymbol{x}_1^{k+1}, \ldots, \boldsymbol{x}_n^{k+1})$.

References

(Razaviyayn, Hong, and Luo 2013) (Sun, Babu, and Palomar 2017).

Outline

Successive Convex Approximation (SCA)

Original Difficult Problem:

$$\begin{array}{ll} \underset{\boldsymbol{x}}{\text{minimize}} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in \mathcal{X} \end{array}$$

where f is the (possibly nonconvex) objective function and $\mathcal{X}$ is the convex feasible set.

Successive Convex Approximation (SCA) Method (Scutari et al. 2014):

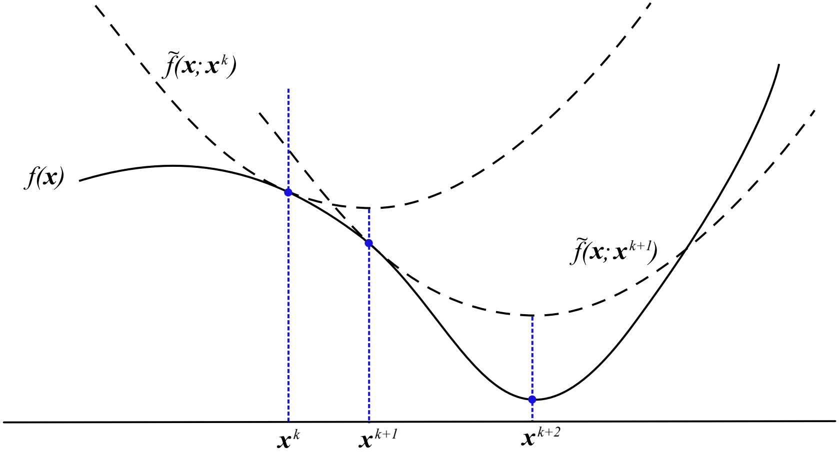
- Approximates a difficult optimization problem by a sequence of simpler problems:

$$\boldsymbol{x}^{k+1} = \arg\min_{\boldsymbol{x}\in\mathcal{X}} \tilde{f}\left(\boldsymbol{x}; \boldsymbol{x}^k\right), \qquad k = 0, 1, 2, \ldots$$

- The function $\tilde{f}\left(\boldsymbol{x}; \boldsymbol{x}^k\right)$ is a surrogate function at iteration k that approximates $f(\boldsymbol{x})$ around the current point $\boldsymbol{x}^k$.
- The method produces a sequence of iterates $\boldsymbol{x}^0, \boldsymbol{x}^1, \boldsymbol{x}^2, \ldots$ that converge to $\boldsymbol{x}^\star$.
- For convergence, a smoothing step is needed to avoid oscillations ($\gamma^k \in (0, 1]$):

$$\left.\begin{aligned} \hat{\boldsymbol{x}}^{k+1} &= \arg\min_{\boldsymbol{x}\in\mathcal{X}} \tilde{f}\left(\boldsymbol{x}; \boldsymbol{x}^k\right) \\ \boldsymbol{x}^{k+1} &= \boldsymbol{x}^k + \gamma^k\left(\hat{\boldsymbol{x}}^{k+1} - \boldsymbol{x}^k\right) \end{aligned}\right. \qquad k = 0, 1, 2, \ldots$$

Illustration of sequence of surrogate problems in SCA:



SCA: Convergence

Conditions for the Surrogate Function $\tilde{f}(\boldsymbol{x}; \boldsymbol{x}^k)$

- Must be strongly convex on the feasible set $\mathcal{X}$.
- Must be differentiable with $\nabla \tilde{f}(\boldsymbol{x}; \boldsymbol{x}^k) = \nabla f(\boldsymbol{x})$.

Stepsize Rules for $\{\gamma^k\}$

- *Bounded stepsize*: γ^k values are sufficiently small (difficult to use in practice).
- *Backtracking line search*: Effective in terms of iterations but costly.
- *Diminishing stepsize*: satisfying $\sum_{k=1}^{\infty} \gamma^k = +\infty$ and $\sum_{k=1}^{\infty} (\gamma^k)^2 < +\infty$.
  - Example 1: $\gamma^{k+1} = \gamma^k \left(1 - \epsilon\gamma^k\right)$, $\gamma^0 < 1/\epsilon$, $\epsilon \in (0, 1)$.
  - Example 2: $\gamma^{k+1} = \frac{\gamma^k + \alpha^k}{1+\beta^k}$, $\gamma^0 = 1$, α^k and β^k satisfy $0 \leq \alpha^k \leq \beta^k$ and $\alpha^k/\beta^k \to 0$.

Examples of α^k and β^k

- $\alpha^k = \alpha$ or $\alpha^k = \log(k)^{\alpha}$.
- $\beta^k = \beta k$ or $\beta^k = \beta\sqrt{k}$.
- Constants $\alpha \in (0, 1)$, $\beta \in (0, 1)$, and $\alpha \leq \beta$.

Advantages of SCA

- Surrogate function is convex by construction.
- Easier to construct a convex surrogate function compared to MM.

Algorithm

SCA algorithm

Initialization:

- Choose initial point $\boldsymbol{x}^0 \in \mathcal{X}$, sequence $\{\gamma^k\}$, and set $k \leftarrow 0$.

Repeat (kth iteration):

1. Construct surrogate of $f(\boldsymbol{x})$ around current point $\boldsymbol{x}^k$ as $\tilde{f}\left(\boldsymbol{x};\boldsymbol{x}^k\right)$.
2. Obtain intermediate point by solving the surrogate convex problem:
$$\hat{\boldsymbol{x}}^{k+1} = \arg\min_{\boldsymbol{x}\in\mathcal{X}} \tilde{f}\left(\boldsymbol{x};\boldsymbol{x}^k\right).$$
3. Obtain next iterate by averaging the intermediate point with the previous one:
$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \gamma^k\left(\hat{\boldsymbol{x}}^{k+1} - \boldsymbol{x}^k\right).$$
4. $k \leftarrow k+1$

Until: convergence

Gradient Descent Method as SCA

- Unconstrained problem

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad f(\boldsymbol{x}),$$

- One possible SCA surrogate function is

$$\tilde{f}\left(\boldsymbol{x};\boldsymbol{x}^k\right) = f\left(\boldsymbol{x}^k\right) + \nabla f\left(\boldsymbol{x}^k\right)^\mathsf{T}\left(\boldsymbol{x}-\boldsymbol{x}^k\right) + \frac{1}{2\alpha^k}\|\boldsymbol{x}-\boldsymbol{x}^k\|^2.$$

- To minimize the surrogate function, set its gradient to zero:

$$\nabla\tilde{f}\left(\boldsymbol{x};\boldsymbol{x}^k\right) = \nabla f\left(\boldsymbol{x}^k\right) + \frac{1}{\alpha^k}(\boldsymbol{x}-\boldsymbol{x}^k) = 0.$$

- Solving for $\boldsymbol{x}$ yields:

$$\boldsymbol{x} = \boldsymbol{x}^k - \alpha^k\nabla f\left(\boldsymbol{x}^k\right).$$

- The resulting iteration process coincides with the gradient descent method:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \alpha^k\nabla f\left(\boldsymbol{x}^k\right), \qquad k = 0, 1, 2, \ldots.$$

Newton's Method as SCA

- By including second-order information, the surrogate function becomes:

$$\tilde{f}\left(\boldsymbol{x};\boldsymbol{x}^k\right)=f\left(\boldsymbol{x}^k\right)+\nabla f\left(\boldsymbol{x}^k\right)^{\mathsf{T}}\left(\boldsymbol{x}-\boldsymbol{x}^k\right)+\frac{1}{2\alpha^k}\left(\boldsymbol{x}-\boldsymbol{x}^k\right)^{\mathsf{T}}\nabla^2 f\left(\boldsymbol{x}^k\right)\left(\boldsymbol{x}-\boldsymbol{x}^k\right).$$

- To minimize the surrogate function, set its gradient to zero:

$$\nabla\tilde{f}\left(\boldsymbol{x};\boldsymbol{x}^k\right)=\nabla f\left(\boldsymbol{x}^k\right)+\frac{1}{\alpha^k}\nabla^2 f\left(\boldsymbol{x}^k\right)(\boldsymbol{x}-\boldsymbol{x}^k)=0.$$

- Solving for $\boldsymbol{x}$ yields:

$$\boldsymbol{x}=\boldsymbol{x}^k-\alpha^k\nabla^2 f\left(\boldsymbol{x}^k\right)^{-1}\nabla f\left(\boldsymbol{x}^k\right).$$

- The resulting iteration process coincides with Newton's method:

$$\boldsymbol{x}^{k+1}=\boldsymbol{x}^k-\alpha^k\nabla^2 f\left(\boldsymbol{x}^k\right)^{-1}\nabla f\left(\boldsymbol{x}^k\right),\qquad k=0,1,2,\ldots.$$

Parallel SCA

Partitioned Variables in SCA

$$\begin{array}{lll} \underset{\boldsymbol{x}}{\text{minimize}} & f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) & \\ \text{subject to} & \boldsymbol{x}_i \in \mathcal{X}_i, & i = 1, \ldots, n, \end{array}$$

with variables are partitioned into n separate blocks: $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$.

Parallel Updates in SCA

- Unlike BCD or MM, SCA updates variables in parallel with surrogates $\tilde{f}_i(\boldsymbol{x}_i; \boldsymbol{x}^k)$.
- The update process for each block i is:

$$\left.\begin{aligned} \hat{\boldsymbol{x}}_i^{k+1} &= \underset{\boldsymbol{x}_i \in \mathcal{X}_i}{\arg\min}\ \tilde{f}_i\left(\boldsymbol{x}_i; \boldsymbol{x}^k\right) \\ \boldsymbol{x}_i^{k+1} &= \boldsymbol{x}_i^k + \gamma^k \left(\hat{\boldsymbol{x}}_i^{k+1} - \boldsymbol{x}_i^k\right) \end{aligned}\right. \qquad i = 1, \ldots, n, \quad k = 0, 1, 2, \ldots$$

  where $\{\gamma^k\}$ is a properly designed sequence with $\gamma^k \in (0, 1]$.

Advantages of Parallel Updates

- Efficiently handles large-scale problems by updating multiple variables simultaneously.
- Reduces computational time compared to sequential updates in BCD or block MM.

SCA: Convergence*

Technical Conditions for the Surrogate Function (Scutari et al. 2014)

- Must be strongly convex on the feasible set $\mathcal{X}$.
- Must be differentiable with $\nabla \tilde{f}(\boldsymbol{x}; \boldsymbol{x}^k) = \nabla f(\boldsymbol{x})$.

Stepsize Rules for $\{\gamma^k\}$

- *Bounded stepsize*: γ^k values are sufficiently small.
- *Backtracking line search*: effective but requires multiple evaluations per iteration.
- *Diminishing stepsize*: satisfying $\sum_{k=1}^{\infty} \gamma^k = +\infty$ and $\sum_{k=1}^{\infty} (\gamma^k)^2 < +\infty$.

Theoretical Convergence

- SCA enjoys strong theoretical convergence properties.
- Convergence results are detailed in (Scutari et al. 2014).

Convergence of SCA

- Suppose the surrogate function $\tilde{f}\left(\boldsymbol{x}; \boldsymbol{x}^k\right)$ (or each $\tilde{f}_i\left(\boldsymbol{x}_i; \boldsymbol{x}^k\right)$ in the parallel version) satisfies the required technical conditions.
- If $\{\gamma^k\}$ is chosen according to the bounded stepsize, diminishing rule, or backtracking line search, then the sequence $\{\boldsymbol{x}^k\}$ converges to a stationary point of the original problem.

SCA Example: $\ell_2 - \ell_1$-Norm Minimization

$\ell_2 - \ell_1$-Norm Minimization Problem:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \tfrac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda\|\boldsymbol{x}\|_1.$$

Solution Methods

- Can be solved via BCD, MM, or a QP solver.
- We will develop an iterative algorithm based on SCA.

Parallel SCA for $\ell_2 - \ell_1$-Norm Minimization

- The variable $\boldsymbol{x}$ is partitioned into elements $(x_1, \ldots, x_n)$.
- Surrogate functions:

$$\tilde{f}\left(\boldsymbol{x}_i; \boldsymbol{x}^k\right) = \frac{1}{2}\left\|\boldsymbol{a}_i x_i - \tilde{\boldsymbol{b}}_i^k\right\|_2^2 + \lambda|x_i| + \frac{\tau}{2}\left(x_i - x_i^k\right)^2,$$

where $\tilde{\boldsymbol{b}}_i^k = \boldsymbol{b} - \sum_{j \neq i} \boldsymbol{a}_j x_j^k$.

Sequence of surrogate problems: For $k = 0, 1, 2, \ldots$ and $i = 1, \ldots, n$:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \tfrac{1}{2}\left\|\boldsymbol{a}_i x_i - \tilde{\boldsymbol{b}}_i^k\right\|_2^2 + \lambda|x_i| + \tau\left(x_i - x_i^k\right)^2$$

SCA Example: $\ell_2 - \ell_1$-Norm Minimization

SCA iterative algorithm:

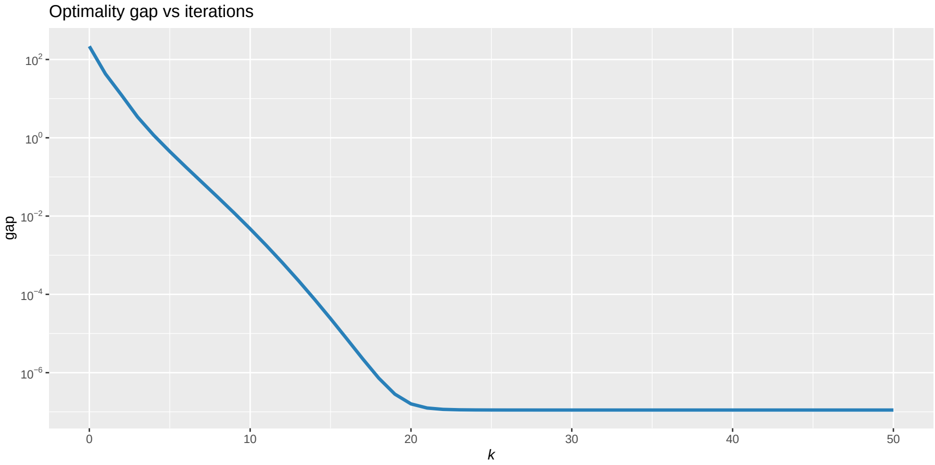
$$\begin{aligned} \hat{x}_i^{k+1} &= \frac{1}{\tau + \|\boldsymbol{a}_i\|^2} \mathcal{S}_\lambda \left(\boldsymbol{a}_i^\mathsf{T} \tilde{\boldsymbol{b}}_i^k + \tau x_i^k \right) \\ x_i^{k+1} &= x_i^k + \gamma^k \left(\hat{x}_i^{k+1} - x_i^k \right) \end{aligned} \qquad i = 1, \ldots, n, \quad k = 0, 1, 2, \ldots$$

where $\mathcal{S}_\lambda(\cdot)$ is the soft-thresholding operator:

$$\mathcal{S}_\lambda(z) = \mathrm{sign}(z) \max(|z| - \lambda, 0).$$

SCA Example: $\ell_2 - \ell_1$-Norm Minimization

Convergence of SCA for the $\ell_2 - \ell_1$-norm minimization:



SCA Example: Dictionary Learning

Dictionary Learning Problem

$$\begin{array}{ll} \underset{\boldsymbol{D},\boldsymbol{X}}{\text{minimize}} & \frac{1}{2}\|\boldsymbol{Y}-\boldsymbol{D}\boldsymbol{X}\|_F^2+\lambda\|\boldsymbol{X}\|_1 \\ \text{subject to} & \|[\boldsymbol{D}]_{:,i}\|\leq 1, \qquad i=1,\ldots,m. \end{array}$$

- $\|\boldsymbol{D}\|_F$: frobenius norm of $\boldsymbol{D}$
- $\|\boldsymbol{X}\|_1$: elementwise ℓ_1-norm of $\boldsymbol{X}$

Matrix Definitions

- $\boldsymbol{D}$: dictionary matrix (fat matrix with columns explaining the columns of $\boldsymbol{Y}$).
- $\boldsymbol{X}$: sparse matrix selecting a few columns of the dictionary.

Bi-Convex Nature

- The problem is not jointly convex in $(\boldsymbol{D},\boldsymbol{X})$, but it is bi-convex.
- For fixed $\boldsymbol{D}$, the problem is convex in $\boldsymbol{X}$.
- For fixed $\boldsymbol{X}$, the problem is convex in $\boldsymbol{D}$.

Solution Methods

- BCD: updates $\boldsymbol{D}$ and $\boldsymbol{X}$ sequentially.
- SCA: allows parallel updates of $\boldsymbol{D}$ and $\boldsymbol{X}$.

SCA Example: Dictionary Learning

SCA Approach: Surrogate functions:

$$\tilde{f}_1\left(\boldsymbol{D};\boldsymbol{X}^k\right) = \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{D}\boldsymbol{X}^k\|_F^2$$
$$\tilde{f}_2\left(\boldsymbol{X};\boldsymbol{D}^k\right) = \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{D}^k\boldsymbol{X}\|_F^2$$

Resulting Convex Problems

- Normalized least squares (LS) problem for $\boldsymbol{D}$:

$$\begin{array}{ll} \underset{\boldsymbol{D}}{\text{minimize}} & \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{D}\boldsymbol{X}^k\|_F^2 \\ \text{subject to} & \|[\boldsymbol{D}]_{:,i}\| \leq 1, \qquad i = 1,\ldots,m \end{array}$$

- Matrix version of the $\ell_2 - \ell_1$-norm problem for $\boldsymbol{X}$:

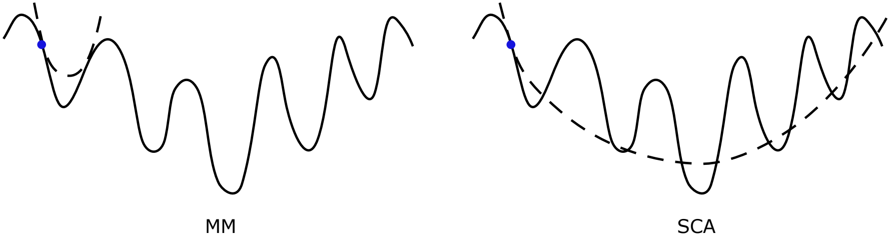
$$\underset{\boldsymbol{X}}{\text{minimize}} \quad \tfrac{1}{2}\|\boldsymbol{Y} - \boldsymbol{D}^k\boldsymbol{X}\|_F^2 + \lambda\|\boldsymbol{X}\|_1$$

  which can be further decomposed into a set of vectorized $\ell_2 - \ell_1$-norm problems for each column of $\boldsymbol{X}$.

MM vs. SCA

Surrogate Function

- MM (Majorization-Minimization):
  - Requires the surrogate function to be a global upper bound.
  - The surrogate function need not be convex.
  - Can be difficult to derive and too restrictive in some cases.
- SCA (Successive Convex Approximation):
  - Relaxes the upper-bound condition.
  - Requires the surrogate function to be strongly convex.



MM vs. SCA

Constraint Set: In principle, both require the feasible set $\mathcal{X}$ to be convex.

- MM:
  - Convergence can be extended to nonconvex $\mathcal{X}$ on a case-by-case basis.
  - Examples of nonconvex $\mathcal{X}$ handled by MM: (Song, Babu, and Palomar 2015; Sun, Babu, and Palomar 2017; Kumar et al. 2019, 2020).
- SCA:
  - Cannot directly handle nonconvex $\mathcal{X}$ (Scutari et al. 2014).
  - Some extensions allow for successive convexification of $\mathcal{X}$, but at the expense of a more complex algorithm (Scutari and Sun 2018).

Schedule of Updates: Both can handle separable variables $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$.

- MM:
  - Requires a sequential update for block variables (Razaviyayn, Hong, and Luo 2013; Sun, Babu, and Palomar 2017).
- SCA:
  - Naturally implements a parallel update, which is more amenable for distributed implementations.

Outline

Alternating Direction Method of Multipliers (ADMM)

- Practical algorithm resembling BCD but can handle coupled block variables in constraints.
- Detailed in (S. Boyd et al. 2010) and (Beck 2017).

Convex Optimization Problem

$$
\begin{array}{ll}
\underset{\boldsymbol{x},\boldsymbol{z}}{\text{minimize}} & f(\boldsymbol{x}) + g(\boldsymbol{z}) \\
\text{subject to} & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} = \boldsymbol{c},
\end{array}
$$

Observe that the variables $\boldsymbol{x}$ and $\boldsymbol{z}$ are coupled via the constraint $\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} = \boldsymbol{c}$.

First Attempt: Dual Ascent Method

First attempt to decouple the variables.

Dual Ascent Method

- Updates dual variable $\boldsymbol{y}$ via gradient method.
- Solves Lagrangian for given $\boldsymbol{y}$:

$$\underset{\boldsymbol{x},\boldsymbol{z}}{\text{minimize}} \quad L(\boldsymbol{x},\boldsymbol{z};\boldsymbol{y}) \triangleq f(\boldsymbol{x}) + g(\boldsymbol{z}) + \boldsymbol{y}^{\mathsf{T}}(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{c})$$

- Decouples into two separate problems over $\boldsymbol{x}$ and $\boldsymbol{z}$:

$$\begin{aligned}
\boldsymbol{x}^{k+1} &= \arg\min_{\boldsymbol{x}} \; f(\boldsymbol{x}) + (\boldsymbol{y}^k)^{\mathsf{T}}\boldsymbol{A}\boldsymbol{x} \\
\boldsymbol{z}^{k+1} &= \arg\min_{\boldsymbol{z}} \; g(\boldsymbol{z}) + (\boldsymbol{y}^k)^{\mathsf{T}}\boldsymbol{B}\boldsymbol{z} \qquad\qquad k = 0,1,2,\ldots \\
\boldsymbol{y}^{k+1} &= \boldsymbol{y}^k + \alpha^k\left(\boldsymbol{A}\boldsymbol{x}^{k+1} + \boldsymbol{B}\boldsymbol{z}^{k+1} - \boldsymbol{c}\right)
\end{aligned}$$

- Requires many technical assumptions and is often slow.

Second Attempt: Method of Multipliers

Second attempt to decouple the variables.

Method of Multipliers

- Uses the augmented Lagrangian:

$$L_\rho(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{y}) \triangleq f(\boldsymbol{x}) + g(\boldsymbol{z}) + \boldsymbol{y}^\mathsf{T}(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{c}) + \frac{\rho}{2}\|\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{c}\|_2^2$$

- The algorithm is:

$$\begin{aligned} \left(\boldsymbol{x}^{k+1}, \boldsymbol{z}^{k+1}\right) &= \arg\min_{\boldsymbol{x},\boldsymbol{z}} \; L_\rho(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{y}^k) \\ \boldsymbol{y}^{k+1} &= \boldsymbol{y}^k + \rho\left(\boldsymbol{A}\boldsymbol{x}^{k+1} + \boldsymbol{B}\boldsymbol{z}^{k+1} - \boldsymbol{c}\right) \end{aligned} \qquad k = 0, 1, 2, \ldots$$

- It converges under more relaxed conditions but cannot decouple $\boldsymbol{x}$ and $\boldsymbol{z}$ due to the $\|\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{c}\|_2^2$ term.

Third Attempt: ADMM

Third and final attempt to decouple the variables.

Alternating Direction Method of Multipliers (ADMM)

- Combines features of dual decomposition and the method of multipliers.
- Minimizes the augmented Lagrangian with a BCD-like method:

$$\begin{aligned} \boldsymbol{x}^{k+1} &= \arg\min_{\boldsymbol{x}} \; L_\rho(\boldsymbol{x}, \boldsymbol{z}^k; \boldsymbol{y}^k) \\ \boldsymbol{z}^{k+1} &= \arg\min_{\boldsymbol{z}} \; L_\rho(\boldsymbol{x}^{k+1}, \boldsymbol{z}; \boldsymbol{y}^k) \qquad\qquad k = 0, 1, 2, \ldots \\ \boldsymbol{y}^{k+1} &= \boldsymbol{y}^k + \rho\left(\boldsymbol{A}\boldsymbol{x}^{k+1} + \boldsymbol{B}\boldsymbol{z}^{k+1} - \boldsymbol{c}\right) \end{aligned}$$

- Successfully decouples the primal variables $\boldsymbol{x}$ and $\boldsymbol{z}$.
- Achieves faster convergence with fewer technical conditions.

Scaled Dual Variable Form

It is common to express the ADMM updates using the scaled dual variable $\boldsymbol{u}^k = \boldsymbol{y}^k/\rho$.

ADMM algorithm

Initialization:

- Choose initial point $(\boldsymbol{x}^0, \boldsymbol{z}^0)$, ρ, and set $k \leftarrow 0$.

Repeat (kth iteration):

1. Iterate primal and dual variables:

$$\begin{aligned}
\boldsymbol{x}^{k+1} &= \arg\min_{\boldsymbol{x}} \; f(\boldsymbol{x}) + \frac{\rho}{2} \left\| \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z}^k - \boldsymbol{c} + \boldsymbol{u}^k \right\|_2^2 \\
\boldsymbol{z}^{k+1} &= \arg\min_{\boldsymbol{z}} \; g(\boldsymbol{z}) + \frac{\rho}{2} \left\| \boldsymbol{A}\boldsymbol{x}^{k+1} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{c} + \boldsymbol{u}^k \right\|_2^2 \\
\boldsymbol{u}^{k+1} &= \boldsymbol{u}^k + \left(\boldsymbol{A}\boldsymbol{x}^{k+1} + \boldsymbol{B}\boldsymbol{z}^{k+1} - \boldsymbol{c} \right);
\end{aligned}$$

2. $k \leftarrow k + 1$

Until: convergence

ADMM: Convergence*

Assumptions

- $f(\boldsymbol{x})$ and $g(\boldsymbol{z})$ are convex.
- Both the $\boldsymbol{x}$-update and the $\boldsymbol{z}$-update are solvable.
- The Lagrangian has a saddle point.

Convergence of ADMM

- Residual convergence (feasibility convergence of iterates): $\boldsymbol{A}\boldsymbol{x}^k + \boldsymbol{B}\boldsymbol{z}^k - \boldsymbol{c} \to \boldsymbol{0}$ as $k \to \infty$.
- Objective convergence: $f(\boldsymbol{x}) + g(\boldsymbol{z}) \to p^\star$ as $k \to \infty$.
- Dual variable convergence: $\boldsymbol{y}^k \to \boldsymbol{y}^\star$ as $k \to \infty$.
- Detailed analysis in (S. Boyd et al. 2010) and references therein.

Practical Considerations

- The sequences $\{\boldsymbol{x}^k\}$ and $\{\boldsymbol{z}^k\}$ need not converge to optimal values without additional assumptions.
- ADMM can be slow to converge to high accuracy.
- It often converges to modest accuracy within a few tens of iterations, which is sufficient for many practical applications.

ADMM Example: Constrained Convex Optimization

Generic Convex Optimization Problem:

$$\begin{array}{ll} \underset{\boldsymbol{x}}{\text{minimize}} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in \mathcal{X}, \end{array}$$

where f is convex and $\mathcal{X}$ is a convex set.

Using ADMM to Transform the Problem

- Define g as the indicator function of the feasible set $\mathcal{X}$:

$$g(\boldsymbol{x}) \triangleq \begin{cases} 0 & \boldsymbol{x} \in \mathcal{X} \\ +\infty & \text{otherwise}, \end{cases}$$

- Formulate the equivalent problem:

$$\begin{array}{ll} \underset{\boldsymbol{x},\boldsymbol{z}}{\text{minimize}} & f(\boldsymbol{x}) + g(\boldsymbol{z}) \\ \text{subject to} & \boldsymbol{x} - \boldsymbol{z} = \mathbf{0}. \end{array}$$

ADMM Example: Constrained Convex Optimization

ADMM Algorithm for the Transformed Problem: The update rules are:

$$\begin{aligned} \boldsymbol{x}^{k+1} &= \arg\min_{\boldsymbol{x}} \; f(\boldsymbol{x}) + \frac{\rho}{2}\left\|\boldsymbol{x} - \boldsymbol{z}^k + \boldsymbol{u}^k\right\|_2^2 \\ \boldsymbol{z}^{k+1} &= \left[\boldsymbol{x}^{k+1} + \boldsymbol{u}^k\right]_{\mathcal{X}} \\ \boldsymbol{u}^{k+1} &= \boldsymbol{u}^k + \left(\boldsymbol{x}^{k+1} - \boldsymbol{z}^{k+1}\right) \end{aligned} \qquad k = 0, 1, 2, \ldots$$

where $[\cdot]_{\mathcal{X}}$ denotes projection on the set $\mathcal{X}$.

Explanation of Steps

- The $\boldsymbol{x}$-update minimizes $f(\boldsymbol{x})$ with a quadratic penalty term.
- The $\boldsymbol{z}$-update projects $\boldsymbol{x}^{k+1} + \boldsymbol{u}^k$ onto the set $\mathcal{X}$.
- The $\boldsymbol{u}$-update updates the scaled dual variable $\boldsymbol{u}$.

Benefits of this Approach

- Transforms a constrained optimization problem into an unconstrained one.
- Leverages the efficiency of ADMM for solving the problem.
- Allows for the use of projection operations to handle constraints.

$\ell_2 - \ell_1$-Norm Minimization Problem:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda\|\boldsymbol{x}\|_1.$$

Reformulated Problem for ADMM:

$$\begin{aligned} \underset{\boldsymbol{x},\boldsymbol{z}}{\text{minimize}} \quad & \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda\|\boldsymbol{z}\|_1 \\ \text{subject to} \quad & \boldsymbol{x} - \boldsymbol{z} = \boldsymbol{0}. \end{aligned}$$

ADMM Example: $\ell_2 - \ell_1$-Norm Minimization

ADMM Algorithm

- The $\boldsymbol{x}$-update solves the unconstrained QP:
$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \frac{\rho}{2}\|\boldsymbol{x} - \boldsymbol{z} + \boldsymbol{u}\|_2^2,$$
which has the solution:
$$\boldsymbol{x} = \left(\boldsymbol{A}^\mathsf{T}\boldsymbol{A} + \rho\boldsymbol{I}\right)^{-1}\left(\boldsymbol{A}^\mathsf{T}\boldsymbol{b} + \rho(\boldsymbol{z} - \boldsymbol{u})\right).$$
- The $\boldsymbol{z}$-update solves:
$$\underset{\boldsymbol{z}}{\text{minimize}} \quad \frac{\rho}{2}\|\boldsymbol{x} - \boldsymbol{z} + \boldsymbol{u}\|_2^2 + \lambda\|\boldsymbol{z}\|_1,$$
which has the solution using the soft-thresholding operator $\mathcal{S}_{\lambda/\rho}(\cdot)$:
$$\boldsymbol{z} = \mathcal{S}_{\lambda/\rho}(\boldsymbol{x} + \boldsymbol{u}).$$
- The $\boldsymbol{u}$-update for the scaled dual variable is:
$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \left(\boldsymbol{x}^{k+1} - \boldsymbol{z}^{k+1}\right).$$

ADMM Example: $\ell_2 - \ell_1$-Norm Minimization

ADMM Iterative Algorithm

The update rules are:

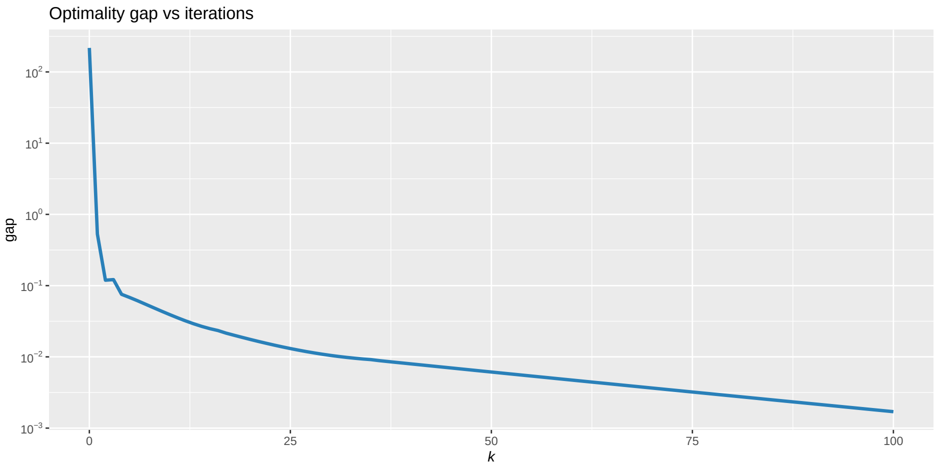
$$
\begin{aligned}
\boldsymbol{x}^{k+1} &= \left(\boldsymbol{A}^\mathsf{T}\boldsymbol{A} + \rho\boldsymbol{I}\right)^{-1}\left(\boldsymbol{A}^\mathsf{T}\boldsymbol{b} + \rho\left(\boldsymbol{z}^k - \boldsymbol{u}^k\right)\right) \\
\boldsymbol{z}^{k+1} &= \mathcal{S}_{\lambda/\rho}\left(\boldsymbol{x}^{k+1} + \boldsymbol{u}^k\right) \qquad\qquad k = 0, 1, 2, \ldots \\
\boldsymbol{u}^{k+1} &= \boldsymbol{u}^k + \left(\boldsymbol{x}^{k+1} - \boldsymbol{z}^{k+1}\right)
\end{aligned}
$$

where $\mathcal{S}_{\lambda/\rho}(z)$ is the soft-thresholding operator:

$$
\mathcal{S}_{\lambda/\rho}(z) = \operatorname{sign}(z)\max(|z| - \lambda/\rho, 0).
$$

ADMM Example: $\ell_2 - \ell_1$-Norm Minimization

Convergence of ADMM for the $\ell_2 - \ell_1$-norm minimization:



Outline

$\ell_2 - \ell_1$**-Norm Minimization Problem:**

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda\|\boldsymbol{x}\|_1$$

Iterates of Various Algorithms:

- **BCD (Gauss-Seidel) iterates:**

$$\boldsymbol{x}^{k+1} = \mathcal{S}_{\frac{\lambda}{\text{diag}(\boldsymbol{A}^T\boldsymbol{A})}}\left(\boldsymbol{x}^k - \frac{\boldsymbol{A}^T\left(\boldsymbol{A}\boldsymbol{x}^{(k,i)} - \boldsymbol{b}\right)}{\text{diag}\left(\boldsymbol{A}^T\boldsymbol{A}\right)}\right), \qquad i = 1, \ldots, n, \quad k = 0, 1, 2, \ldots$$

  where $\boldsymbol{x}^{(k,i)} \triangleq \left(x_1^{k+1}, \ldots, x_{i-1}^{k+1}, x_i^k, \ldots, x_n^k\right)$

- **Parallel BCD (Jacobi) iterates:**

$$\boldsymbol{x}^{k+1} = \mathcal{S}_{\frac{\lambda}{\text{diag}(\boldsymbol{A}^T\boldsymbol{A})}}\left(\boldsymbol{x}^k - \frac{\boldsymbol{A}^T\left(\boldsymbol{A}\boldsymbol{x}^k - \boldsymbol{b}\right)}{\text{diag}\left(\boldsymbol{A}^T\boldsymbol{A}\right)}\right), \qquad i = 1, \ldots, n, \quad k = 0, 1, 2, \ldots$$

Numerical Comparison

Iterates of Various Algorithms (cont'd):

- **MM iterates:**

$$\boldsymbol{x}^{k+1} = \mathcal{S}_{\frac{\lambda}{\kappa}}\left(\boldsymbol{x}^k - \frac{1}{\kappa}\boldsymbol{A}^\mathsf{T}\left(\boldsymbol{A}\boldsymbol{x}^k - \boldsymbol{b}\right)\right), \qquad k = 0, 1, 2, \ldots.$$

- **Accelerated MM iterates:**

$$\begin{aligned}
\boldsymbol{r}^k &= R(\boldsymbol{x}^k) \triangleq \mathsf{MM}(\boldsymbol{x}^k) - \boldsymbol{x}^k \\
\boldsymbol{v}^k &= R(\mathsf{MM}(\boldsymbol{x}^k)) - R(\boldsymbol{x}^k) \\
\alpha^k &= -\max\left(1, \|\boldsymbol{r}^k\|_2 / \|\boldsymbol{v}^k\|_2\right) \qquad k = 0, 1, 2, \ldots. \\
\boldsymbol{y}^k &= \boldsymbol{x}^k - \alpha^k \boldsymbol{r}^k \\
\boldsymbol{x}^{k+1} &= \mathsf{MM}(\boldsymbol{y}^k)
\end{aligned}$$

Iterates of Various Algorithms (cont'd):

- **SCA iterates:**

$$\begin{aligned} \hat{\boldsymbol{x}}^{k+1} &= \mathcal{S}_{\frac{\lambda}{\tau+\mathrm{diag}(\boldsymbol{A}^\mathsf{T}\boldsymbol{A})}}\left(\boldsymbol{x}^k - \frac{\boldsymbol{A}^\mathsf{T}\left(\boldsymbol{A}\boldsymbol{x}^k - \boldsymbol{b}\right)}{\tau + \mathrm{diag}\left(\boldsymbol{A}^\mathsf{T}\boldsymbol{A}\right)}\right) \\ \boldsymbol{x}^{k+1} &= \gamma^k \hat{\boldsymbol{x}}^{k+1} + \left(1-\gamma^k\right)\boldsymbol{x}^k \end{aligned} \qquad k = 0, 1, 2, \ldots$$

- **ADMM iterates:**

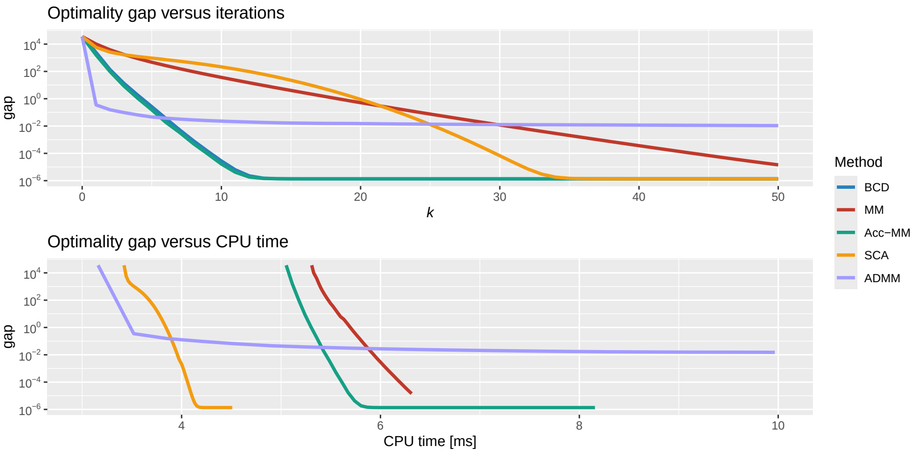
$$\begin{aligned} \boldsymbol{x}^{k+1} &= \left(\boldsymbol{A}^\mathsf{T}\boldsymbol{A} + \rho\boldsymbol{I}\right)^{-1}\left(\boldsymbol{A}^\mathsf{T}\boldsymbol{b} + \rho\left(\boldsymbol{z}^k - \boldsymbol{u}^k\right)\right) \\ \boldsymbol{z}^{k+1} &= \mathcal{S}_{\lambda/\rho}\left(\boldsymbol{x}^{k+1} + \boldsymbol{u}^k\right) \\ \boldsymbol{u}^{k+1} &= \boldsymbol{u}^k + \left(\boldsymbol{x}^{k+1} - \boldsymbol{z}^{k+1}\right) \end{aligned} \qquad k = 0, 1, 2, \ldots$$

Comparison of Methods

- The BCD method updates each element sequentially ($n = 100$), which leads to a high computational cost (CPU time).
- The Jacobi method is the parallel version of BCD, is not guaranteed to converge, and is similar to SCA but lacks τ and a smoothing step.
- The MM method requires computing the largest eigenvalue of $\boldsymbol{A}^\mathsf{T}\boldsymbol{A}$ and uses a conservative upper-bound κ for all elements.
- The SCA method uses $\text{diag}\left(\boldsymbol{A}^\mathsf{T}\boldsymbol{A}\right)$ instead of a common κ, which leads to faster convergence due to element-specific updates.
- The ADMM method converges with lower accuracy but is often sufficient for practical applications.

Numerical Comparison

Comparison of different iterative methods for the $\ell_2 - \ell_1$-norm minimization:



Outline

Summary

- Solvers for convex and nonconvex problems are available in all programming languages, often used via modeling frameworks.
- Solvers use methods like gradient descent, Newton's method, and interior-point methods, but users typically don't need to understand these details.
- Advanced users may develop custom algorithms for specific problems, requiring more effort and knowledge, such as the Dinkelbach method or Charnes-Cooper-Schaible transform for fractional problems.
- Iterative algorithmic frameworks break complex problems into easier ones:
  - Bisection
  - Block Coordinate Descent (BCD)
  - Majorization-Minimization (MM)
  - Successive Convex Approximation (SCA)
  - Alternating Direction Method of Multipliers (ADMM)

Beck, A. 2017. *First-Order Methods in Optimization*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics (SIAM).

Bertsekas, D. P. 1999. *Nonlinear Programming*. Athena Scientific.

Bertsekas, D. P., and J. N. Tsitsiklis. 1997. *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific.

Boyd, S. P., and L. Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press.

Boyd, S., N. Parikh, E. Chu, B. Peleato, and J. Eckstein. 2010. *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*. Foundations and Trends in Machine Learning, Now Publishers.

Fu, A., B. Narasimhan, and S. Boyd. 2020. "CVXR: An R Package for Disciplined Convex Optimization." *Journal of Statistical Software* 94 (14): 1–34.

References II

Grant, M., and S. Boyd. 2008. "Graph Implementations for Nonsmooth Convex Programs." In *Recent Advances in Learning and Control*, edited by V. Blondel, S. Boyd, and H. Kimura, 95–110. Lecture Notes in Control and Information Sciences. Springer-Verlag.

———. 2014. *CVX: Matlab Software for Disciplined Convex Programming*. http://cvxr.com/cvx.

Grippo, L., and M. Sciandrone. 2000. "On the Convergence of the Block Nonlinear Gauss–Seidel Method Under Convex Constraints." *Operations Research Letters* 26 (3): 127–36.

Hunter, D. R., and K. Lange. 2004. "A Tutorial on MM Algorithms." *The American Statistician* 58: 30–37.

Kumar, S., J. Ying, J. V. M. Cardoso, and D. P. Palomar. 2019. "Structured Graph Learning via Laplacian Spectral Constraints." In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*. Vancouver, Canada.

———. 2020. "A Unified Framework for Structured Graph Learning via Spectral Constraints." *Journal of Machine Learning Research (JMLR)*, 1–60.

Löfberg, J. 2004. "YALMIP: A Toolbox for Modeling and Optimization in MATLAB." In *Proceedings of the CACSD Conference*. Taipei, Taiwan.

Nemirovski, A. 2001. "Lectures on Modern Convex Optimization." In *Society for Industrial and Applied Mathematics (SIAM)*.

Nesterov, Y. 2018. *Lectures on Convex Optimization*. 2nd ed. Springer.

Nesterov, Y., and A. Nemirovskii. 1994. *Interior-Point Polynomial Algorithms in Convex Programming*. Philadelphia, PA: SIAM.

Nocedal, J., and S. J. Wright. 2006. *Numerical Optimization*. Springer Verlag.

References IV

Palomar, D. P. 2025. *Portfolio Optimization: Theory and Application*. Cambridge University Press.

Razaviyayn, M., M. Hong, and Z. Luo. 2013. "A Unified Convergence Analysis of Block Successive Minimization Methods for Nonsmooth Optimization." *SIAM Journal on Optimization* 23 (2): 1126–53.

Scutari, G., F. Facchinei, P. Song, D. P. Palomar, and J.-S. Pang. 2014. "Decomposition by Partial Linearization: Parallel Optimization of Multi-Agent Systems." *IEEE Transactions on Signal Processing* 62 (3): 641–56.

Scutari, G., and Y. Sun. 2018. "Parallel and Distributed Successive Convex Approximation Methods for Big-Data Optimization." In *Multi-Agent Optimization*, edited by F. Facchinei and J. S. Pang, 141–308. Lecture Notes in Mathematics, Springer.

Song, J., P. Babu, and D. P. Palomar. 2015. "Sparse Generalized Eigenvalue Problem via Smooth Optimization." *IEEE Transactions on Signal Processing* 63 (7): 1627–42.

Sun, Y., P. Babu, and D. P. Palomar. 2017. "Majorization-Minimization Algorithms in Signal Processing, Communications, and Machine Learning." *IEEE Transactions on Signal Processing* 65 (3): 794–816.

Varadhan, R., and C. Roland. 2008. "Simple and Globally Convergent Methods for Accelerating the Convergence of Any EM Algorithm." *Scandinavian Journal of Statistics* 35 (2): 335–53.

Zibulevsky, M., and M. Elad. 2010. "L1 - L2 Optimization in Signal and Image Processing." *IEEE Signal Processing Magazine*, May, 76–88.