IMPLEMENTING NONLINEAR OPTIMIZATION ALGORITHMS IN PYTHON

Truncated Newton · Sequential Penalty · Filled Functions

Continuous Optimization · Alessandro Rapo · Sapienza University of Rome · 2024–2025

01 — ABSTRACT

This article documents the Python implementation of three fundamental algorithms from continuous nonlinear optimization: the Truncated Newton method, the Sequential Penalty method, and the Filled Function algorithm for global optimization. Every component — from the inner conjugate gradient solver to the Armijo line search and the Gaussian hill construction — was built from scratch in NumPy, with particular attention to numerical stability and adherence to theoretical convergence guarantees.

02 — INTRODUCTION

Nonlinear optimization is at the heart of machine learning, engineering design, financial modeling, and operations research. While production-grade solvers such as IPOPT, KNITRO, or scipy.optimize are widely available, implementing the underlying algorithms by hand remains one of the most effective ways to develop a deep, operational understanding of their behavior — including their failure modes and sensitivity to parameter choices.

The project addresses three distinct but related problems:

- → **NT:** Unconstrained local minimization via the Truncated Newton method.
- → **PS:** Constrained local minimization via Sequential Penalty functions.
- → **FF:** Global minimization via additive Filled Functions.

All three algorithms are implemented in pure NumPy without automatic differentiation. Gradients are provided analytically or approximated with central finite differences, and Hessian-vector products are computed via a matrix-free finite-difference scheme, avoiding the $O(n^2)$ cost of assembling the full Hessian.

03 — TRUNCATED NEWTON METHOD (NEWTONT.PY)

The Truncated Newton method is designed for large-scale unconstrained optimization. At each outer iteration k, instead of solving the Newton system $H_k d = -g_k$ exactly, an approximate solution is computed using the Conjugate Gradient method (Steihaug-CG), truncating early when negative curvature is detected or when the residual is small enough.

▸ *Hessian-Vector Products*

The key computational primitive is the matrix-free approximation of $H(x)\cdot v$ via central finite differences on the gradient:

```
H(x)·v  ≈  [ ∇f(x + h·v^) − ∇f(x − h·v^) ] / (2h / ‖v‖)      h = 1e-5
```

This avoids forming the n×n Hessian explicitly and keeps memory usage at $O(n)$, which is critical for high-dimensional problems.

▸ *Truncated Conjugate Gradient (GCT)*

The inner CG loop solves $H\cdot d \approx -g$ starting from $d = 0$. Two truncation criteria are enforced:

- → Negative curvature: if $s^T H s \leq 0$, return the current iterate (descent is guaranteed).
- → Relative residual: stop when $\|r_k\| \leq \eta_k \cdot \|g\|$, with $\eta_k = \min(0.5, \sqrt{\|g\|})$, ensuring superlinear convergence.

▸ *Armijo Line Search*

```
f(x + α·d)  ≤  f(x) + α · γ · ∇f(x)^T d        γ = 1e-4,   β = 0.5
```

If the CG direction is not a descent direction (possible near saddle points), the algorithm falls back to the steepest descent direction $-g$.

04 — SEQUENTIAL PENALTY METHOD (PENALITA.PY)

The Sequential Penalty method converts a constrained problem into a sequence of unconstrained ones by penalizing constraint violations. Given $\min f(x)$ s.t. $g(x) \leq 0$, $h(x) = 0$, the penalized objective is:

$$F_\varepsilon(x) = f(x) + (1/\varepsilon) \cdot \Sigma \max(0, g_i(x))^2 + (1/\varepsilon) \cdot \Sigma h_j(x)^2$$

As $\varepsilon \to 0$, the penalty weight $1/\varepsilon \to \infty$, forcing the minimizer of F_ε towards the feasible region. The algorithm follows the four-step schema from the lecture notes:

→ P1: Estimate KKT multipliers as $\lambda_i = (2/\varepsilon)\cdot\max(0, g_i(x))$ and $\mu_j = (2/\varepsilon)\cdot h_j(x)$. Check KKT conditions.

→ P2: Minimize F_ε approximately using Truncated Newton, stopping when $\|\nabla F_\varepsilon\| \leq \delta$.

→ P3: If constraint violation did not decrease by factor θ_1, reduce $\varepsilon \leftarrow \theta_2\cdot\varepsilon$ (increase penalty weight).

→ P4: Tighten inner tolerance: $\delta \leftarrow \theta_3\cdot\delta$. Repeat from P1.

▸ *KKT Verification*

KKT conditions are checked across four criteria at each outer iteration: stationarity of the Lagrangian ($\|\nabla L\| \leq$ tol), primal feasibility, dual feasibility ($\lambda \geq 0$), and complementary slackness ($\lambda_i\cdot g_i(x) = 0$). Termination occurs as soon as all four hold within tolerance.

▸ *Parameter Choices*

We use $\varepsilon_0 = 1.0$ (moderate initial penalty), $\theta_1 = 0.25$ (strict improvement threshold), $\theta_2 = 0.1$ (aggressive penalty growth), and $\theta_3 = 0.5$ (gradual tightening). This aggressive schedule avoids the ill-conditioning that arises when ε is reduced too slowly.

05 — FILLED FUNCTION ALGORITHM (FILLED.PY)

Local optimization methods converge to the nearest local minimum, which may be far from the global one. The Filled Function method provides a principled escape strategy: construct an auxiliary function that eliminates the current local minimum and guides the search towards a better basin.

▸ *The Additive Filled Function — Type 1*

Given the current local minimizer x*_k, the function is:

$$U_k(x) = \tau \cdot (\min\{0, f(x) - f(x^*_k) + \rho\})^3 + \exp(-\|x - x^*_k\|^2 / \gamma^2)$$

The two terms serve complementary roles:

→ Gaussian term: creates a hill centered at x*_k, making it a strict local maximum of U_k — any local minimizer of U_k starting from x*_k is forced to move elsewhere.

→ Cubic penalty: is zero where $f(x) \geq f(x^*_k) - \rho$ and strongly negative where $f(x) < f(x^*_k) - \rho$, pulling the search towards regions of lower objective value.

For sufficiently large τ and $0 < \rho < f(x^*_k) - f^*$, all global minimizers of U_k lie in $\{x : f(x) < f(x^*_k)\}$, guaranteeing progress towards the global optimum.

▸ *Multi-Start Strategy*

→ Geometric perturbations along each coordinate axis at scales $\gamma \cdot 10^0$, $\gamma \cdot 10^{-1}$, $\gamma \cdot 10^{-2}$, $\gamma \cdot 10^{-3}$.
→ Uniformly random restarts sampled across the feasible box $[box\_lo, box\_hi]^n$.
→ Any candidate z with $f(z) < f(x^*_k)$ is accepted, then refined with a full NT run on f.

06 — EXPERIMENTAL RESULTS

▸ *Sequential Penalty — Selected Problems*

→ **P3 (Rosenbrock + constraints):** $x^* = (0.5, 0.866)$, $f^* = 38.199$. Active constraints: circle $g_3 = 0$ and bound $g_4 = 0$.
→ **P7 (minimum distance + equalities):** KKT at iteration 10. $x^* \approx (2.0, 2.0, 0.849, 1.131)$, $f^* = 13.858$.
→ **P11 (maximize x_1 + mixed):** KKT at iteration 15. $x^* = (1, 1, 0, 0)$, $f^* = -1.0$ — global optimum confirmed.

▸ *Filled Functions — Selected Problems*

→ **F1 (Rosenbrock 4D):** $f^* = 0.0$ at $x^* = (1,1,1,1)$ — global minimum confirmed.
→ **F5 (Wood function):** $f^* = 0.0$ at $x^* = (1,1,1,1)$ — global minimum confirmed.
→ **F6 (Six-Hump Camel):** $f^* = -1.031628$ at $x^* \approx (-0.090, 0.713)$ — matches known global minimum.
→ **F129 (Box volume):** $f^* = -3456.0$ at $x^* = (12, 12)$ — analytic global minimum recovered.

07 — IMPLEMENTATION NOTES & LESSONS LEARNED

→ **Finite difference step size:** h = 1e-5 for Hessian-vector products and gradients. Too small (< 1e-8) causes cancellation; too large (> 1e-3) corrupts the CG direction.
→ **Adaptive CG tolerance:** $\eta_k = \min(0.5, \sqrt{\|g_k\|})$ rather than fixed — coarse solutions early, refined solutions near optimality — dramatically improves convergence.
→ **Filled function γ sensitivity:** Controls the basin of repulsion around x^*_k. Problem-specific tuning ($\gamma \in \{0.05, 0.5, 1.0, 2.0\}$) proved necessary — no single value works universally.

→ **Penalty ill-conditioning:** As $\varepsilon \to 0$ the Hessian of F_ε becomes ill-conditioned. Adaptive inner tolerance $\delta \leftarrow \theta_3 \cdot \delta$ compensates by requiring less precision when the landscape is steep.

08 — CONCLUSION

Implementing these algorithms from scratch — without auto-diff or external solvers — offers a level of insight that using a library simply cannot provide. Every hyperparameter has a theoretical justification rooted in convergence proofs; every failure mode reveals something concrete about the geometry of the problem.

The Truncated Newton inner solver proved to be the critical shared component across all three algorithms: a robust, numerically stable GCT implementation is what enables both the penalty method and the filled function framework to converge reliably. The full implementation — funzioni.py, NewtonT.py, penalita.py, Filled.py — covers 20+ benchmark problems across constrained and global optimization.

— — REFERENCES

---

[1] S. Lucidi, Appunti dalle lezioni di Ottimizzazione Continua, Sapienza, A.A. 2024–2025.

[2] J. Nocedal, S. J. Wright, Numerical Optimization, 2nd ed., Springer, 2006.

[3] T. Steihaug, The conjugate gradient method and trust regions in large scale optimization, SIAM J. Numer. Anal., 1983.

[4] Y.-P. Zhang, Filled functions for unconstrained global optimization, J. Global Optim., 2001.

[5] R. Fletcher, Practical Methods of Optimization, 2nd ed., Wiley, 1987.