

Untitled

Numerical Optimization Methods Summary

Line Search Based Methods

General Formula:

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

Variable Explanations:

- x_k : The current iterate (approximation to the solution) at iteration k .
- x_{k+1} : The next iterate.
- α_k : The step length determined by a line search procedure, deciding how far to move along the search direction.
- p_k : The search direction vector, indicating the direction of movement from x_k .

1. Steepest Descent

Formula:

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

Variable Explanations:

- $\nabla f(x_k)$: The gradient vector of the objective function f evaluated at x_k . **Explanation:** The most basic method, using the negative gradient as the search direction. It often converges slowly, especially for ill-conditioned problems, with a linear convergence rate. **Pros:**
- Simple to implement and understand.
- Guaranteed to be a descent direction if $\nabla f(x_k) \neq 0$. **Cons:**
- Often exhibits very slow (linear) convergence, especially zigzagging behavior.
- Sensitive to poor scaling of variables.

2. Newton's Method (Unconstrained)

Formula:

$$\begin{aligned} \nabla^2 f(x_k) p_k^N &= -\nabla f(x_k) \\ x_{k+1} &= x_k + \alpha_k p_k^N \quad (\text{often } \alpha_k = 1) \end{aligned}$$

Variable Explanations:

- $\nabla^2 f(x_k)$: The Hessian matrix (matrix of second partial derivatives) of f evaluated at x_k .
- p_k^N : The Newton search direction. **Explanation:** 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. **Pros:**
- Very fast (quadratic) convergence rate near the solution.
- Scale-invariant. **Cons:**
- Requires computation (and inversion/factorization) of the Hessian matrix, which can be expensive.

- Requires the Hessian to be positive definite for the direction to be a descent direction and for stability; may fail or converge to saddle points otherwise.

3. Newton's Method with Hessian Modification

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

- Extends Newton's method to handle non-convex regions.
- Retains fast convergence when the true Hessian is positive definite. **Cons:**
- Adds complexity and computational cost compared to pure Newton.
- Modification strategy can impact performance.

3a. Eigenvalue Modification

Formula:

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

$$B_k p_k = -\nabla f_k$$

Variable Explanations:

- B_k : The modified Hessian approximation.
- Q, Λ' : Factors from the modified spectral decomposition of $\nabla^2 f_k$. **Explanation:** Computes the Hessian's spectral decomposition and replaces non-positive eigenvalues. Ensures the modified Hessian B_k is positive definite. **Pros:**
- Directly ensures positive definiteness. **Cons:**
- Requires full eigenvalue decomposition, which is very expensive ($O(n^3)$).

3b. Adding a Multiple of the Identity

Formula:

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

$$B_k p_k = -\nabla f_k$$

Variable Explanations:

- τ : A positive scalar shift parameter.
- I : The identity matrix. **Explanation:** Adds a scaled identity matrix to the Hessian to make it positive definite. Simple but finding an appropriate τ can be complex. **Pros:**
- Relatively simple to implement. **Cons:**
- Choosing an appropriate τ can be heuristic or require iterative refinement.
- May modify the Hessian more than necessary.

3c. Modified Cholesky Factorization

Formula: Factorize $P^T(\nabla^2 f_k + E)P = LL^T$ (E is diagonal correction) Solve $LL^T(P^T p_k) = -P^T \nabla f_k$ **Variable Explanations:**

- P : Permutation matrix.
- E : Non-negative diagonal correction matrix added during factorization.

- L: Lower triangular Cholesky factor. **Explanation:** Computes a Cholesky factorization while adding diagonal elements if needed. Ensures positive definiteness during the factorization process. **Pros:**
- Computationally cheaper than eigenvalue decomposition ($O(n^3)$) but often faster in practice).
- Adds correction only when needed during factorization. **Cons:**
- Resulting matrix $B_k = P L L^T P^T$ depends on the order of computation (pivoting P).
- Can still be expensive for large n.

3d. Modified Symmetric Indefinite Factorization

Formula: Factorize $P^T(\nabla^2 f_k + E)P = LBL^T$ (B is block diagonal) **Variable Explanations:**

- B: Block diagonal matrix (1x1 or 2x2 blocks). **Explanation:** Uses a symmetric indefinite factorization, adding modifications E to ensure sufficient positive definiteness. Handles indefinite Hessians directly. **Pros:**
- Can be more stable than modified Cholesky for indefinite matrices. **Cons:**
- More complex factorization than Cholesky.

Trust-Region Based Methods

General Subproblem:

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

Variable Explanations:

- p: The step (search direction) to be determined.
- $m_k(p)$: A model function (usually quadratic) approximating the objective function change around x_k .
- f_k : The objective function value $f(x_k)$.
- ∇f_k : The gradient $\nabla f(x_k)$.
- B_k : An approximation to the Hessian $\nabla^2 f(x_k)$ (often the true Hessian or a Quasi-Newton approximation).
- Δ_k : The trust-region radius, defining the region around x_k where the model m_k is trusted.
- $\|\cdot\|$: A norm, typically the Euclidean norm.

4. Dogleg Method

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

- p_k^C : Cauchy point (minimizer along steepest descent within the trust region).
- p_k^N : Full (unconstrained) Newton step for the model m_k . **Explanation:** Approximates the trust-region subproblem solution using a path made of the steepest descent direction and the Newton direction. Efficient for small/medium problems. **Pros:**
- Relatively inexpensive way to find an approximate solution to the trust-region subproblem.
- Effective when the Hessian approximation B_k is positive definite. **Cons:**
- Less effective or needs modification if B_k is indefinite.
- Specifically designed path may not always be the best choice.

5. Two-Dimensional Subspace Minimization

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

- α : A non-negative scalar (can be 0 if B_k is positive definite).

- $\text{span}\{u, v\}$: The vector space spanned by vectors u and v . **Explanation:** Solves the trust-region subproblem by minimizing the model over a 2D subspace. Captures essential information from gradient and Newton directions. **Pros:**
- Can handle indefinite B_k more naturally than the dogleg method.
- Often provides a better approximation to the subproblem solution than dogleg. **Cons:**
- Requires solving a 2D minimization problem, which is slightly more work than dogleg path calculation.

6. Trust-Region Newton Method

Formula: Solve the general trust-region subproblem (see above) with $B_k = \nabla^2 f_k$ or approximation. Update $x_{k+1} = x_k + p_k$ and adjust Δ_k . **Explanation:** The fundamental trust-region approach using a quadratic model based on the Hessian. Radius Δ_k is adjusted based on model agreement. **Pros:**

- Strong convergence properties (globally convergent under reasonable conditions).
- Handles indefinite Hessians naturally within the subproblem solution. **Cons:**
- Requires solving the trust-region subproblem at each iteration, which can be computationally expensive (though often done approximately).
- Performance depends on the quality of the Hessian approximation B_k .

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

Algorithm: Uses Conjugate Gradient (CG) method (Algorithm 7.1). **Explanation:** 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. **Pros:**

- Avoids explicit factorization of B_k , suitable for large-scale problems where only matrix-vector products are feasible.
- Can handle indefinite B_k by terminating CG if negative curvature is detected. **Cons:**
- Provides an approximate solution to the subproblem.
- Performance depends on the effectiveness of CG (e.g., requires B_k to be positive definite for standard CG theory, although modifications exist).

Conjugate Gradient Methods

8. Linear Conjugate Gradient Method

Formula: (Iterative updates involving α_k, β_k - See Algorithm 5.2) $\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$, $\beta_{k+1} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$

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

$$r_{k+1} = r_k - \alpha_k A p_k$$

$$p_{k+1} = r_{k+1} + \beta_{k+1} p_k$$