PART I. OPTIMIZATION: CLASSICAL APPROACHES
(LECTURE 9)

Shpilev Petr Valerievich
Faculty of Mathematics and Mechanics, SPbU

September, 2025

Санкт-Петербургский государственный университет

Comments

In this lecture, we extend our study of linear and quadratic programming algorithms. We begin with the two-phase simplex method, discussing challenges such as degeneracy, cycling, and perturbation strategies to ensure progress. We then introduce the dual simplex method, exploring its step-by-step procedure and illustrating its application through examples. The lecture also covers presolving techniques in linear programming, including detection of redundant or dominated constraints and problem simplification. Building on this foundation, we transition to quadratic programming, with portfolio optimization as a motivating example. We analyze equality-constrained QPs, properties of the KKT matrix such as nonsingularity and inertia, and solution methods including Schur-complement and null-space approaches. Finally, we discuss iterative methods for large-scale problems, focusing on preconditioned conjugate gradients for reduced systems and the projected CG method.

Example: Inequality-Constrained Linear Program

Consider: $\min 3x_1 + x_2 + x_3$, s.t. $2x_1 + x_2 + x_3 \leq 2$, $x_1 - x_2 - x_3 \leq -1$, $x \geq 0$.

Convert to standard form and set up Phase I to find a feasible starting point.

Standard Form: Add slack variables x_4, x_5:

$$\min 3x_1 + x_2 + x_3, \text{ s.t.}$$
$$2x_1 + x_2 + x_3 + x_4 = 2,$$
$$x_1 - x_2 - x_3 + x_5 = -1,$$
$$x \geq 0.$$

Point $x = (0, 0, 0, 2, 0)$ is feasible for first constraint, not second.

Phase I Setup: Add one artificial variable z_2 to second constraint:

$$\min z_2, \text{ s.t.}$$
$$2x_1 + x_2 + x_3 + x_4 = 2,$$
$$x_1 - x_2 - x_3 + x_5 - z_2 = -1,$$
$$(x, z_2) \geq 0.$$

Feasible point: $(x, z_2) = ((0, 0, 0, 2, 0), 1)$, with basis matrix $B = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ (invertible).

Note: B is formed from columns of x_4 and z_2 (basic variables); x_4 acts as artificial variable for first constraint, so no z_1 needed.

Comments

To illustrate the mechanics of the two-phase simplex method, consider a small linear program with inequalities. The objective is to minimize $3x_1 + x_2 + x_3$, subject to two constraints: first, $2x_1 + x_2 + x_3 \leq 2$; second, $x_1 - x_2 - x_3 \leq -1$. All variables are required to be nonnegative.

The first step is to rewrite the inequalities in standard form by introducing slack variables. Adding slack variable x_4 to the first inequality converts it into an equality. Adding slack variable x_5 to the second inequality does the same. However, the second equality is problematic: it reads $x_1 - x_2 - x_3 + x_5 = -1$. This right-hand side is negative, which immediately makes the naive choice of slacks infeasible. To resolve this, an artificial variable z_2 is added to absorb the infeasibility. The modified system then includes z_2 in the second constraint.

The auxiliary Phase One objective is simply to minimize z_2. The starting feasible point assigns all decision variables to zero, sets $x_4 = 2$, and sets $z_2 = 1$. At this point, the system is consistent, and the basis matrix B is invertible, constructed from the columns corresponding to x_4 and z_2. This demonstrates how slack variables sometimes automatically act as artificial variables, and how artificial variables are introduced only where truly necessary. The procedure guarantees a feasible starting point for Phase One, after which the artificial objective will be minimized to check whether feasibility of the original problem is achievable.

Simplex Method: Degenerate Steps and Cycling

Challenges in simplex method when updating basis for $\min c^T x$, s.t. $Ax = b$, $x \geq 0$.

Degenerate Steps:

- Occur when entering variable x_q cannot increase without violating $x \geq 0$, if some basic variable $x_i = 0$ and $d_i > 0$ ($d = B^{-1}A_q$).
- No change in x or objective $c^T x$, but basis changes, potentially nearing optimal basis.

Cycling Issue:

- Repeated degenerate steps may return to a previous basis, causing infinite loop.
- Common in large integer program relaxations; requires avoidance strategies.

Perturbation Strategy:

- For degenerate basis $\hat{\mathcal{B}}$ with matrix $\hat{B}$, modify constraints:

$$b(\epsilon) = b + \hat{B} \begin{bmatrix} \epsilon \\ \epsilon^2 \\ \vdots \\ \epsilon^m \end{bmatrix}, \quad \epsilon > 0 \text{ small.}$$

- A very small positive number ϵ shifts right-hand side to avoid degenerate cycling.

Comments

One of the subtle difficulties in the simplex method arises from the phenomenon of degeneracy. Degeneracy occurs when, during a pivot operation, the entering variable cannot increase because one or more basic variables are already equal to zero. In this case, although the basis changes, the vector of decision variables and the value of the objective function remain unchanged. In words, the algorithm performs a step that does not advance the solution in the objective space.

While this may sound harmless, repeated degenerate steps create the risk of cycling. Cycling refers to the possibility that the simplex algorithm revisits the same basis multiple times, which can theoretically lead to an infinite loop without ever reaching an optimal solution.

To mitigate this issue, a perturbation strategy is introduced. The idea is to slightly shift the right-hand side vector of the system of equations in a carefully controlled manner. If the current degenerate basis is denoted by $\hat{\mathcal{B}}$, with basis matrix $\hat{B}$, then we define a perturbed right-hand side as vector $b(\epsilon)$ equals

$$b + \hat{B} \begin{bmatrix} \epsilon \\ \epsilon^2 \\ \vdots \\ \epsilon^m \end{bmatrix}.$$

Here ϵ is a small positive scalar. This construction slightly perturbs the problem so that every basic variable is strictly positive, avoiding degeneracy. Although ϵ is taken arbitrarily small, the modification is sufficient to guarantee that each basis step produces an actual improvement or movement, thereby preventing endless cycling.

Intuitively, the perturbation acts like nudging the system away from exact degeneracy, ensuring that pivots lead to meaningful changes. Importantly, once the algorithm terminates under the perturbed system, the solution can be mapped back to the original problem, preserving correctness while ensuring finite termination.

Prevent cycling in degenerate simplex steps for $\min c^T x$, s.t. $Ax = b$, $x \geq 0$.

Perturbation Effect:

- Perturbed constraints

$$Ax = b + \hat{B} \begin{bmatrix} \epsilon \\ \epsilon^2 \\ \vdots \\ \epsilon^m \end{bmatrix}$$ shift basic solution:

$$x_{\mathcal{B}}(\epsilon) = x_{\mathcal{B}} + \sum_{k=1}^{m} (B^{-1}\hat{B})_{:k}\epsilon^k,$$

$(B^{-1}\hat{B})_{:k}$ - the kth column of $B^{-1}\hat{B}$ and $x_{\mathcal{B}}$ - the basic solution.

Ensures $x_{\mathcal{B}}(\epsilon)_i > 0$ for all ϵ sufficiently small, making bases nondegenerate (all basic variables positive).

Nondegeneracy and Termination:

- Nondegeneracy proven: If $x_{\mathcal{B}}(\epsilon)_i = 0$, then $(x_{\mathcal{B}})_i = 0$ and i-th row of $B^{-1}\hat{B} = 0$, impossible since B, $\hat{B}$ invertible.
- Prevents revisiting bases, ensures finite termination; reset $x_{\mathcal{B}} = B^{-1}b$ for original solution.

Lexicographic Strategy:

- Tracks coefficients of $\epsilon, \epsilon^2, \ldots, \epsilon^m$ in $x_{\mathcal{B}}(\epsilon)$, selects leaving variable by lexicographically minimizing $x_{\mathcal{B}}(\epsilon)_i/d_i$, updates pivot to maintain ϵ-dependence.

Comments

The perturbation strategy becomes more precise when we analyze its algebraic effect. Suppose the current basis matrix is B and the degenerate basis we perturb is $\hat{B}$. Then the new right-hand side is vector b plus matrix $\hat{B}$ times the vector ϵ, ϵ^2, and so on.

Solving for the basic variables gives the perturbed basic solution. Concretely, the basic solution $x_{\mathcal{B}}(\epsilon)$ equals the unperturbed solution $x_{\mathcal{B}}$ plus the sum from $k = 1$ to m of the k-th column of the product $B^{-1}\hat{B}$, multiplied by ϵ^k. This expansion shows that each basic variable is shifted slightly upward by a positive power of ϵ.

The crucial property is that every basic variable becomes strictly positive for sufficiently small ϵ. To see this, note that if a perturbed variable were zero, then both its unperturbed value and all coefficients multiplying ϵ terms would have to vanish. But since B and $\hat{B}$ are invertible matrices, such a simultaneous vanishing is impossible. This argument establishes nondegeneracy rigorously.

Once nondegeneracy is guaranteed, the simplex algorithm can no longer revisit the same basis, and termination in finitely many steps is assured. After finishing under the perturbed system, we simply set ϵ back to zero, recovering the original solution.

In practice, instead of explicitly adding symbolic perturbations, the algorithm can adopt a lexicographic pivoting rule. This rule compares ratios in a lexicographic manner by tracking the coefficients of ϵ, ϵ^2, and so on in the expansion of the basic variables. The leaving variable is chosen by minimizing the ratio in lexicographic order, ensuring consistency. This approach emulates perturbation implicitly, without requiring explicit small numbers, and is the standard way to enforce progress in degenerate situations.

Dual Simplex Method

The dual simplex method is a faster variant (in many cases) for $\min c^{\mathrm{T}}x$, s.t. $Ax = b$, $x \geq 0$, using basis updates like primal simplex.

Overview:

- Starts with dual feasible point (λ, s): $s_{\mathcal{B}} = 0$, $s_{\mathcal{N}} \geq 0$, but primal $x_{\mathcal{B}} = B^{-1}b$ may have negative components.
- Updates basis B, N to reach primal feasibility ($x_{\mathcal{B}} \geq 0$), achieving optimality.
- Unlike primal, B is nonsingular but not always a basis matrix (may not satisfy $x_{\mathcal{B}} = B^{-1}b \geq 0$).

Single Step:

- Define variables:

$$x_{\mathcal{B}} = B^{-1}b, \quad x_{\mathcal{N}} = 0, \quad \lambda = B^{-\mathrm{T}}c_{\mathcal{B}}, \quad s_{\mathcal{B}} = 0, \quad s_{\mathcal{N}} = c_{\mathcal{N}} - N^{\mathrm{T}}\lambda \geq 0.$$

- If $x_{\mathcal{B}} \geq 0$, the current point (x, λ, s) satisfies the optimality (KKT) conditions, and we are done.

Comments

The dual simplex method provides an alternative pivoting scheme that often proves faster than the primal simplex in applications.

The primal simplex begins with a feasible primal solution and maintains feasibility while driving the objective function toward optimality.

The dual simplex takes the opposite perspective: it begins from a dual feasible solution and then works toward primal feasibility, all while maintaining dual feasibility. Ultimately, the method converges to a solution that satisfies both primal and dual conditions, which is equivalent to optimality.

To be more precise, suppose we partition the variables into basic and nonbasic sets corresponding to matrices B and N. Then we can write the primal and dual variables explicitly.

The primal basic variables equal $B^{-1}b$. The primal nonbasic variables are zero. The dual vector λ is defined as $B^{-\mathrm{T}}c_{\mathcal{B}}$. The dual slacks are zero for the basic set, and equal the cost vector for nonbasic variables minus the transpose of N times λ.

If the primal basic variables are nonnegative, then the point defined by x, λ, and s satisfies all Karush-Kuhn-Tucker conditions, meaning we have reached an optimal solution. The distinguishing feature of the dual simplex method is that it can start from a situation where primal feasibility is violated—some basic variables are negative—but dual feasibility still holds.

The algorithm then corrects these violations step by step, making the basic variables nonnegative, while preserving the dual conditions.

This makes the dual simplex particularly efficient in reoptimization problems, such as when new constraints are added to a model or in large-scale branch-and-bound procedures for integer programming.

Dual Simplex Method: Step Details

Index Selection:

- Select a leaving index $q \in \mathcal{B}$ such that $x_q < 0$ to move x_q to zero (ensuring nonnegativity);
- Identify an entering index $r \in \mathcal{N}$ such that s_r becomes zero, while x_r increases from zero, moving q from $\mathcal{B}$ to $\mathcal{N}$, r from $\mathcal{N}$ to $\mathcal{B}$.

Dual Variable Updates:

- Update $s_{\mathcal{B}}$: $s_{\mathcal{B}}^+ = s_{\mathcal{B}} + \alpha e_q$, where e_q is a vector of length m with 1 at the position of q in $\mathcal{B}$, zeros elsewhere, for some positive scalar α to be determined update λ^+ and $s_{\mathcal{B}}^+$ for some vector v:

$$s_{\mathcal{B}}^+ = s_{\mathcal{B}} + \alpha e_q, \quad \lambda^+ = \lambda + \alpha v.$$

Deriving Update Vector:

- Since $s_{\mathcal{B}}^+ = c_{\mathcal{B}} - B^T\lambda^+$ must hold, we derive the system of equations that we have to solve to obtain v:

$$s_{\mathcal{B}} + \alpha e_q = c_{\mathcal{B}} - B^T(\lambda + \alpha v) \implies \alpha e_q = -B^T \alpha v \implies$$
$$e_q = -B^T v \implies v = -B^{-T} e_q$$

Comments

Let us now examine the mechanics of a single pivot step in the dual simplex method. The first task is to identify a leaving variable among the basic set. Since primal feasibility is violated, there must exist a basic variable with a negative value. Denote its index as q.

This variable will be forced to zero in the next step, which moves it out of the basis. The second task is to identify an entering variable from the nonbasic set. Denote its index as r.

The entering variable increases from zero while its corresponding dual slack decreases to zero. In this way, variables exchange roles between the basis and the nonbasis.

To maintain dual feasibility, the update of dual variables must be carefully orchestrated. We adjust the dual slacks for the basic set by adding α times the unit vector e_q, and simultaneously update the dual vector λ by adding α times another vector v. The scalar α is chosen so that one nonbasic slack becomes zero exactly at the right moment, enabling variable r to enter the basis.

The structure of the update ensures consistency with complementary slackness. To determine the vector v, we impose the defining relation between slacks and costs. The updated slacks for the basis must equal the cost vector of the basis minus the transpose of matrix B times the updated λ.

Substituting the proposed form yields an equation that simplifies to $e_q = -B^T v$. Solving this system gives $v = -B^{-T} e_q$.This explicit formula for v completes the update rule. Thus, each pivot step in the dual simplex adjusts both primal and dual variables in tandem, steadily reducing infeasibility and progressing toward optimality.

Dual Objective Change:

- Using $\lambda^+ = \lambda + \alpha v$, $v = -B^{-T}e_q$, $x_{\mathcal{B}} = B^{-1}b$, and $x_q = x_{\mathcal{B}}^T e_q$, derive the dual objective change:

$$b^T\lambda^+ = b^T\lambda + \alpha b^T v = b^T\lambda - \alpha b^T B^{-T}e_q = b^T\lambda - \alpha x_{\mathcal{B}}^T e_q = b^T\lambda - \alpha x_q.$$

Maximizing the Dual Objective:

- Since $x_q < 0$, choose α as large as possible to maximize the dual objective $b^T\lambda$, subject to the constraint $s_{\mathcal{N}}^+ \geq 0$.

Computing α

- Update $s_{\mathcal{N}}^+ = c_{\mathcal{N}} - N^T\lambda^+ = s_{\mathcal{N}} - \alpha w$, where $w = N^T v = -N^T B^{-T}e_q$. The largest α for which $s_{\mathcal{N}}^+ \geq 0$ is:

$$\alpha = \min_{j \in \mathcal{N}, w_j > 0} \frac{s_j}{w_j}.$$

Comments

When analyzing the dual simplex method at this stage, the focus is on how the dual objective changes as the iteration progresses. The dual variable is updated according to the formula $\lambda^+ = \lambda + \alpha v$, with $v = -B^{-T}e_q$.

Using the relationship $x_{\mathcal{B}} = B^{-1}b$ and $x_q = x_{\mathcal{B}}^T e_q$, we can derive how the dual objective $b^T\lambda$ changes. Substituting, we obtain that $b^T\lambda^+ = b^T\lambda - \alpha x_q$.

This simple but crucial identity tells us that the effect of increasing α is directly tied to the current value of x_q. Now, since x_q is negative, subtracting α times x_q actually increases the dual objective value. In other words, the negative component in the primal basic solution drives improvement in the dual objective. Therefore, the strategy is to choose α as large as possible while still respecting feasibility in the dual slack variables.

This feasibility constraint appears through the update $s_{\mathcal{N}}^+ = s_{\mathcal{N}} - \alpha w$, where $w = -N^T B^{-T}e_q$. To preserve nonnegativity of the dual slack variables, we must bound α so that $s_{\mathcal{N}}^+$ remains nonnegative. This gives the condition that α is less than or equal to the minimum ratio of $\frac{s_j}{w_j}$ across all nonbasic indices j with $w_j > 0$.

The minimum ratio rule here mirrors the logic of the standard simplex method but in the dual setting: it guarantees that one of the dual constraints becomes tight exactly at the point where α reaches its maximum permissible value.

Thus, the step simultaneously improves the dual objective while preserving feasibility, establishing the foundation for moving towards optimality.

Entering Index:

- Define the entering index r as the index at which the minimum in $\alpha = \min_{j \in \mathcal{N}, w_j > 0} \frac{s_j}{w_j}$ is achieved, so:

$$s_r^+ = 0 \quad \text{and} \quad w_r = A_r^T v > 0,$$

- where A_r is the r-th column of A.

Updating Primal Variables:

- Set $x_q^+ = 0$ for the leaving index q, allow x_r^+ nonzero for the entering index r. Define the direction d for $x_{\mathcal{B}}$:

$$Bd = \sum_{i \in \mathcal{B}} A_i d_i = A_r.$$

$$\sum_{i \in \mathcal{B}} A_i x_i = b \implies \sum_{i \in \mathcal{B}} A_i (x_i - \gamma d_i) + A_r \gamma = b, \quad \gamma = \frac{x_q}{d_q}.$$

Deriving Direction:

- Since d_q must be nonzero, we have $d_q < 0$, since:

$$d_q = d^T e_q = A_r^T B^{-T} e_q = -A_r^T v = -w_r < 0,$$

using $Bd = A_r$, $e_q = -B^T v$, and $w_r = A_r^T v > 0$.

Comments

At this stage, we determine the entering index in the dual simplex iteration. The entering index r is defined as the position at which the minimum ratio $\frac{s_j}{w_j}$ is achieved among all nonbasic indices with $w_j > 0$.

By this choice, s_r^+ becomes zero, and the corresponding w_r is strictly positive. This identifies the variable that should enter the basis. Importantly, the column A_r associated with this index provides the structure needed to compute the direction of movement in the primal space.

Once the entering variable is identified, we must update the primal solution to reflect the change in basis. The leaving variable x_q is set to zero, while x_r becomes nonzero. To maintain primal feasibility, the basic variables adjust along a direction vector d, defined by the equation $Bd = A_r$.

This equation ensures that the linear constraints remain satisfied after the exchange of basis elements. Next, to maintain consistency, the adjustment is parameterized by γ, defined as $\frac{x_q}{d_q}$.

This ratio captures how far we can move in the computed direction before the leaving variable becomes zero, which matches the rule that enforces primal feasibility. At the same time, we recognize that d_q cannot be zero, since otherwise the exchange would not reduce infeasibility.

Further analysis shows that d_q must in fact be negative. Specifically, $d_q = -w_r$, which is less than zero because w_r is positive. This condition confirms that γ is positive, ensuring that the update preserves feasibility and progresses towards optimality. Thus, the entering and leaving indices together guide the iteration toward an improved basis.

Updating Primal Variables:

- Since $x_q < 0$, it follows from $\gamma = \frac{x_q}{d_q}$ that $\gamma > 0$. Following $\sum_{i\in\mathcal{B}} A_i(x_i - \gamma d_i) + A_r\gamma = b$, define the updated vector x^+:

$$x_i^+ = \begin{cases} x_i - \gamma d_i, & \text{for } i \in \mathcal{B} \text{ with } i \neq q, \\ 0, & \text{for } i = q, \\ 0, & \text{for } i \in \mathcal{N} \text{ with } i \neq r, \\ \gamma, & \text{for } i = r. \end{cases}$$

Example: Dual Simplex Method

Consider the problem:

$$\min -4x_1 - 2x_2, \text{ s.t. } x_1 + x_2 + x_3 = 5,\ 2x_1 + \frac{1}{2}x_2 + x_4 = 8,\ x \geq 0.$$

- Basis: $\mathcal{B} = \{2,3\}$, $\mathcal{N} = \{1,4\}$. Compute $x_\mathcal{B} = B^{-1}b$, $B = \begin{bmatrix} 1 & 1 \\ \frac{1}{2} & 0 \end{bmatrix}$:

$$x_\mathcal{B} = \begin{bmatrix} 0 & 2 \\ 1 & -2 \end{bmatrix} \begin{bmatrix} 5 \\ 8 \end{bmatrix} = \begin{bmatrix} 16 \\ -11 \end{bmatrix}, \quad x_2 = 16, \quad x_3 = -11, \quad x_1 = x_4 = 0.$$

Comments

The update of the primal variables now becomes explicit. Since x_q is negative and d_q is also negative, then, as we mentioned earlier, their ratio $\gamma = \frac{x_q}{d_q}$ is strictly positive.

This scalar determines the amount by which the new entering variable increases. The updated primal solution x^+ is then defined piecewise: each remaining basic variable is adjusted by subtracting γ times the corresponding component of d, the leaving variable x_q is set to zero, all other nonbasic variables remain zero except for the new entering variable x_r, which is assigned the value γ.

This update guarantees that the new solution continues to satisfy all primal constraints while eliminating the infeasibility caused by the negative basic variable. The result is a feasible basis, at least with respect to the variable that previously violated nonnegativity.

Importantly, the update preserves the balance between primal and dual feasibility conditions, ensuring consistency of the simplex method framework.

To illustrate the procedure concretely, consider the example problem where the objective is to minimize $-4x_1 - 2x_2$, subject to two equality constraints with nonnegativity conditions. The chosen basis consists of variables 2 and 3, while variables 1 and 4 are nonbasic. By computing the basic solution $x_\mathcal{B} = B^{-1}b$, we obtain values sixteen and negative eleven for x_2 and x_3, respectively.

The negative value of x_3 signals the necessity of applying the dual simplex method. This example provides a concrete demonstration of the abstract derivations, showing how a negative component in the primal solution triggers a corrective iteration that not only restores feasibility but also improves the dual objective simultaneously.

- Since $x_3 < 0$, choose $q = 2$ (corresponding to the second component). Dual: Solve $B^T\lambda = c_{\mathcal{B}} = \begin{bmatrix}-2\\0\end{bmatrix}$, get $\lambda = \begin{bmatrix}0\\-4\end{bmatrix}$. Compute $s_{\mathcal{N}} = c_{\mathcal{N}} - N^T\lambda$:

$$s_{\mathcal{N}} = \begin{bmatrix}-4\\0\end{bmatrix} - \begin{bmatrix}1 & 2\\0 & 1\end{bmatrix}\begin{bmatrix}0\\-4\end{bmatrix} = \begin{bmatrix}4\\4\end{bmatrix}.$$

$$v = -B^{-T}e_2 = \begin{bmatrix}-1\\2\end{bmatrix}, \quad w = N^T v = \begin{bmatrix}3\\2\end{bmatrix}, \quad \alpha = \min\left\{\frac{4}{3}, \frac{4}{2}\right\} = 4/3, \quad r = 1.$$

- Compute direction: Solve $Bd = A_1 = \begin{bmatrix}1\\2\end{bmatrix}$:

$$d = \begin{bmatrix}0 & 2\\1 & -2\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix} = \begin{bmatrix}4\\-3\end{bmatrix}, \quad \gamma = \frac{x_q}{d_q} = \frac{-11}{-3} = \frac{11}{3}.$$

- Update: $x_2^+ = 16 - \frac{11}{3}\cdot 4 = \frac{4}{3}$, $x_3^+ = 0$, $x_1^+ = \frac{11}{3}$, $x_4^+ = 0$. New basis: $\mathcal{B} = \{2, 1\}$. And since all $x_i > 0$ the problem is solved with $z = -\frac{52}{3}$.

Comments

Continuing with the example, the negative value of x_3 identifies the leaving index q. Solving the dual system $B^T\lambda = c_{\mathcal{B}}$ yields the dual variable values, which in this case are zero and negative four.

Using these dual multipliers, we compute the reduced costs for the nonbasic variables as $s_{\mathcal{N}} = c_{\mathcal{N}} - N^T\lambda$, which gives positive values. This confirms dual feasibility while the primal is still infeasible. Next, the search direction is determined.

The vector v is computed as $-B^{-T}e_2$, resulting in entries negative one and two. From this, the vector $w = N^T v$ is obtained as entries three and two. The maximum permissible step length α is the minimum of $\frac{s_j}{w_j}$ for $w_j > 0$, giving four thirds in this case.

The corresponding entering index r is one, associated with the first variable. With this choice, we then compute the direction d by solving $Bd = A_1$, leading to the vector four, negative three.

Using the ratio test, γ equals negative eleven divided by negative three equals eleven thirds.

The update then produces new values for the primal variables: x_2 becomes four thirds, x_3 becomes zero, x_1 becomes eleven thirds, and x_4 remains zero. The new basis is thus formed by variables two and one.

Since all variables are now nonnegative, feasibility is restored, and the problem is solved with the optimal objective value negative fifty-two thirds.

This sequence of calculations demonstrates how the dual simplex method systematically removes infeasibilities while moving towards optimality in a finite number of steps.

Presolving in Linear Programming

Presolving (preprocessing) reduces the size of an LP problem before passing it to the solver. It's used in both simplex and interior-point methods.

LP Formulation for Discussion: $\min c^T x$, subject to $Ax = b, l \le x \le u$.
Some components $l_i (u_i)$ of the lower(upper) bound vector may be $-\infty$ (∞).

Key Presolving Techniques:

Row Singleton:

- If constraint k involves only variable j ($A_{kj} \neq 0$, $A_{ki} = 0$ for $i \neq j$).
- Set $x_j = b_k / A_{kj}$ and eliminate x_j.
- If x_j violates bounds ($x_j < l_j$ or $x_j > u_j$), the problem is infeasible.

Free Column Singleton:

- Variable x_j appears in only one equality constraint k ($A_{kj} \neq 0$, $A_{lj} = 0$ for all $l \neq k$) and is free ($l_j = -\infty$, $u_j = +\infty$).
- Use constraint k to eliminate x_j: $x_j = \dfrac{b_k - \sum_{p \neq j} A_{kp} x_p}{A_{kj}}$.
- Update cost vector: $c_p \leftarrow c_p - c_j A_{kp} / A_{kj}$ for all $p \neq j$.
- Dual variable for constraint k: $\sum_{i=1}^{m} A_{ij} \lambda_i = c_j \Rightarrow \lambda_k = c_j / A_{kj}$.

Comments

In large-scale linear programming, presolving plays a crucial role in reducing the complexity of the problem before it is passed to the main solver. The central idea is to exploit the structure of the model, eliminate redundancies, and simplify constraints in ways that preserve equivalence of the feasible set and the optimal solution. Consider the general form of a linear program: minimize $c^T x$, subject to $Ax = b$, with bounds $l \le x \le u$. Here, the bound vectors l and u may contain infinite components, reflecting variables that are unbounded in one direction.

One fundamental presolving rule is the row singleton case. If a constraint involves only a single variable, then that variable is uniquely determined: we set $x_j = b_k$ divided by A_{kj}. Once substituted, the variable and the associated row can be removed from the system. If this substitution forces x_j outside its lower or upper bounds, then the entire problem is immediately infeasible. This simple detection can save the solver from wasting effort on an impossible problem.

Another powerful rule is the free column singleton. Suppose a variable appears in only one equation and has no finite bounds, meaning $l_j = -\infty$ and $u_j = +\infty$. In this case, the variable can be eliminated directly by rewriting the equation to solve for x_j in terms of the remaining variables. This elimination also requires updating the objective function coefficients, ensuring equivalence in the reduced model. Moreover, the dual variable corresponding to the eliminated constraint can be derived explicitly as $\lambda_k = c_j$ divided by A_{kj}.

Through such transformations, presolving reduces problem size, accelerates convergence, and often determines feasibility early. In practice, these seemingly small reductions accumulate to substantial efficiency gains in large optimization tasks.

Presolving: Zero Rows/Columns & Forcing Constraints

Zero Rows and Columns in A*:*

Zero Row ($A_{ki} = 0$ **for all** i)**:**

- If $b_k = 0$, delete the row. λ_k can be arbitrary.
- If $b_k \neq 0$, the problem is infeasible.

Zero Column ($A_{kj} = 0$ **for all** k)**:**

- Determine x_j based on c_j and bounds $[l_j, u_j]$:
  - If $c_j < 0$: set $x_j = u_j$. If $u_j = +\infty$, problem is unbounded.
  - If $c_j > 0$: set $x_j = l_j$. If $l_j = -\infty$, problem is unbounded.
  - If $c_j = 0$: x_j can be any value in $[l_j, u_j]$.

Forcing/Dominated Constraints: Some constraints can only be satisfied if certain variables are at their bounds. **Example:** Consider $5x_1 - x_4 + 2x_5 = 10$ with bounds:

$$0 \leq x_1 \leq 1, \quad -1 \leq x_4 \leq 5, \quad 0 \leq x_5 \leq 2.$$

To satisfy the equality, we must have:

- $x_1 = 1$ (to maximize $5x_1$)
- $x_4 = -1$ (to minimize $-x_4$, effectively maximizing it as $+1$)
- $x_5 = 2$ (to maximize $2x_5$)

Thus, set $x_1 = 1, x_4 = -1, x_5 = 2$, and eliminate these variables and the constraint.

Comments

Beyond singletons, presolving also addresses cases where entire rows or columns in the coefficient matrix vanish. A zero row corresponds to an equation where all coefficients are zero. If the right-hand side b_k is also zero, then the constraint is redundant and can be deleted without consequence. However, if b_k is nonzero, then no assignment of variables can satisfy the equality, and the problem is infeasible. This quick detection prevents unnecessary processing of inconsistent models.

A zero column indicates a variable that never appears in any constraint. In this case, the variable influences only the objective function through its cost coefficient c_j and its bounds. If c_j is negative, the optimal strategy is to maximize the variable, pushing it to its upper bound u_j. If that bound is infinite, the objective becomes unbounded, and the problem has no finite solution. Conversely, if c_j is positive, the optimizer drives the variable to its lower bound l_j, unless that bound is $-\infty$, again leading to unboundedness. Finally, if c_j is zero, the variable is irrelevant to the objective and can take any feasible value within its bounds.

Another critical category consists of forcing or dominated constraints, where the structure of the equality combined with variable bounds forces specific variables to their extremal values. For instance, an equation such as $5x_1 - x_4 + 2x_5 = 10$ with restricted ranges can only be satisfied if $x_1 = 1, x_4 = -1, \text{and} x_5 = 2$. Thus, the constraint forces variables to particular values, enabling their elimination.

By exploiting such structural properties, presolving strips away redundancy, sharpens problem formulation, and simplifies the path to solution.

1. Dominated Constraints:

- Implicitly tightened bounds for a variable due to other constraints.
- Example: Consider $2x_2 + x_6 - 3x_7 = 8$ with bounds:

$$-10 \le x_2 \le 10, \quad 0 \le x_6 \le 1, \quad 0 \le x_7 \le 2.$$

- Rearranging and using bounds on x_6, x_7:

$$\begin{aligned} x_2 &= 4 - (1/2)x_6 + (3/2)x_7 \\ \Rightarrow \quad 4 - (1/2)(1) + (3/2)(0) &\le x_2 \le 4 - (1/2)(0) + (3/2)(2) \\ \Rightarrow \quad 7/2 &\le x_2 \le 7. \end{aligned}$$

- Original bounds $[-10, 10]$ on x_2 are redundant; x_2 is implicitly confined to $[7/2, 7]$.
- We can drop the original bounds and treat x_2 as a free variable (within its new implicit bounds).

Comments

Presolving also leverages the concept of dominated constraints, which serve to tighten variable bounds implicitly. This process refines feasible regions without altering the actual problem. Consider the equality $2x_2 + x_6 - 3x_7 = 8$. With the original bounds $-10 \le x_2 \le 10$, $0 \le x_6 \le 1$, and $0 \le x_7 \le 2$, one might initially think that x_2 can vary widely. However, rearranging the equation yields $x_2 = 4 - \frac{1}{2}x_6 + \frac{3}{2}x_7$. Substituting the bounds of x_6 and x_7 produces implicit limits: $\frac{7}{2} \le x_2 \le 7$. These are far narrower than the original interval, rendering the earlier bounds redundant.

This example illustrates how presolving captures hidden relationships between variables. Instead of treating the constraints and bounds as separate entities, presolving integrates them to uncover more precise restrictions. The benefit is twofold: solvers work with tighter feasible regions, reducing exploration of infeasible areas, and redundant inequalities are discarded, simplifying the optimization.

Such reasoning is particularly impactful in large-scale systems, where thousands of constraints interact. By uncovering dominated constraints, presolving reduces dimensionality not by eliminating variables outright but by sharpening their permissible ranges. This yields a model that is leaner, yet faithfully equivalent to the original.

Ultimately, dominated constraints exemplify how structural reasoning turns a superficially loose formulation into a tightly defined optimization task, ensuring solvers converge more quickly and robustly.

2. Recursive Application of Presolving:

- Eliminating variables/constraints can create new opportunities for further eliminations.
- Example:

$$3x_2 = 6 \quad \text{(Row Singleton)}$$
$$x_2 + 4x_5 = 10$$

- First constraint implies $x_2 = 2$. Eliminate x_2 and first constraint.
- Second constraint becomes $4x_5 = 10 - 2 = 8$, which is now a Row Singleton.
- This leads to $x_5 = 2$, eliminating x_5 and the second constraint.

Note: Detailed information on presolving is scarce due to its commercial value in LP software.

Comments

An important observation is that presolving operates not just once, but recursively. Eliminating a variable or a constraint can expose new opportunities for further simplification.

For example, suppose the system contains the equations $3x_2 = 6$ and $x_2 + 4x_5 = 10$. The first relation is a row singleton, immediately yielding $x_2 = 2$. Substituting this value eliminates both the variable and the equation. The second relation then reduces to $4x_5 = 8$, which itself is a row singleton, leading directly to $x_5 = 2$. Through this chain, two variables and both constraints vanish entirely.

This recursive nature is what makes presolving extremely powerful. A single reduction may propagate through the model, triggering cascades of simplifications. The end result is a much leaner system, one that preserves the same optimal solution but requires far less computational effort to solve.

However, despite its importance, detailed methodologies of presolving are not widely published. The reason lies in their commercial value: state-of-the-art optimization solvers often rely heavily on proprietary presolving routines that distinguish their performance from competitors. This scarcity of open literature highlights how central presolving has become in practical optimization.

In essence, presolving embodies the principle of problem transformation. Rather than tackling a complex formulation head-on, it reshapes the problem into a smaller, sharper, and more efficient form. Recursive application ensures no opportunity for reduction is missed, making presolving one of the cornerstones of modern large-scale linear programming.

Definition: Quadratic Program

A *quadratic program* (QP) is an optimization problem:

$$\min_x \; q(x) = \frac{1}{2} x^T G x + x^T c \text{ s.t.}$$
$$a_i^T x = b_i,\ i \in \mathcal{E},$$
$$a_i^T x \geq b_i,\ i \in \mathcal{I},$$

G: symmetric $n \times n$ matrix; c, x, a_i: vectors in $\mathbb{R}^n$; $\mathcal{E}, \mathcal{I}$: finite index sets.

- QPs arise as subproblems in sequential quadratic programming, augmented Lagrangian, and interior-point methods.
- Always solvable (or infeasible) in finite computation; effort depends on objective function and number of inequality constraints.
- *Convex QP*: G positive semidefinite, similar in difficulty to linear programs.
- *Strictly convex QP*: G positive definite.
- *Nonconvex QP*: G indefinite, may have multiple stationary points and local minima.

Convex QPs are computationally tractable; nonconvex QPs are more challenging.

Comments

Up to this point, we have focused on linear programming and related optimization frameworks. These methods allowed us to describe and solve a wide range of problems where both the objective function and the constraints were linear. However, many important applications in science, engineering, and economics involve objectives that are not simply linear but quadratic in nature. To handle such cases, we turn to the theory of quadratic programming, which generalizes linear programming by incorporating quadratic terms in the objective function while preserving linear constraints. This extension significantly broadens the modeling power of optimization.

A quadratic program is defined by an objective function that combines both quadratic and linear terms. Concretely, the goal is to minimize the function $\frac{1}{2} x^T G x + x^T c$, subject to linear constraints. Here, G is a symmetric matrix, c is a vector of coefficients, and the feasible set is determined by equalities and inequalities of the form $a_i^T x$ equal to or greater than b_i. This structure retains the clarity of linear programming but introduces curvature into the optimization landscape.

The nature of the problem depends heavily on the matrix G. If G is positive semidefinite, the objective function is convex, and the problem behaves much like a linear program in terms of difficulty. If G is strictly positive definite, we obtain a strictly convex optimization problem, which guarantees a unique solution. By contrast, if G is indefinite, the problem becomes nonconvex, leading to multiple stationary points, including possibly many local minima. This distinction is central in practice because convex quadratic programs can be solved reliably and efficiently, whereas nonconvex ones are substantially more challenging.

Quadratic programming plays an essential role not only as a standalone tool but also as a subproblem within more advanced methods. For example, sequential quadratic programming and interior-point algorithms rely on solving quadratic subproblems as intermediate steps toward tackling more complex nonlinear optimization tasks. The appeal of quadratic programming is that, while the problem may appear more difficult than linear programming, in its convex form it is computationally tractable and enjoys strong theoretical guarantees.

Example: Portfolio Optimization

Portfolio theory models the tradeoff between *risk* and *return*: higher expected return requires tolerating greater risks.

- Consider n investments with returns r_i, $i = 1, \ldots, n$, assumed to be random variables (often normal).
- Characterized by:
  - Expected value: $\mu_i = E[r_i]$,
  - Variance: $\sigma_i^2 = E[(r_i - \mu_i)^2]$ (larger $\sigma_i \Rightarrow$ riskier).
- Returns not independent; correlation between r_i, r_j:

$$\rho_{ij} = \frac{E[(r_i - \mu_i)(r_j - \mu_j)]}{\sigma_i \sigma_j}, \quad i, j = 1, \ldots, n.$$

- ρ_{ij} measures co-movement: $\rho_{ij} \approx 1 \Rightarrow$ returns track closely; negative $\rho_{ij} \Rightarrow$ opposite movements.
- Portfolio: fraction x_i of funds in investment i, $i = 1, \ldots, n$.
- Constraints (full investment, no short-selling): $\sum_{i=1}^{n} x_i = 1, \quad x \geq 0.$

Comments

One of the most celebrated applications of quadratic programming is found in portfolio optimization, a foundational concept in modern finance. The central idea is to model the tradeoff between risk and return: higher expected returns typically come with higher risk, and investors must balance these two objectives when allocating their wealth across different assets.

Consider a situation with n possible investments, each associated with a random return denoted by r_i. For each return, the expected value μ_i represents the average gain, while the variance σ_i^2 measures the volatility, or riskiness, of the investment. A higher variance indicates that returns are more uncertain. However, investments are rarely independent. The correlation coefficient ρ_{ij} between two returns r_i and r_j captures how their outcomes move together. A correlation close to one means the assets behave almost identically, while a negative correlation suggests that gains in one tend to be offset by losses in the other.

A portfolio is represented by a vector x, where each component x_i specifies the fraction of wealth allocated to asset i. Standard constraints ensure that all available capital is invested — meaning the sum of x_i equals one — and that allocations are nonnegative, ruling out short selling. This simple framework already demonstrates the richness of the model, as the introduction of correlation among assets provides investors with the possibility of diversification: spreading investments across imperfectly correlated assets reduces overall risk without necessarily lowering expected return.

Portfolio Optimization (continued)

- Portfolio return: $R = \sum_{i=1}^{n} x_i r_i$.
- Expected return: $E[R] = E\left[\sum_{i=1}^{n} x_i r_i\right] = \sum_{i=1}^{n} x_i E[r_i] = \sum_{i=1}^{n} x_i \mu_i = x^T \mu$.
- Variance:
$$Var[R] = \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j \sigma_i \sigma_j \rho_{ij} = x^T G x,$$
where $G_{ij} = \rho_{ij}\sigma_i\sigma_j$ is the symmetric positive semidefinite *covariance matrix*.

Goal: Maximize expected return $x^T\mu$ while minimizing variance $x^T G x$.

- Markowitz model combines objectives using risk tolerance $\kappa \geq 0$:
$$\max\ x^T\mu - \kappa x^T G x, \quad \text{s.t.} \sum_{i=1}^{n} x_i = 1,\ x \geq 0.$$
- κ reflects investor preference:
  - Large κ: Conservative, emphasizes low variance (risk).
  - Small κ: Daring, prioritizes high return.
- Challenge: Estimating μ_i, σ_i^2, ρ_{ij} using historical data and financial insights.

Markowitz model balances return and risk via κ.

Comments

The mathematical formulation of portfolio return builds directly on these definitions. The portfolio return R is a weighted sum of the individual asset returns, where the weights are the allocations x_i. Its expected value, E, is simply the dot product $x^T\mu$, reflecting the linear combination of expected asset returns. The variance of the portfolio, however, captures interactions between all pairs of assets.

Specifically, it can be written as $x^T G x$, where G is the covariance matrix, built from the correlations and standard deviations of individual assets. This quadratic structure makes portfolio optimization a natural quadratic programming problem.

The classical Markowitz model seeks to balance these competing objectives. It introduces a parameter κ, called risk tolerance, that determines the tradeoff. The optimization goal is to maximize expected return minus κ times the variance. A large value of κ corresponds to a conservative investor who places high emphasis on reducing risk. A small value of κ corresponds to a more aggressive investor willing to tolerate volatility in pursuit of higher returns.

A crucial challenge lies in estimating the inputs: the expected returns μ_i, the variances σ_i^2, and the correlations ρ_{ij}. In practice, these are derived from historical data and financial modeling, but they are subject to uncertainty and estimation error. Nevertheless, the Markowitz framework has had enormous influence because it formalizes the intuition that diversification reduces risk and because it provides a systematic method for designing portfolios that reflect different levels of risk aversion.

Equality-Constrained QP

Equality-constrained QPs, critical for general QP algorithms, are formulated as:

$$\min_{x} \frac{1}{2} x^T G x + x^T c \text{ s.t. } Ax = b,$$

where G is a symmetric $n \times n$ matrix, A is an $m \times n$ Jacobian ($m \leq n$, full row rank), and $b \in \mathbb{R}^m$. The constraint $Ax = b$ ensures feasibility, with full rank guaranteeing consistency.

- **First-order optimality (KKT system)**: Solution x^* requires a Lagrange multiplier vector λ^* satisfying the KKT system:

$$\begin{bmatrix} G & -A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -c \\ b \end{bmatrix}.$$

  This system balances the gradient $\nabla q(x^*) = Gx^* + c$ with constraint forces $A^T \lambda^*$.

- Computational approach: Express $x^* = x + p$, where p is the step from estimate x. Solve:

$$\begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} -p \\ \lambda^* \end{bmatrix} = \begin{bmatrix} c + Gx \\ Ax - b \end{bmatrix}.$$

Here, $c + Gx$ is the gradient at x, and $Ax - b$ measures constraint violation and the matrix is called the Karush-Kuhn-Tucker (KKT) matrix.

Comments

Equality-constrained quadratic programs represent a very important subclass of quadratic optimization problems. They consist of minimizing a quadratic function subject only to equality constraints. Concretely, the objective has the form $\frac{1}{2} x^T G x + x^T c$, where G is a symmetric matrix and c is a vector. The constraints are linear equalities written as $Ax = b$. Here A is an $m \times n$ matrix of full row rank, and b is a vector in m dimensions. The full rank assumption ensures that the equalities are consistent and that the feasible region is well-defined.

To find an optimal solution, one uses the first-order optimality conditions, often called the Karush-Kuhn-Tucker or KKT system. The idea is that at a solution x^*, there exists a multiplier vector λ^* such that the gradient of the objective, which is $Gx^* + c$, is exactly balanced by the constraint forces $A^T \lambda^*$. While keeping the restrictions on the x^*: $Ax^* = b$.

From a computational perspective, one often updates an approximate solution x by considering a correction step p, with the new solution being $x + p$. The corresponding linear system can again be written in block form, now with G in the top-left and A^T in the top-right, and A and zero on the bottom row.

On the right-hand side, the top block is the gradient at the current x, namely $c + Gx$, and the bottom block is the constraint residual, namely $Ax - b$. Solving this system produces both the step direction p and the updated multipliers. This KKT matrix, while structured, is often large and indefinite, making its numerical solution a central challenge in quadratic programming.

The KKT matrix arises in equality-constrained QPs. Let Z be an $n \times (n-m)$ matrix with full rank, forming a basis for the null space of A (i.e., $AZ = 0$).

Lemma 10: KKT Nonsingularity

If A has full row rank and $Z^T GZ$ is positive definite, then the KKT matrix

$$K = \begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix}$$

is nonsingular, ensuring a unique solution (x^*, λ^*) to the KKT system.

Proof: Suppose $\exists w, v$ such that

$$\begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} w \\ v \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

We have from this equation (since $Aw = 0$), that

$$0 = \begin{bmatrix} w \\ v \end{bmatrix}^T \begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} w \\ v \end{bmatrix} = w^T G w.$$

Since w lies in the null space of A, it can be written as $w = Zu$ for some vector $u \in \mathbb{R}^{n-m}$.

Comments

When analyzing equality-constrained quadratic programs, one of the central questions is whether the Karush-Kuhn-Tucker, or KKT, system admits a unique solution. The KKT matrix in this setting is built as a block structure with the Hessian matrix, denoted by capital G, in the upper-left block, the transpose of capital A in the upper-right block, capital A itself in the lower-left block, and a block of zeros in the lower-right. The uniqueness of the solution depends critically on two conditions.

First, the matrix capital A must have full row rank, meaning its rows are linearly independent.

Second, the reduced Hessian, which takes the form $Z^T GZ$, must be positive definite. Here, Z is a basis matrix for the null space of A, that is, any vector w satisfying $Aw = 0$ can be written as Z times some vector u.

The proof of nonsingularity is straightforward but important. Suppose the KKT system has a nontrivial solution vector, written as w stacked with v. Multiplying through shows that $w^T Gw = 0$. Since w lies in the null space of A, we can express it as Z times u.

From $w = Zu$, we have

$$0 = w^T G w = u^T Z^T G Z u.$$

Since $Z^T G Z$ is positive definite, $u = 0$, so $w = 0$. Then, $A^T v = 0$, and full row rank of A implies $v = 0$. Thus, only $w = 0$, $v = 0$ satisfies the KKT null space, proving nonsingularity. □

Example: Equality-Constrained QP

Consider:

$$\min\ 3x_1^2+2x_1x_2+x_1x_3+2.5x_2^2+2x_2x_3+2x_3^2-8x_1-3x_2-3x_3 \text{ s.t. } x_1+x_3=3,\ x_2+x_3=0.$$

Form: $G = \begin{bmatrix} 6 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 4 \end{bmatrix}$, $c = \begin{bmatrix} -8 \\ -3 \\ -3 \end{bmatrix}$, $A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$, $b = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$. Solution:

$x^* = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$, $\lambda^* = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$, with G positive definite, $Z = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}$.

Positive definite G and full rank A ensure unique QP solution.

Comments

Substituting gives $u^T Z^T G Z u = 0$. But if the reduced Hessian is positive definite, the only solution is $u = 0$, hence $w = 0$.

Then from $A^T v = 0$, and the full row rank of A, it follows that v must also vanish.

The only solution is the trivial one, proving that the KKT matrix is nonsingular. This establishes the foundation for the existence of a unique solution pair, denoted by x^* and λ^*.

To see how this theory works in practice, consider an equality-constrained quadratic program in three variables.

The objective is a quadratic function: $3x_1^2 + 2x_1x_2 + x_1x_3 + 2.5x_2^2 + 2x_2x_3 + 2x_3^2 - 8x_1 - 3x_2 - 3x_3$.

The constraints are linear equalities: $x_1 + x_3 = 3$, and $x_2 + x_3 = 0$. In matrix notation, the Hessian G is a three-by-three symmetric matrix with entries six, two, one in the first row, two, five, two in the second, and one, two, four in the third.

The vector c is minus eight, minus three, minus three.

The constraint matrix A is two by three with first row one, zero, one and second row zero, one, one.

The right-hand side vector b is three and zero.

Solving this quadratic program yields the optimal point x^* equal to the vector two, minus one, one, and the corresponding multipliers $\lambda^* = (3, -2)^T$.

The Hessian is positive definite, and the null-space basis matrix Z can be taken as the vector $(-1, -1, 1)^T$. This example illustrates that with a positive definite G and a full-rank A, the problem admits a unique solution.

Theorem 35: Global Solution

If A has full row rank and assume that the reduced-Hessian matrix Z^TGZ is positive definite, then x^* satisfying the KKT system is the unique global solution of the equality-constrained QP.

Proof: For feasible x ($Ax = b$), define $p = x^* - x$, so $Ap = 0$. The objective expands as:

$$q(x) = \frac{1}{2}(x^* - p)^T G(x^* - p) + c^T(x^* - p) = \frac{1}{2}p^T Gp - p^T Gx^* - c^T p + q(x^*).$$

Since $Gx^* = -c + A^T\lambda^*$ (satisfying the KKT system) and $Ap = 0$, we get:

$$q(x) = \frac{1}{2}p^T Gp - p^T(-c + A^T\lambda^*) - c^T p + q(x^*) = \frac{1}{2}p^T Gp + q(x^*).$$

As $p = Zu$ we have $q(x) = \frac{1}{2}u^T Z^T GZu + q(x^*)$ and positive definiteness of Z^TGZ ensures $q(x) > q(x^*)$ unless $p = 0$ (i.e., $x = x^*$). □

- Positive semidefinite Z^TGZ with zero eigenvalues: x^* is a local minimizer.
- Negative eigenvalues in Z^TGZ: x^* is a stationary point.

Positive definite Z^TGZ ensures a unique global minimum.

Comments

The discussion of nonsingularity sets the stage for a deeper result: the global solution theorem for equality-constrained quadratic programs.

It asserts that if the constraint matrix A has full row rank and the reduced Hessian Z^TGZ is positive definite, then any point x^* that satisfies the KKT system is not only a stationary point but the unique global minimizer of the problem. The proof relies on examining any feasible vector x, that is, any x with $Ax = b$. Let p be defined as $x^* - x$, so that $Ap = 0$.

Expanding the quadratic objective function q(x) leads to a decomposition. Specifically, q(x) equals $\frac{1}{2}p^T Gp - p^T Gx^* - c^T p + q(x^*)$. From the KKT system we know that $Gx^* = -c + A^T\lambda^*$.

Since $Ap = 0$, the cross terms vanish, and we are left with $q(x) = \frac{1}{2}p^T Gp + q(x^*)$.

By expressing p as Zu, the expression becomes $\frac{1}{2}u^T Z^T GZu + q(x^*)$.

If the reduced Hessian is positive definite, this expression is strictly greater than $q(x^*)$ unless $u = 0$.

Thus, no feasible point has a lower objective value than x^*, and the minimizer is unique.

If Z^TGZ is only positive semidefinite, the situation weakens: x^* remains a local minimizer but not necessarily unique.

If Z^TGZ has negative eigenvalues, then x^* is merely a stationary point without optimality guarantees.

This reasoning emphasizes how definiteness conditions translate into global solution properties.

The KKT system for equality-constrained QPs is indefinite for $m \geq 1$.

The inertia of a symmetric matrix K is defined as $\text{inertia}(K) = (n_+, n_-, n_0)$, where n_+, n_-, n_0 are the numbers of positive, negative, and zero eigenvalues.

Theorem 36: KKT Inertia

For $K = \begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix}$ with A of rank m, the inertia is:

$$\text{inertia}(K) = \text{inertia}(Z^T G Z) + (m, m, 0).$$

If $Z^T G Z$ is positive definite, $\text{inertia}(K) = \langle n, m, 0 \rangle$.

Proof in: N. I. M. Gould, *On practical conditions for the existence and uniqueness of solutions to the general equality quadratic programming problem*, Math. Program., 32 (1985), pp. 90–99 (see Lemma 3.4, p. 96).

Inertia informs efficient KKT system solution strategies.

Comments

The final step is to connect nonsingularity and optimality with the spectral structure of the KKT system itself. Because the KKT matrix combines both the Hessian and the constraint structure, it is not positive definite in general. For any problem with at least one equality constraint, that is, when $m \geq 1$, the KKT system is indefinite.

This means its spectrum contains both positive and negative eigenvalues. To analyze this systematically, we use the concept of inertia of a symmetric matrix. The inertia is a triplet consisting of the numbers of positive, negative, and zero eigenvalues.

For the KKT matrix, defined as the block matrix with G and A^T in the top row, and A and zero in the bottom row, the inertia can be expressed in terms of the reduced Hessian.

Specifically, the inertia of the KKT matrix equals the inertia of $Z^T G Z$ plus the triplet $(m, m, 0)$. An important corollary arises when the reduced Hessian is positive definite. In that case, $Z^T G Z$ contributes only positive eigenvalues, and the inertia of the full KKT matrix becomes $(n, m, 0)$.

Here $n - m$ counts the positive eigenvalues corresponding to the reduced Hessian, while and the remaining m of positive eigenvalues and m of negative eigenvalues counts both positive and negative directions associated with the constraints. This result, due to Nicholas Gould in the mid-1980s, is crucial for practical algorithms, because inertia reveals whether a factorization of the KKT system is numerically stable and whether the resulting search directions are well-defined. Understanding inertia therefore provides both theoretical insight and practical guidance for solving quadratic programs efficiently.

Solving the KKT System

The KKT matrix is indefinite for $m \geq 1$, ruling out Cholesky factorization. Gaussian elimination with partial pivoting ignores symmetry, so symmetric indefinite factorization is preferred:

$$P^T K P = L B L^T,$$

where P is a permutation matrix, L is unit lower triangular, and B is block-diagonal (1×1 or 2×2 blocks).

- Solution steps for KKT system:

$$\text{solve } Lz = P^T \begin{bmatrix} c + Gx \\ Ax - b \end{bmatrix} \text{ for } z;$$

$$\text{solve } B\hat{z} = z \text{ for } \hat{z};$$

$$\text{solve } L^T \bar{z} = \hat{z} \text{ for } \bar{z};$$

$$\text{set } \begin{bmatrix} -p \\ \lambda^* \end{bmatrix} = P\bar{z}, \text{ (recall: } x^* = x + p)$$

- Permutations (P, P^T) are inexpensive; B systems are small; triangular solves with L, L^T depend on sparsity.

Symmetric factorization is efficient but costly if L loses sparsity.

Comments

The Karush–Kuhn–Tucker system involves an indefinite matrix, which immediately eliminates the possibility of Cholesky factorization. Instead, symmetric indefinite factorization is the natural choice. This approach rearranges the system using permutation matrices and reduces it to triangular solves with a block-diagonal middle matrix.

The procedure unfolds in three simple steps: forward substitution, block solve, and backward substitution. Each of these respects the sparsity structure, making the method efficient when the underlying problem is sparse.

The final step reconstructs both the primal direction and the Lagrange multipliers. What is remarkable here is that the expensive work is confined to factorizations, while the application of permutations and triangular solves is relatively cheap. Nonetheless, if the triangular factor loses too much sparsity during factorization, the overall method can become costly.

Hence, there is a delicate balance: symmetry is preserved, stability is ensured, but computational effort depends strongly on how much structure the factorization destroys.

Schur-Complement Method for KKT

Assuming G is positive definite, multiply the first KKT equation by AG^{-1} and subtract the second to get:

$$(AG^{-1}A^{T})\lambda^{*} = AG^{-1}(c + Gx) - (Ax - b).$$

Solve this for λ^{*}, then recover p from:

$$Gp = A^{T}\lambda^{*} - (c + Gx).$$

Block Gaussian elimination on the KKT matrix yields:

$$\begin{bmatrix} G & A^{T} \\ 0 & -AG^{-1}A^{T} \end{bmatrix}.$$

- Effective when: G is well-conditioned (e.g., diagonal); G^{-1} is known; or m is small (so that the number of backsolves needed to form the matrix $AG^{-1}A^{T}$ is not too large).

Schur-Complement method leverages G^{-1} for efficient KKT solutions.

Comments

When the Hessian matrix G is positive definite, the KKT system can be represented as two subsystems separated through the Schur complement. Multiplying the first KKT equation by the AG^{-1}, followed by subtraction, produces a reduced system in terms of the multipliers λ^{*}. The central object here is the matrix $AG^{-1}A^{T}$ which captures the constraint interactions.

Once the multipliers are determined from this reduced system, the search direction for the primal variables is recovered by solving a linear system with G. Conceptually, this approach transforms the saddle-point structure into two coupled problems: one of size equal to the number of constraints, and another of size equal to the number of primal variables.

Efficiency arises when G is simple, such as diagonal, or when its inverse is easily available. In such situations, forming and solving with the Schur complement is highly attractive. However, when G is large and dense, explicitly building $AG^{-1}A^{T}$ may be too expensive.

Therefore, the method shines when the number of constraints is modest and the Hessian has a structure that makes inversion cheap.

The matrix $AG^{-1}A^T$ is the Schur complement of G in the KKT matrix $K = \begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix}$. Block elimination on the KKT system yields the Schur-Complement method equations.

Explicit inverse of K:

$$\begin{bmatrix} G & A^T \\ A & 0 \end{bmatrix}^{-1} = \begin{bmatrix} C & E \\ E^T & F \end{bmatrix},$$

- $C = G^{-1} - G^{-1}A^T(AG^{-1}A^T)^{-1}AG^{-1}$,
- $E = G^{-1}A^T(AG^{-1}A^T)^{-1}$,
- $F = -(AG^{-1}A^T)^{-1}$.

Multiplying the KKT system's right-hand side by this inverse, grouping terms, recovers the Schur-Complement method.

Explicit inverse links directly to Schur-Complement solution.

Comments

The Schur complement provides not only a computational strategy but also a clean algebraic perspective.

If we consider the KKT matrix as a block matrix, its inverse can be written explicitly in terms of G^{-1} and the inverse of the Schur complement $AG^{-1}A^T$.

Each block of this inverse has an interpretation: the C block corresponds to adjustments in the primal variables, the E block links constraints to variables, and the F block represents interactions between constraints themselves. By multiplying the right-hand side of the KKT equations with this inverse, we recover exactly the Schur-complement equations previously derived.

Thus, the inverse form and the elimination approach are simply two views of the same underlying mechanism.

The explicit inverse is rarely formed in practice, because it is dense and costly. Yet, it provides valuable theoretical insight, clarifying how constraints project onto the variable space and how feasibility and optimality conditions are enforced simultaneously.

The null-space method, unlike Schur-Complement, does not require G nonsingularity, only A full row rank and $Z^T GZ$ positive definite. It uses a null-space basis Z to decouple the KKT system into smaller systems.

Decompose p as:

$$p = Yp_Y + Zp_Z,$$

where Z is $n \times (n - m)$ null-space matrix ($AZ = 0$), Y is $n \times m$ such that $[Y \mid Z]$ is nonsingular, $p_Y \in \mathbb{R}^m$, $p_Z \in \mathbb{R}^{n-m}$. Substituting into the KKT system's second equation gives:

$$(AY)p_Y = -(Ax - b).$$

- Yp_Y is a particular solution to $A(x + p) = b$; Zp_Z moves along constraints.

Null-space method leverages Z for broader applicability.

Comments

The null-space method offers an alternative to the Schur complement approach for solving the Karush–Kuhn–Tucker, or KKT, system. Unlike the Schur complement, it does not require the Hessian matrix, denoted as G, to be invertible. Instead, the only requirements are that the constraint matrix A has full row rank, and that the matrix $Z^T GZ$ is positive definite. Here Z is a basis for the null space of A, meaning that $AZ = 0$.

The key idea is to split the step vector p into two parts: a particular part that ensures feasibility, and a null-space part that moves within the feasible region. Formally, we write $p = Yp_Y + Zp_Z$. The columns of Y are chosen so that together with the columns of Z they form a full basis of the entire space. Substituting this decomposition into the KKT equations shows that the component Yp_Y ensures that $A(x + p) = b$ is satisfied. Meanwhile, the component Zp_Z represents directions that stay within the feasible set, since $AZ = 0$.

Thus, the null-space method eliminates the need to solve for all variables simultaneously. Instead, it first enforces feasibility through Yp_Y, and then optimizes within the feasible set by solving for p_Z. This makes the method broadly applicable even when G is singular or poorly conditioned.

Null-Space Method (Continued)

Given A rank m and nonsingular $[Y \mid Z]$, AY is an $m \times m$ nonsingular matrix, as $A[Y \mid Z] = [AY \mid 0]$ has rank m. Solve $(AY)p_Y = -(Ax - b)$ for p_Y, the component of p in the range of Y.

Substitute $p = Yp_Y + Zp_Z$ into the KKT system's first equation:

$$-GYp_Y - GZp_Z + A^T\lambda^* = c + Gx.$$

Multiply by Z^T (since $AZ = 0$) to isolate p_Z:

$$(Z^TGZ)p_Z = -Z^TGYp_Y - Z^T(c + Gx).$$

Solve for p_Z using Cholesky factorization of positive definite Z^TGZ, then compute total step $p = Yp_Y + Zp_Z$.

- For λ^*, solve $(AY)^T\lambda^* = Y^T(c + Gx + Gp)$, using the computed p.

Null-space method efficiently decouples KKT system via Y and Z.

Comments

The construction of the null-space method becomes clearer once we look at how the system of equations decouples.

Recall that we decomposed the step vector p into $Yp_Y + Zp_Z$. Because the block matrix formed by concatenating Y and Z is nonsingular, the submatrix AY is square and invertible. This allows us to compute the component p_Y uniquely by solving the linear system $(AY)p_Y = -(Ax - b)$. In other words, p_Y is the correction that ensures feasibility of the equality constraints.

Next, we substitute the full decomposition $p = Yp_Y + Zp_Z$ into the first block of the KKT equations, which originally reads $-Gp + A^T\lambda^* = c + Gx$. After substitution, this becomes $-GYp_Y - GZp_Z + A^T\lambda^* = c + Gx$. To separate the null-space component, we multiply the equation by Z^T. Since $AZ = 0$, this operation eliminates the dependence on λ^* and isolates p_Z.

The resulting equation is $(Z^TGZ)p_Z = -Z^TGYp_Y - Z^T(c + Gx)$. The matrix Z^TGZ is symmetric and positive definite, and therefore well-suited for numerical solution using Cholesky factorization.

Once p_Z is obtained, the complete step is reconstructed as $Yp_Y + Zp_Z$. Finally, with p known, the multipliers λ^* can be recovered by solving the smaller system involving the transpose of AY. Altogether, the method achieves an elegant decoupling: feasibility is handled through p_Y, optimality is enforced through p_Z, and the multipliers are obtained as a final adjustment.

Example: Applying Null-Space Method

For the QP with G, c, A, b as defined in previous Example: $G = \begin{bmatrix} 6 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 4 \end{bmatrix}$, $c = \begin{bmatrix} -8 \\ -3 \\ -3 \end{bmatrix}$, $A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$, $b = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$, choose: $Y = \begin{bmatrix} 2/3 & -1/3 \\ -1/3 & 2/3 \\ 1/3 & 1/3 \end{bmatrix}$, $Z = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}$, $AY = I$. At $x = (0,0,0)^T$ compute $Ax - b = -b$, $c + Gx = (-8,-3,-3)^T$. Solve $(AY)p_Y = b$ to get $p_Y = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$, from $(Z^T G Z)p_Z = -Z^T G Y p_Y - Z^T(c + Gx)$ get $p_Z = [0]$. Total step: $p = Yp_Y + Zp_Z = (2,-1,1)^T$.

- Solve $(AY)^T \lambda^* = Y^T(c + Gx + Gp)$ to obtain $\lambda^* = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$.
- Solution: $x^* = x + p = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$, $\lambda^* = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$.

Null-space method computes x^* and λ^* efficiently.

Comments

To illustrate the mechanics of the null-space method, let us work through a concrete quadratic programming problem. The Hessian matrix G is given as the three-by-three matrix with rows: six, two, one; two, five, two; one, two, four. The cost vector c is negative eight, negative three, negative three.

The constraint matrix A has two rows: the first row is one, zero, one; the second row is zero, one, one. The right-hand side vector b is three, zero. A suitable choice of bases is Y equal to the three-by-two matrix with columns two-thirds, minus one-third, one-third and minus one-third, two-thirds, one-third; and Z equal to the three-by-one vector minus one, minus one, one. With this choice, AY equals the identity matrix. We begin at the point $x = (0,0,0)^T$.

Then $Ax - b$ equals minus b, so the feasibility correction is determined by solving $(AY)p_Y = b$. Since AY is the identity, this immediately gives p_Y equal to the vector three, zero.

Next, we compute the null-space component by solving the reduced system $(Z^T G Z)p_Z = -Z^T G Y p_Y - Z^T(c + Gx)$. Substitution shows that the right-hand side vanishes, so $p_Z = 0$. Therefore the full step is $Yp_Y + Zp_Z$, which evaluates to the vector two, negative one, one.

Finally, to compute the multipliers, we solve the system $(AY)^T \lambda^* = Y^T(c+Gx+Gp)$. This yields λ^* equal to the vector three, negative two. Thus, the optimal solution of this quadratic program is x^* equal to two, negative one, one, with associated multipliers λ^* equal to three and negative two. The example demonstrates how the method enforces feasibility via p_Y, finds the optimal feasible step using p_Z, and then recovers the multipliers in a clean sequence of linear solves, each of relatively small dimension.

Iterative Null-Space Method

Iterative methods suit large KKT systems, leveraging parallelization. CG is unstable for indefinite systems, so Krylov methods (GMRES, QMR, LSQR) or CG on the reduced system are preferred, assuming $Z^T G Z$ positive definite.

Decompose solution: $x^* = Y x_Y + Z x_Z$, where $x_Y \in \mathbb{R}^m$, $x_Z \in \mathbb{R}^{n-m}$, Z is $n \times (n-m)$ null-space matrix ($AZ = 0$), Y is $n \times m$ such that $[Y \mid Z]$ is nonsingular. Constraints $Ax = b$ give: $AYx_Y = b$.
Substitute into the QP to get the reduced problem:

$$\min_{x_Z} \frac{1}{2} x_Z^T Z^T G Z x_Z + x_Z^T c_z, \text{ where } c_z = (2Z^T G Y x_Y + Z^T c).$$

Solve $(Z^T G Z) x_Z = -c_z$ using CG, then compute x^*.

- Preconditioner W_{zz} enhances CG convergence.
- Assumes A full rank, $Z^T G Z$ positive definite.

CG on reduced system efficiently solves equality-constrained QP.

Comments

When the number of variables is very large, direct methods for solving the Karush–Kuhn–Tucker system become prohibitively expensive. In such cases, iterative approaches are preferable. Among the iterative methods, the conjugate gradient method is widely used, but it is only reliable when applied to positive definite systems.

Since the KKT system is indefinite, we cannot apply conjugate gradient directly. Instead, we work with the reduced null-space formulation. The idea is to decompose the solution into two components: a feasible part that enforces the constraints and a null-space part that optimizes within the feasible set. More precisely, the solution vector x^* is written as $Yx_Y + Zx_Z$, where the matrix Z has full rank, and $AZ = 0$.

The equality constraints $Ax = b$ then reduce to $(AY)x_Y = b$. Once this feasible component is determined, the optimization reduces to minimizing a quadratic function of the null-space variables. The reduced objective function is $\frac{1}{2} x_Z^T Z^T G Z x_Z + x_Z^T c_z$. The vector c_z equals $2Z^T G Y x_Y + Z^T c$. The resulting linear system is $(Z^T G Z) x_Z = -c_z$.

This system is symmetric and positive definite under standard assumptions that A has a full rank and $Z^T G Z$ is positive definite, which makes it suitable for solution by conjugate gradient.

Once x_Z is found, the full solution x^* is reconstructed as $Yx_Y + Zx_Z$. Preconditioning plays a crucial role: a carefully chosen preconditioner matrix W_{zz} can dramatically accelerate convergence (recall that the preconditioner is a special matrix used in the conjugate gradient method to improve the convergence rate. We discussed this earlier.).

Preconditioned CG for Reduced System

Goal: Solve $(Z^T G Z)x_Z = -c_z$ iteratively using CG with preconditioner W_{zz} (which is assumed to be given) for efficiency. **Algorithm: Preconditioned CG**

1: Choose initial x_Z;
2: Compute $r_z = Z^T G Z x_Z + c_z$, $g_z = W_{zz}^{-1} r_z$, $d_z = -g_z$;
3: for iterations until $r_z^T W_{zz}^{-1} r_z$ is sufficiently small do
4: $\quad \alpha \leftarrow r_z^T g_z / d_z^T Z^T G Z d_z$;
5: $\quad x_Z \leftarrow x_Z + \alpha d_z$;
6: $\quad r_z^+ \leftarrow r_z + \alpha Z^T G Z d_z$;
7: $\quad g_z^+ \leftarrow W_{zz}^{-1} r_z^+$;
8: $\quad \beta \leftarrow (r_z^+)^T g_z^+ / r_z^T g_z$;
9: $\quad d_z \leftarrow -g_z^+ + \beta d_z$;
10: $\quad g_z \leftarrow g_z^+$, $r_z \leftarrow r_z^+$;
11: end for

Note: No explicit $Z^T G Z$ or Z needed as long as we are able to compute products of Z and Z^T with arbitrary vectors. These products are often cheaper to compute than Z itself (for sparse Z).

Comments

At this stage, we arrive at the reduced system $(Z^T G Z)x_Z = -c_z$. This is exactly the type of system for which the conjugate gradient method is designed: it is symmetric and positive definite.

Instead of solving it by direct factorization, which would be too expensive for large-scale problems, we employ the preconditioned conjugate gradient algorithm that we already studied earlier in the course.

So, conceptually, nothing new is happening here—we are simply applying Algorithm 10, the preconditioned conjugate gradient method, but now in the special setting of quadratic programming with equality constraints. Let me briefly recall the main idea behind conjugate gradients. Ordinary gradient descent follows the steepest descent direction at each step, but this often leads to slow zig-zagging.

The conjugate gradient method improves on this by building a sequence of search directions that are conjugate with respect to the matrix of the system. In other words, each new direction is chosen not only to reduce the error but also to avoid undoing progress made in earlier steps. This property guarantees convergence to the solution in at most n steps, and often much faster in practice.

The role of the preconditioner W_{zz} is to accelerate this process. If $Z^T G Z$ has eigenvalues spread across a wide range, the convergence of conjugate gradients can be painfully slow. The preconditioner reshapes the system so that the eigenvalues are better clustered, which leads to faster convergence.

Importantly, we do not need to build $Z^T G Z$ explicitly. It is sufficient to compute products of Z and Z^T with vectors, which can be done efficiently in structured or sparse cases. Thus, the message here is simple: we reduce the KKT problem to a symmetric positive definite system in the null space, and then solve it using a preconditioned conjugate gradient method that we already understand well.

Projected CG Method

- Preconditioner $W_{zz} = Z^T HZ$, with H symmetric and $Z^T HZ$ positive definite, clusters eigenvalues of $W_{zz}^{-1/2}(Z^T GZ)W_{zz}^{-1/2}$ to accelerate CG convergence.
- Ideal case: $W_{zz} = Z^T GZ$, making the system trivial.
- Projected CG solves $AYx_Y = b$ and updates x_Z in $x = Zx_Z + Yx_Y$ implicitly in n-dimensional space, using projection matrix:

$$P = Z(Z^T HZ)^{-1}Z^T.$$

Operates with n-vectors: $x = Zx_Z + Yx_Y$, $Z^T r = r_z$, $g = Zg_z$, $d = Zd_z$. This avoids explicit Z, reducing computational cost.

- Robust to ill-conditioned A or poor Z, as P projects onto null space.
- Solves reduced system $(Z^T GZ)x_Z = -c_z$ using matrix-vector products.
- Ensures numerical stability via orthogonal Z assumption.

Projected CG efficiently handles large systems implicitly.

Comments

Another approach provide by the projected conjugate gradient method is to work entirely in the original n-dimensional space and avoid building an explicit null-space basis Z. The problem is that the reduced system $(Z^T GZ)x_Z = -c_z$, although smaller in dimension, can still be very costly to solve if we form it explicitly.

To overcome this, one can rearrange the previous algorithm so that all computations are carried out in the original n-dimensional space, but implicitly within the null space of the constraints.

This is the essence of the projected conjugate gradient approach. The idea is to define a projection operator $P = Z(Z^T HZ)^{-1}Z^T$, where H is some symmetric positive definite matrix.

This projection maps arbitrary vectors into the null space of the constraint matrix A. By doing so, we ensure that all search directions generated by the conjugate gradient iterations remain feasible with respect to the constraints. In practice, this means that feasibility is preserved automatically at each step, without requiring explicit enforcement.

Choosing an effective preconditioner is crucial. The matrix $W_{zz} = Z^T HZ$ serves this role. Its effect is to cluster the eigenvalues of the transformed operator $W_{zz}^{-1/2}(Z^T GZ)W_{zz}^{-1/2}$. When eigenvalues are tightly clustered, conjugate gradients converge in far fewer iterations.

The ideal case, though usually impractical, would be to take W_{zz} exactly equal to $Z^T GZ$, which would reduce the problem to a trivial one-step solution. A key advantage of the projected approach is efficiency. We avoid forming Z explicitly, which can be extremely expensive for large systems. Instead, everything is expressed in terms of n-vectors such as $x = Zx_Z + Yx_Y$. The updates are done directly in the full space, but in such a way that the iterations live in the null space. This ensures both numerical stability and robustness, even when the constraint matrix A is ill-conditioned or the choice of basis Z is less than ideal.