# Portfolio Optimization

## Optimization Algorithms

Daniel P. Palomar (2024). *Portfolio Optimization: Theory and Application.*
Cambridge University Press.

portfoliooptimizationbook.com

Latest update: September 17, 2024

# Outline

1. Solvers
2. Gradient methods
3. Interior-point methods (IPM)
4. Fractional programming (FP) methods
5. BCD
6. MM
7. SCA
8. ADMM
9. Numerical comparison
10. Summary

# Abstract

Over the past century, the development of efficient algorithms for solving convex optimization problems has seen significant advancements. In 1947, Dantzig introduced the simplex method for linear programming (LP), which, despite its exponential worst-case complexity, became widely used. In 1984, Karmarkar's interior-point method revolutionized LP by offering polynomial time complexity. This innovation spurred further research, extending interior-point methods to quadratic programming (QP) and linear complementarity problems. In 1994, Nesterov and Nemirovskii advanced the field with the theory of self-concordant functions, enabling the application of log-barrier function-based algorithms to a broader range of convex problems, including semidefinite programming (SDP) and second-order cone programming (SOCP). Additionally, various specialized techniques like block-coordinate descent, majorization-minimization, and successive convex approximation have been developed to create customized algorithms for specific problems, often enhancing complexity and convergence rates. These slides will delve into a wide array of such practical algorithms (Palomar 2024, Appendix B).

# Outline

1. Solvers
2. Gradient methods
3. Interior-point methods (IPM)
4. Fractional programming (FP) methods
5. BCD
6. MM
7. SCA
8. ADMM
9. Numerical comparison
10. Summary

# 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 $O(n^3 L)$.
  - $n$: number of variables.
  - $L$: number of accuracy digits of the solution.
- **Specific complexities:**
  - **LP:** $O((m+n)^{3/2} n^2 L)$.
  - **QCQP:** $O(\sqrt{m}(m+n)n^2 L)$.
  - **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 $O(n^6 L)$.
  - 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 $\ell_2$-norm:**

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

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

# Interface with solvers

- **Norm approximation with Chebyshev or $\ell_\infty$-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)
```

# Interface with solvers

- **Norm approximation problem with Manhattan or $\ell_1$-norm:**

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

  - Rewritten as LP:

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

  - Equivalent form:

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

  - Matlab code:

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

# Interface with solvers

- **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

- **Euclidean norm approximation problem with linear constraints: (cont'd)**
  - 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:

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

# Interface with solvers

- **Euclidean norm approximation problem with linear constraints: (cont'd)**
  - **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.
  - Supports integer constraints.
  - 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.

# Modeling frameworks

- **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
```

# Modeling frameworks

- **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

1. Solvers
2. Gradient methods
3. Interior-point methods (IPM)
4. Fractional programming (FP) methods
5. BCD
6. MM
7. SCA
8. ADMM
9. Numerical comparison
10. Summary

# 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 $\boldsymbol{x}^0, \boldsymbol{x}^1, \boldsymbol{x}^2, \dots$
  - Sequence may or may not converge to an optimal solution $\boldsymbol{x}^\star$.
- **Ideal case with convex $f$:**
  - As iterations proceed ($k \rightarrow \infty$):
    - Objective function converges to the optimal value:

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

    - Gradient tends to zero:

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

- **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.*
    - $\alpha^{k}$: *stepsize* at iteration $k$.
- **Descent property:**
  - For a sufficiently small step, $\boldsymbol{d}$ must satisfy:
    $$\nabla f\left(\boldsymbol{x}\right)^{\mathsf{T}}\boldsymbol{d} < 0$$
  - $\alpha$ must be properly chosen (if too large, the descent property may be violated even with a descent direction).

# Line search

- **Line search:**
  - Procedure to choose the stepsize $\alpha$.
  - Two widely used methods due to good theoretical convergence and practical performance:
- **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

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

- **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.

# Gradient descent method

## Gradient descent method

**Initialization:**

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

**Repeat ($k$th 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:

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

  - 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).

# Newton's method

## Newton's method

**Initialization:**

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

**Repeat (*k*th 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

- **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 $\alpha^k$ 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 $\alpha$.
  - 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.
    - $\alpha^k$ 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:**

$$\begin{aligned} \bar{\boldsymbol{x}}^k &= \left[\boldsymbol{x}^k - s^k \nabla f\left(\boldsymbol{x}^k\right)\right]_{\mathcal{X}} \\ \boldsymbol{x}^{k+1} &= \boldsymbol{x}^k + \alpha^k \left(\bar{\boldsymbol{x}}^k - \boldsymbol{x}^k\right), \end{aligned}$$

  - $\bar{\boldsymbol{x}}^k$: Projection of $\boldsymbol{x}^k - s^k \nabla f\left(\boldsymbol{x}^k\right)$ onto the set $\mathcal{X}$.
  - $\alpha^k$: 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 $\boldsymbol{x} \in \mathcal{X}$.
- **Constrained Newton's method:**
  - **Iterative update:**

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

    - $\bar{\boldsymbol{x}}^k$: Solution to the quadratic subproblem.
    - $\alpha^k$: 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 $\alpha^k$ 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

1. Solvers
2. Gradient methods
3. **Interior-point methods (IPM)**
4. Fractional programming (FP) methods
5. BCD
6. MM
7. SCA
8. ADMM
9. Numerical comparison
10. Summary

# 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 $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$:**
  - Solutions can be expressed as:

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

  - $\boldsymbol{x}_0$ is any particular solution to $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$.
  - $\boldsymbol{F} \in \mathbb{R}^{n\times(n-p)}$ spans the nullspace of $\boldsymbol{A}$, i.e., $\boldsymbol{A}\boldsymbol{F} = \boldsymbol{0}$.
- **Reduced or eliminated problem:**
  - Equivalent to the original problem:

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

  - Gradients and Hessians:

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

# Indicator function

- **Reformulation via indicator function:**
  - Equivalent form of the original problem:

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

  - Indicator function definition:

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

- **Characteristics of the reformulated problem:**
  - Inequality constraints are eliminated.
  - Indicator function is included in the objective.
  - Drawbacks:
    - Indicator function is noncontinuous.
    - Indicator function is nondifferentiable.
    - Not practical for optimization.

# Logarithmic barrier

- **Smooth approximation of the indicator function:**
  - Popular choice: **logarithmic barrier**:

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

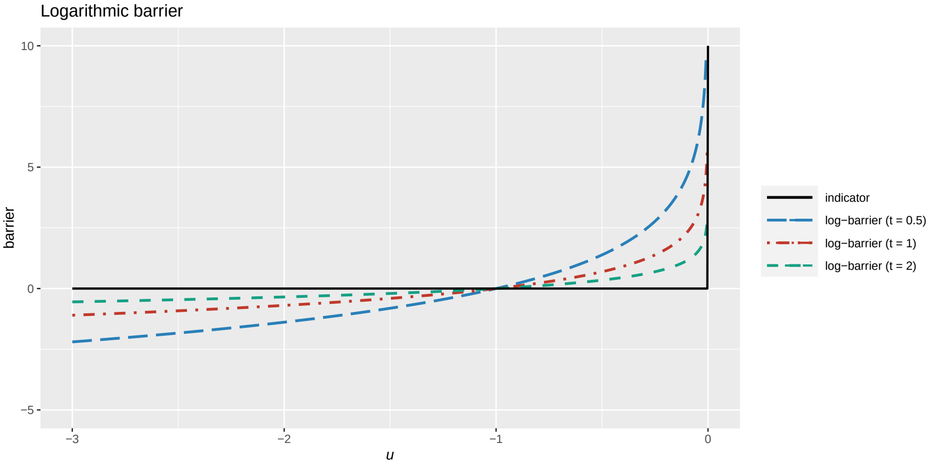
  - Parameter $t > 0$ controls the approximation.
  - Approximation improves as $t \to \infty$.

- **Approximate problem using the logarithmic barrier:**
  - Reformulated problem:

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

# Logarithmic barrier

Logarithmic barrier for several values of the parameter $t$:



# Logarithmic barrier

- **Logarithmic barrier function:**
  - Overall barrier function (excluding the $1/t$ factor):

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

    which is convex (from composition rules).
  - Gradient and Hessian:

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

# Central path

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

$$\begin{array}{ll} \underset{\boldsymbol{x}}{\text{minimize}} & tf_0(\boldsymbol{x}) + \phi(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \end{array}$$

  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 $\lambda_i^\star(t) = 1/(-tf_i(\boldsymbol{x}^\star(t)))$.
  - $\boldsymbol{x}^\star(t)$ 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:** $f_0(\boldsymbol{x}^\star(t)) \to p^\star$ as $t \to \infty$.
  - From Lagrange duality theory:

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

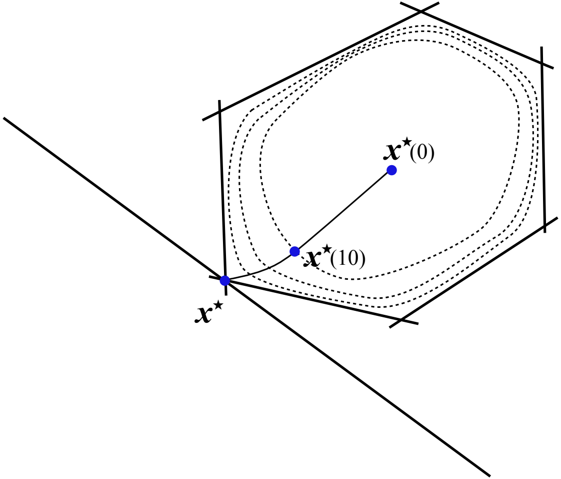
- **Connection with KKT conditions:**
  - $\boldsymbol{x}^\star(t)$ and $\boldsymbol{\lambda}^\star(t)$ satisfy:

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

  - Difference with original KKT conditions:
    - Complementary slackness is approximately satisfied.
    - Approximation improves as $t \to \infty$.

# Central path

Example of central path of an LP:



# Barrier method

- **Smooth approximation with logarithmic barrier:**
  - Log-barrier problem is a smooth approximation of the original problem.
  - 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.
    - 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

- **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 $\mu$:**
    - Large $\mu$ 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.

# Barrier method

## Barrier method for constrained optimization

**Initialization:**

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

**Repeat ($k$th iteration):**

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

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

# Barrier method

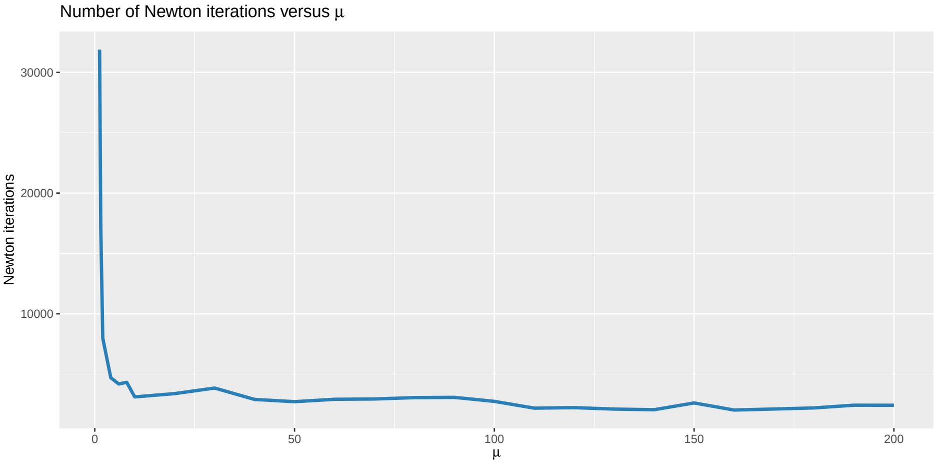
- **Example: Barrier method for LP:**
  - Consider the LP:

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

  - Use the barrier method with different $\mu$ values.
  - **Convergence analysis:**
    - Case: $m = 100$ inequalities, $n = 50$ variables.
    - $\epsilon = 10^{-6}$ for the duality gap.
    - Centering problem solved via Newton's method.
  - **Observation:**
    - Total number of Newton iterations is not very sensitive to $\mu$ as long as $\mu \geq 10$.

# Barrier method

Convergence of barrier method for an LP for different values of $\mu$:



# Convergence

- **Termination criterion:**
  - Number of outer iterations (centering steps) required:

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

  - Solving for $k$:

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

  - $\lceil\cdot\rceil$ is the ceiling operator.
- **Convergence of centering steps:**
  - Characterized via the convergence for Newton's method.
  - Specific updates for $\mu$ 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:**
  - 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:

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

  - 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:

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

  - **Constructing a strictly feasible point:**
    - Choose any $\boldsymbol{x}$ that satisfies the equality constraints.
    - 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:**
  - Obtain $(\boldsymbol{x}^\star, s^\star)$.
  - If $s^\star < 0$:
    - $\boldsymbol{x}^\star$ is a strictly feasible point.
    - Can be used in the barrier method to solve the original problem.
  - If $s^\star > 0$:
    - No feasible point exists.
    - No need to attempt solving 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

1. Solvers
2. Gradient methods
3. Interior-point methods (IPM)
4. Fractional programming (FP) methods
5. BCD
6. MM
7. SCA
8. ADMM
9. Numerical comparison
10. Summary

# Fractional programming (FP) methods

- **Concave-convex fractional program (FP):**

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

  - **Properties:**
    - $f(\boldsymbol{x})$ is a concave function
    - $g(\boldsymbol{x}) > 0$ is a convex function
    - $\mathcal{X}$ is a convex feasible set
- **Nature of FPs:**
  - Nonconvex problems, generally difficult to solve
  - Concave-convex FP is a quasiconvex optimization problem, making it more tractable
- **Methods to solve concave-convex FP:**
  - **Iterative bisection method**
  - **Dinkelbach method**
  - **Schaible transform**

# FP: Bisection method

- **Problem reformulation:**
  - Solve a sequence of convex feasibility problems:

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

  - $t > 0$ 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

# FP: Bisection method

- 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 $\epsilon$ 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:**

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

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

# FP: Dinkelbach method

- 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 ($k$th 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 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:

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

# FP: Schaible transform

- **Concave-convex fractional program (FP):**

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

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

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

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

# 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

1. Solvers
2. Gradient methods
3. Interior-point methods (IPM)
4. Fractional programming (FP) methods
5. **BCD**
6. MM
7. SCA
8. ADMM
9. Numerical comparison
10. Summary

# BCD

- **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:**

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

  where $f$ is the (possibly nonconvex) objective function and each $\mathcal{X}_i$ is a convex set.
  - **Partitioning:** Variable $\boldsymbol{x}$ is partitioned into $n$ blocks $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$
- **Method description:**
  - **Iterative process:** Produces a sequence of iterates $\boldsymbol{x}^0, \boldsymbol{x}^1, \boldsymbol{x}^2, \ldots$ that converge to $\boldsymbol{x}^\star$
  - **Update rule:** Optimize the problem with respect to each block $\boldsymbol{x}_i$ sequentially
  - **Inner iterations:** At each outer iteration $k$, execute $n$ inner iterations sequentially:

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

# BCD

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

## BCD for separable problems

**Initialization:**

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

**Repeat ($k$th iteration):**

1. Execute $n$ inner iterations sequentially:

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

2. $k \leftarrow k + 1$

**Until:** convergence

# BCD: Convergence

- BCD enjoys monotonicity, i.e., $f\left(\boldsymbol{x}^{k+1}\right) \leq f\left(\boldsymbol{x}^{k}\right)$
- **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):**
  - **Objective:** Execute $n$ inner iterations in parallel instead of sequentially
  - **Update rule:**

$$\mathbf{x}_i^{k+1} = \underset{\mathbf{x}_i \in \mathcal{X}_i}{\arg\min}\ f\left(\mathbf{x}_1^k, \dots, \mathbf{x}_{i-1}^k, \mathbf{x}_i, \mathbf{x}_{i+1}^k, \dots, \mathbf{x}_n^k\right), \quad i = 1, \dots, n$$

- **Jacobi method:**
  - **Description:** Parallel update of block variables
  - **Algorithmic attractiveness:** Potentially faster due to parallel execution
- **Convergence properties:**
  - **Issue:** Jacobi method does not enjoy nice convergence properties
  - **Condition for convergence:** Convergence is guaranteed if the mapping defined by $T(\mathbf{x}) = \mathbf{x} - \gamma \nabla f(\mathbf{x})$ is a contraction for some $\gamma$
  - **Reference:** (Bertsekas 1999)

# BCD example: Soft-thresholding operator

- **Univariate convex optimization problem:**

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

- **Solution:**

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

  - Sign function:

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

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

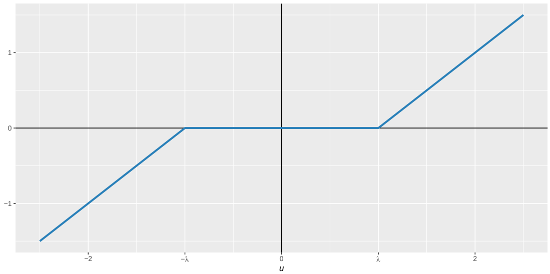
# BCD example: Soft-thresholding operator

- **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:**

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

- **Solution approach:**
  - **Standard method:** Can be solved with a QP solver
  - **Iterative algorithm via BCD:** via soft-thresholding operator (Zibulevsky and Elad 2010)
- **BCD method:**
  - **Variable partitioning:** Divide the variable into each constituent element $\boldsymbol{x} = (x_1, \dots, x_n)$
  - **Sequence of problems** at each iteration $k = 0, 1, 2, \dots$ for each element $i = 1, \dots, n$:

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

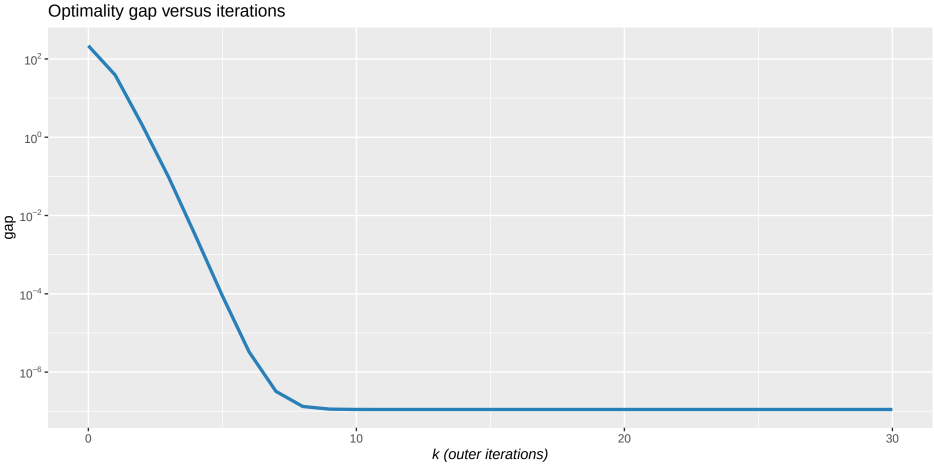
  where $\tilde{\boldsymbol{b}}_i^k \triangleq \boldsymbol{b} - \sum_{j<i}\boldsymbol{a}_j x_j^{k+1} - \sum_{j>i}\boldsymbol{a}_j x_j^k$.

- **Iterative algorithm:** For $k = 0, 1, 2, \dots$:

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

# BCD example: $\ell_2 - \ell_1$-norm minimization

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



# Outline

1. Solvers
2. Gradient methods
3. Interior-point methods (IPM)
4. Fractional programming (FP) methods
5. BCD
6. MM
7. SCA
8. ADMM
9. Numerical comparison
10. Summary

# MM

- **Majorization-minimization (MM) method:**
  - **Objective:** Approximate a difficult optimization problem by a sequence of simpler problems.
  - **References:**
    - Concise tutorial: (Hunter and Lange 2004)
    - Long tutorial with applications: (Sun, Babu, and Palomar 2017)
    - Convergence analysis: (Razaviyayn, Hong, and Luo 2013)
- **Original 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
- $\mathcal{X}$ is a (possibly nonconvex) set.

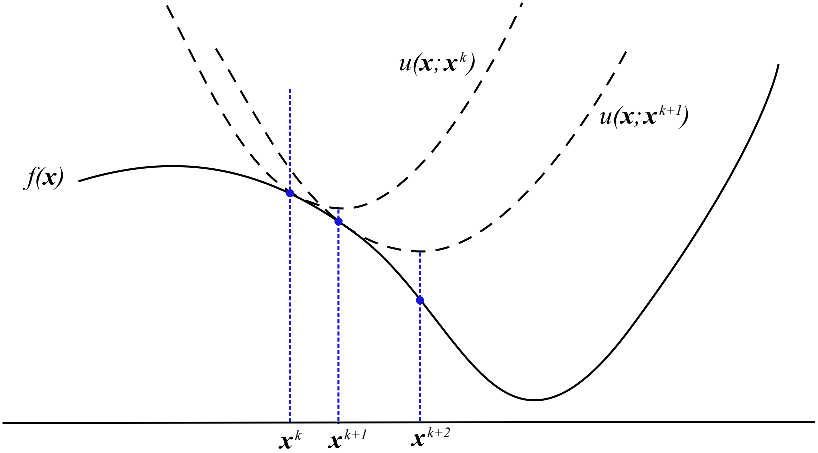
# MM

- **MM method:**
  - **Iterative process:** Produces a sequence of iterates $\boldsymbol{x}^0, \boldsymbol{x}^1, \boldsymbol{x}^2, \ldots$ that converge to $\boldsymbol{x}^\star$.
  - **Surrogate function:** At iteration $k$, approximate $f(\boldsymbol{x})$ by a surrogate function $u\left(\boldsymbol{x}; \boldsymbol{x}^k\right)$ around the current point $\boldsymbol{x}^k$.
  - **Sequence of problems:**

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

# MM

Illustration of sequence of surrogate problems in MM:



# MM: Convergence

- **Conditions for surrogate function $u\left(\mathbf{x};\mathbf{x}^{k}\right)$:**
  - **Upper-bound property:** $u\left(\mathbf{x};\mathbf{x}^{k}\right)\geq f\left(\mathbf{x}\right)$
  - **Touching property:** $u\left(\mathbf{x}^{k};\mathbf{x}^{k}\right)=f\left(\mathbf{x}^{k}\right)$
  - **Tangent property:** $u\left(\mathbf{x};\mathbf{x}^{k}\right)$ must be differentiable with $\nabla u\left(\mathbf{x};\mathbf{x}^{k}\right)=\nabla f\left(\mathbf{x}\right)$
- **Consequences:**
  - **Monotonicity:** $f\left(\mathbf{x}^{k+1}\right)\leq f\left(\mathbf{x}^{k}\right)$
  - **Convergence:** If $\mathcal{X}$ is convex, every limit point of the sequence $\left\{\mathbf{x}^{k}\right\}$ is a stationary point of the original problem
- **Majorizer construction:**
  - **Objective:** Find an appropriate majorizer $u\left(\mathbf{x};\mathbf{x}^{k}\right)$ that satisfies the technical conditions and leads to a simpler surrogate problem
  - **Techniques and examples:** Refer to (Sun, Babu, and Palomar 2017)

# MM

## MM algorithm

**Initialization:**

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

**Repeat ($k$th iteration):**

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

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

3. $k \leftarrow k+1$

**Until:** convergence

# MM: Convergence

- **Versatility of MM framework:**
  - **Objective:** Derive practical algorithms
  - **Theoretical guarantees:** Convergence properties are well-established
- **Assumptions:**
  - Majorizer $u\left(\boldsymbol{x};\boldsymbol{x}^{k}\right)$ satisfies the technical conditions.
  - Feasible set $\mathcal{X}$ is convex.
- **Convergence:** Every limit point of the sequence $\{\boldsymbol{x}^{k}\}$ is a stationary point of the original problem
- **Nonconvex feasible set $\mathcal{X}$:**
  - Convergence must be studied on a case-by-case basis.
  - Examples: (Song, Babu, and Palomar 2015; Sun, Babu, and Palomar 2017; Kumar et al. 2019, 2020).

# MM: Acceleration techniques

- **MM convergence speed:**
  - **Issue:** MM may require many iterations to converge if the surrogate function $u\left(\boldsymbol{x};\boldsymbol{x}^{k}\right)$ 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:**

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

  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: $\boldsymbol{x}^\star = (\boldsymbol{A}^\mathsf{T}\boldsymbol{A})^{-1}\boldsymbol{A}^\mathsf{T}\boldsymbol{b}$.

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

# MM example: Nonnegative LS

- **Objective function:** $f(\boldsymbol{x}) = \frac{1}{2}\left\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\right\|_2^2$
- **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:** $u\left(\boldsymbol{x};\boldsymbol{x}^k\right) \geq f\left(\boldsymbol{x}\right)$ (proved using Jensen's inequality)
  - **Touching property:** $u\left(\boldsymbol{x}^k;\boldsymbol{x}^k\right) = f\left(\boldsymbol{x}^k\right)$
  - **Tangent property:** $\nabla u\left(\boldsymbol{x}^k;\boldsymbol{x}^k\right) = \nabla f\left(\boldsymbol{x}^k\right)$

# MM example: Nonnegative LS

- **Sequence of majorized problems:**

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

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

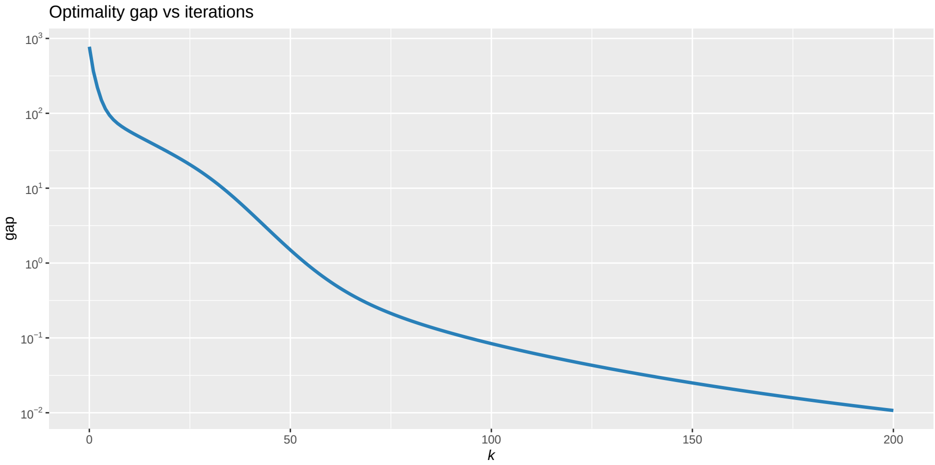
- **Iterative update:**

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

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 elementwise product.

# MM example: Nonnegative LS

Convergence of MM for the nonnegative LS:



# 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:**

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

  - **Partitioning:** Variables are partitioned into $n$ blocks $\boldsymbol{x} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$.
  - **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 $u(\boldsymbol{x}_i; \boldsymbol{x}^k)$.
- **References:** (Razaviyayn, Hong, and Luo 2013) (Sun, Babu, and Palomar 2017).

# Block MM Procedure

1. **Initialization:** Start with an initial guess $\boldsymbol{x}^0 = (\boldsymbol{x}_1^0, \ldots, \boldsymbol{x}_n^0)$
2. **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:** $\boldsymbol{x}^{k+1} = (\boldsymbol{x}_1^{k+1}, \ldots, \boldsymbol{x}_n^{k+1})$

# Outline

1. Solvers
2. Gradient methods
3. Interior-point methods (IPM)
4. Fractional programming (FP) methods
5. BCD
6. MM
7. SCA
8. ADMM
9. Numerical comparison
10. Summary

# SCA

- **Successive convex approximation (SCA) method:**
  - Approximates a difficult optimization problem by a sequence of simpler convex problems.
  - Produces a sequence of iterates $\boldsymbol{x}^0, \boldsymbol{x}^1, \boldsymbol{x}^2, \ldots$ that converge to $\boldsymbol{x}^\star$.

- **Problem formulation:**

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

where

- $f$ is a (possibly nonconvex) objective function
- $\mathcal{X}$ is a convex set (nonconvex sets can be accommodated with more complexity).

# SCA

- **Iteration process:**
  - At iteration $k$, approximate $f(\mathbf{x})$ by a surrogate function $\tilde{f}\left(\mathbf{x};\mathbf{x}^k\right)$ around $\mathbf{x}^k$.
  - Solve the sequence of simpler problems:

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

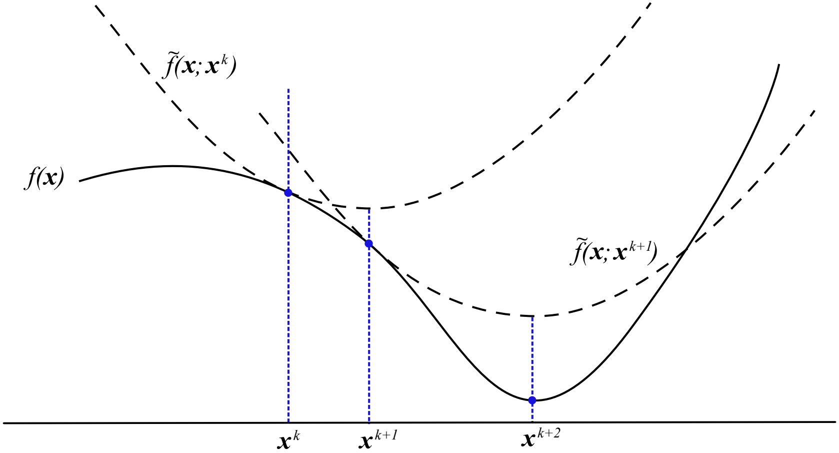
  - Introduce a smoothing step to avoid oscillations:

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

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

# SCA

Illustration of sequence of surrogate problems in SCA:



$\tilde{f}(\boldsymbol{x};\boldsymbol{x}^k)$
$f(\boldsymbol{x})$
$\tilde{f}(\boldsymbol{x};\boldsymbol{x}^{k+1})$
$\boldsymbol{x}^k$
$\boldsymbol{x}^{k+1}$
$\boldsymbol{x}^{k+2}$

# SCA: Convergence

- **Conditions for surrogate function $\tilde{f}\left(\boldsymbol{x};\boldsymbol{x}^k\right)$:**
  - Must be strongly convex on the feasible set $\mathcal{X}$.
  - Must be differentiable with $\nabla\tilde{f}\left(\boldsymbol{x};\boldsymbol{x}^k\right)=\nabla f\left(\boldsymbol{x}\right)$.
- **Stepsize rules for $\{\gamma^k\}$:**
  - *Bounded stepsize*: $\gamma^k$ values are sufficiently small (difficult to use in practice).
  - *Backtracking line search*: Effective in terms of iterations but costly.
  - *Diminishing stepsize*: Practical choice 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: $\gamma^{k+1}=\frac{\gamma^k+\alpha^k}{1+\beta^k}$, $\gamma^0=1$, $\alpha^k$ and $\beta^k$ satisfy $0\leq\alpha^k\leq\beta^k$ and $\alpha^k/\beta^k\to 0$.
- **Examples of $\alpha^k$ and $\beta^k$:**
  - $\alpha^k=\alpha$ 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.

# SCA

## SCA algorithm

**Initialization:**

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

**Repeat ($k$th iteration):**

1. Construct surrogate of $f(\mathbf{x})$ around current point $\mathbf{x}^k$ as $\tilde{f}\left(\mathbf{x};\mathbf{x}^k\right)$.
2. Obtain intermediate point by solving the surrogate convex problem:

$$\hat{\mathbf{x}}^{k+1} = \underset{\mathbf{x}\in\mathcal{X}}{\arg\min}\ \tilde{f}\left(\mathbf{x};\mathbf{x}^k\right).$$

3. Obtain next iterate by averaging the intermediate point with the previous one:

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \gamma^k\left(\hat{\mathbf{x}}^{k+1} - \mathbf{x}^k\right).$$

4. $k \leftarrow k+1$

**Until:** convergence

# Gradient descent method as SCA

- **Unconstrained problem:**

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

- **SCA with surrogate function:**
  - Surrogate function:

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

- **Minimizing the surrogate function:**
  - Set the gradient of $\tilde{f}\left(\boldsymbol{x};\boldsymbol{x}^k\right)$ to zero:

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

  - Solve for $\boldsymbol{x}$:

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

- **Iteration process:**
  - Update rule coincides with the gradient descent method:

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

# Newton method as SCA

- **Including second-order information:**
  - Surrogate function with Hessian:

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

- **Minimizing the surrogate function:**
  - Set the gradient of $\tilde{f}\left(\boldsymbol{x};\boldsymbol{x}^{k}\right)$ to zero:

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

  - Solve for $\boldsymbol{x}$:

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

- **Iteration process:**
  - Update rule coincides with Newton's method:

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

# Parallel SCA

- **Partitioned variables in SCA:**

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

  where 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 surrogate functions $\tilde{f}_i\left(\boldsymbol{x}_i; \boldsymbol{x}^k\right)$.
  - Update process for each block $i$:

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

    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 surrogate function:**
  - Must be strongly convex on the feasible set $\mathcal{X}$.
  - Must be differentiable with $\nabla \tilde{f}\left(\boldsymbol{x};\boldsymbol{x}^k\right) = \nabla f\left(\boldsymbol{x}\right)$.
- **Stepsize rules for $\{\gamma^k\}$:**
  - *Bounded stepsize*: $\gamma^k$ values are sufficiently small.
  - *Backtracking line search*: Effective but requires multiple evaluations per iteration.
  - *Diminishing stepsize*: Practical choice 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:**
  - Partition variable $\boldsymbol{x}$ 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:**
  - Update rule:

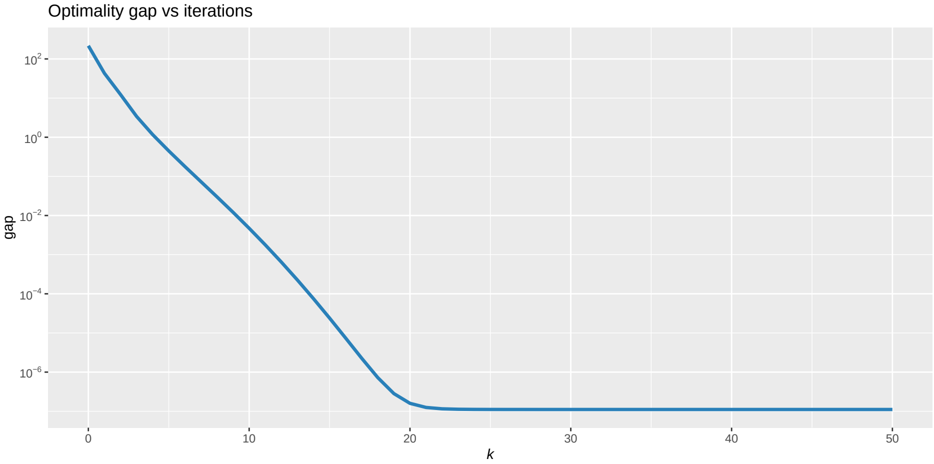
$$\begin{aligned} \hat{x}_i^{k+1} &= \frac{1}{\tau + \|\boldsymbol{a}_i\|^2} \mathcal{S}_\lambda \left( \boldsymbol{a}_i^\mathsf{T} \tilde{\boldsymbol{b}}_i^k + \tau x_i^k \right) \\ x_i^{k+1} &= x_i^k + \gamma^k \left( \hat{x}_i^{k+1} - x_i^k \right) \end{aligned} \qquad i = 1, \dots, n, \quad k = 0,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

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

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

# 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:**

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

  - $\|\boldsymbol{D}\|_F$: Frobenius norm of $\boldsymbol{D}$
  - $\|\boldsymbol{X}\|_1$: elementwise $\ell_1$-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:**
  - 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:**

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

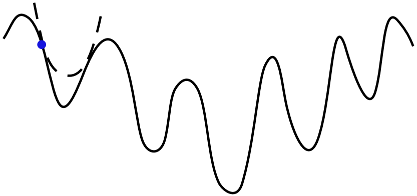
  - **Matrix version of the $\ell_2 - \ell_1$-norm problem:**

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

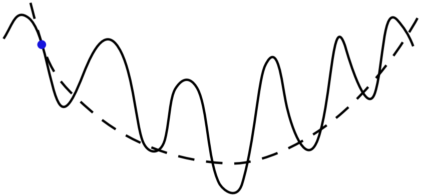
which can be further decomposed into a set of vectorized $\ell_2 - \ell_1$-norm problems for each column of $\boldsymbol{X}$.

# MM versus 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



SCA

# MM versus 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 $\mathcal{X}$.
        - 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 $\boldsymbol{x} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$.
    - **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

1. Solvers
2. Gradient methods
3. Interior-point methods (IPM)
4. Fractional programming (FP) methods
5. BCD
6. MM
7. SCA
8. ADMM
9. Numerical comparison
10. Summary

# ADMM

- **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 $\boldsymbol{y}$:

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

- 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 augmented Lagrangian:

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

- Algorithm:

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

- Converges under more relaxed conditions but cannot decouple $\boldsymbol{x}$ and $\boldsymbol{z}$ due to $\|\boldsymbol{Ax} + \boldsymbol{Bz} - \boldsymbol{c}\|_2^2$ term.

# Third Attempt: ADMM

Third an final attempt to decouple the variables.

**ADMM:**

- Combines features of dual decomposition and method of multipliers.
- Minimizes augmented Lagrangian with BCD 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 primal variables $\boldsymbol{x}$ and $\boldsymbol{z}$.
- Faster convergence with fewer technical conditions.
- Common to express ADMM updates using scaled dual variable $\boldsymbol{u}^k = \boldsymbol{y}^k/\rho$ as in the next algorithm.

# ADMM

## ADMM algorithm

**Initialization:**

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

**Repeat ($k$th 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:** $\boldsymbol{A}\boldsymbol{x}^k + \boldsymbol{B}\boldsymbol{z}^k - \boldsymbol{c} \to \boldsymbol{0}$ as $k \to \infty$
    - Iterates approach feasibility.
  - **Objective convergence:** $f(\boldsymbol{x}) + g(\boldsymbol{z}) \to p^\star$ as $k \to \infty$
    - Objective function of the iterates approaches the optimal value.
  - **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:**
  - $\{\boldsymbol{x}^k\}$ and $\{\boldsymbol{z}^k\}$ need not converge to optimal values without additional assumptions.
  - ADMM can be slow to converge to high accuracy.
  - Often converges to modest accuracy within a few tens of iterations, which is sufficient for many practical applications.
  - Different from the fast convergence of Newton's method.

# 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:**
  - Update rules:

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

  - $[\cdot]_{\mathcal{X}}$ denotes projection on the set $\mathcal{X}$.
- **Explanation of steps:**
  - ***x*-update:** Minimize $f(\boldsymbol{x})$ with a quadratic penalty term.
  - ***z*-update:** Project $\boldsymbol{x}^{k+1} + \boldsymbol{u}^k$ onto the set $\mathcal{X}$.
  - ***u*-update:** Update 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.

# ADMM example: $\ell_2 - \ell_1$-norm minimization

- **$\ell_2 - \ell_1$-norm minimization problem:**

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

- **Reformulated problem for ADMM:**

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

# ADMM example: $\ell_2 - \ell_1$-norm minimization

- **ADMM algorithm:**
  - ***x*-update:**
    - Given $\boldsymbol{z}$ and scaled dual variable $\boldsymbol{u}$, solve:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \tfrac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \tfrac{\rho}{2}\,\|\boldsymbol{x} - \boldsymbol{z} + \boldsymbol{u}\|_2^2$$

    - Solution:

$$\boldsymbol{x} = \left(\boldsymbol{A}^\mathsf{T}\boldsymbol{A} + \rho\boldsymbol{I}\right)^{-1}\left(\boldsymbol{A}^\mathsf{T}\boldsymbol{b} + \rho(\boldsymbol{z} - \boldsymbol{u})\right)$$

  - ***z*-update:**
    - Given $\boldsymbol{x}$ and $\boldsymbol{u}$, solve:

$$\underset{\boldsymbol{z}}{\text{minimize}} \quad \tfrac{\rho}{2}\,\|\boldsymbol{x} - \boldsymbol{z} + \boldsymbol{u}\|_2^2 + \lambda\|\boldsymbol{z}\|_1$$

    - Solution using the soft-thresholding operator $\mathcal{S}_{\lambda/\rho}(\cdot)$:

$$\boldsymbol{z} = \mathcal{S}_{\lambda/\rho}\left(\boldsymbol{x} + \boldsymbol{u}\right)$$

  - ***u*-update:**
    - Update the scaled dual variable:

$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \left(\boldsymbol{x}^{k+1} - \boldsymbol{z}^{k+1}\right)$$

# ADMM example: $\ell_2 - \ell_1$-norm minimization

- **ADMM iterative algorithm:**
  - Update rules:

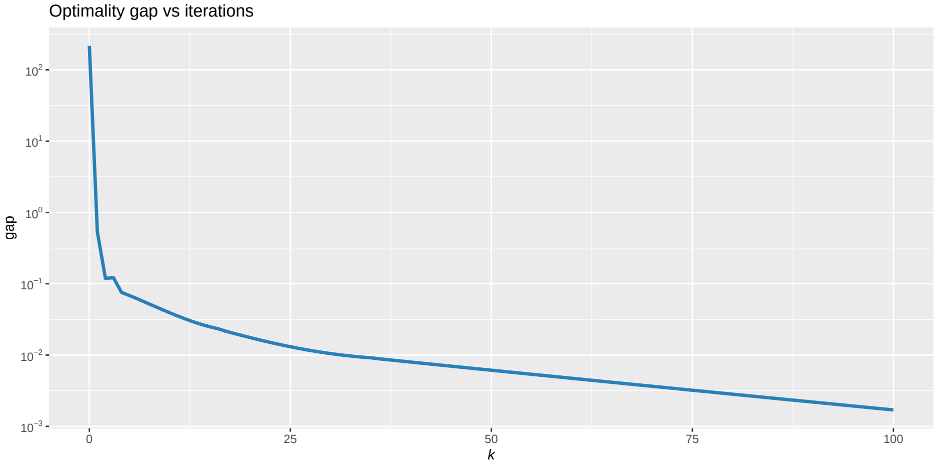
$$\begin{aligned} \boldsymbol{x}^{k+1} &= \left(\boldsymbol{A}^\mathsf{T}\boldsymbol{A} + \rho\boldsymbol{I}\right)^{-1}\left(\boldsymbol{A}^\mathsf{T}\boldsymbol{b} + \rho\left(\boldsymbol{z}^k - \boldsymbol{u}^k\right)\right) \\ \boldsymbol{z}^{k+1} &= \mathcal{S}_{\lambda/\rho}\left(\boldsymbol{x}^{k+1}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

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

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

# ADMM example: $\ell_2 - \ell_1$-norm minimization

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



# Outline

1. Solvers
2. Gradient methods
3. Interior-point methods (IPM)
4. Fractional programming (FP) methods
5. BCD
6. MM
7. SCA
8. ADMM
9. Numerical comparison
10. Summary

# Numerical comparison

- $\ell_2 - \ell_1$**-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:**

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

    - $\boldsymbol{x}^{(k,i)} \triangleq \left(x_1^{k+1},\dots,x_{i-1}^{k+1},x_i^k,\dots,x_n^k\right)$
  - **Parallel BCD (Jacobi) iterates:**

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

# Numerical comparison

- **Iterates of various algorithms: (cont'd)**
  - **MM iterates:**

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

  - **Accelerated MM iterates:**

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

# Numerical comparison

- **Iterates of various algorithms: (cont'd)**
  - **SCA iterates:**

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

  - **ADMM iterates:**

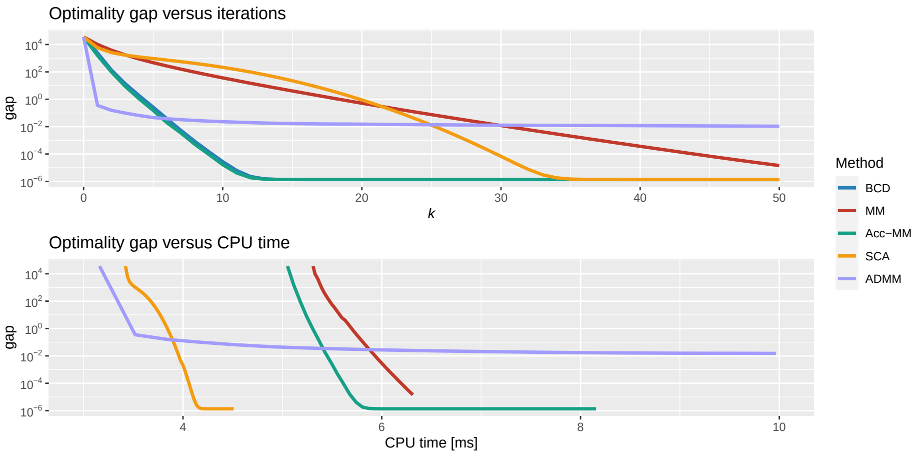
$$\begin{aligned} \boldsymbol{x}^{k+1} &= \left(\boldsymbol{A}^\mathsf{T}\boldsymbol{A} + \rho\boldsymbol{I}\right)^{-1}\left(\boldsymbol{A}^\mathsf{T}\boldsymbol{b} + \rho\left(\boldsymbol{z}^k - \boldsymbol{u}^k\right)\right) \\ \boldsymbol{z}^{k+1} &= \mathcal{S}_{\lambda/\rho}\left(\boldsymbol{x}^{k+1}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

# Numerical comparison

- **Comparison of methods:**
  - **BCD:**
    - Updates each element sequentially ($n = 100$).
    - High computational cost (CPU time) due to sequential updates.
  - **Jacobi:**
    - Parallel version of BCD.
    - Not guaranteed to converge.
    - Similar to SCA but lacks $\tau$ and smoothing step.
  - **MM:**
    - Requires computing the largest eigenvalue of $\boldsymbol{A}^\mathsf{T}\boldsymbol{A}$.
    - Conservative upper-bound $\kappa$ used for all elements.
  - **SCA:**
    - Uses $\text{diag}\left(\boldsymbol{A}^\mathsf{T}\boldsymbol{A}\right)$ instead of a common $\kappa$.
    - Faster convergence due to element-specific updates.
  - **ADMM:**
    - Converges with lower accuracy.
    - Often sufficient for practical applications.

# Numerical comparison

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



# Outline

1. Solvers
2. Gradient methods
3. Interior-point methods (IPM)
4. Fractional programming (FP) methods
5. BCD
6. MM
7. SCA
8. ADMM
9. Numerical comparison
10. Summary

# 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)

# References I

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.

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.

# References II

———. 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.

# References III

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.

Palomar, D. P. 2024. *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.

# References IV

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.