Model-Based Derivative-Free Optimization: Algorithms and Approximation Theory

Part 1: Overview of DFO

Lindon Roberts, University of Melbourne (lindon.roberts@unimelb.edu.au)
Supported by the Australian Research Council (DE240100006)

MoCaO Lectures
27 July 2026

Overview

The aim of these lectures is to give an overview of the mathematical theory underpinning *model-based derivative-free optimization* (MBDFO), part of the broader class of *derivative-free optimization* (DFO) algorithms.

Plan for this week:

1. Overview of DFO: motivation, applications, algorithm types
2. Key concepts in MBDFO: algorithmic framework, approximation theory, convergence guarantees
3. Towards practical MBDFO algorithms: incremental geometry management, constraints, handling noise, exploiting structure
4. Zaikun Zhang (Sun Yat-Sen University): large-scale problems
5. Sara Shashaani (North Carolina State University): stochastic problems

Key references

The main reference for these lectures is:

- LR, Introduction to Model-Based Derivative-Free Optimization, arXiv:2510.04473 (2025).

Other important references are:

- C. Audet & W. Hare, Derivative-Free and Blackbox Optimization, Springer (2026).
  - *Broad and accessible introduction to DFO methods*
- A. R. Conn, K. Scheinberg & L. N. Vicente, Introduction to Derivative-Free Optimization, SIAM (2009).
  - *Foundational theory of MBDFO*
- J. Larson, M. Menickelly & S. M. Wild, Derivative-Free Optimization Methods, Acta Numerica (2019).
  - *Broad literature survey of DFO methods*

Outline

1. **Motivation**
2. Applications
3. Overview of Algorithms
4. Model-Based DFO

Gradient Descent

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} \; f(\boldsymbol{x})$$

For any $\boldsymbol{x}$, the vector $\nabla f(\boldsymbol{x})$ points in the direction of fastest ascent (locally).

Gradient Descent

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x})$$

For any $\boldsymbol{x}$, the vector $\nabla f(\boldsymbol{x})$ points in the direction of fastest ascent (locally).

Gradient descent iterates to a solution by stepping in the $-\nabla f$ direction

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)$$

Gradient Descent

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x})$$

For any $\boldsymbol{x}$, the vector $\nabla f(\boldsymbol{x})$ points in the direction of fastest ascent (locally).

Gradient descent iterates to a solution by stepping in the $-\nabla f$ direction

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)$$

Theorem

Suppose $f \in C^1(\mathbb{R}^n)$ bounded below and has L-Lipschitz continuous gradients.

If $\alpha_k = 1/L$ for all k, then $\lim_{k\to\infty} \nabla f(\boldsymbol{x}_k) = \boldsymbol{0}$.

Gradient Descent

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x})$$

For any $\boldsymbol{x}$, the vector $\nabla f(\boldsymbol{x})$ points in the direction of fastest ascent (locally).

Gradient descent iterates to a solution by stepping in the $-\nabla f$ direction

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)$$

Theorem

Suppose $f \in C^1(\mathbb{R}^n)$ bounded below and has L-Lipschitz continuous gradients.

If $\alpha_k = 1/L$ for all k, then $\lim_{k\to\infty} \nabla f(\boldsymbol{x}_k) = \mathbf{0}$.

Can get stronger conclusions under different assumptions, e.g. if f is strongly convex, then $\boldsymbol{x}_k \to \boldsymbol{x}^*$.

Newton's Method

The "direction of fastest ascent/descent" is derived from a linear Taylor series.

Newton's Method

The "direction of fastest ascent/descent" is derived from a linear Taylor series. If we have second-order information, then we can use the better Taylor approximation:

$$f(\boldsymbol{x}_k + \boldsymbol{s}) \approx f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T \boldsymbol{s} + \frac{1}{2}\boldsymbol{s}^T \nabla^2 f(\boldsymbol{x}_k)\boldsymbol{s}$$

Newton's Method

The "direction of fastest ascent/descent" is derived from a linear Taylor series. If we have second-order information, then we can use the better Taylor approximation:

$$f(\boldsymbol{x}_k + \boldsymbol{s}) \approx f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T \boldsymbol{s} + \frac{1}{2} \boldsymbol{s}^T \nabla^2 f(\boldsymbol{x}_k) \boldsymbol{s}$$

If $\nabla^2 f(\boldsymbol{x}_k)$ is positive definite, then we can minimize the Taylor approximation, giving Newton's method

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k [\nabla^2 f(\boldsymbol{x}_k)]^{-1} \nabla f(\boldsymbol{x}_k)$$

Newton's Method

The "direction of fastest ascent/descent" is derived from a linear Taylor series. If we have second-order information, then we can use the better Taylor approximation:

$$f(\boldsymbol{x}_k + \boldsymbol{s}) \approx f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T \boldsymbol{s} + \frac{1}{2}\boldsymbol{s}^T \nabla^2 f(\boldsymbol{x}_k)\boldsymbol{s}$$

If $\nabla^2 f(\boldsymbol{x}_k)$ is positive definite, then we can minimize the Taylor approximation, giving Newton's method

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k [\nabla^2 f(\boldsymbol{x}_k)]^{-1} \nabla f(\boldsymbol{x}_k)$$

Theorem

Suppose $f \in C^2(\mathbb{R}^n)$ with Lipschitz continuous Hessian, and $\boldsymbol{x}^*$ is a local minimizer with $\nabla^2 f(\boldsymbol{x}^*) \succ 0$. If $\boldsymbol{x}_0$ is sufficiently close to $\boldsymbol{x}^*$ and $\alpha_k \equiv 1$, then $\boldsymbol{x}_k \to \boldsymbol{x}^*$ quadratically.

Gradient Descent: Practicalities

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k) \qquad \text{or} \qquad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k [\nabla^2 f(\boldsymbol{x}_k)]^{-1} \nabla f(\boldsymbol{x}_k)$$

- How to calculate derivatives of f?
  - Write code by hand
  - Finite differences, $f'(x) \approx \frac{f(x+h)-f(x)}{h}$
  - Algorithmic differentiation/backpropagation

Gradient Descent: Practicalities

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k) \qquad \text{or} \qquad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k [\nabla^2 f(\boldsymbol{x}_k)]^{-1} \nabla f(\boldsymbol{x}_k)$$

- How to calculate derivatives of f?
  - Write code by hand
  - Finite differences, $f'(x) \approx \frac{f(x+h)-f(x)}{h}$
  - Algorithmic differentiation/backpropagation
- Impractical if function evaluation is black-box, computationally expensive or noisy

Gradient Descent: Practicalities

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k) \qquad \text{or} \qquad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k [\nabla^2 f(\boldsymbol{x}_k)]^{-1} \nabla f(\boldsymbol{x}_k)$$

- How to calculate derivatives of f?
  - Write code by hand
  - Finite differences, $f'(x) \approx \frac{f(x+h)-f(x)}{h}$
  - Algorithmic differentiation/backpropagation
- Impractical if function evaluation is black-box, computationally expensive or noisy
- How to pick stepsize/learning rate?
  - Calculate L
  - Hyperparameter tuning
  - Adaptive procedures (e.g. linesearch)

Gradient Descent: Practicalities

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k) \qquad \text{or} \qquad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k [\nabla^2 f(\boldsymbol{x}_k)]^{-1} \nabla f(\boldsymbol{x}_k)$$

- How to calculate derivatives of f?
  - Write code by hand
  - Finite differences, $f'(x) \approx \frac{f(x+h)-f(x)}{h}$
  - Algorithmic differentiation/backpropagation
- Impractical if function evaluation is black-box, computationally expensive or noisy
- How to pick stepsize/learning rate?
  - Calculate L
  - Hyperparameter tuning
  - Adaptive procedures (e.g. linesearch)
- Prefer adaptive procedures (no tuning, fits to local curvature, L often not known)

Derivative-Free Optimization

For black-box, computationally expensive and/or noisy functions, we cannot assume access to gradient information.

Alternative: derivative-free optimization (DFO)

Derivative-Free Optimization

For black-box, computationally expensive and/or noisy functions, we cannot assume access to gradient information.

Alternative: derivative-free optimization (DFO)

- Assume can evaluate $f(\boldsymbol{x})$ but not $\nabla f(\boldsymbol{x})$, etc. (but still assume f is differentiable)
  - Harder again are *comparison oracles*: given $\boldsymbol{x}$ and $\boldsymbol{y}$, say which is better (but no values of f), e.g. survey results [Cai et al., 2022; Scheinberg & Xiong, 2026]
- We will focus on local minimization methods (actually, approximate stationary point: $\|\nabla f(\boldsymbol{x})\| \leq \epsilon$)
- Focus on efficient & adaptive methods with no need for parameter tuning/knowledge of Lipschitz constants

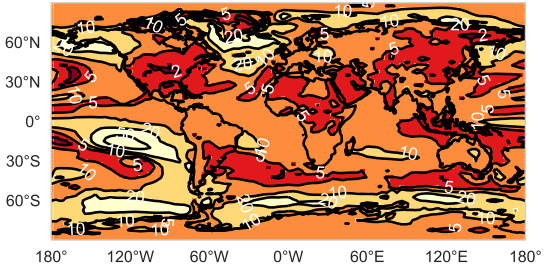
Outline

1. Motivation
2. **Applications**
3. Overview of Algorithms
4. Model-Based DFO

Application 1: Climate Modelling

[Tett et al., 2022]

- Parameter calibration for global climate models (least squares minimization)
- One model run = simulate global climate for 5 years = expensive
- Very complicated, chaotic physics = black-box & noisy



Application 2: Quantum Approximate Optimization Algorithm [Farhi et al., 2014]

- Solve discrete optimization problems (e.g. travelling salesman problem) using quantum hardware
- Convert to QUBO, $\max_{\boldsymbol{v} \in \{0,1\}^m} \boldsymbol{v}^T Q \boldsymbol{v}$, then to equivalent Hamiltonian/maximum eigenvalue problem
- Optimize (continuous) parameters of a simple quantum circuit to approximate the Hamiltonian

$$\max_{\boldsymbol{x} \in \mathbb{R}^n} \mathbb{E}[\varphi(\boldsymbol{x})^* H \varphi(\boldsymbol{x})]$$

  where randomness in $\varphi(\boldsymbol{x})$ comes from stochastic nature of quantum hardware
- Derivatives can be computed, but implementing this is (currently) expensive and noisy due to hardware limitations

Application 3: Adversarial Example Generation [Alzantot et al., 2019]

- Find perturbations of neural network inputs which are misclassified (min. probability of correct label/max. probability of desired incorrect label)
- Neural network structure assumed to be unknown = black-box
- Want to test very few examples $\approx$ expensive
- Useful for copyright protection of artists' work against generative AI [Shan et al., 2024]

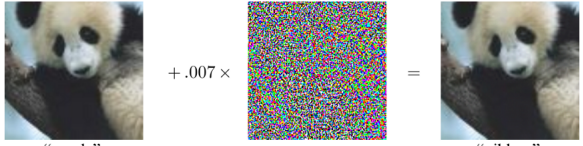


Image from [Goodfellow et al., 2015]

Application 4: Styrene Process Optimization [Audet et al., 2008]

- Key component of polystyrene, produced from ethylbenzene
- Ethylbenzene is heated over a catalyst, to produce styrene + ethylbenzene, extracted via distillation
- Process is simulated using a complex combination of numerical methods (e.g. ODE and nonlinear equation solvers)
- Goal is to optimize the operating parameters of the process (e.g. heating temperature, reactor dimensions) to maximize the net present value of the process, subject to several constraints (e.g. minimum purity of styrene, environmental regulations, max total investment)
- Components of the objective are complex, black-box simulations
- Some constraints are unquantifiable: just get feasible/infeasible flag

Application 5: Simulation Optimization [Kim et al., 2015]

Problems of the form $\min_{\boldsymbol{x} \in \mathbb{R}^n} \mathbb{E}_\omega[f(\boldsymbol{x}, \omega)]$, where $f(\boldsymbol{x}, \omega)$ that can only be simulated.

Application 5: Simulation Optimization

[Kim et al., 2015]

Problems of the form $\min_{\boldsymbol{x} \in \mathbb{R}^n} \mathbb{E}_\omega[f(\boldsymbol{x}, \omega)]$, where $f(\boldsymbol{x}, \omega)$ that can only be simulated.

Bus Arrival Problem

People arrive at a bus stop randomly between times $t = 0$ and $t = 1$ (uniform distribution). A bus will leave at $t = 1$, and we want to schedule a second bus to leave at another time $t \in (0, 1)$. What other bus time minimizes the average waiting time?

Application 5: Simulation Optimization [Kim et al., 2015]

Problems of the form $\min_{\boldsymbol{x} \in \mathbb{R}^n} \mathbb{E}_\omega[f(\boldsymbol{x}, \omega)]$, where $f(\boldsymbol{x}, \omega)$ that can only be simulated.

Bus Arrival Problem

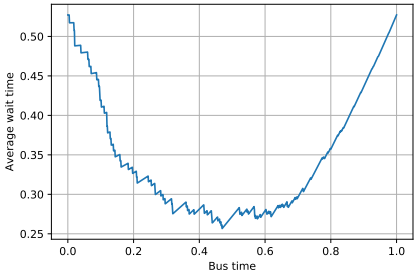
People arrive at a bus stop randomly between times $t = 0$ and $t = 1$ (uniform distribution). A bus will leave at $t = 1$, and we want to schedule a second bus to leave at another time $t \in (0, 1)$. What other bus time minimizes the average waiting time?

Analytic solution easy, but in general (e.g. location of ambulance bases) we would:

- Generate a large number of random passenger arrival times
- For a given t, calculate the average waiting time for these passengers
- Minimize this quantity with respect to t

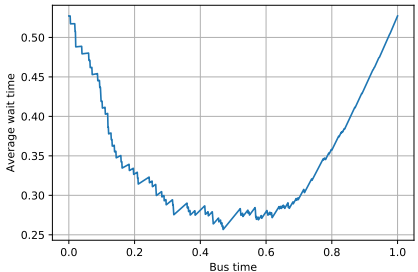
Applications

For example, with $N = 100$ samples, an approximate objective function is:



Applications

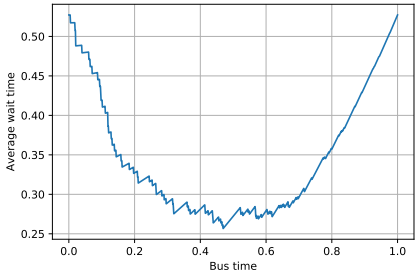
For example, with $N = 100$ samples, an approximate objective function is:



True objective is quadratic, but any finite-N approximation is piecewise linear and *strictly increasing* when continuous. Analytic derivatives are useless, finite differencing often unreliable.

Applications

For example, with $N = 100$ samples, an approximate objective function is:



True objective is quadratic, but any finite-N approximation is piecewise linear and *strictly increasing* when continuous. Analytic derivatives are useless, finite differencing often unreliable. Similar issues arise for more complex examples (e.g. optimizing ambulance base locations).

Applications

In general, DFO is used for optimization problems involving:

- Black-box components (e.g. complex simulations, physical experiments, legacy code libraries)
- Noise (e.g. Monte Carlo simulations, chaotic dynamics)
- Expensive-to-evaluate functions — subjective!
- Except for ML, usually low–medium scale problems (e.g. $n \leq \mathcal{O}(100)$ variables)

Applications

In general, DFO is used for optimization problems involving:

- Black-box components (e.g. complex simulations, physical experiments, legacy code libraries)
- Noise (e.g. Monte Carlo simulations, chaotic dynamics)
- Expensive-to-evaluate functions — subjective!
- Except for ML, usually low–medium scale problems (e.g. $n \leq \mathcal{O}(100)$ variables)
- Inexperienced/lazy/time-constrained users: "I just need a solver that doesn't require me to do anything"

Applications

In general, DFO is used for optimization problems involving:

- Black-box components (e.g. complex simulations, physical experiments, legacy code libraries)
- Noise (e.g. Monte Carlo simulations, chaotic dynamics)
- Expensive-to-evaluate functions — subjective!
- Except for ML, usually low–medium scale problems (e.g. $n \leq \mathcal{O}(100)$ variables)
- Inexperienced/lazy/time-constrained users: "I just need a solver that doesn't require me to do anything"

Hundreds of example use cases in energy systems, material science, astrophysics, etc., and growing use in ML [Alarie et al., 2015; Lin et al., 2026]

Outline

1. Motivation
2. Applications
3. **Overview of Algorithms**
4. Model-Based DFO

Finite Differencing

Is [gradient-based algorithm] with finite differencing a DFO method?

Finite Differencing

Is [gradient-based algorithm] with finite differencing a DFO method?

Usually, no: in practice, finite differencing is treated as a way to produce gradients which are treated as exact.

Finite Differencing

Is [gradient-based algorithm] with finite differencing a DFO method?

Usually, no: in practice, finite differencing is treated as a way to produce gradients which are treated as exact.

Exception are methods where the FD perturbation size is explicitly controlled (e.g. implicit filtering takes perturbation to zero in an outer loop) [Kelley, 2011]

Finite Differencing

Is [gradient-based algorithm] with finite differencing a DFO method?

Usually, no: in practice, finite differencing is treated as a way to produce gradients which are treated as exact.

Exception are methods where the FD perturbation size is explicitly controlled (e.g. implicit filtering takes perturbation to zero in an outer loop) [Kelley, 2011]

My opinion, other DFO people may disagree

Stochastic Gradients

Is [gradient-based algorithm] with stochastic gradients a DFO method?

Stochastic Gradients

Is [gradient-based algorithm] with stochastic gradients a DFO method?

No: a stochastic gradient oracle gives you much more information (n pieces of information). These are gradient-based methods with inexact gradient information.

Global Optimization Algorithms

Most global optimization algorithms are derivative-free (since gradients give purely local information), usually for bound-constrained problems (so a global solution exists).

Global Optimization Algorithms

Most global optimization algorithms are derivative-free (since gradients give purely local information), usually for bound-constrained problems (so a global solution exists).

Global DFO **Local DFO**

Global Optimization Algorithms

Most global optimization algorithms are derivative-free (since gradients give purely local information), usually for bound-constrained problems (so a global solution exists).

Global DFO

- Get global optimum
- Needs to explore whole space

Local DFO

Global Optimization Algorithms

Most global optimization algorithms are derivative-free (since gradients give purely local information), usually for bound-constrained problems (so a global solution exists).

Global DFO

- Get global optimum
- Needs to explore whole space

Local DFO

- Fewer evaluations needed
- More scalable
- Can exploit good starting point

Global Optimization Algorithms

Most global optimization algorithms are derivative-free (since gradients give purely local information), usually for bound-constrained problems (so a global solution exists).

Global DFO

- Get global optimum
- Needs to explore whole space

Local DFO

- Fewer evaluations needed
- More scalable
- Can exploit good starting point

Focus this week is on a specific class of local DFO algorithms (that borrow heavily from gradient-based methods). But first, let's look at some alternatives...

Global DFO: Evolutionary/Genetic Algorithms

Evolutionary algorithms aim to mimic Darwinian evolution:

- Start with a collection ('population') of points & evaluate objective for each
- Delete 'bad' points from the population

Global DFO: Evolutionary/Genetic Algorithms

Evolutionary algorithms aim to mimic Darwinian evolution:

- Start with a collection ('population') of points & evaluate objective for each
- Delete 'bad' points from the population
- Use the remaining 'good' points to randomly build a new population
  - e.g. pick two 'parents' $\boldsymbol{x}_1, \boldsymbol{x}_2$ and create offspring $\boldsymbol{x} = \theta \boldsymbol{x}_1 + (1-\theta)\boldsymbol{x}_2$ with $\theta = \frac{f(\boldsymbol{x}_2)}{f(\boldsymbol{x}_1)+f(\boldsymbol{x}_2)}$

Global DFO: Evolutionary/Genetic Algorithms

Evolutionary algorithms aim to mimic Darwinian evolution:

- Start with a collection ('population') of points & evaluate objective for each
- Delete 'bad' points from the population
- Use the remaining 'good' points to randomly build a new population
  - e.g. pick two 'parents' $\boldsymbol{x}_1, \boldsymbol{x}_2$ and create offspring $\boldsymbol{x} = \theta \boldsymbol{x}_1 + (1-\theta)\boldsymbol{x}_2$ with $\theta = \frac{f(\boldsymbol{x}_2)}{f(\boldsymbol{x}_1)+f(\boldsymbol{x}_2)}$

Very flexible (can be adapted to many settings), but usually inefficient.

Global DFO: Evolutionary/Genetic Algorithms

Evolutionary algorithms aim to mimic Darwinian evolution:

- Start with a collection ('population') of points & evaluate objective for each
- Delete 'bad' points from the population
- Use the remaining 'good' points to randomly build a new population
  - e.g. pick two 'parents' $\boldsymbol{x}_1, \boldsymbol{x}_2$ and create offspring $\boldsymbol{x} = \theta \boldsymbol{x}_1 + (1-\theta)\boldsymbol{x}_2$ with $\theta = \frac{f(\boldsymbol{x}_2)}{f(\boldsymbol{x}_1)+f(\boldsymbol{x}_2)}$

Very flexible (can be adapted to many settings), but usually inefficient.

A reasonable variant with some theory is CMA-ES (population drawn from multivariate Gaussian, mean/covariance adjusted based on subset of population with small objective value). [Hansen & Ostermeier, 1996]

Global DFO: DIRECT

DIRECT (dividing rectangles) is similar to branch-and-bound: [Jones et al., 1993]

- Divide (bounded) domain into rectangles and evaluate objective in the center of each

Global DFO: DIRECT

DIRECT (dividing rectangles) is similar to branch-and-bound: [Jones et al., 1993]

- Divide (bounded) domain into rectangles and evaluate objective in the center of each
- At each iteration, pick rectangle(s) to subdivide based on
  - Current center has small objective value — *exploitation*
  - Current rectangle is large — *exploration*

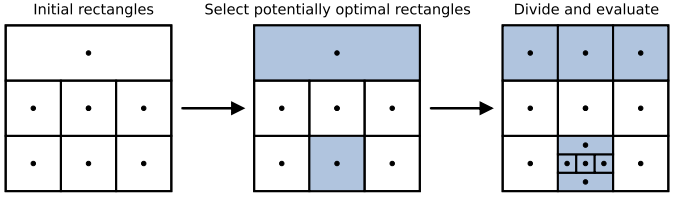


Image based on [Jones et al., 1993]

Global DFO: Bayesian Optimization

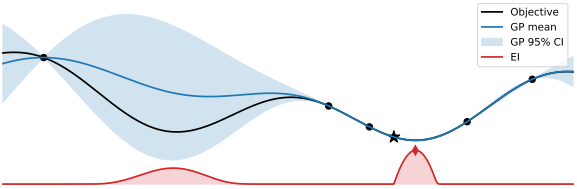
Bayesian Optimization uses objective values to build a global surrogate (i.e. global approximation) to the objective: [Shahriari et al., 2016]

- Given all previous evaluations, construct a Gaussian process surrogate
  - GP surrogate includes mean & uncertainty information (far from evaluations = high uncertainty), $f(\boldsymbol{x}) \approx m(\boldsymbol{x}) \sim \mathcal{N}(\mu(\boldsymbol{x}), \sigma^2(\boldsymbol{x}))$

Global DFO: Bayesian Optimization

Bayesian Optimization uses objective values to build a global surrogate (i.e. global approximation) to the objective: [Shahriari et al., 2016]

- Given all previous evaluations, construct a Gaussian process surrogate
  - GP surrogate includes mean & uncertainty information (far from evaluations = high uncertainty), $f(\boldsymbol{x}) \approx m(\boldsymbol{x}) \sim \mathcal{N}(\mu(\boldsymbol{x}), \sigma^2(\boldsymbol{x}))$
- Globally optimize a quantity related to the (cheap-to-evaluate) surrogate to get the next evaluation
  - e.g. maximize expected improvement, $\text{EI}(\boldsymbol{x}) := \mathbb{E}[\max(f_{\text{min}} - m(\boldsymbol{x}), 0)]$



Local DFO: Direct Search

Direct search methods undertake local optimization by sampling points in a neighborhood of the current iterate using structured/geometric approaches to select new points. [Kolda et al., 2003]

Three important classes of direct search methods are:

- Nelder–Mead (the other simplex method)
- Generalized Pattern Search (GPS)
- Mesh Adaptive Direct Search (MADS)

Local DFO: Direct Search

Direct search methods undertake local optimization by sampling points in a neighborhood of the current iterate using structured/geometric approaches to select new points. [Kolda et al., 2003]

Three important classes of direct search methods are:

- Nelder–Mead (the other simplex method)
- Generalized Pattern Search (GPS)
- Mesh Adaptive Direct Search (MADS)

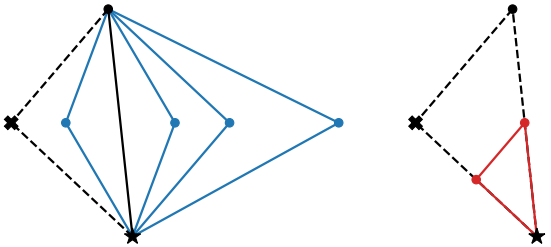
Compared to model-based DFO (the focus of these lectures), direct search methods are usually* less efficient, but their simplicity means they are more developed for more complex problems (mixed integer, nonsmooth, multi-objective, ...).

** unless they are accelerated using model-based techniques*

Local DFO: Nelder–Mead

Nelder–Mead/simplex method maintains a simplex ($n+1$ points in $\mathbb{R}^n$), which is changed at each iteration. [Nelder & Mead, 1965]

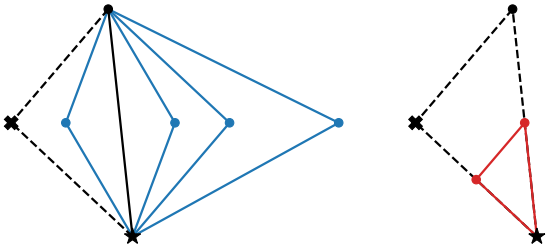
At each iteration, points are ordered by objective value, and (usually) the worst is replaced by a point on the line through the centroid of the rest:



Local DFO: Nelder–Mead

Nelder–Mead/simplex method maintains a simplex ($n+1$ points in $\mathbb{R}^n$), which is changed at each iteration. [Nelder & Mead, 1965]

At each iteration, points are ordered by objective value, and (usually) the worst is replaced by a point on the line through the centroid of the rest:



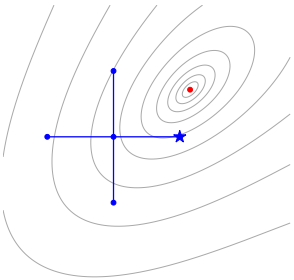
Very popular (Numerical Recipes), but provably non-convergent without modifications (e.g. sufficient decrease) & practically inefficient [McKinnon, 1998; Kelley, 1999]

Local DFO: Generalized Pattern Search

In generalized pattern search (GPS), we search finitely many directions around the current iterate, accepting any improvement and dynamically adjusting the stepsize.

Local DFO: Generalized Pattern Search

In generalized pattern search (GPS), we search finitely many directions around the current iterate, accepting any improvement and dynamically adjusting the stepsize.



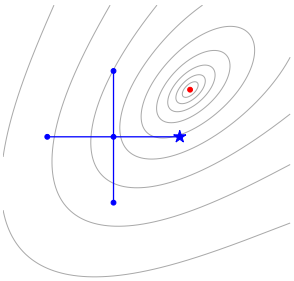
- Poll directions must always contain a descent direction

$$\max_{\boldsymbol{d}\in\mathcal{D}_k} \frac{\boldsymbol{d}^T(-\nabla f(\boldsymbol{x}_k))}{\|\boldsymbol{d}\|\,\|\nabla f(\boldsymbol{x}_k)\|} \geq \kappa > 0$$

- Either accept based on sufficient decrease, or ensure all points lie on a rational lattice

Local DFO: Generalized Pattern Search

In generalized pattern search (GPS), we search finitely many directions around the current iterate, accepting any improvement and dynamically adjusting the stepsize.



- Poll directions must always contain a descent direction

$$\max_{\boldsymbol{d} \in \mathcal{D}_k} \frac{\boldsymbol{d}^T(-\nabla f(\boldsymbol{x}_k))}{\|\boldsymbol{d}\| \, \|\nabla f(\boldsymbol{x}_k)\|} \geq \kappa > 0$$

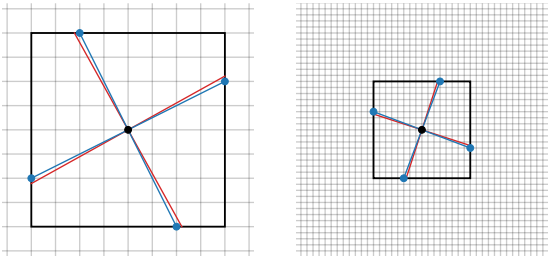
- Either accept based on sufficient decrease, or ensure all points lie on a rational lattice

Standard choice is $\mathcal{D}_k = \{\pm \boldsymbol{e}_1, \ldots, \pm \boldsymbol{e}_n\}$, but theoretical & practical speedups possible using randomized directions (only contain descent direction with some probability).

[Gratton et al., 2015; LR & Royer, 2023]

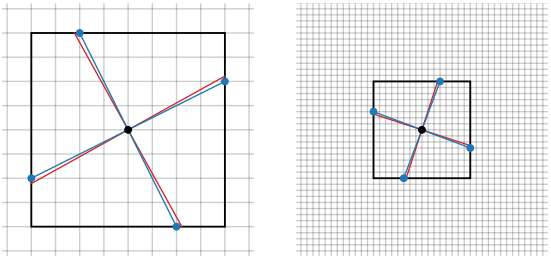
Local DFO: Mesh Adaptive Direct Search

Mesh Adaptive Direct Search (MADS) is similar to GPS, but poll directions are taken from an increasingly fine lattice. [Audet & Dennis, 2006; Abramson et al., 2009]



Local DFO: Mesh Adaptive Direct Search

Mesh Adaptive Direct Search (MADS) is similar to GPS, but poll directions are taken from an increasingly fine lattice. [Audet & Dennis, 2006; Abramson et al., 2009]



Over infinitely many iterations, poll directions are dense on the unit sphere $\implies$ convergence for nonsmooth problems (inc. arbitrary constraints). Basis for widely used NOMAD software. [Le Digabel, 2011; Audet et al., 2022]

Local DFO: Randomized Finite Differencing

In ML, where n is usually very large, zeroth order optimization is usually performed using randomized finite differencing:

$$\nabla f(\boldsymbol{x}) \approx \boldsymbol{g}(\boldsymbol{x}) := \frac{1}{N} \sum_{i=1}^{N} \frac{f(\boldsymbol{x} + h\boldsymbol{u}_i) - f(\boldsymbol{x})}{h} \boldsymbol{u}_i,$$

for finite difference parameter $h > 0$ and random directions $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_N$ (e.g. standard Gaussian). [Ghadimi & Lan, 2013; Nesterov & Spokoiny, 2017]

Local DFO: Randomized Finite Differencing

In ML, where n is usually very large, zeroth order optimization is usually performed using randomized finite differencing:

$$\nabla f(\boldsymbol{x}) \approx \boldsymbol{g}(\boldsymbol{x}) := \frac{1}{N} \sum_{i=1}^{N} \frac{f(\boldsymbol{x} + h\boldsymbol{u}_i) - f(\boldsymbol{x})}{h} \boldsymbol{u}_i,$$

for finite difference parameter $h > 0$ and random directions $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_N$ (e.g. standard Gaussian). [Ghadimi & Lan, 2013; Nesterov & Spokoiny, 2017]

Allows us to get a full-dimensional (i.e. n entry) gradient approximation with few evaluations of $f \implies$ use in your favorite stochastic gradient method.

Local DFO: Randomized Finite Differencing

In ML, where n is usually very large, zeroth order optimization is usually performed using randomized finite differencing:

$$\nabla f(\boldsymbol{x}) \approx \boldsymbol{g}(\boldsymbol{x}) := \frac{1}{N} \sum_{i=1}^{N} \frac{f(\boldsymbol{x} + h\boldsymbol{u}_i) - f(\boldsymbol{x})}{h} \boldsymbol{u}_i,$$

for finite difference parameter $h > 0$ and random directions $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_N$ (e.g. standard Gaussian). [Ghadimi & Lan, 2013; Nesterov & Spokoiny, 2017]

Allows us to get a full-dimensional (i.e. n entry) gradient approximation with few evaluations of $f \implies$ use in your favorite stochastic gradient method.

Can prove bounds reminiscent of SGD analysis: if ∇f is L-Lipschitz continuous,

$$\|\mathbb{E}[\boldsymbol{g}(\boldsymbol{x})] - \nabla f(\boldsymbol{x})\| \leq \sqrt{n} L h \quad \text{and} \quad \|\operatorname{Cov}[\boldsymbol{g}(\boldsymbol{x})]\| \leq \mathcal{O}\left(\frac{\|\nabla f(\boldsymbol{x})\|^2 + n^2 L^2 h^2}{N}\right)$$

[Berahas et al., 2022]

Outline

1. Motivation
2. Applications
3. Overview of Algorithms
4. **Model-Based DFO**

Model-Based DFO

The focus of this week is model-based DFO (MBDFO). MBDFO follows the basic framework of standard derivative-based algorithms, but replaces Taylor-type approximations with locally accurate models.

Model-Based DFO

The focus of this week is model-based DFO (MBDFO). MBDFO follows the basic framework of standard derivative-based algorithms, but replaces Taylor-type approximations with locally accurate models.

The basic approach behind an MBDFO algorithm is:

- Use evaluations at points near $\boldsymbol{x}_k$ to build a local model for the objective
- Perform one iteration of a standard algorithm (using this model instead of a Taylor-type approximation) to get $\boldsymbol{x}_{k+1}$

Model-Based DFO

The focus of this week is model-based DFO (MBDFO). MBDFO follows the basic framework of standard derivative-based algorithms, but replaces Taylor-type approximations with locally accurate models.

The basic approach behind an MBDFO algorithm is:

- Use evaluations at points near $\boldsymbol{x}_k$ to build a local model for the objective
- Perform one iteration of a standard algorithm (using this model instead of a Taylor-type approximation) to get $\boldsymbol{x}_{k+1}$

Theoretically, the core idea is:

- Select 'good points' to build the model
- Model error $= \mathcal{O}$(Taylor series error)
- Derivative-based convergence theory still holds

Assumptions

When we analyze MBDFO, we will usually consider the smooth, nonconvex, unconstrained problem

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x})$$

where f is differentiable (but we don't have access to ∇f in our algorithm)

Assumptions

When we analyze MBDFO, we will usually consider the smooth, nonconvex, unconstrained problem

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x})$$

where f is differentiable (but we don't have access to ∇f in our algorithm)

Assumption (standard)

$f : \mathbb{R}^n \to \mathbb{R}$ has L-Lipschitz gradient and is bounded below by f_{low}

This is standard to prove convergence of algorithms to first-order optimal points (i.e. $\|\nabla f(\boldsymbol{x}_k)\| \to 0$).

Assumptions

When we analyze MBDFO, we will usually consider the smooth, nonconvex, unconstrained problem

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x})$$

where f is differentiable (but we don't have access to ∇f in our algorithm)

Assumption (standard)

$f : \mathbb{R}^n \to \mathbb{R}$ has L-Lipschitz gradient and is bounded below by f_{low}

This is standard to prove convergence of algorithms to first-order optimal points (i.e. $\|\nabla f(\boldsymbol{x}_k)\| \to 0$). Lipschitz gradient implies an error bound on a first-order Taylor series:

$$|f(\boldsymbol{y}) - f(\boldsymbol{x}) - \nabla f(\boldsymbol{x})^T(\boldsymbol{y} - \boldsymbol{x})| \leq \frac{L}{2}\|\boldsymbol{y} - \boldsymbol{x}\|^2, \qquad \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$$

Motivation

Question

What type of (standard) derivative-based algorithm is best suited to MBDFO?

Motivation

Question

What type of (standard) derivative-based algorithm is best suited to MBDFO?

Suppose we analyze a linear model derived from finite differencing:

$$f(\boldsymbol{y}) \approx m_k(\boldsymbol{y}) := f(\boldsymbol{x}_k) + \boldsymbol{g}_k^T(\boldsymbol{y} - \boldsymbol{x}_k), \qquad \text{where} \quad [\boldsymbol{g}_k]_i = \frac{f(\boldsymbol{x}_k + h\boldsymbol{e}_i) - f(\boldsymbol{x}_k)}{h}$$

Motivation

Question

What type of (standard) derivative-based algorithm is best suited to MBDFO?

Suppose we analyze a linear model derived from finite differencing:

$$f(\boldsymbol{y}) \approx m_k(\boldsymbol{y}) := f(\boldsymbol{x}_k) + \boldsymbol{g}_k^T(\boldsymbol{y} - \boldsymbol{x}_k), \qquad \text{where} \quad [\boldsymbol{g}_k]_i = \frac{f(\boldsymbol{x}_k + h\boldsymbol{e}_i) - f(\boldsymbol{x}_k)}{h}$$

Standard error analysis for finite differencing gives e.g. [Nocedal & Wright, 2006]

$$\|\boldsymbol{g}_k - \nabla f(\boldsymbol{x}_k)\|_\infty \leq \frac{Lh}{2} \qquad \implies \qquad \|\boldsymbol{g}_k - \nabla f(\boldsymbol{x}_k)\| \leq \frac{\sqrt{n}Lh}{2}$$

Question

What type of (standard) derivative-based algorithm is best suited to MBDFO?

Suppose we analyze a linear model derived from finite differencing:

$$f(\boldsymbol{y}) \approx m_k(\boldsymbol{y}) := f(\boldsymbol{x}_k) + \boldsymbol{g}_k^T(\boldsymbol{y} - \boldsymbol{x}_k), \qquad \text{where} \quad [\boldsymbol{g}_k]_i = \frac{f(\boldsymbol{x}_k + h\boldsymbol{e}_i) - f(\boldsymbol{x}_k)}{h}$$

Standard error analysis for finite differencing gives e.g. [Nocedal & Wright, 2006]

$$\|\boldsymbol{g}_k - \nabla f(\boldsymbol{x}_k)\|_\infty \leq \frac{Lh}{2} \qquad \implies \qquad \|\boldsymbol{g}_k - \nabla f(\boldsymbol{x}_k)\| \leq \frac{\sqrt{n}Lh}{2}$$

So, the model error is

$$\begin{aligned} |f(\boldsymbol{y}) - m_k(\boldsymbol{y})| &\leq |f(\boldsymbol{y}) - f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_k)^T(\boldsymbol{y} - \boldsymbol{x}_k)| + |(\nabla f(\boldsymbol{x}_k) - \boldsymbol{g}_k)^T(\boldsymbol{y} - \boldsymbol{x}_k)| \\ &\leq \frac{L}{2}\|\boldsymbol{y} - \boldsymbol{x}_k\|^2 + \frac{\sqrt{n}Lh}{2}\|\boldsymbol{y} - \boldsymbol{x}_k\| \end{aligned}$$

Motivation

So, if we choose $\boldsymbol{x}_{k+1}$ to decrease our model, we will need to quantify the decrease in the objective via

$$|f(\boldsymbol{x}_{k+1}) - m_k(\boldsymbol{x}_{k+1})| \leq \frac{L}{2}\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|^2 + \frac{\sqrt{n}Lh}{2}\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|$$

Motivation

So, if we choose $\boldsymbol{x}_{k+1}$ to decrease our model, we will need to quantify the decrease in the objective via

$$|f(\boldsymbol{x}_{k+1}) - m_k(\boldsymbol{x}_{k+1})| \leq \frac{L}{2}\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|^2 + \frac{\sqrt{n}Lh}{2}\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|$$

The first term gives us a limit on how accurate our model can be (Taylor error), so it makes sense to pick h to make the second term a similar size (too small = significant rounding errors)

$$h = \mathcal{O}(\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|)$$

Motivation

So, if we choose $\boldsymbol{x}_{k+1}$ to decrease our model, we will need to quantify the decrease in the objective via

$$|f(\boldsymbol{x}_{k+1}) - m_k(\boldsymbol{x}_{k+1})| \leq \frac{L}{2}\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|^2 + \frac{\sqrt{n}Lh}{2}\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|$$

The first term gives us a limit on how accurate our model can be (Taylor error), so it makes sense to pick h to make the second term a similar size (too small = significant rounding errors)

$$h = \mathcal{O}(\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|)$$

But, we need h to form $\boldsymbol{g}_k$, before $\boldsymbol{x}_{k+1}$ is known!

Motivation

So, if we choose $\boldsymbol{x}_{k+1}$ to decrease our model, we will need to quantify the decrease in the objective via

$$|f(\boldsymbol{x}_{k+1}) - m_k(\boldsymbol{x}_{k+1})| \leq \frac{L}{2}\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|^2 + \frac{\sqrt{n}Lh}{2}\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|$$

The first term gives us a limit on how accurate our model can be (Taylor error), so it makes sense to pick h to make the second term a similar size (too small = significant rounding errors)

$$h = \mathcal{O}(\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|)$$

But, we need h to form $\boldsymbol{g}_k$, before $\boldsymbol{x}_{k+1}$ is known!

Question

Is there a framework that allows us to know $\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|$ a priori?

Trust-Region Methods

Trust-region methods are a standard class of nonconvex optimization algorithms, widely used in practical software. [Conn, Gould & Toint, 2000]

At each iteration, we update a radius $\Delta_k > 0$ that controls the maximum allowable stepsize, based on how accurate the model has been in practice.

Trust-Region Methods

Trust-region methods are a standard class of nonconvex optimization algorithms, widely used in practical software. [Conn, Gould & Toint, 2000]

At each iteration, we update a radius $\Delta_k > 0$ that controls the maximum allowable stepsize, based on how accurate the model has been in practice.

- Given $\boldsymbol{x}_k$, define a local model (e.g. Taylor series, quasi-Newton approximation)

$$f(\boldsymbol{y}) \approx m_k(\boldsymbol{y}) = c + \boldsymbol{g}_k^T(\boldsymbol{y} - \boldsymbol{x}_k) + \frac{1}{2}(\boldsymbol{y} - \boldsymbol{x}_k)^T H_k(\boldsymbol{y} - \boldsymbol{x}_k)$$

Trust-Region Methods

Trust-region methods are a standard class of nonconvex optimization algorithms, widely used in practical software. [Conn, Gould & Toint, 2000]

At each iteration, we update a radius $\Delta_k > 0$ that controls the maximum allowable stepsize, based on how accurate the model has been in practice.

- Given $\boldsymbol{x}_k$, define a local model (e.g. Taylor series, quasi-Newton approximation)

$$f(\boldsymbol{y}) \approx m_k(\boldsymbol{y}) = c + \boldsymbol{g}_k^T(\boldsymbol{y} - \boldsymbol{x}_k) + \frac{1}{2}(\boldsymbol{y} - \boldsymbol{x}_k)^T H_k(\boldsymbol{y} - \boldsymbol{x}_k)$$

- Locally minimize the model to get a tentative step *(how? → tomorrow)*

$$\boldsymbol{s}_k \approx \underset{\|\boldsymbol{s}\| \leq \Delta_k}{\arg\min}\, m_k(\boldsymbol{x}_k + \boldsymbol{s})$$

Trust-Region Methods

- Given tentative step $\boldsymbol{s}_k$, evaluate $f(\boldsymbol{x}_k + \boldsymbol{s}_k)$ and assess the quality of the step

$$\rho_k = \frac{\text{actual decrease}}{\text{predicted decrease}} = \frac{f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k + \boldsymbol{s}_k)}{m_k(\boldsymbol{x}_k) - m_k(\boldsymbol{x}_k + \boldsymbol{s}_k)}$$

Trust-Region Methods

- Given tentative step $\boldsymbol{s}_k$, evaluate $f(\boldsymbol{x}_k + \boldsymbol{s}_k)$ and assess the quality of the step

$$\rho_k = \frac{\text{actual decrease}}{\text{predicted decrease}} = \frac{f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k + \boldsymbol{s}_k)}{m_k(\boldsymbol{x}_k) - m_k(\boldsymbol{x}_k + \boldsymbol{s}_k)}$$

- Accept/reject the step and update trust-region radius:
  - (*Very successful*) If $\rho_k \geq 0.7$, set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$ and $\Delta_{k+1} = 2\Delta_k$
  - (*Successful*) If $0.1 \leq \rho_k < 0.7$, set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$ and $\Delta_{k+1} = \Delta_k$
  - (*Unsuccessful*) If $\rho_k < 0.1$, set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$ and $\Delta_{k+1} = \frac{1}{2}\Delta_k$

Trust-Region Methods

- Given tentative step $\boldsymbol{s}_k$, evaluate $f(\boldsymbol{x}_k + \boldsymbol{s}_k)$ and assess the quality of the step

$$\rho_k = \frac{\text{actual decrease}}{\text{predicted decrease}} = \frac{f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k + \boldsymbol{s}_k)}{m_k(\boldsymbol{x}_k) - m_k(\boldsymbol{x}_k + \boldsymbol{s}_k)}$$

- Accept/reject the step and update trust-region radius:
  - (*Very successful*) If $\rho_k \geq 0.7$, set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$ and $\Delta_{k+1} = 2\Delta_k$
  - (*Successful*) If $0.1 \leq \rho_k < 0.7$, set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$ and $\Delta_{k+1} = \Delta_k$
  - (*Unsuccessful*) If $\rho_k < 0.1$, set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$ and $\Delta_{k+1} = \frac{1}{2}\Delta_k$

  In practice, accept any step with decrease ($\rho_k > 0$), but decrease radius when $\rho_k < 0.1$.

Trust-Region Methods

- Given tentative step $\boldsymbol{s}_k$, evaluate $f(\boldsymbol{x}_k + \boldsymbol{s}_k)$ and assess the quality of the step

$$\rho_k = \frac{\text{actual decrease}}{\text{predicted decrease}} = \frac{f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k + \boldsymbol{s}_k)}{m_k(\boldsymbol{x}_k) - m_k(\boldsymbol{x}_k + \boldsymbol{s}_k)}$$

- Accept/reject the step and update trust-region radius:
  - (*Very successful*) If $\rho_k \geq 0.7$, set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$ and $\Delta_{k+1} = 2\Delta_k$
  - (*Successful*) If $0.1 \leq \rho_k < 0.7$, set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$ and $\Delta_{k+1} = \Delta_k$
  - (*Unsuccessful*) If $\rho_k < 0.1$, set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$ and $\Delta_{k+1} = \frac{1}{2}\Delta_k$

  In practice, accept any step with decrease ($\rho_k > 0$), but decrease radius when $\rho_k < 0.1$.

For convergence, need an accurate model (e.g. Taylor series) and sufficiently good step calculation (predicted decrease sufficiently large).

MBDFO Algorithm

MBDFO

For MBDFO, the accuracy of our model will depend on the desired radius of approximation — trust-region methods guarantee our next iterate lies in this radius.

MBDFO

For MBDFO, the accuracy of our model will depend on the desired radius of approximation — trust-region methods guarantee our next iterate lies in this radius.

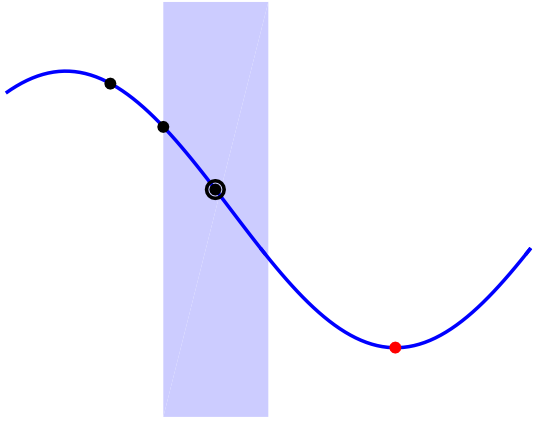
A trust-region MBDFO algorithm looks like:

- Given $\boldsymbol{x}_k$ and Δ_k, evaluate f at some points $\mathcal{Y}_k$ close to $B(\boldsymbol{x}_k, \Delta_k)$ and form an interpolating quadratic model:

$$f(\boldsymbol{y}) \approx m_k(\boldsymbol{y}) := f(\boldsymbol{x}_k) + \boldsymbol{g}_k^T(\boldsymbol{y} - \boldsymbol{x}_k) + \frac{1}{2}(\boldsymbol{y} - \boldsymbol{x}_k)^T H_k(\boldsymbol{y} - \boldsymbol{x}_k)$$

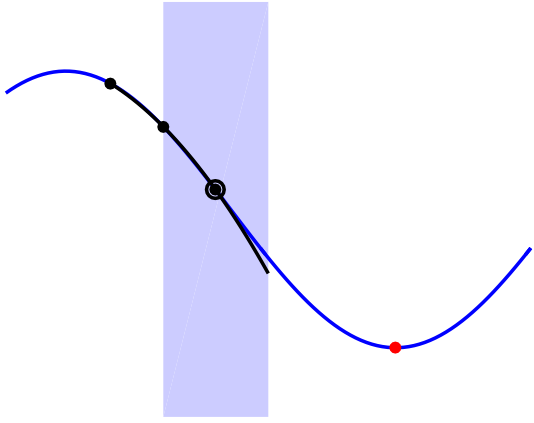
  such that $m_k(\boldsymbol{y}) = f(\boldsymbol{y})$ for all $\boldsymbol{y} \in \mathcal{Y}_k$
- Run a standard trust-region iteration to get $\boldsymbol{x}_{k+1}$ and Δ_{k+1} and use this information to pick $\mathcal{Y}_{k+1}$

Example: Model-Based DFO



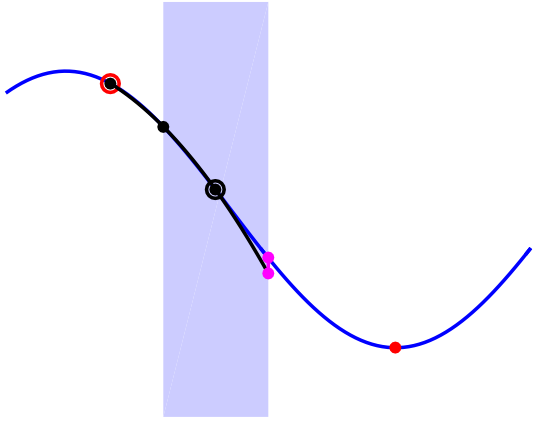
1. Choose interpolation set

Example: Model-Based DFO



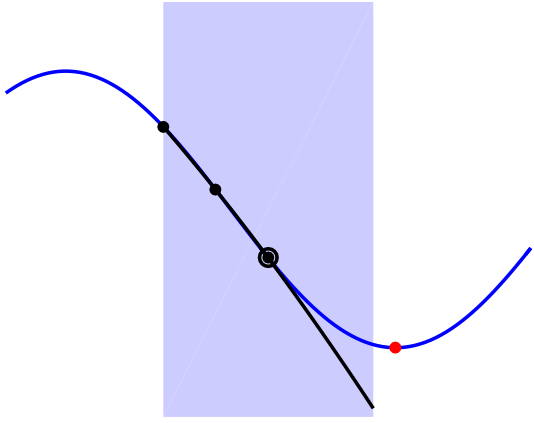
2. Interpolate & minimize...

Example: Model-Based DFO



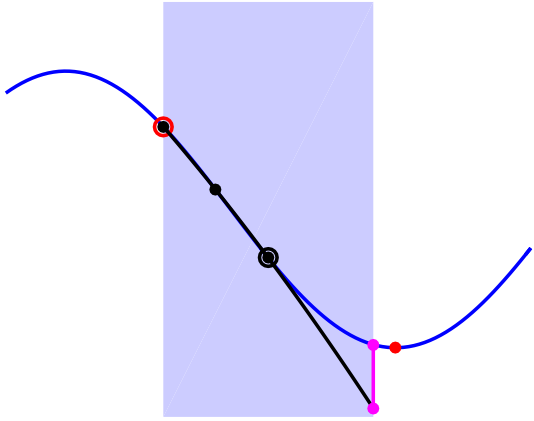
3. Add new point to interpolation set (replace a bad point)

Example: Model-Based DFO



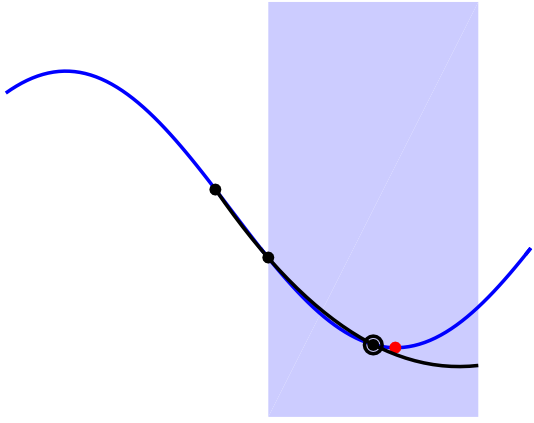
4. Repeat with new interpolation set & model

Example: Model-Based DFO



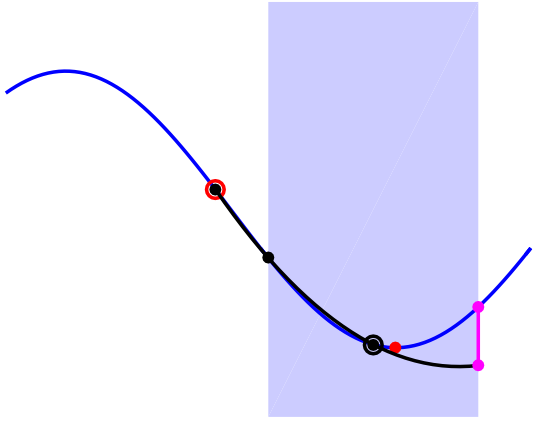
4. Repeat with new interpolation set & model

Example: Model-Based DFO



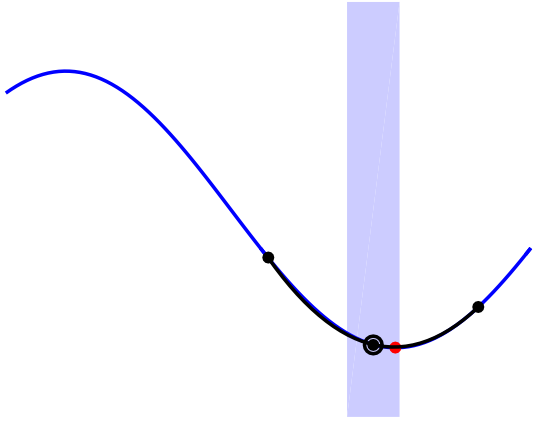
4. Repeat with new interpolation set & model

Example: Model-Based DFO



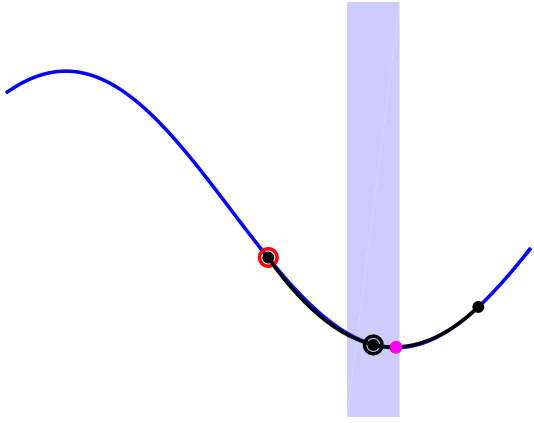
4. Repeat with new interpolation set & model

Example: Model-Based DFO



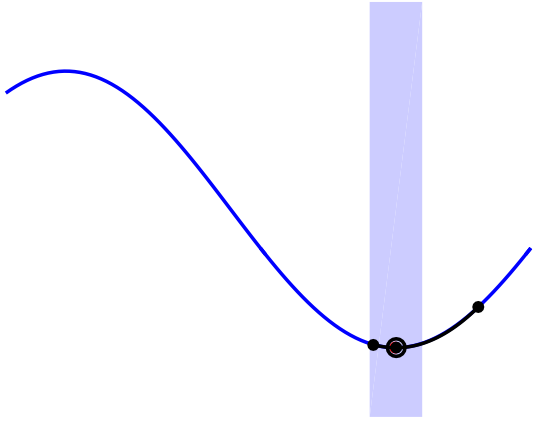
4. Repeat with new interpolation set & model

Example: Model-Based DFO



4. Repeat with new interpolation set & model

Example: Model-Based DFO



4. Repeat with new interpolation set & model

Next steps

There are several important questions that we need to answer tomorrow:

- What interpolation model error bounds are achievable?
- What (if any) modifications are needed to the standard trust-region iteration?
- What is the theoretical convergence penalty (if any) from the absence of gradients?
- How should the interpolation set $\mathcal{Y}_k$ be chosen/updated to guarantee accurate models while minimizing the number of (possibly expensive) objective evaluations required?

References i

M. A. Abramson, C. Audet, J. E. Dennis, and S. L. Digabel, *OrthoMADS: A deterministic MADS instance with orthogonal directions*, SIAM Journal on Optimization, 20 (2009), pp. 948–966.

S. Alarie, C. Audet, A. E. Gheribi, M. Kokkolaras, and S. Le Digabel, *Two decades of blackbox optimization applications*, EURO Journal on Computational Optimization, 9 (2021), p. 100011.

M. Alzantot, Y. Sharma, S. Chakraborty, H. Zhang, C.-J. Hsieh, and M. B. Srivastava, *GenAttack: Practical black-box attacks with gradient-free optimization*, in Proceedings of the Genetic and Evolutionary Computation Conference, Prague, Czech Republic, 2019, ACM, pp. 1111–1119.

C. Audet, V. Béchard, and S. L. Digabel, *Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search*, Journal of Global Optimization, 41 (2008), pp. 299–318.

C. Audet and J. E. Dennis, *Mesh adaptive direct search algorithms for constrained optimization*, SIAM Journal on Optimization, 17 (2006), pp. 188–217.

C. Audet and W. Hare, *Derivative-Free and Blackbox Optimization*, Springer Series in Operations Research and Financial Engineering, Springer, Cham, Switzerland, 2nd ed ed., 2026.

References ii

C. Audet, S. Le Digabel, V. R. Montplaisir, and C. Tribes, *Algorithm 1027: NOMAD version 4: Nonlinear optimization with the MADS algorithm*, ACM Transactions on Mathematical Software, 48 (2022), pp. 1–22.

A. S. Berahas, L. Cao, K. Choromanski, and K. Scheinberg, *A theoretical and empirical comparison of gradient approximations in derivative-free optimization*, Foundations of Computational Mathematics, 22 (2022), pp. 507–560.

H. Cai, D. McKenzie, W. Yin, and Z. Zhang, *A one-bit, comparison-based gradient estimator*, Applied and Computational Harmonic Analysis, 60 (2022), pp. 242–266.

A. R. Conn, N. I. M. Gould, and P. L. Toint, *Trust-Region Methods*, vol. 1 of MPS-SIAM Series on Optimization, MPS/SIAM, Philadelphia, 2000.

A. R. Conn, K. Scheinberg, and L. N. Vicente, *Introduction to Derivative-Free Optimization*, vol. 8 of MPS-SIAM Series on Optimization, MPS/SIAM, Philadelphia, 2009.

E. Farhi, J. Goldstone, and S. Gutmann, *A quantum approximate optimization algorithm*. **arXiv preprint arXiv:1411.4028, 2014.**

References iii

S. Ghadimi and G. Lan, *Stochastic first- and zeroth-order methods for nonconvex stochastic programming*, SIAM Journal on Optimization, 23 (2013), pp. 2341–2368.

I. J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and harnessing adversarial examples*, in 3rd International Conference on Learning Representations ICLR, San Diego, 2015.

S. Gratton, C. W. Royer, L. N. Vicente, and Z. Zhang, *Direct search based on probabilistic descent*, SIAM Journal on Optimization, 25 (2015), pp. 1515–1541.

N. Hansen and A. Ostermeier, *Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation*, in Proceedings of IEEE International Conference on Evolutionary Computation, Nagoya, Japan, 1996, IEEE, pp. 312–317.

D. R. Jones, C. D. Perttunen, and B. E. Stuckman, *Lipschitzian optimization without the Lipschitz constant*, Journal of Optimization Theory and Applications, 79 (1993), pp. 157–181.

C. T. Kelley, *Detection and remediation of stagnation in the Nelder–Mead algorithm using a sufficient decrease condition*, SIAM Journal on Optimization, 10 (1999), pp. 43–55.

———, *Implicit Filtering*, SIAM, Philadelphia, 2011.

References iv

S. Kim, R. Pasupathy, and S. G. Henderson, *A guide to sample average approximation*, in Handbook of Simulation Optimization, M. C. Fu, ed., vol. 216 of International Series in Operations Research & Management Science, Springer New York, New York, NY, 2015.

T. G. Kolda, R. M. Lewis, and V. Torczon, *Optimization by direct search: New perspectives on some classical and modern methods*, SIAM Review, 45 (2003), pp. 385–482.

J. Larson, M. Menickelly, and S. M. Wild, *Derivative-free optimization methods*, Acta Numerica, 28 (2019), pp. 287–404.

S. Le Digabel, *Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm*, ACM Transactions on Mathematical Software, 37 (2011), pp. 1–15.

L. Lin, H. Ma, J. Wang, and S. Yang, *A survey on zeroth-order optimization for machine learning*, in Web Information Systems and Applications, S. Yang, J. Mao, M. Yu, C. Jin, and L. Chao, eds., vol. 16299, Springer Nature Singapore, Singapore, 2026, pp. 481–497.

K. I. M. McKinnon, *Convergence of the Nelder–Mead simplex method to a nonstationary point*, SIAM Journal on Optimization, 9 (1998), pp. 148–158.

References v

J. A. Nelder and R. Mead, *A simplex method for function minimization*, The Computer Journal, 7 (1965), pp. 308–313.

Y. Nesterov and V. Spokoiny, *Random gradient-free minimization of convex functions*, Foundations of Computational Mathematics, 17 (2017), pp. 527–566.

J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer Series in Operations Research and Financial Engineering, Springer, New York, 2nd ed., 2006.

L. Roberts, *Introduction to model-based derivative-free optimization*.
arXiv preprint arXiv:2510.04473, 2025.

L. Roberts and C. W. Royer, *Direct search based on probabilistic descent in reduced spaces*, SIAM Journal on Optimization, 33 (2023), pp. 3057–3082.

K. Scheinberg and Z. Xiong, *Function-free optimization via comparison oracles*.
arXiv preprint arXiv:2604.26867, 2026.

B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, *Taking the human out of the loop: A review of Bayesian optimization*, Proceedings of the IEEE, 104 (2016), pp. 148–175.

References vi

S. Shan, W. Ding, J. Passananti, S. Wu, H. Zheng, and B. Y. Zhao, *Nightshade: Prompt-specific poisoning attacks on text-to-image generative models*, in 2024 IEEE Symposium on Security and Privacy (SP), 2024, pp. 807–825.

S. F. B. Tett, J. M. Gregory, N. Freychet, C. Cartis, M. J. Mineter, and L. Roberts, *Does model calibration reduce uncertainty in climate projections?*, Journal of Climate, 35 (2022), pp. 2585–2602.