Derivative-Free Optimization: A Brief Overview and Some Recent Advances

Zaikun Zhang

AMA, The Hong Kong Polytechnic University

Joint works with

S. Gratton, T. M. Ragonneau, C. W. Royer, L. N. Vicente

Zeroth-Order Optimization Workshop

The Chinese University of Hong Kong, Shenzhen, May 24, 2023

Outline

Table of contents

Why optimize a function without using derivatives?

> I started to write computer programs in Fortran at Harwell in 1962. ... after moving to Cambridge in 1976 ... I became a consultant for IMSL. One product they received from me was the `TOLMIN` package for optimization ... which requires first derivatives ... Their customers, however, prefer methods that are without derivatives, so IMSL forced my software to employ difference approximations ... I was not happy ... Thus there was strong motivation to try to construct some better algorithms.

— M. J. D. Powell

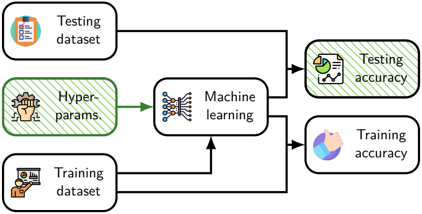
A view of algorithms for optimization without derivatives, 2007, Hong Kong

Derivative-free optimization (DFO)

$$x \longrightarrow \boxed{f} \longrightarrow f(x)$$

- Minimize f using function values, not derivatives (1st-order information).
- A typical case: f is a black box without an explicit formula.
- The dimension of x may be small, but the evaluation of f is expensive.
- Closely related: black-box optimization, simulation-based optimization, gradient-free optimization, zeroth-order optimization.

An example — hyperparameter tuning



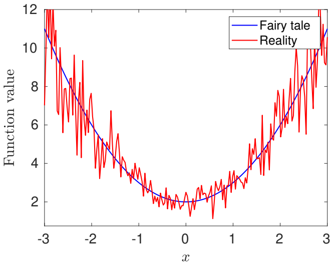
$$\max \{\Pi(x) : x \in \mathcal{X}\}$$

- x represents the hyperparameters for a machine learning model.
- $\Pi(x)$ quantifies the performance of the model (e.g., accuracy) for x.
- Π does not have an explicit formulation.
- The dimension of x is often small, but each evaluation of Π is expensive.
- How to choose x in order to "optimize"/improve the performance?
- A "small" problem defined by big data.

Words of caution

- The reason for not using derivatives is not nonsmoothness but rather the unavailability of the first-order information.
- It is most likely a bad idea to use derivative-free optimization methods if any kind of first-order information is available.
- A real problem is often a gray box instead of a black box. Any known structure should be exploited.

DFO is no fairy-tale world



(Yes, this is your favorite convex quadratic function)

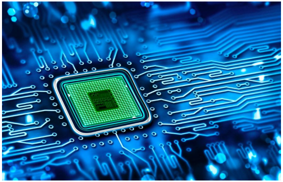
- The black box defining f can be extremely noisy.
- The function evaluation can be extremely expensive.
- The budget can be extremely low.

Be realistic

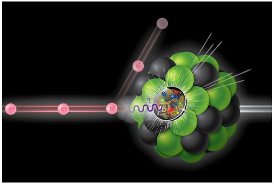
(In real applications) ... one almost never reaches a solution but even 1% improvement can be extremely valuable.

— A. R. Conn

Inversion, history matching, clustering and linear algebra, 2015, Toulouse



Circuit design

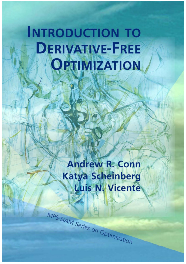


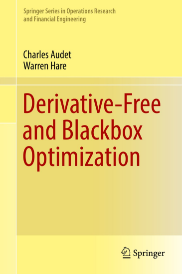
Computational nuclear phsics

Machine learning

- Ciccazzo *et al.*, Derivative-free robust optimization for circuit design, *J. Optim. Theory Appl.*, 2015
- Wild, Sarich, and Schunck, Derivative-free optimization for parameter estimation in computational nuclear physics, *J. Phys. G*, 2015
- Ghanbari and Scheinberg, Black-box optimization in machine learning with trust region based derivative free algorithm, arXiv:1703.06925, 2017

Well-developed theory and methods





- Powell, Direct search algorithms for optimization calculations, 1998
- Powell, A view of algorithms for optimization without derivatives, 2007
- Conn, Scheinberg, and Vicente, *Introduction to Derivative-Free Optimization*, 2007
- Audet and Warren, *Derivative-Free and Blackbox Optimization*, 2017
- Larson, Menickelly, and Wild, Derivative-free optimization methods, 2019

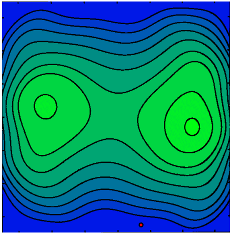
Table of contents

Two main classes of methods

- Model-based methods:
  iterates are determined according to models built using function values
  - Powell's trust-region DFO methods
  - SOLNP and SOLNP+ by Ye *et al*
- Direct search methods:
  iterates are determined by simple comparison of objective function values
  - Nelder-Mead simplex method

Methods not covered by these two classes:

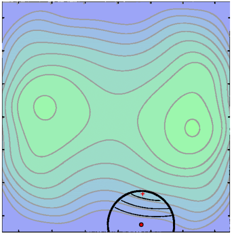
Bayesian optimization, genetic methods, simulated annealing, grid search, ...



$$x_{k+1} \approx x_k + \underset{\|d\| \leq \Delta_k}{\arg\min}\, m_k(x_k + d)$$

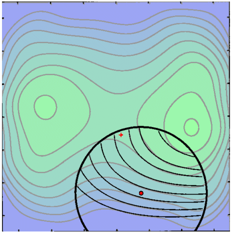
- m_k is the trust-region model (surrogate)
  - $m_k(x) \approx f(x)$ around x_k
  - m_k interpolates f on a set $\mathcal{X}_k$ consisting of previous iterates
- $\|d\| \leq \Delta_k$ is the trust-region constraint
  - If "things work well", increase Δ_k
  - Otherwise, decrease Δ_k

Trust-region DFO methods based on interpolation models



$$x_{k+1} \approx x_k + \operatorname*{arg\,min}_{\|d\| \le \Delta_k} m_k(x_k + d)$$

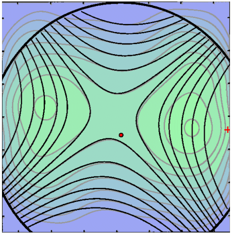
- m_k is the trust-region model (surrogate)
  - $m_k(x) \approx f(x)$ around x_k
  - m_k interpolates f on a set $\mathcal{X}_k$ consisting of previous iterates
- $\|d\| \le \Delta_k$ is the trust-region constraint
  - If "things work well", increase Δ_k
  - Otherwise, decrease Δ_k



$$x_{k+1} \approx x_k + \operatorname*{arg\,min}_{\|d\| \leq \Delta_k} m_k(x_k + d)$$

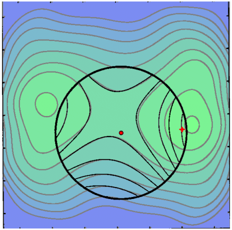
- m_k is the trust-region model (surrogate)
  - $m_k(x) \approx f(x)$ around x_k
  - m_k interpolates f on a set $\mathcal{X}_k$ consisting of previous iterates
- $\|d\| \leq \Delta_k$ is the trust-region constraint
  - If "things work well", increase Δ_k
  - Otherwise, decrease Δ_k

Trust-region DFO methods based on interpolation models



$$x_{k+1} \approx x_k + \operatorname*{arg\,min}_{\|d\| \le \Delta_k} m_k(x_k + d)$$

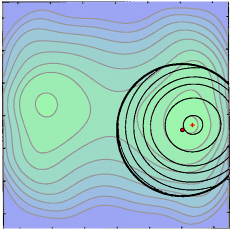
- m_k is the trust-region model (surrogate)
  - $m_k(x) \approx f(x)$ around x_k
  - m_k interpolates f on a set $\mathcal{X}_k$ consisting of previous iterates
- $\|d\| \le \Delta_k$ is the trust-region constraint
  - If "things work well", increase Δ_k
  - Otherwise, decrease Δ_k



$$x_{k+1} \approx x_k + \underset{\|d\| \leq \Delta_k}{\arg\min}\, m_k(x_k + d)$$

- m_k is the trust-region model (surrogate)
  - $m_k(x) \approx f(x)$ around x_k
  - m_k interpolates f on a set $\mathcal{X}_k$ consisting of previous iterates
- $\|d\| \leq \Delta_k$ is the trust-region constraint
  - If "things work well", increase Δ_k
  - Otherwise, decrease Δ_k

Trust-region DFO methods based on interpolation models



$$x_{k+1} \approx x_k + \mathop{\arg\min}_{\|d\| \le \Delta_k} m_k(x_k + d)$$

- m_k is the trust-region model (surrogate)
  - $m_k(x) \approx f(x)$ around x_k
  - m_k interpolates f on a set $\mathcal{X}_k$ consisting of previous iterates
- $\|d\| \le \Delta_k$ is the trust-region constraint
  - If "things work well", increase Δ_k
  - Otherwise, decrease Δ_k

Maintenance of the interpolation set

The interpolation set $\mathcal{X}_k$ must be updated with care.

- $\mathcal{X}_k$ should reuse previous iterates as much as possible.
- The geometry of $\mathcal{X}_k$ must be good enough so that the problem

$$m_k(x) = f(x) \qquad x \in \mathcal{X}_k$$

  is well conditioned.
- Normally, $\mathcal{X}_{k+1} = \mathcal{X}_k \cup \{x_{k+1}\} \setminus \{$a "bad" point$\}$.
- When m_k does not work well and the geometry of $\mathcal{X}_k$ is "bad", geometry-improving steps must be taken.

Table of contents

Curse of dimensionality

"Large" problems are challenging in DFO:

- Trust-region methods normally employ quadratic models of f:

$$f(x) \approx m_k(x) \equiv c_k + {g_k}^\top (x - x_k) + \frac{1}{2}(x - x_k)^\top B_k (x - x_k).$$

  m_k has $\mathcal{O}(n^2)$ degrees of freedom. Difficult to construct if n is large.

- Directional direct search methods typically search along $\mathcal{O}(n)$ directions at each iteration, necessitating $\mathcal{O}(n)$ function evaluations per iteration.

How to reduce the cost by randomization?

Randomized trust-region method: random interpolation set

Idea: construct sparse quadratic models by interpolating f on randomly sampled interpolation sets with much less than $\mathcal{O}(n^2)$ points.

- Bandeira, Scheinberg, and Vicente, Computation of sparse low degree interpolating polynomials and their application to derivative-free optimization, *Math. Program.*, 2012

Under some conditions, the models constructed in this way are "second-order approximations" of f in a probabilistic sense.

Randomized trust-region method: random interpolation set

Global convergence of methods using such models:

- Bandeira, Scheinberg, and Vicente, Convergence of trust-region methods based on probabilistic models, *SIAM J. Optim.*, 2014

Tool: martingale theory.

Worst-case complexity of methods using such models:

- Gratton, Royer, Vicente, and Zhang, Complexity and global rates of trust-region methods based on probabilistic models, *IMA J. Numer. Anal.*, 2017

Tool: measure concentration inequalities (Chernoff).

There are many other results in this direction. See Dr. Cao Liyuan's talk.

- Cartis and Scheinberg, Global convergence rate analysis of unconstrained optimization methods based on probabilistic models, *Math. Program.*, 2018

Randomized trust-region method: random subspace

Idea: Randomly choose a subspace of dimension $\mathcal{O}(1)$, optimize f using a trust-region method (models are cheap), and iterate.

- Cartis and Roberts, Scalable subspace methods for derivative-free nonlinear least-squares optimization, *Math. Program.*, 2022

Tool: Johnson-Lindenstrauss Embeddings

Randomized direct search method

Idea:

- Traditional: search along a deterministic set of directions $\mathcal{D}$, $|\mathcal{D}| = \mathcal{O}(n)$.
- Randomized: search along a random set of directions $\mathcal{D}_k$, $|\mathcal{D}_k| = \mathcal{O}(1)$.

It turns out that $|\mathcal{D}_k| = 2$ is enough in the unconstrained case, no matter how large is the problem.

- Gratton, Royer, Vicente, Zhang, Direct search based on probabilistic descent, *SIAM J. Optim.*, 2015
- Gratton, Royer, Vicente, Zhang, Direct search based on probabilistic feasible descent for bound and linearly constrained problems, *Comput. Optim. Appl.*, 2019

Tool: measure concentration inequalities (Chernoff).

Table of contents

算法 5.18. (NEWUOAs)

步 *1.* 取正数序列 $\{h_k\}$、$\{p_k\}$ 和常数 $\varepsilon \geq 0$. 确定初始点 x_1; $s_0 := 0$; $k := 1$.

步 *2.* 选取整数 $m_k \in [n+1, (n+1)(n+2)/2]$, 调用 MODEL($x_k$, h_k, m_k), 获得 x_k 处的近似梯度 $\tilde{g}_k$. 若 $h_k < \varepsilon$ 且 $\|\tilde{g}_k\| < \varepsilon$, 终止. 令

$$\mathcal{S}_k = \operatorname{span}\{\tilde{g}_k, s_{k-1}\}. \tag{5.59}$$

步 *3.* 设置 RHOEND $= p_k$, 调用 NEWUOA 求解子问题

$$\min_{d \in \mathcal{S}_k} f(x_k + d) \tag{5.60}$$

得到 d_k.

步 *4.* 若 $f(x_k + d_k) < f(x_k)$, 则 $x_{k+1} := x_k + d_k$, $s_k := d_k$; 否则 $x_{k+1} := x_k$, $s_k := s_{k-1}$. $k := k+1$. 转步 *2.*

NEWUOAs (screen-shot taken from Sec. 5.3.3 of my Ph.D. thesis, 2012)

More information on NEWUOAs

- NEWUOAs is fully described in Sec. 5.3 of
  Zhang, *On Derivative-Free Optimization Methods* (无导数优化方法的研究), Ph.D. thesis, Chinese Academy of Sciences, Beijing, 2012
- A general framework for subspace methods is given in Sec. 5.3.1 of the thesis.
- Implementation details of NEWUOAs are presented in Sec. 5.3.5 of the thesis.
- The MATLAB implementation of NEWUOAs is available at

  `https://github.com/newuoas/newuoas` .

- NEWUOAs is included as a DFO solver in the open-source package cm3 at

  `https://github.com/modula3/cm3`

  under `caltech-other/newuoa/src/NewUOAs.m3` .

`github.com/newuoas` `github.com/cm3`

NEWUOAs solving some 2000-dimensional Problems

	f_{start}	f_{best}	$\#f$	CPU (s)
ARWHEAD	5.997000E+03	0.000000E+00	16095	6.42
BRYBND	7.200000E+04	6.486038E−09	50000	26.09
DIXMAANE	1.471453E+04	1.000000E+00	36264	21.12
DIXMAANF	2.734976E+04	1.000000E+00	36384	31.07
DIXMAANG	5.069653E+04	1.000000E+00	36393	22.72
DQRTIC	6.376035E+15	1.214880E−38	40854	14.70
GENHUMPS	5.122260E+07	1.624799E−26	36467	23.54
LIARWHD	1.170000E+06	2.428807E−24	16208	6.73
POWER	2.668667E+09	1.423292E−11	20130	19.19
SPARSQUR	5.627812E+05	6.381755E−30	16209	9.87

N.B.: These results are from 2012. In 2023, we can do much better.

Table of contents

Powell's algorithms and software

- COBYLA: solving general nonlinearly constrained problems using linear models; code released in 1992; paper published in 1994
- UOBYQA: solving unconstrained problems using quadratic models; code released in 2000; paper published in 2002
- NEWUOA: solving unconstrained problems using quadratic models; code released in 2004; paper published in 2006
- BOBYQA: solving bound constrained problems using quadratic models; code released and paper written in 2009
- LINCOA: solving linearly constrained problems using quadratic models; code released in 2013; no paper written

How do these algorithms look like?

DIRECT SEARCH CONSTRAINED OPTIMIZATION 55

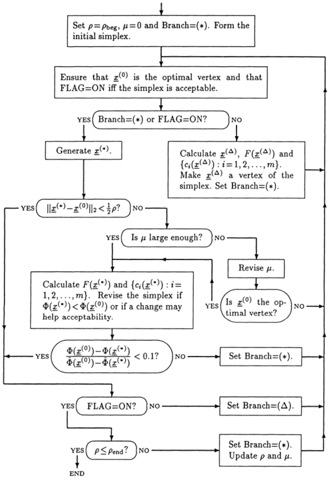
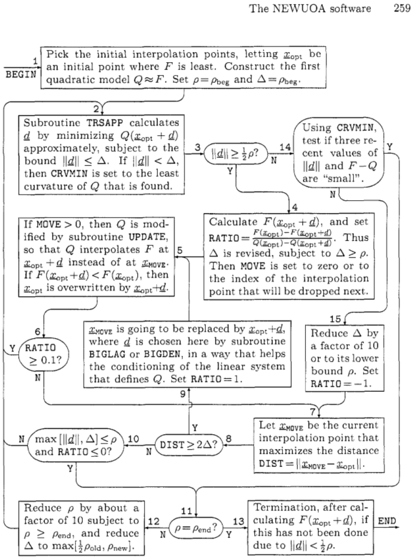


Figure 1: A summary of the algorithm

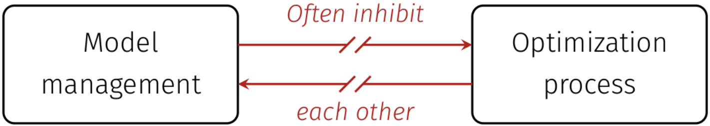
COBYLA

How do these algorithms look like?



NEWUOA

The central difficulty



Implementation of these methods is HARD

> The development of NEWUOA has taken nearly three years. The work was very frustrating [...]

— M. J. D. Powell

The NEWUOA software for unconstrained optimization without derivatives, 2006

Powell's implementation

- Powell implemented all his five methods into publicly available solvers.
- The solvers are widely used by scientists and engineers.
- They are the default benchmarks when designing new algorithms.
- However, the implementation was in Fortran 77, with plenty of GOTOs: in total, 7939 lines of code with 249 GOTOs!

A modernized implementation is greatly needed.

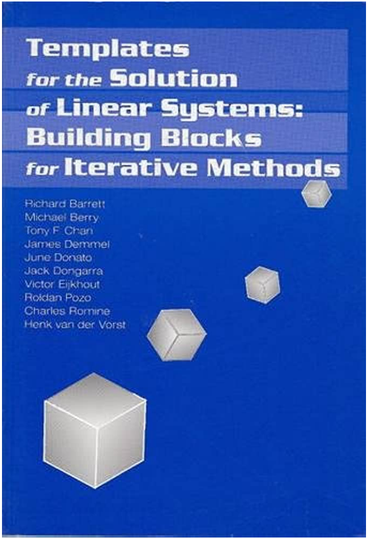
libprima.net

PRIMA is a acronym for

"Reference Implementation for Powell's Methods
with Modernization and Amelioration",

"P" for Powell.

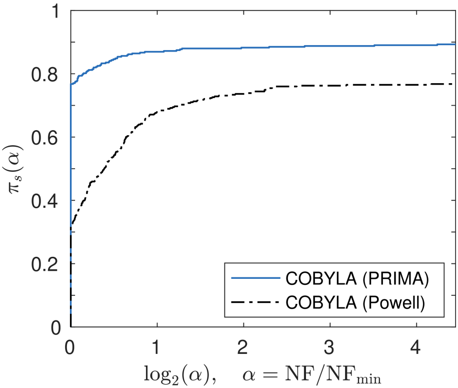
An inspiration



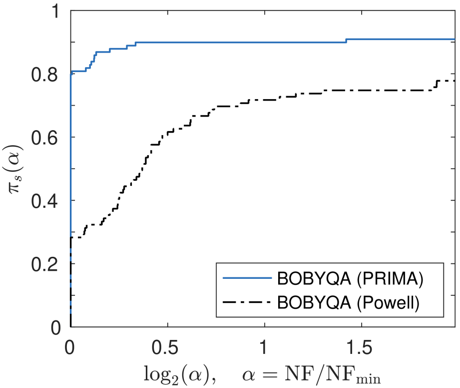
Features of PRIMA

- A structured and modularized way so that they are understandable, maintainable, extendable, fault tolerant, and future proof.
- The code will have no GOTO and will use matrix-vector procedures instead of loops whenever possible.
- The implementation is mathematically equivalent to Powell's except for the bug fixes and improvements we introduce intentionally.
- The implementation is extensively tested by GitHub Actions. The total testing time has exceeded 20 years as of May 2023.
- The inclusion of PRIMA into SciPy is under discussion, and the major SciPy maintainers are positive about it.

The improvement of PRIMA over Powell's implementation



The improvement of PRIMA over Powell's implementation



Summary

- Motivation of derivative-free optimization (DFO)
- Basic methods of DFO
- Randomization strategy for scaling DFO methods to larger problems
- Subspace strategy for scaling DFO methods to larger problems
- Software packages: NEWUOAs, PRIMA, and COBYQA

`github.com/newuoas` `libprima.net` `cobyqa.com`

Thank you!