

Numerical Optimization Methods

Numerical Optimization Methods Summary

Line Search Based Methods

$$x_{k+1} = x_k + \alpha_k p_k$$

Where p_k is the search direction and α_k is the step length determined by a line search.

1. Steepest Descent

$$p_k = -\nabla f(x_k)$$

The Steepest Descent method moves along the negative gradient, which indicates the direction of the steepest decrease in the objective function. It's simple but can be slow. It often converges slowly, especially for ill-conditioned problems, with a linear convergence rate.

Pros:

- Simple to implement.
- Guaranteed to decrease the objective function if the step size is chosen correctly.

Cons:

- Can converge very slowly, especially for ill-conditioned problems.
- Zig-zagging behavior.

2. Newton's Method (Unconstrained)

$$\nabla^2 f(x_k) p_k^N = -\nabla f(x_k)$$

$$x_{k+1} = x_k + \alpha_k p_k^N \quad (\text{often } \alpha_k = 1)$$

Uses a quadratic model of the function and steps to its minimum. Converges quadratically near a solution if the Hessian is positive definite but requires Hessian computation.

3. Newton's Method with Hessian Modification

Modifies the Hessian $\nabla^2 f_k$ when it's not positive definite to ensure a descent direction. Various strategies exist for this modification.

3a. Eigenvalue Modification

$$B_k = Q \Lambda' Q^T \quad (\text{where } \Lambda' \text{ has modified eigenvalues of } \nabla^2 f_k)$$

$$B_k p_k = -\nabla f_k$$

Computes the Hessian's spectral decomposition and replaces non-positive eigenvalues. Ensures the modified Hessian B_k is positive definite.

$$\hat{H}_k = H_k + \lambda_k I$$

Eigenvalue modification adds a multiple of the identity matrix to the Hessian to ensure it is positive definite. This helps to guarantee descent directions.

Pros:

- Ensures that the search direction is a descent direction.
- Can improve convergence when the Hessian has negative eigenvalues.

Cons:

- Can distort the Hessian, leading to slower convergence.
- Requires eigenvalue computation, which can be expensive.

3b. Adding a Multiple of the Identity

$$B_k = \nabla^2 f_k + \tau I \quad (\text{for sufficiently large } \tau > 0)$$

$$B_k p_k = -\nabla f_k$$

Adds a scaled identity matrix to the Hessian to make it positive definite. Simple but finding an appropriate τ can be complex.

3c. Modified Cholesky Factorization

Factorize $P^T(\nabla^2 f_k + E)P = LL^T$ (E is diagonal correction)

Solve $LL^T(P^T p_k) = -P^T \nabla f_k$

Computes a Cholesky factorization while adding diagonal elements if needed. Ensures positive definiteness during the factorization process.

Modified Cholesky factorization modifies the diagonal elements of the Hessian during Cholesky factorization to ensure stability.

Pros:

- Efficiently computes a factorization of a positive definite matrix related to the Hessian.
- Well-suited for sparse matrices.

Cons:

- More complex to implement.
- May not work well for highly indefinite Hessians

3d. Modified Symmetric Indefinite Factorization

Factorize $P^T(\nabla^2 f_k + E)P = LBL^T$ (B is block diagonal)

Uses a symmetric indefinite factorization, adding modifications E to ensure sufficient positive definiteness. Handles indefinite Hessians directly.

A modified symmetric indefinite factorization method factors the symmetric indefinite Hessian matrix into LDL^T form, where L is a unit lower triangular matrix, D is a block diagonal matrix with 1x1 and 2x2 blocks.

Modifications are applied to D to control the inertia and ensure numerical stability.

Pros:

- Handles indefinite Hessians directly.
- Numerically stable.

Cons:

- More complex to implement than Cholesky factorization.
- Can be more computationally expensive.

Trust-Region Based Methods

General Subproblem:

$$\min_{p \in \mathbb{R}^n} m_k(p) = f_k + \nabla f_k^T p + \frac{1}{2} p^T B_k p \quad \text{s.t.} \quad \|p\| \leq \Delta_k$$

Where $m_k(p)$ is a model (usually quadratic) and Δ_k is the trust-region radius.

4. Dogleg Method

Path combines Cauchy point p_k^C and Newton step p_k^N . Solution p_k lies on this path within radius Δ_k .

The dogleg method combines the steepest descent direction and the Newton step. It follows the steepest descent direction until it reaches the boundary of the trust region or a multiple of the full Newton step, then switches to the Newton direction.

Pros:

- Combines the advantages of steepest descent and Newton's method.
- More efficient than the Cauchy point method.

Cons:

- Can be more complex to implement than the Cauchy point method.
- Performance depends on the accuracy of the quadratic model.

5. Two-Dimensional Subspace Minimization

Minimize $m_k(p)$ over $p \in \text{span}\{\nabla f_k, (B_k + \alpha I)^{-1} \nabla f_k\}$ within $\|p\| \leq \Delta_k$.

Solves the trust-region subproblem by minimizing the model over a 2D subspace. Captures essential information from gradient and Newton directions.

6. Trust-Region Newton Method

Solve the general trust-region subproblem (see above) with

$$p \in \text{span}\{\nabla f_k, (B_k + \alpha I)^{-1} \nabla f_k\}, \quad \text{with} \quad \|p\| \leq \Delta_k.$$

The fundamental trust-region approach using a quadratic model based on the Hessian. Radius Δ_k is adjusted based on model agreement.

7. Steihaug's Method (Trust-Region Newton-CG)

Algorithm: Uses Conjugate Gradient (CG) method (Algorithm 7.1).

Applies CG iteratively to approximately solve the trust-region subproblem $B_k p = -\nabla f_k$. Stops early if trust boundary is hit or negative curvature found; suitable for large scale.

The Trust-Region Newton-CG method uses the Conjugate Gradient method to approximately solve the trust-region subproblem.

Pros:

- Effective for large-scale problems.
- Does not require the explicit computation of the Hessian.

Cons:

- Requires careful implementation to ensure convergence.
- The Conjugate Gradient method may not converge quickly if the Hessian is ill-conditioned.

Trust-Region Newton-Lanczos Method

The Trust-Region Newton-Lanczos method uses the Lanczos method to find an approximate solution of the trust-region subproblem. Lanczos is an iterative method for finding a few of the eigenvalues and eigenvectors of a large sparse matrix.

Pros:

- Effective for very large-scale problems with sparse Hessians.
- Can compute an approximate solution to the trust-region subproblem more efficiently than other methods.

Cons:

- More complex to implement.
- Requires careful handling of numerical issues.

Conjugate Gradient Methods

8. Linear Conjugate Gradient Method

(Iterative updates involving α_k, β_k - See Algorithm 5.2)

$$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}, \quad \beta_{k+1} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$$

Solves symmetric positive definite linear systems $Ax=b$ iteratively. Crucial for large-scale optimization, finds exact solution in n steps (exact arithmetic).

$$p_k = r_k + \beta_k p_{k-1}, \quad \beta_k = \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}}$$

The Linear Conjugate Gradient method is an iterative method for solving systems of linear equations where the matrix is symmetric and positive definite. It generates search directions that are conjugate to each other.

Pros:

- Efficient for solving linear systems with symmetric positive definite matrices.
- Does not require the computation or storage of the matrix.

Cons:

- Only applicable to linear systems with symmetric positive definite matrices.
- Convergence rate depends on the condition number of the matrix.

9. Nonlinear Conjugate Gradient Methods

$$p_k = -\nabla f_k + \beta_k p_{k-1}$$

Adapts linear CG for general nonlinear optimization, avoiding Hessian storage. Different methods use different formulas for the scalar β_k .

9a. Fletcher-Reeves

$$\beta_k^{FR} = \frac{||\nabla f_k||^2}{||\nabla f_{k-1}||^2}$$

One choice for β_k in nonlinear CG. Has strong convergence theory but can perform poorly in practice.

$$\beta_k = \frac{\nabla f_k^T \nabla f_k}{\nabla f_{k-1}^T \nabla f_{k-1}}$$

The Fletcher-Reeves method is a nonlinear conjugate gradient method. It uses the gradient at the current and previous iterations to compute the conjugate direction.

Pros:

- Relatively simple to implement.
- Does not require the computation or storage of the Hessian matrix.

Cons:

- Can exhibit poor performance if the line search is inaccurate.
- May not converge for non-smooth functions.

9b. Polak-Ribière

$$\beta_k^{PR} = \frac{\nabla f_k^T (\nabla f_k - \nabla f_{k-1})}{||\nabla f_{k-1}||^2} \quad (\text{often use } \max(\beta_k^{PR}, 0))$$

Another choice for β_k , often performs better than Fletcher-Reeves. Convergence theory is weaker, modifications often needed.

$$\beta_k = \frac{\nabla f_k^T (\nabla f_k - \nabla f_{k-1})}{\nabla f_{k-1}^T \nabla f_{k-1}}$$

The Polak-Ribière method is another nonlinear conjugate gradient method. It often performs better than the Fletcher-Reeves method in practice.

Pros:

- Often more efficient than the Fletcher-Reeves method.
- Does not require the computation or storage of the Hessian matrix.

Cons:

- Can exhibit instability.
- May not converge for non-smooth functions.

Quasi-Newton Methods

General Idea: Build Hessian approximation B_k (or inverse H_k) using gradient information.

General Formula: Search direction $p_k = -H_k \nabla f_k$ or solve $B_k p_k = -\nabla f_k$.

10. BFGS Method

(Rank-two update for B_k or H_k , see Eq 6.17/6.19)

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}$$

Most popular quasi-Newton method using a rank-two update. Requires $s_k^T y_k > 0$ (via line search) and typically converges superlinearly.

$$x_{k+1} = x_k - \alpha_k B_k^{-1} \nabla f_k$$

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}$$

The Broyden–Fletcher–Goldfarb–Shanno (BFGS) method approximates the Hessian matrix and updates it at each iteration. It generally has good convergence properties.

Pros:

- Superlinear convergence.
- Does not require the computation of the Hessian matrix.

Cons:

- More complex to implement than steepest descent.
- Requires more storage.

11. SR1 Method

(Rank-one update, see Eq 6.24)

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}$$

Uses a symmetric rank-one update. Can handle indefinite Hessians but update may need skipping if the denominator is small.

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}$$

The Symmetric Rank 1 (SR1) method is another quasi-Newton update. It can be indefinite, which can be both an advantage and a disadvantage.

Pros:

- Can be more efficient than BFGS in some cases.
- Lower computational cost per iteration compared to BFGS.

Cons:

- The updated matrix can be indefinite.
- Can be unstable if the denominator is close to zero.

Broyden Class

$$B_{k+1} = (1 - \phi)B_{k+1}^{BFGS} + \phi B_{k+1}^{SR1}$$

The Broyden class is a family of quasi-Newton updates that includes BFGS and SR1 as special cases. The parameter ϕ controls the balance between these two updates.

Pros:

- Provides a flexible framework that includes BFGS and SR1.
- Can potentially achieve better performance by tuning the parameter ϕ .

Cons:

- Requires tuning of the parameter ϕ .
- The updated matrix can be indefinite for some values of ϕ .

12. Limited-Memory BFGS (L-BFGS)

Implicit update using last m pairs (s_i, y_i) . Direction via two-loop recursion (Algorithm 7.4).

Variant of BFGS for large problems, storing only recent m correction pairs instead of full matrix. Uses recursion to compute search direction.

The Limited-Memory BFGS (L-BFGS) method is an approximation to the BFGS method that uses only a limited amount of computer memory. It stores only a few vectors that represent the history of the iterations.

Pros:

- Efficient for large-scale problems.
- Superlinear convergence.

Cons:

- More complex to implement than BFGS.
- Requires careful tuning of the number of stored vectors.

13. Compact Representation of BFGS Updating

Represents B_k using compact matrix products (Eq 7.20-7.23).

Alternative way to represent BFGS updates. Useful for understanding L-BFGS and connections to other methods.

14. Sparse Quasi-Newton Updates

Attempts to maintain sparsity in the Hessian approximation B_k . Generally less effective than L-BFGS for large-scale problems.

Derivative-Free Optimization (DFO) Methods

15. Model-Based DFO Methods

Minimize model $m_k(p) \approx f(x_k + p)$ subject to $\|p\| \leq \Delta_k$.

Builds and minimizes a model (e.g., quadratic) based on function values at sampled points. Manages interpolation set geometry.

16. Coordinate Search Method

Algorithm: Explores along coordinate directions $\pm e_i$.

Very simple DFO method that searches along axes. Can be very slow but easy to implement.

The coordinate search method minimizes the objective function by iteratively changing one variable at a time while holding the others constant.

Pros:

- Simple to implement.
- Does not require derivatives.

Cons:

- Can be very slow, especially for functions with many variables.
- May fail to converge if the variables are highly dependent.

Pattern-Search Methods

Pattern search methods evaluate the objective function at a set of points (a pattern) around the current iterate. The pattern is moved if a better point is found.

Pros:

- Do not require derivatives.
- Can be effective for non-smooth functions.

Cons:

- Can be slow.
- Convergence depends on the choice of the pattern.

Conjugate-Direction Method

Conjugate direction methods search along directions that are conjugate to each other with respect to the Hessian of the objective function.

Pros:

- Faster convergence than steepest descent.
- Effective for quadratic functions.

Cons:

- Requires knowledge of conjugate directions.
- Performance depends on the accuracy of line searches.

Nelder-Mead Method

The Nelder-Mead method is a direct search method that uses a simplex (a polytope of $n + 1$ vertices in n dimensions) to search for the minimum of a function.

Pros:

- Does not require derivatives.
- Can be effective for non-smooth functions.

Cons:

- Can be slow and may converge to a non-stationary point.
- Performance depends on the initial simplex.

Implicit Filtering

Implicit filtering is a derivative-free optimization method that uses a filtered response surface model to guide the search.

Pros:

- Robust to noise.
- Effective for problems with expensive function evaluations.

Cons:

- Computationally expensive.
- Complex to implement.

Finite-Difference Derivative Approximations

Approximating the Gradient

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x + he_i) - f(x)}{h}$$

Approximating the gradient using forward difference.

Pros:

- Simple to implement.
- Does not require access to the analytical form of the derivative.

Cons:

- Can be inaccurate if the step size h is not chosen carefully.
- Computationally expensive for functions with many variables.

Approximating a Sparse Jacobian

Approximating a sparse Jacobian efficiently by exploiting the sparsity pattern to reduce the number of function evaluations.

Pros:

- Reduces the number of function evaluations compared to dense Jacobian approximation.
- Useful for problems where Jacobian matrix is sparse.

Cons:

- More complex to implement.
- Requires knowledge of the sparsity pattern.

Approximating the Hessian

Approximating the Hessian matrix using finite differences, often using central difference approximations for better accuracy.

$$\frac{\partial^2 f}{\partial x_i \partial x_j} \approx \frac{f(x + h_i e_i + h_j e_j) - f(x + h_i e_i) - f(x + h_j e_j) + f(x)}{h_i h_j}$$

Pros:

- Does not require analytical second derivatives.

Cons:

- Computationally expensive.
- Accuracy depends on the step size.
- Can be numerically unstable.

Approximating a Sparse Hessian

Approximating a sparse Hessian efficiently by exploiting the sparsity pattern to reduce the number of function evaluations.

Pros:

- Reduces the computational cost.
- Useful for problems with sparse Hessian matrices.

Cons:

- More complex to implement.
- Requires knowledge of the sparsity pattern.

Automatic Differentiation

Forward Mode

Automatic differentiation computes derivatives by applying the chain rule to each elementary operation in the code. Forward mode computes the derivative of a function at a particular input.

Pros:

- Computes derivatives with machine precision.
- Efficient for functions with many outputs.

Cons:

- Can be inefficient for functions with many inputs.
- Requires modifying the source code.

Reverse Mode

Reverse mode computes the derivative of a function by traversing the computational graph in reverse. Efficient for functions with many inputs and few outputs.

Pros:

- Efficient for functions with many inputs and few outputs.
- Computes derivatives with machine precision.

Cons:

- Can be more complex to implement than forward mode.
- Requires storing intermediate variables.

Model-Based Methods

A Method Based on Minimum-Change Updating

This method constructs a model of the objective function and updates it by making the smallest possible change that is consistent with the new information.

Pros:

- Can be effective when derivatives are unavailable or unreliable.
- Provides a framework for updating the model.

Cons:

- Performance depends on the choice of the model.
- Can be computationally expensive.

Local Algorithms

Newton's Method for Nonlinear Equations

$$x_{k+1} = x_k - J_k^{-1} r_k$$

Newton's method is an iterative method for finding the roots of a system of nonlinear equations. It uses the Jacobian matrix of the system.

Pros:

- Quadratic convergence near the solution.
- Well-understood convergence properties.

Cons:

- Requires the computation and inversion of the Jacobian matrix.
- May not converge if the initial guess is far from the solution.

Inexact Newton Methods

Inexact Newton methods are variants of Newton's method where the Newton equations are solved only approximately.

Pros:

- Can be more efficient than the standard Newton's method, especially for large-scale problems.

- Allows for the use of iterative methods to solve the linear systems.

Cons:

- Requires careful control of the accuracy of the approximate solution.
- Convergence rate depends on the accuracy of the solution of the linear systems.

Broyden's Method

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k) s_k^T}{s_k^T s_k}$$

Broyden's method is a quasi-Newton method for solving nonlinear equations. It approximates the Jacobian matrix and updates it at each iteration.

Pros:

- Does not require the computation of the Jacobian matrix.
- Superlinear convergence.

Cons:

- More complex to implement than the secant method.
- May not converge if the initial guess is far from the solution.

Tensor Methods

Tensor methods use higher-order derivatives (tensors) to obtain a more accurate model of the function being optimized.

Pros:

- Faster convergence than Newton's method in some cases.
- Can handle problems where the Hessian is singular.

Cons:

- Computationally expensive.
- Complex to implement.

Practical Methods

Merit Functions

A merit function is a function that combines the objective function and the constraints of a constrained optimization problem into a single function. Merit functions are used to measure the progress of an algorithm.

Pros:

- Provide a way to measure progress in constrained optimization.
- Can be used to ensure global convergence.

Cons:

- Requires careful selection of parameters.
- Can be difficult to optimize.

Continuation/Homotopy Methods

Continuation/homotopy methods solve a difficult optimization problem by solving a series of simpler problems. The simpler problems are constructed by introducing a parameter that gradually transforms the simpler problem into the original problem.

Pros:

- Can be used to find global solutions.
- Can be effective for problems with multiple local minima.

Cons:

- Can be computationally expensive.
- Requires careful construction of the continuation path.

Linear Programming

The Simplex Method

The simplex method is a classic algorithm for solving linear programming problems. It moves from one vertex of the feasible region to another, improving the objective function at each step.

$$\text{Objective: } \max c^T x$$

$$\text{Constraints: } Ax \leq b, \quad x \geq 0$$

Pros:

- Guaranteed to find the optimal solution if it exists.
- Efficient for many practical problems.

Cons:

- Can be slow for large problems.
- May cycle if the problem is degenerate.

Dual Simplex Method

The dual simplex method is a variant of the simplex method that starts with a solution that is dual feasible but primal infeasible and iterates until primal feasibility is achieved while maintaining dual feasibility.

Pros:

- Useful when an optimal solution to a related problem is known.
- Can be more efficient than the regular simplex method in some cases.

Cons:

- More complex to implement.
- May not be as intuitive as the regular simplex method.

Interior-Point Methods

Primal-Dual Methods

Primal-dual methods for linear programming move through the interior of the feasible region, simultaneously improving the primal and dual solutions.

Pros:

- Efficient for large-scale problems.
- Polynomial-time complexity.

Cons:

- More complex to implement than the simplex method.
- Can be sensitive to the choice of parameters.

Potential-Reduction Methods

Potential-reduction methods are interior-point methods that reduce a potential function at each iteration. This potential function measures both the duality gap and the feasibility of the solution.

Pros:

- Guaranteed polynomial-time convergence.
- Can be very efficient in practice.

Cons:

- More complex than other interior-point methods.
- Performance depends on the choice of the potential function.

Quadratic Programming

Equality-Constrained Quadratic Programs

An equality-constrained quadratic program is an optimization problem where the objective function is quadratic and the constraints are linear equalities.

$$\begin{aligned} \min \quad & \frac{1}{2} x^T Q x + c^T x \\ \text{subject to} \quad & Ax = b \end{aligned}$$

Pros:

- Can be solved efficiently using direct methods.
- The solution can be characterized by a system of linear equations.

Cons:

- Only applicable to problems with equality constraints.
- The matrix Q must be positive definite for the problem to be convex.

Active-Set Methods for Convex QPs

Active-set methods for convex quadratic programming maintain a set of active constraints (those that are satisfied with equality at the current solution estimate) and iteratively update this set until the optimal active set is identified.

Pros:

- Can be efficient for problems with many constraints.
- Exploits the structure of the problem.

Cons:

- Can be complex to implement.
- Performance depends on the effectiveness of the active-set strategy.

Gradient Projection Method

The gradient projection method is used for solving constrained optimization problems, including quadratic programs with simple constraints (e.g., box constraints). It iteratively moves the current point in the direction of the negative gradient and then projects the resulting point back onto the feasible set.

Pros:

- Relatively simple to implement, especially for problems with simple constraints.
- Can handle non-smooth constraints.

Cons:

- Can be slow for problems with complex constraints.
- Convergence rate depends on the geometry of the feasible set.

Penalty and Augmented Lagrangian Methods

Quadratic Penalty Method

The quadratic penalty method replaces a constrained optimization problem with an unconstrained one by adding a penalty term to the objective function for constraint violations.

$$\min f(x) + \frac{\rho}{2} \sum_{i=1}^m c_i(x)^2$$

Pros:

- Simple to implement.
- Can be applied to a wide range of constrained optimization problems.

Cons:

- Can suffer from ill-conditioning as the penalty parameter ρ increases.
- The solution of the unconstrained problem may not be feasible for the original constrained problem.

Nonsmooth Penalty Functions

Nonsmooth penalty functions, like the l_1 penalty, penalize constraint violations using a nonsmooth term.

Pros:

- Can promote sparsity in the constraint multipliers.
- More robust to ill-conditioning than smooth penalty functions.

Cons:

- The objective function becomes nonsmooth, requiring specialized optimization methods.

- The solution may not be feasible for the original constrained problem.

Augmented Lagrangian Method: Equality Constraints

The augmented Lagrangian method combines the Lagrangian function with a penalty term for constraint violations.

$$L_A(x, \lambda, \rho) = f(x) + \lambda^T c(x) + \frac{\rho}{2} |c(x)|^2$$

Pros:

- More robust than the penalty method.
- Can converge to a feasible solution without requiring the penalty parameter to go to infinity.

Cons:

- More complex to implement than the penalty method.
- Requires updating the Lagrange multipliers.

Sequential Quadratic Programming

Local SQP Method

The local SQP method solves a sequence of quadratic programming subproblems to approximate the solution of a nonlinear programming problem.

Pros:

- Fast convergence near the solution.
- Effective for a wide range of nonlinear programming problems.

Cons:

- May not converge if the initial guess is far from the solution.
- Requires solving a quadratic programming subproblem at each iteration.

Line Search SQP Method

Line search SQP methods add a line search procedure to the basic SQP algorithm to improve its global convergence properties.

Pros:

- Improved global convergence compared to local SQP.
- More robust.

Cons:

- More complex to implement.
- Requires a suitable merit function.

Trust-Region SQP Methods

Trust-region SQP methods use a trust-region approach to improve the global convergence of the SQP algorithm.

Pros:

- Robust global convergence properties.
- Can handle problems where the Hessian of the Lagrangian is indefinite.

Cons:

- More complex to implement than line search SQP.
- Requires solving a constrained quadratic subproblem at each iteration.

Nonlinear Gradient Projection

Nonlinear gradient projection extends the gradient projection method to handle nonlinear constraints.

Pros:

- Can handle nonlinear constraints.
- Relatively simple to implement for some types of constraints.

Cons:

- Can be slow for problems with highly nonlinear constraints.
- Convergence depends on the geometry of the feasible set.

Interior-Point Methods for Nonlinear Programming

Primal Log-Barrier Method

The primal log-barrier method is an interior-point method for nonlinear programming that adds a logarithmic barrier term to the objective function to keep the iterates away from the boundary of the feasible region.

$$\min f(x) - \mu \sum_{i=1}^m \log(-c_i(x))$$

Pros:

- Effective for solving nonlinear programming problems.
- Can handle inequality constraints directly.

Cons:

- Requires careful selection of the barrier parameter μ .
- Can be inefficient if the initial point is close to the boundary of the feasible region.