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

Part 3: Practical Model-Based DFO Algorithms

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

MoCaO Lectures
29 July 2026

Model-Based DFO

We want to solve

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

where f is nonconvex and has L-Lipschitz continuous gradient (not accessible to algorithm!).

Model-Based DFO

We want to solve

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

where f is nonconvex and has L-Lipschitz continuous gradient (not accessible to algorithm!).

MBDFO trust-region method steps (convergence/complexity proven last time):

- Evaluate f at ($\geq n+1$ well-spaced) points near $\boldsymbol{x}_k$
- Using these evaluations, build a local quadratic interpolation model to approximate f near $\boldsymbol{x}_k$
- Calculate a step $\|\boldsymbol{s}_k\| \leq \Delta_k$ that decreases the model and accept/reject based on $f(\boldsymbol{x}_k + \boldsymbol{s}_k)$

e.g. [Conn et al., 2009]

Outline

1. **Incremental Geometry Improvement**
2. Constrained Optimization
3. Noisy Problems
4. Structured Problems
5. Software & Outlook

Incremental Geometry Improvement

An important use case of DFO is where objective evaluations are expensive.

Incremental Geometry Improvement

An important use case of DFO is where objective evaluations are expensive.

The existing MBDFO method is based on structured interpolation sets, e.g. $\{\boldsymbol{x}_k, \boldsymbol{x}_k + \Delta_k \boldsymbol{e}_1, \ldots, \boldsymbol{x}_k + \Delta_k \boldsymbol{e}_n\}$, so evaluations at these points can't be reused between iterations.

Incremental Geometry Improvement

An important use case of DFO is where objective evaluations are expensive.

The existing MBDFO method is based on structured interpolation sets, e.g. $\{\boldsymbol{x}_k, \boldsymbol{x}_k + \Delta_k \boldsymbol{e}_1, \ldots, \boldsymbol{x}_k + \Delta_k \boldsymbol{e}_n\}$, so evaluations at these points can't be reused between iterations.

Is there a sensible way to retain interpolation points across iterations?

Incremental Geometry Improvement

An important use case of DFO is where objective evaluations are expensive.

The existing MBDFO method is based on structured interpolation sets, e.g. $\{\boldsymbol{x}_k, \boldsymbol{x}_k + \Delta_k \boldsymbol{e}_1, \ldots, \boldsymbol{x}_k + \Delta_k \boldsymbol{e}_n\}$, so evaluations at these points can't be reused between iterations.

Is there a sensible way to retain interpolation points across iterations?

If yes, how do we ensure fully linear models? How do we decide which points are retained, and which points are changed (e.g. to include $\boldsymbol{x}_k + \boldsymbol{s}_k$)?

Lagrange Polynomials

We will use the Lagrange polynomials associated with an interpolation set.

Lagrange Polynomials

We will use the Lagrange polynomials associated with an interpolation set.

Given a set of points $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_p \in \mathbb{R}^n$, the associated Lagrange polynomials $\ell_1, \ldots, \ell_p : \mathbb{R}^n \to \mathbb{R}$ (one per point) satisfy $\ell_i(\boldsymbol{y}_j) = \delta_{i,j}$ and have the correct degree corresponding to the type of interpolation we are doing.

Lagrange Polynomials

We will use the Lagrange polynomials associated with an interpolation set.

Given a set of points $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_p \in \mathbb{R}^n$, the associated Lagrange polynomials $\ell_1, \ldots, \ell_p : \mathbb{R}^n \to \mathbb{R}$ (one per point) satisfy $\ell_i(\boldsymbol{y}_j) = \delta_{i,j}$ and have the correct degree corresponding to the type of interpolation we are doing.

Theorem (e.g. Trefethen, 2020)

For distinct interpolation points $y_1, \ldots, y_{d+1} \in [-1, 1]$, fit a polynomial of degree d to $f : [-1, 1] \to \mathbb{R}$. Then (note $\|f\|_\infty := \max_{y \in [-1,1]} |f(y)|$)

$$\|f - p_d\|_\infty \leq (\Lambda_1 + 1)\|f - p_d^*\|_\infty$$

where p_d^* is the optimal degree-d approximation to f, and $\Lambda_1 := \max_{y \in [-1,1]} \sum_{i=1}^{d+1} |\ell_i(y)|$ is the Lebesgue constant of the interpolation set.

Lagrange Polynomials

Suppose we are doing linear interpolation with points $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{n+1} \in \mathbb{R}^n$.

Lagrange Polynomials

Suppose we are doing linear interpolation with points $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{n+1} \in \mathbb{R}^n$. The associated Lagrange polynomials satisfy

$$\ell_i(\boldsymbol{y}) = c_i + \boldsymbol{g}_i^T(\boldsymbol{y} - \boldsymbol{x}_k), \qquad \text{such that} \quad \underbrace{\begin{bmatrix} 1 & (\boldsymbol{y}_1 - \boldsymbol{x}_k)^T/\Delta_k \\ \vdots & \vdots \\ 1 & (\boldsymbol{y}_p - \boldsymbol{x}_k)^T/\Delta_k \end{bmatrix}}_{=:\hat{M}} \begin{bmatrix} c_i \\ \Delta_k \boldsymbol{g}_i \end{bmatrix} = \boldsymbol{e}_i$$

where $\hat{M}$ is the same matrix used for linear interpolation yesterday.

Lagrange Polynomials

Suppose we are doing linear interpolation with points $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{n+1} \in \mathbb{R}^n$. The associated Lagrange polynomials satisfy

$$\ell_i(\boldsymbol{y}) = c_i + \boldsymbol{g}_i^T(\boldsymbol{y} - \boldsymbol{x}_k), \qquad \text{such that} \quad \underbrace{\begin{bmatrix} 1 & (\boldsymbol{y}_1 - \boldsymbol{x}_k)^T/\Delta_k \\ \vdots & \vdots \\ 1 & (\boldsymbol{y}_p - \boldsymbol{x}_k)^T/\Delta_k \end{bmatrix}}_{=:\hat{M}} \begin{bmatrix} c_i \\ \Delta_k \boldsymbol{g}_i \end{bmatrix} = \boldsymbol{e}_i$$

where $\hat{M}$ is the same matrix used for linear interpolation yesterday.

As a consequence,

$$\ell_i(\boldsymbol{y}) = \begin{bmatrix} c_i \\ \Delta_k \boldsymbol{g}_i \end{bmatrix}^T \begin{bmatrix} 1 \\ (\boldsymbol{y} - \boldsymbol{x}_k)/\Delta_k \end{bmatrix} = \boldsymbol{e}_i^T \hat{M}^{-T} \begin{bmatrix} 1 \\ (\boldsymbol{y} - \boldsymbol{x}_k)/\Delta_k \end{bmatrix}$$

Lagrange Polynomials

Suppose we are doing linear interpolation with points $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{n+1} \in \mathbb{R}^n$. The associated Lagrange polynomials satisfy

$$\ell_i(\boldsymbol{y}) = c_i + \boldsymbol{g}_i^T(\boldsymbol{y} - \boldsymbol{x}_k), \qquad \text{such that} \quad \underbrace{\begin{bmatrix} 1 & (\boldsymbol{y}_1 - \boldsymbol{x}_k)^T/\Delta_k \\ \vdots & \vdots \\ 1 & (\boldsymbol{y}_p - \boldsymbol{x}_k)^T/\Delta_k \end{bmatrix}}_{=:\hat{M}} \begin{bmatrix} c_i \\ \Delta_k \boldsymbol{g}_i \end{bmatrix} = \boldsymbol{e}_i$$

where $\hat{M}$ is the same matrix used for linear interpolation yesterday.

As a consequence,

$$\boldsymbol{\lambda}(\boldsymbol{y}) := \begin{bmatrix} \ell_1(\boldsymbol{y}) \\ \vdots \\ \ell_p(\boldsymbol{y}) \end{bmatrix} = \hat{M}^{-T} \begin{bmatrix} 1 \\ (\boldsymbol{y} - \boldsymbol{x}_k)/\Delta_k \end{bmatrix}$$

Lagrange Polynomials

Yesterday, we got an error bound for linear interpolation models ($\|\boldsymbol{y}_i - \boldsymbol{x}_k\| \leq \beta\Delta_k$):

$$|m_k(\boldsymbol{y}) - f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_k)^T(\boldsymbol{y} - \boldsymbol{x}_k)| \leq \frac{L}{2}\beta^2\Delta_k^2\|\boldsymbol{v}\|_1 \quad \text{where} \quad \hat{M}^T\boldsymbol{v} = \begin{bmatrix} 1 \\ (\boldsymbol{y} - \boldsymbol{x}_k)/\Delta_k \end{bmatrix}$$

Lagrange Polynomials

Yesterday, we got an error bound for linear interpolation models ($\|\boldsymbol{y}_i - \boldsymbol{x}_k\| \leq \beta\Delta_k$):

$$|m_k(\boldsymbol{y}) - f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_k)^T(\boldsymbol{y} - \boldsymbol{x}_k)| \leq \frac{L}{2}\beta^2\Delta_k^2\|\boldsymbol{v}\|_1 \quad \text{where} \quad \hat{M}^T\boldsymbol{v} = \begin{bmatrix} 1 \\ (\boldsymbol{y} - \boldsymbol{x}_k)/\Delta_k \end{bmatrix}$$

That is, $\boldsymbol{v} = \boldsymbol{\lambda}(\boldsymbol{y})$, and we get something like the Lebesgue measure

$$|m_k(\boldsymbol{y}) - f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_k)^T(\boldsymbol{y} - \boldsymbol{x}_k)| \leq \frac{L}{2}\beta^2\Delta_k^2\|\boldsymbol{\lambda}(\boldsymbol{y})\|_1$$

Lagrange Polynomials

Yesterday, we got an error bound for linear interpolation models ($\|\boldsymbol{y}_i - \boldsymbol{x}_k\| \leq \beta\Delta_k$):

$$|m_k(\boldsymbol{y}) - f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_k)^T(\boldsymbol{y} - \boldsymbol{x}_k)| \leq \frac{L}{2}\beta^2\Delta_k^2\|\boldsymbol{v}\|_1 \quad \text{where} \quad \hat{M}^T\boldsymbol{v} = \begin{bmatrix} 1 \\ (\boldsymbol{y} - \boldsymbol{x}_k)/\Delta_k \end{bmatrix}$$

That is, $\boldsymbol{v} = \boldsymbol{\lambda}(\boldsymbol{y})$, and we get something like the Lebesgue measure

$$|m_k(\boldsymbol{y}) - f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_k)^T(\boldsymbol{y} - \boldsymbol{x}_k)| \leq \frac{L}{2}\beta^2\Delta_k^2\|\boldsymbol{\lambda}(\boldsymbol{y})\|_1$$

Although the 1-norm gives better bounds, it won't give a practical algorithm, so instead we use the ∞-norm, $\|\boldsymbol{\lambda}(\boldsymbol{y})\|_1 \leq p\|\boldsymbol{\lambda}(\boldsymbol{y})\|_\infty$.

Λ-poisedness

This motivates the following definition: [Conn et al., 2007]

Definition

Given $\Lambda > 0$, an interpolation set is Λ-poised for interpolation in $B(\boldsymbol{x}_k, \Delta_k)$ if

$$\max_{\boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k)} \|\boldsymbol{\lambda}(\boldsymbol{y})\|_\infty \leq \Lambda$$

Λ-poisedness

This motivates the following definition: [Conn et al., 2007]

Definition

Given $\Lambda > 0$, an interpolation set is Λ-poised for interpolation in $B(\boldsymbol{x}_k, \Delta_k)$ if

$$\max_{\boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k)} \|\boldsymbol{\lambda}(\boldsymbol{y})\|_\infty \leq \Lambda$$

Λ measures the 'quality of the the interpolation set' (in a geometric sense), smaller values are better.

Λ-poisedness

This motivates the following definition: [Conn et al., 2007]

Definition

Given $\Lambda > 0$, an interpolation set is Λ-poised for interpolation in $B(\boldsymbol{x}_k, \Delta_k)$ if

$$\max_{\boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k)} \|\boldsymbol{\lambda}(\boldsymbol{y})\|_\infty \leq \Lambda$$

Λ measures the 'quality of the the interpolation set' (in a geometric sense), smaller values are better. Note: if $\boldsymbol{x}_k$ is an interpolation point (usually true) then $\Lambda \geq 1$.

Λ-poisedness

This motivates the following definition: [Conn et al., 2007]

Definition

Given $\Lambda > 0$, an interpolation set is Λ-poised for interpolation in $B(\boldsymbol{x}_k, \Delta_k)$ if

$$\max_{\boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k)} \|\boldsymbol{\lambda}(\boldsymbol{y})\|_\infty \leq \Lambda$$

Λ measures the 'quality of the the interpolation set' (in a geometric sense), smaller values are better. Note: if $\boldsymbol{x}_k$ is an interpolation point (usually true) then $\Lambda \geq 1$.

Corollary

A Λ-poised linear interpolation model is fully linear with $\kappa_{\text{mf}}, \kappa_{\text{mg}} = \mathcal{O}(L\beta^2 p\Lambda)$.

Λ-poisedness

Similar results hold for fully quadratic interpolation.

Λ-poisedness

Similar results hold for fully quadratic interpolation.

For minimum Frobenius norm (MFN) interpolation, need to define Lagrange polynomials in an MFN-type way:

$$\min_{c_i, \boldsymbol{g}_i, H_i} \|H_i\|_F^2 \qquad \text{s.t.} \quad H_i = H_i^T \text{ and } \ell_i(\boldsymbol{y}_j) = \delta_{i,j} \ \forall i, j = 1, \ldots, p$$

then can define Λ-poisedness the same way, $\max_{\boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k)} \|\boldsymbol{\lambda}(\boldsymbol{y})\|_\infty \leq \Lambda$.

Λ-poisedness

Similar results hold for fully quadratic interpolation.

For minimum Frobenius norm (MFN) interpolation, need to define Lagrange polynomials in an MFN-type way:

$$\min_{c_i, \boldsymbol{g}_i, H_i} \|H_i\|_F^2 \qquad \text{s.t.} \quad H_i = H_i^T \text{ and } \ell_i(\boldsymbol{y}_j) = \delta_{i,j} \ \forall i, j = 1, \ldots, p$$

then can define Λ-poisedness the same way, $\max_{\boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k)} \|\boldsymbol{\lambda}(\boldsymbol{y})\|_\infty \leq \Lambda$.

Then, we get a similar result:

Corollary [LR, 2025], originally [Conn et al., 2008]

A Λ-poised MFN interpolation model is fully linear with $\kappa_{\text{mf}}, \kappa_{\text{mg}} = \mathcal{O}((L + \kappa_H)\beta^2 p \Lambda)$, where $\|H_k\| \leq \kappa_H = \mathcal{O}(Lp\beta^2\Lambda)$.

Geometry Improvement

The reason Lagrange polynomials and Λ-poisedness are so useful is that we get a clear way to check/improve the interpolation set.

Geometry Improvement

The reason Lagrange polynomials and Λ-poisedness are so useful is that we get a clear way to check/improve the interpolation set.

Geometry improvement algorithm:

- Calculate Lagrange polynomials and evaluate $\Lambda = \max_{\boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k)} \max_{i=1,\ldots,p} |\ell_i(\boldsymbol{y})|$

Geometry Improvement

The reason Lagrange polynomials and Λ-poisedness are so useful is that we get a clear way to check/improve the interpolation set.

Geometry improvement algorithm:

- Calculate Lagrange polynomials and evaluate $\Lambda = \max_{\boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k)} \max_{i=1,\ldots,p} |\ell_i(\boldsymbol{y})|$
- If $\Lambda \leq \Lambda^*$, terminate (good geometry)

Geometry Improvement

The reason Lagrange polynomials and Λ-poisedness are so useful is that we get a clear way to check/improve the interpolation set.

Geometry improvement algorithm:

- Calculate Lagrange polynomials and evaluate $\Lambda = \max_{\boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k)} \max_{i=1,\ldots,p} |\ell_i(\boldsymbol{y})|$
- If $\Lambda \leq \Lambda^*$, terminate (good geometry)
- Otherwise, find $\boldsymbol{y}$ and $i \in \{1, \ldots, p\}$ with $|\ell_i(\boldsymbol{y})| > \Lambda^*$ and replace $\boldsymbol{y}_i$ with $\boldsymbol{y}$ in the interpolation set

Geometry Improvement

The reason Lagrange polynomials and Λ-poisedness are so useful is that we get a clear way to check/improve the interpolation set.

Geometry improvement algorithm:

- Calculate Lagrange polynomials and evaluate $\Lambda = \max_{\boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k)} \max_{i=1,\ldots,p} |\ell_i(\boldsymbol{y})|$
- If $\Lambda \leq \Lambda^*$, terminate (good geometry)
- Otherwise, find $\boldsymbol{y}$ and $i \in \{1, \ldots, p\}$ with $|\ell_i(\boldsymbol{y})| > \Lambda^*$ and replace $\boldsymbol{y}_i$ with $\boldsymbol{y}$ in the interpolation set

Theorem

Provided $\Lambda^* > 1$, this algorithm terminates and returns a Λ^*-poised set in finite time.

Geometry Improvement

The reason Lagrange polynomials and Λ-poisedness are so useful is that we get a clear way to check/improve the interpolation set.

Geometry improvement algorithm:

- Calculate Lagrange polynomials and evaluate $\Lambda = \max_{\boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k)} \max_{i=1,\dots,p} |\ell_i(\boldsymbol{y})|$
- If $\Lambda \leq \Lambda^*$, terminate (good geometry)
- Otherwise, find $\boldsymbol{y}$ and $i \in \{1, \dots, p\}$ with $|\ell_i(\boldsymbol{y})| > \Lambda^*$ and replace $\boldsymbol{y}_i$ with $\boldsymbol{y}$ in the interpolation set

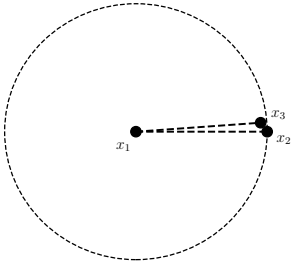
Theorem

Provided $\Lambda^* > 1$, this algorithm terminates and returns a Λ^*-poised set in finite time.

Proof idea: one replacement increases determinant of interpolation matrix by a factor of at least $|\ell_i(\boldsymbol{y})| > \Lambda^*$ but determinant is bounded since points stay in $B(\boldsymbol{x}_k, \Delta_k)$.

Geometry Improvement

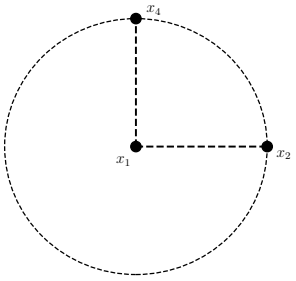
2D example: improving geometry in $B(\mathbf{0}, 1)$



$(0, 0)$**,** $(1, 0)$ **and** $(0.95, 0.07)$ **give** $\Lambda \approx 14.3$

Geometry Improvement

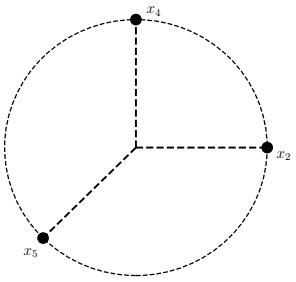
2D example: improving geometry in $B(\mathbf{0}, 1)$



$(0, 0)$, $(1, 0)$ and $(0, 1)$ give $\Lambda \approx 2.41$

Geometry Improvement

2D example: improving geometry in $B(\mathbf{0}, 1)$



$(-0.7071, -0.7071)$**,** $(1, 0)$ **and** $(0, 1)$ **give** $\Lambda \approx 1.06$

Incremental Geometry Algorithm

To give fully linear models, our interpolation set needs to have:

Incremental Geometry Algorithm

To give fully linear models, our interpolation set needs to have:

- All points close to $\boldsymbol{x}_k$, i.e. $\|\boldsymbol{y}_i - \boldsymbol{x}_k\| \leq \beta\Delta_k$ for all $i = 1, \ldots, p$

Incremental Geometry Algorithm

To give fully linear models, our interpolation set needs to have:

- All points close to $\boldsymbol{x}_k$, i.e. $\|\boldsymbol{y}_i - \boldsymbol{x}_k\| \leq \beta\Delta_k$ for all $i = 1, \ldots, p$
- Λ-poisedness, i.e. $\max_{\boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k)} |\ell_i(\boldsymbol{y})| \leq \Lambda$ for all $i = 1, \ldots, p$

Incremental Geometry Algorithm

To give fully linear models, our interpolation set needs to have:

- All points close to $\boldsymbol{x}_k$, i.e. $\|\boldsymbol{y}_i - \boldsymbol{x}_k\| \leq \beta\Delta_k$ for all $i = 1, \ldots, p$
- Λ-poisedness, i.e. $\max_{\boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k)} |\ell_i(\boldsymbol{y})| \leq \Lambda$ for all $i = 1, \ldots, p$

This idea motivates a incremental geometry improvement algorithm that has convergence theory and is practically useful. [Scheinberg & Toint, 2010]

Incremental Geometry Algorithm

To give fully linear models, our interpolation set needs to have:

- All points close to $\boldsymbol{x}_k$, i.e. $\|\boldsymbol{y}_i - \boldsymbol{x}_k\| \leq \beta\Delta_k$ for all $i = 1, \ldots, p$
- Λ-poisedness, i.e. $\max_{\boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k)} |\ell_i(\boldsymbol{y})| \leq \Lambda$ for all $i = 1, \ldots, p$

This idea motivates a incremental geometry improvement algorithm that has convergence theory and is practically useful. [Scheinberg & Toint, 2010]

Question

But first, do we genuinely need to consider geometry (Λ-poisedness)? Why not just ensure all points close to $\boldsymbol{x}_k$?

Incremental Geometry Algorithm

To give fully linear models, our interpolation set needs to have:

- All points close to $\boldsymbol{x}_k$, i.e. $\|\boldsymbol{y}_i - \boldsymbol{x}_k\| \leq \beta\Delta_k$ for all $i = 1, \ldots, p$
- Λ-poisedness, i.e. $\max_{\boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k)} |\ell_i(\boldsymbol{y})| \leq \Lambda$ for all $i = 1, \ldots, p$

This idea motivates a incremental geometry improvement algorithm that has convergence theory and is practically useful. [Scheinberg & Toint, 2010]

Question

But first, do we genuinely need to consider geometry (Λ-poisedness)? Why not just ensure all points close to $\boldsymbol{x}_k$?

Geometry matters! For some problems, taking only trust-region steps puts all points in a subspace.

Incremental Geometry Algorithm

Incremental Geometry Algorithm:

- Given current interpolation set, construct interpolation model
- Minimize model to get $\boldsymbol{s}_k$, evaluate $f(\boldsymbol{x}_k + \boldsymbol{s}_k)$ and set $\boldsymbol{x}_{k+1}$ as before
- Update interpolation set:

Incremental Geometry Algorithm

Incremental Geometry Algorithm:

- Given current interpolation set, construct interpolation model
- Minimize model to get $\boldsymbol{s}_k$, evaluate $f(\boldsymbol{x}_k + \boldsymbol{s}_k)$ and set $\boldsymbol{x}_{k+1}$ as before
- Update interpolation set:
  - Successful step: $\Delta_{k+1} = 2\Delta_k$, update set in any manner

Incremental Geometry Algorithm

Incremental Geometry Algorithm:

- Given current interpolation set, construct interpolation model
- Minimize model to get $\boldsymbol{s}_k$, evaluate $f(\boldsymbol{x}_k + \boldsymbol{s}_k)$ and set $\boldsymbol{x}_{k+1}$ as before
- Update interpolation set:
  - Successful step: $\Delta_{k+1} = 2\Delta_k$, update set in any manner
  - Else, replace far point: if $\|\boldsymbol{y}_i - \boldsymbol{x}_k\| > \beta\Delta_k$, replace $\boldsymbol{y}_i$ with a point in $B(\boldsymbol{x}_k, \Delta_k)$, keep $\Delta_{k+1} = \Delta_k$

Incremental Geometry Algorithm

Incremental Geometry Algorithm:

- Given current interpolation set, construct interpolation model
- Minimize model to get $\boldsymbol{s}_k$, evaluate $f(\boldsymbol{x}_k + \boldsymbol{s}_k)$ and set $\boldsymbol{x}_{k+1}$ as before
- Update interpolation set:
  - Successful step: $\Delta_{k+1} = 2\Delta_k$, update set in any manner
  - Else, replace far point: if $\|\boldsymbol{y}_i - \boldsymbol{x}_k\| > \beta\Delta_k$, replace $\boldsymbol{y}_i$ with a point in $B(\boldsymbol{x}_k, \Delta_k)$, keep $\Delta_{k+1} = \Delta_k$
  - Else, replace bad point: if any $|\ell_i(\boldsymbol{y})| > \Lambda^*$, replace $\boldsymbol{y}_i$ with $\boldsymbol{y}$, keep $\Delta_{k+1} = \Delta_k$

Incremental Geometry Algorithm

Incremental Geometry Algorithm:

- Given current interpolation set, construct interpolation model
- Minimize model to get $\boldsymbol{s}_k$, evaluate $f(\boldsymbol{x}_k + \boldsymbol{s}_k)$ and set $\boldsymbol{x}_{k+1}$ as before
- Update interpolation set:
  - Successful step: $\Delta_{k+1} = 2\Delta_k$, update set in any manner
  - Else, replace far point: if $\|\boldsymbol{y}_i - \boldsymbol{x}_k\| > \beta\Delta_k$, replace $\boldsymbol{y}_i$ with a point in $B(\boldsymbol{x}_k, \Delta_k)$, keep $\Delta_{k+1} = \Delta_k$
  - Else, replace bad point: if any $|\ell_i(\boldsymbol{y})| > \Lambda^*$, replace $\boldsymbol{y}_i$ with $\boldsymbol{y}$, keep $\Delta_{k+1} = \Delta_k$
  - Else, unsuccessful step: shrink $\Delta_{k+1} = 0.5\Delta_k$, update set in any manner

Incremental Geometry Algorithm

Incremental Geometry Algorithm:

- Given current interpolation set, construct interpolation model
- Minimize model to get $\boldsymbol{s}_k$, evaluate $f(\boldsymbol{x}_k + \boldsymbol{s}_k)$ and set $\boldsymbol{x}_{k+1}$ as before
- Update interpolation set:
  - Successful step: $\Delta_{k+1} = 2\Delta_k$, update set in any manner
  - Else, replace far point: if $\|\boldsymbol{y}_i - \boldsymbol{x}_k\| > \beta\Delta_k$, replace $\boldsymbol{y}_i$ with a point in $B(\boldsymbol{x}_k, \Delta_k)$, keep $\Delta_{k+1} = \Delta_k$
  - Else, replace bad point: if any $|\ell_i(\boldsymbol{y})| > \Lambda^*$, replace $\boldsymbol{y}_i$ with $\boldsymbol{y}$, keep $\Delta_{k+1} = \Delta_k$
  - Else, unsuccessful step: shrink $\Delta_{k+1} = 0.5\Delta_k$, update set in any manner

Key idea: only shrink Δ_k when we had a bad step *and* the model was definitely good (otherwise, try to fix model first).

Incremental Geometry Algorithm

Incremental Geometry Algorithm:

- Given current interpolation set, construct interpolation model
- Minimize model to get $\boldsymbol{s}_k$, evaluate $f(\boldsymbol{x}_k + \boldsymbol{s}_k)$ and set $\boldsymbol{x}_{k+1}$ as before
- Update interpolation set:
  - Successful step: $\Delta_{k+1} = 2\Delta_k$, update set in any manner
  - Else, replace far point: if $\|\boldsymbol{y}_i - \boldsymbol{x}_k\| > \beta\Delta_k$, replace $\boldsymbol{y}_i$ with a point in $B(\boldsymbol{x}_k, \Delta_k)$, keep $\Delta_{k+1} = \Delta_k$
  - Else, replace bad point: if any $|\ell_i(\boldsymbol{y})| > \Lambda^*$, replace $\boldsymbol{y}_i$ with $\boldsymbol{y}$, keep $\Delta_{k+1} = \Delta_k$
  - Else, unsuccessful step: shrink $\Delta_{k+1} = 0.5\Delta_k$, update set in any manner

Key idea: only shrink Δ_k when we had a bad step *and* the model was definitely good (otherwise, try to fix model first). At most 2 evaluations per iteration!

Incremental Geometry Algorithm

Theoretically, the key idea is: can only have finitely many consecutive ‘replace far point’ or ‘replace bad point’ iterations.

Incremental Geometry Algorithm

Theoretically, the key idea is: can only have finitely many consecutive 'replace far point' or 'replace bad point' iterations.

Assumption

We cannot have more than $G_{\max} \in \mathbb{N}$ consecutive 'replace far point' or 'replace bad point' iterations, *regardless of the starting interpolation set*.

Incremental Geometry Algorithm

Theoretically, the key idea is: can only have finitely many consecutive 'replace far point' or 'replace bad point' iterations.

Assumption

We cannot have more than $G_{\max} \in \mathbb{N}$ consecutive 'replace far point' or 'replace bad point' iterations, *regardless of the starting interpolation set*.

If we use full interpolation in a p-dimensional polynomial space (e.g. linear or fully quadratic interpolation), then $G_{\max} = \mathcal{O}(p \log n)$. [Scheinberg & Chaudhry, 2026]

Incremental Geometry Algorithm

Theoretically, the key idea is: can only have finitely many consecutive 'replace far point' or 'replace bad point' iterations.

Assumption

We cannot have more than $G_{\max} \in \mathbb{N}$ consecutive 'replace far point' or 'replace bad point' iterations, *regardless of the starting interpolation set*.

If we use full interpolation in a p-dimensional polynomial space (e.g. linear or fully quadratic interpolation), then $G_{\max} = \mathcal{O}(p \log n)$. [Scheinberg & Chaudhry, 2026]

Theorem

$\|\nabla f(\boldsymbol{x}_k)\| < \epsilon$ *first occurs after* $\mathcal{O}(G_{\max}\kappa_H(\kappa_{\text{m}} + \kappa_H)^2\epsilon^{-2})$ *iterations/evaluations.*

Incremental Geometry Algorithm

Theoretically, the key idea is: can only have finitely many consecutive 'replace far point' or 'replace bad point' iterations.

Assumption

We cannot have more than $G_{\max} \in \mathbb{N}$ consecutive 'replace far point' or 'replace bad point' iterations, *regardless of the starting interpolation set*.

If we use full interpolation in a p-dimensional polynomial space (e.g. linear or fully quadratic interpolation), then $G_{\max} = \mathcal{O}(p \log n)$. [Scheinberg & Chaudhry, 2026]

Theorem

$\|\nabla f(\boldsymbol{x}_k)\| < \epsilon$ *first occurs after* $\mathcal{O}(G_{\max}\kappa_H(\kappa_{\mathrm{m}} + \kappa_H)^2\epsilon^{-2})$ *iterations/evaluations.*

Worse than full resampling by factor $G_{\max}/p = \mathcal{O}(\log n)$; practical performance better.

Incremental Geometry Algorithm

Practical Issue

How do we update the interpolation set on successful/unsuccessful iterations?

Incremental Geometry Algorithm

Practical Issue

How do we update the interpolation set on successful/unsuccessful iterations?

In practice, always add $\boldsymbol{x}_k + \boldsymbol{s}_k$ (even on unsuccessful iterations) to the interpolation set.

Incremental Geometry Algorithm

Practical Issue

How do we update the interpolation set on successful/unsuccessful iterations?

In practice, always add $\boldsymbol{x}_k + \boldsymbol{s}_k$ (even on unsuccessful iterations) to the interpolation set.

Replace a point that is either far away, or where replacing would improve poisedness, e.g.

$$\underset{i=1,\ldots,p}{\arg\max} \left\{ \max\left(\frac{\|\boldsymbol{y}_i - \boldsymbol{x}_k\|^3}{\Delta_k^3}, 1 \right) \cdot |\ell_i(\boldsymbol{x}_k + \boldsymbol{s}_k)| \right\}$$

Incremental Geometry Algorithm

Practical Issue

How do we update the interpolation set on successful/unsuccessful iterations?

In practice, always add $\boldsymbol{x}_k + \boldsymbol{s}_k$ (even on unsuccessful iterations) to the interpolation set.

Replace a point that is either far away, or where replacing would improve poisedness, e.g.

$$\underset{i=1,\ldots,p}{\arg\max} \left\{ \max\left(\frac{\|\boldsymbol{y}_i - \boldsymbol{x}_k\|^3}{\Delta_k^3}, 1 \right) \cdot |\ell_i(\boldsymbol{x}_k + \boldsymbol{s}_k)| \right\}$$

Why add $\boldsymbol{x}_k + \boldsymbol{s}_k$ on unsuccessful steps?

Incremental Geometry Algorithm

Practical Issue

How do we update the interpolation set on successful/unsuccessful iterations?

In practice, always add $\boldsymbol{x}_k + \boldsymbol{s}_k$ (even on unsuccessful iterations) to the interpolation set.

Replace a point that is either far away, or where replacing would improve poisedness, e.g.

$$\underset{i=1,\ldots,p}{\arg\max} \left\{ \max \left(\frac{\|\boldsymbol{y}_i - \boldsymbol{x}_k\|^3}{\Delta_k^3}, 1 \right) \cdot |\ell_i(\boldsymbol{x}_k + \boldsymbol{s}_k)| \right\}$$

Why add $\boldsymbol{x}_k + \boldsymbol{s}_k$ on unsuccessful steps? It tells the next model where *not* to look.

Outline

1. Incremental Geometry Improvement
2. **Constrained Optimization**
3. Noisy Problems
4. Structured Problems
5. Software & Outlook

Simple Constraints

What happens if our problem has simple constraints, such as bounds

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x}) \qquad \text{s.t.} \quad \boldsymbol{a} \leq \boldsymbol{x} \leq \boldsymbol{b}$$

Simple Constraints

What happens if our problem has simple constraints, such as bounds

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \qquad \text{s.t.} \quad \boldsymbol{a} \leq \boldsymbol{x} \leq \boldsymbol{b}$$

A more general framework is to look at convex constraints

$$\min_{\boldsymbol{x}\in\Omega} f(\boldsymbol{x})$$

for some convex set $\Omega \subset \mathbb{R}^n$.

Simple Constraints

What happens if our problem has simple constraints, such as bounds

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \qquad \text{s.t.} \quad \boldsymbol{a} \leq \boldsymbol{x} \leq \boldsymbol{b}$$

A more general framework is to look at convex constraints

$$\min_{\boldsymbol{x}\in\Omega} f(\boldsymbol{x})$$

for some convex set $\Omega \subset \mathbb{R}^n$.

Trust-region theory follows without much modification: [Conn et al., 2000]

Simple Constraints

What happens if our problem has simple constraints, such as bounds

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \qquad \text{s.t.} \quad \boldsymbol{a} \leq \boldsymbol{x} \leq \boldsymbol{b}$$

A more general framework is to look at convex constraints

$$\min_{\boldsymbol{x}\in\Omega} f(\boldsymbol{x})$$

for some convex set $\Omega \subset \mathbb{R}^n$.

Trust-region theory follows without much modification: [Conn et al., 2000]

- Measure first-order stationarity via ($\pi(\boldsymbol{x}) = \|\nabla f(\boldsymbol{x})\|$ if $\Omega = \mathbb{R}^n$)

$$\pi(\boldsymbol{x}) := -\min_{\boldsymbol{d}\in\mathbb{R}^n} \nabla f(\boldsymbol{x})^T \boldsymbol{d} \qquad \text{s.t.} \quad \boldsymbol{x} + \boldsymbol{d} \in \Omega \text{ and } \|\boldsymbol{d}\| \leq 1$$

Simple Constraints

What happens if our problem has simple constraints, such as bounds

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x}) \qquad \text{s.t.} \quad \boldsymbol{a} \leq \boldsymbol{x} \leq \boldsymbol{b}$$

A more general framework is to look at convex constraints

$$\min_{\boldsymbol{x} \in \Omega} f(\boldsymbol{x})$$

for some convex set $\Omega \subset \mathbb{R}^n$.

Trust-region theory follows without much modification: [Conn et al., 2000]

- Measure first-order stationarity via ($\pi(\boldsymbol{x}) = \|\nabla f(\boldsymbol{x})\|$ if $\Omega = \mathbb{R}^n$)

$$\pi(\boldsymbol{x}) := - \min_{\boldsymbol{d} \in \mathbb{R}^n} \nabla f(\boldsymbol{x})^T \boldsymbol{d} \qquad \text{s.t.} \quad \boldsymbol{x} + \boldsymbol{d} \in \Omega \text{ and } \|\boldsymbol{d}\| \leq 1$$

- More complex procedure for achieving Cauchy decrease for $\boldsymbol{s}_k$

Model Accuracy

Especially with bounds, it is important for algorithms to be strictly feasible: every objective evaluation is at a feasible point.

Model Accuracy

Especially with bounds, it is important for algorithms to be strictly feasible: every objective evaluation is at a feasible point.

Problem

If we build interpolation models using only feasible evaluations, we cannot guarantee they are fully linear (at least with reasonable constants).

Model Accuracy

Especially with bounds, it is important for algorithms to be strictly feasible: every objective evaluation is at a feasible point.

Problem

If we build interpolation models using only feasible evaluations, we cannot guarantee they are fully linear (at least with reasonable constants).

The issue is that fully linear is too strict: for all $\boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k)$

$$|f(\boldsymbol{y}) - m_k(\boldsymbol{y})| \leq \kappa_{\text{mf}} \Delta_k^2 \qquad \text{and} \qquad \|\nabla f(\boldsymbol{x}_k) - \nabla m_k(\boldsymbol{x}_k)\| \leq \kappa_{\text{mg}} \Delta_k$$

Model Accuracy

Especially with bounds, it is important for algorithms to be strictly feasible: every objective evaluation is at a feasible point.

Problem

If we build interpolation models using only feasible evaluations, we cannot guarantee they are fully linear (at least with reasonable constants).

The issue is that fully linear is too strict: for all $\boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k)$

$$|f(\boldsymbol{y}) - m_k(\boldsymbol{y})| \leq \kappa_{\text{mf}} \Delta_k^2 \qquad \text{and} \qquad \|\nabla f(\boldsymbol{x}_k) - \nabla m_k(\boldsymbol{x}_k)\| \leq \kappa_{\text{mg}} \Delta_k$$

We don't care about $\boldsymbol{y} \notin \Omega$!

Model Accuracy

Especially with bounds, it is important for algorithms to be strictly feasible: every objective evaluation is at a feasible point.

Problem

If we build interpolation models using only feasible evaluations, we cannot guarantee they are fully linear (at least with reasonable constants).

The issue is that fully linear is too strict: for all $\boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k)$

$$|f(\boldsymbol{y}) - m_k(\boldsymbol{y})| \leq \kappa_{\text{mf}} \Delta_k^2 \qquad \text{and} \qquad \|\nabla f(\boldsymbol{x}_k) - \nabla m_k(\boldsymbol{x}_k)\| \leq \kappa_{\text{mg}} \Delta_k$$

We don't care about $\boldsymbol{y} \notin \Omega$! Use a weaker definition: [Hough & LR, 2022]

$$|f(\boldsymbol{y}) - m_k(\boldsymbol{y})| \leq \kappa_{\text{mf}} \Delta_k^2 \qquad \text{for all } \boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k) \cap \Omega$$

$$|(\nabla f(\boldsymbol{x}_k) - \nabla m_k(\boldsymbol{x}_k))^T \boldsymbol{d}| \leq \kappa_{\text{mg}} \Delta_k \qquad \text{for all } \|\boldsymbol{d}\| \leq 1,\ \boldsymbol{x}_k + \boldsymbol{d} \in \Omega$$

Model Accuracy

Can this weaker notion (Ω-fully linear) be satisfied using feasible points?

Model Accuracy

Can this weaker notion (Ω-fully linear) be satisfied using feasible points?

Definition

An interpolation set is Λ-poised for interpolation in $B(\boldsymbol{x}_k, \Delta_k) \cap \Omega$ if

$$\max_{\boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k) \cap \Omega} \|\boldsymbol{\lambda}(\boldsymbol{y})\|_\infty \leq \Lambda$$

Model Accuracy

Can this weaker notion (Ω-fully linear) be satisfied using feasible points?

Definition

An interpolation set is Λ-poised for interpolation in $B(\boldsymbol{x}_k, \Delta_k) \cap \Omega$ if

$$\max_{\boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k) \cap \Omega} \|\boldsymbol{\lambda}(\boldsymbol{y})\|_\infty \leq \Lambda$$

Theorem (Hough & LR, 2022; LR, 2025)

For linear and MFN quadratic interpolation, if $\|\boldsymbol{y}_i - \boldsymbol{x}_k\| \leq \beta \Delta_k^2$ and the set is Λ-poised in $B(\boldsymbol{x}_k, \Delta_k) \cap \Omega$, then the model is Ω-fully linear.

Model Accuracy

Can this weaker notion (Ω-fully linear) be satisfied using feasible points?

Definition

An interpolation set is Λ-poised for interpolation in $B(\boldsymbol{x}_k, \Delta_k) \cap \Omega$ if

$$\max_{\boldsymbol{y} \in B(\boldsymbol{x}_k, \Delta_k) \cap \Omega} \|\boldsymbol{\lambda}(\boldsymbol{y})\|_\infty \leq \Lambda$$

Theorem (Hough & LR, 2022; LR, 2025)

For linear and MFN quadratic interpolation, if $\|\boldsymbol{y}_i - \boldsymbol{x}_k\| \leq \beta \Delta_k^2$ and the set is Λ-poised in $B(\boldsymbol{x}_k, \Delta_k) \cap \Omega$, then the model is Ω-fully linear.

Modified trust-region algorithm retains $\mathcal{O}(\epsilon^{-2})$ complexity to get $\pi(\boldsymbol{x}_k) < \epsilon$.

Nonlinear Constraints

What about problems with general nonlinear constraints,

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \qquad \text{s.t.} \quad \boldsymbol{c}_E(\boldsymbol{x}) = \boldsymbol{0}, \;\; \boldsymbol{c}_I(\boldsymbol{x}) \leq \boldsymbol{0}$$

Nonlinear Constraints

What about problems with general nonlinear constraints,

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \qquad \text{s.t.} \quad \boldsymbol{c}_E(\boldsymbol{x}) = \boldsymbol{0}, \;\; \boldsymbol{c}_I(\boldsymbol{x}) \leq \boldsymbol{0}$$

Several algorithms have been proposed, such as:

- Explicit constraint handling if $\boldsymbol{c}_E, \boldsymbol{c}_I$ not derivative-free [Conn et al., 1998]
- Restricted feasible sets [Augustin & Marzouk, 2014; Regis & Wild, 2017]
- Augmented Lagrangian [Conejo et al., 2015]
- Sequential Quadratic Programming (SQP) [Tröltzsch, 2016]
- Two-phase (feasibility then optimization) [Bajaj et al., 2018]

Nonlinear Constraints

What about problems with general nonlinear constraints,

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \qquad \text{s.t.} \quad \boldsymbol{c}_E(\boldsymbol{x}) = \boldsymbol{0},\ \ \boldsymbol{c}_I(\boldsymbol{x}) \leq \boldsymbol{0}$$

Several algorithms have been proposed, such as:

- Explicit constraint handling if $\boldsymbol{c}_E, \boldsymbol{c}_I$ not derivative-free [Conn et al., 1998]
- Restricted feasible sets [Augustin & Marzouk, 2014; Regis & Wild, 2017]
- Augmented Lagrangian [Conejo et al., 2015]
- Sequential Quadratic Programming (SQP) [Tröltzsch, 2016]
- Two-phase (feasibility then optimization) [Bajaj et al., 2018]

But nothing is really 'established' at this stage for MBDFO (unlike, e.g. IPOPT).

COBYQA

The best attempt to build a standard constrained MBDFO solver is COBYQA, which has been included in SciPy since 2024. [Ragonneau, 2022]

COBYQA

The best attempt to build a standard constrained MBDFO solver is COBYQA, which has been included in SciPy since 2024. [Ragonneau, 2022]

COBYQA is an SQP algorithm:

- Build interpolation models for objective/Lagrangian (quadratic) & constraints (linear)
- Solve a linearly constrained trust-region subproblem to get a step $\boldsymbol{s}_k$
- Determine a suitable merit function, e.g. $f(\boldsymbol{x}) + \gamma\Phi(\boldsymbol{x})$, where $\Phi(\boldsymbol{x}) \geq 0$ is ℓ_2 constraint violation and $\gamma > 0$ sufficiently large
- Evaluate objective & constraints at $\boldsymbol{x}_k + \boldsymbol{s}_k$, accept/reject step and update interpolation set in standard ways

Outline

1. Incremental Geometry Improvement
2. Constrained Optimization
3. **Noisy Problems**
4. Structured Problems
5. Software & Outlook

Noisy Problems

One important use case of DFO is for problems with noise (e.g. Monte Carlo simulations, chaotic dynamics, lab experiments). What happens to MBDFO when we have noise?

Noisy Problems

One important use case of DFO is for problems with noise (e.g. Monte Carlo simulations, chaotic dynamics, lab experiments). What happens to MBDFO when we have noise?

Consider two types of noise:

- Deterministic/bounded: $|\tilde{f}(\boldsymbol{x}) - f(\boldsymbol{x})| \leq \epsilon_f$ for all $\boldsymbol{x}$
- Stochastic: $\mathbb{E}[\tilde{f}(\boldsymbol{x})] = f(\boldsymbol{x})$ with some distribution

Noisy Problems

One important use case of DFO is for problems with noise (e.g. Monte Carlo simulations, chaotic dynamics, lab experiments). What happens to MBDFO when we have noise?

Consider two types of noise:

- Deterministic/bounded: $|\tilde{f}(\boldsymbol{x}) - f(\boldsymbol{x})| \leq \epsilon_f$ for all $\boldsymbol{x}$
- Stochastic: $\mathbb{E}[\tilde{f}(\boldsymbol{x})] = f(\boldsymbol{x})$ with some distribution

Key difference: stochastic noise can be made arbitrarily small by repeated sampling (e.g. averaging), deterministic noise cannot be reduced.

Noisy Problems

One important use case of DFO is for problems with noise (e.g. Monte Carlo simulations, chaotic dynamics, lab experiments). What happens to MBDFO when we have noise?

Consider two types of noise:

- Deterministic/bounded: $|\tilde{f}(\boldsymbol{x}) - f(\boldsymbol{x})| \leq \epsilon_f$ for all $\boldsymbol{x}$
- Stochastic: $\mathbb{E}[\tilde{f}(\boldsymbol{x})] = f(\boldsymbol{x})$ with some distribution

Key difference: stochastic noise can be made arbitrarily small by repeated sampling (e.g. averaging), deterministic noise cannot be reduced.

Why is noise difficult?

Noise in objective evaluations means our interpolation model is less accurate, and we can't be sure if $\boldsymbol{x}_k + \boldsymbol{s}_k$ is really an improvement or not.

Deterministic Noise

Example: suppose have deterministic noise of size ϵ_f and do finite differencing:

$$[\boldsymbol{g}_k]_i = \frac{\tilde{f}(\boldsymbol{x}_k + h\boldsymbol{e}_i) - \tilde{f}(\boldsymbol{x}_k)}{h}, \quad i = 1, \ldots, n$$

Deterministic Noise

Example: suppose have deterministic noise of size ϵ_f and do finite differencing:

$$[\boldsymbol{g}_k]_i = \frac{\tilde{f}(\boldsymbol{x}_k + h\boldsymbol{e}_i) - \tilde{f}(\boldsymbol{x}_k)}{h}, \quad i = 1, \ldots, n$$

By comparing to 'noiseless' estimated gradient, we get

$$\|\boldsymbol{g}_k - \nabla f(\boldsymbol{x}_k)\|_\infty \leq \frac{Lh}{2} + \frac{2\epsilon_f}{h}$$

Deterministic Noise

Example: suppose have deterministic noise of size ϵ_f and do finite differencing:

$$[\boldsymbol{g}_k]_i = \frac{\tilde{f}(\boldsymbol{x}_k + h\boldsymbol{e}_i) - \tilde{f}(\boldsymbol{x}_k)}{h}, \quad i = 1, \ldots, n$$

By comparing to 'noiseless' estimated gradient, we get

$$\|\boldsymbol{g}_k - \nabla f(\boldsymbol{x}_k)\|_\infty \leq \frac{Lh}{2} + \frac{2\epsilon_f}{h}$$

Optimal error bound is $\mathcal{O}(\sqrt{L\epsilon_f})$ (nonzero!) from $h = \mathcal{O}(\sqrt{\epsilon_f / L})$. [Shi et al., 2022]

Deterministic Noise

Example: suppose have deterministic noise of size ϵ_f and do finite differencing:

$$[\boldsymbol{g}_k]_i = \frac{\tilde{f}(\boldsymbol{x}_k + h\boldsymbol{e}_i) - \tilde{f}(\boldsymbol{x}_k)}{h}, \quad i = 1, \ldots, n$$

By comparing to 'noiseless' estimated gradient, we get

$$\|\boldsymbol{g}_k - \nabla f(\boldsymbol{x}_k)\|_\infty \leq \frac{Lh}{2} + \frac{2\epsilon_f}{h}$$

Optimal error bound is $\mathcal{O}(\sqrt{L\epsilon_f})$ (nonzero!) from $h = \mathcal{O}(\sqrt{\epsilon_f / L})$. [Shi et al., 2022]

Theorem

Linear/MFN interpolation to noisy evaluations gives "noisy" fully linear bounds

$$|f(\boldsymbol{y}) - m_k(\boldsymbol{y})| \leq \kappa_{\text{mf}}\Delta_k^2 + \mathcal{O}(\epsilon_f) \quad \textit{and} \quad \|\nabla f(\boldsymbol{x}_k) - \nabla m_k(\boldsymbol{x}_k)\| \leq \kappa_{\text{mg}}\Delta_k + \mathcal{O}\left(\frac{\epsilon_f}{\Delta_k}\right)$$

Deterministic Noise

Perhaps surprisingly, no algorithm modification is needed to handle deterministic noise (although ratio test can be safeguarded if ϵ_f is known).

Deterministic Noise

Perhaps surprisingly, no algorithm modification is needed to handle deterministic noise (although ratio test can be safeguarded if ϵ_f is known).

Main complexity bound is identical to the noise-free case:

Theorem (LR, 2025)

$\|\nabla f(\boldsymbol{x}_k)\| < \epsilon$ *first occurs after* $\mathcal{O}(\kappa_H(\kappa_{\text{m}} + \kappa_H)^2\epsilon^{-2})$ *iterations, provided* $\epsilon > \mathcal{O}(\sqrt{\epsilon_f})$.

Deterministic Noise

Perhaps surprisingly, no algorithm modification is needed to handle deterministic noise (although ratio test can be safeguarded if ϵ_f is known).

Main complexity bound is identical to the noise-free case:

Theorem (LR, 2025)

$\|\nabla f(\boldsymbol{x}_k)\| < \epsilon$ *first occurs after* $\mathcal{O}(\kappa_H(\kappa_{\text{m}} + \kappa_H)^2\epsilon^{-2})$ *iterations, provided* $\epsilon > \mathcal{O}(\sqrt{\epsilon_f})$.

This analysis also gives a practical heuristic: for noisy problems, decrease Δ_k very slowly on unsuccessful iterations (e.g. $\Delta_{k+1} = 0.98\Delta_k$).

Stochastic Noise

For stochastic noise, we can always take more samples at any point to get a better objective estimate (with high probability).

Stochastic Noise

For stochastic noise, we can always take more samples at any point to get a better objective estimate (with high probability).

How accurate do our estimates need to be?

Stochastic Noise

For stochastic noise, we can always take more samples at any point to get a better objective estimate (with high probability).

How accurate do our estimates need to be? Always need $|\tilde{f}(\boldsymbol{x}) - f(\boldsymbol{x})| \leq \mathcal{O}(\Delta_k^2)$ (for interpolation to be fully linear and ρ_k calculation).

Stochastic Noise

For stochastic noise, we can always take more samples at any point to get a better objective estimate (with high probability).

How accurate do our estimates need to be? Always need $|\tilde{f}(\boldsymbol{x}) - f(\boldsymbol{x})| \leq \mathcal{O}(\Delta_k^2)$ (for interpolation to be fully linear and ρ_k calculation).

Standard concentration bounds give results like

Chebyshev's Inequality, e.g. [Boucheron et al., 2013]

If $\mathbb{E}[\tilde{f}(\boldsymbol{x})] = f(\boldsymbol{x})$ and $\text{Var}(\tilde{f}(\boldsymbol{x})) = \sigma^2 < \infty$, then taking an average of N noisy samples guarantees error $\mathcal{O}(\Delta_k^2)$ with probability $1 - \delta$, provided $N \geq \mathcal{O}(\sigma^2 \delta^{-1} \Delta_k^{-4})$.

Stochastic Noise

For stochastic noise, we can always take more samples at any point to get a better objective estimate (with high probability).

How accurate do our estimates need to be? Always need $|\tilde{f}(\boldsymbol{x}) - f(\boldsymbol{x})| \leq \mathcal{O}(\Delta_k^2)$ (for interpolation to be fully linear and ρ_k calculation).

Standard concentration bounds give results like

Chebyshev's Inequality, e.g. [Boucheron et al., 2013]

If $\mathbb{E}[\tilde{f}(\boldsymbol{x})] = f(\boldsymbol{x})$ and $\text{Var}(\tilde{f}(\boldsymbol{x})) = \sigma^2 < \infty$, then taking an average of N noisy samples guarantees error $\mathcal{O}(\Delta_k^2)$ with probability $1 - \delta$, provided $N \geq \mathcal{O}(\sigma^2 \delta^{-1} \Delta_k^{-4})$.

In theory, always take $\mathcal{O}(\Delta_k^{-4})$ samples.

Stochastic Noise

For stochastic noise, we can always take more samples at any point to get a better objective estimate (with high probability).

How accurate do our estimates need to be? Always need $|\tilde{f}(\boldsymbol{x}) - f(\boldsymbol{x})| \leq \mathcal{O}(\Delta_k^2)$ (for interpolation to be fully linear and ρ_k calculation).

Standard concentration bounds give results like

Chebyshev's Inequality, e.g. [Boucheron et al., 2013]

If $\mathbb{E}[\tilde{f}(\boldsymbol{x})] = f(\boldsymbol{x})$ and $\text{Var}(\tilde{f}(\boldsymbol{x})) = \sigma^2 < \infty$, then taking an average of N noisy samples guarantees error $\mathcal{O}(\Delta_k^2)$ with probability $1 - \delta$, provided $N \geq \mathcal{O}(\sigma^2 \delta^{-1} \Delta_k^{-4})$.

In theory, always take $\mathcal{O}(\Delta_k^{-4})$ samples.

Since $\Delta_k \to 0$, in practice take fewer (e.g. $\mathcal{O}(\Delta_k^{-1})$).

Stochastic Noise

In theory, $N = \mathcal{O}(\Delta_k^{-4})$ sampling gives fully linear models & accurate ρ_k with high probability.

Stochastic Noise

In theory, $N = \mathcal{O}(\Delta_k^{-4})$ sampling gives fully linear models & accurate ρ_k with high probability.

Question

What happens to trust-region methods if problem information is only accurate with high probability?

Stochastic Noise

In theory, $N = \mathcal{O}(\Delta_k^{-4})$ sampling gives fully linear models & accurate ρ_k with high probability.

Question

What happens to trust-region methods if problem information is only accurate with high probability?

Assume:

- Model is fully linear with probability $\geq \alpha_m$
- Estimates used for ρ_k are sufficiently accurate with probability $\geq \alpha_f$

Stochastic Noise

In theory, $N = \mathcal{O}(\Delta_k^{-4})$ sampling gives fully linear models & accurate ρ_k with high probability.

Question

What happens to trust-region methods if problem information is only accurate with high probability?

Assume:

- Model is fully linear with probability $\geq \alpha_m$
- Estimates used for ρ_k are sufficiently accurate with probability $\geq \alpha_f$

Theorem (Blanchet et al., 2019)

If α_m and α_f are sufficiently large, the expected # iterations to achieve $\|\nabla f(\boldsymbol{x}_k)\| < \epsilon$ is $\mathcal{O}(\epsilon^{-2}/(\alpha_m\alpha_f - \frac{1}{2}))$.

Outline

1. Incremental Geometry Improvement
2. Constrained Optimization
3. Noisy Problems
4. **Structured Problems**
5. Software & Outlook

Black-Box Structure

So far, we have worked with simple derivative-free objectives, $f : \mathbb{R}^n \to \mathbb{R}$. In practice, sometimes a black-box comes with more structure than this.

Black-Box Structure

So far, we have worked with simple derivative-free objectives, $f : \mathbb{R}^n \to \mathbb{R}$. In practice, sometimes a black-box comes with more structure than this.

For example, the climate modelling application (lecture 1) is a parameter fitting problem: parameters $\boldsymbol{x}$ are used to define a system of PDEs, which is solved numerically; the solution should match known observations.

Black-Box Structure

So far, we have worked with simple derivative-free objectives, $f : \mathbb{R}^n \to \mathbb{R}$. In practice, sometimes a black-box comes with more structure than this.

For example, the climate modelling application (lecture 1) is a parameter fitting problem: parameters $\boldsymbol{x}$ are used to define a system of PDEs, which is solved numerically; the solution should match known observations. This is naturally formulated as a nonlinear least-squares problem

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} \sum_{i=1}^{N_{\text{obs}}} \left(\text{model}(\boldsymbol{x}, \boldsymbol{v}_i) - y_i\right)^2$$

where $\text{model}(\cdot, \cdot)$ takes PDE parameters $\boldsymbol{x}$ and information $\boldsymbol{v}_i$ about the specific desired observation, and should match known data y_i.

Black-Box Structure

In DFO, problems with some structure are sometimes called grey-box or glass-box:

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x}) = h(\boldsymbol{r}(\boldsymbol{x})),$$

where $\boldsymbol{r}(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^m$ is a complicated, black-box function (no derivatives), but $h : \mathbb{R}^m \to \mathbb{R}$ is a known function.

Black-Box Structure

In DFO, problems with some structure are sometimes called grey-box or glass-box:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) = h(\boldsymbol{r}(\boldsymbol{x})),$$

where $\boldsymbol{r}(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^m$ is a complicated, black-box function (no derivatives), but $h : \mathbb{R}^m \to \mathbb{R}$ is a known function.

The nonlinear least-squares (NLLS) case is $h(\boldsymbol{r}) = \frac{1}{2}\|\boldsymbol{r}\|^2$, so we are solving

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{r}(\boldsymbol{x})\|^2 = \frac{1}{2}\sum_{i=1}^{m} r_i(\boldsymbol{x})^2$$

for a (derivative-free) residual function $\boldsymbol{r} : \mathbb{R}^n \to \mathbb{R}^m$.

Black-Box Structure

In DFO, problems with some structure are sometimes called grey-box or glass-box:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) = h(\boldsymbol{r}(\boldsymbol{x})),$$

where $\boldsymbol{r}(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^m$ is a complicated, black-box function (no derivatives), but $h : \mathbb{R}^m \to \mathbb{R}$ is a known function.

The nonlinear least-squares (NLLS) case is $h(\boldsymbol{r}) = \frac{1}{2}\|\boldsymbol{r}\|^2$, so we are solving

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{r}(\boldsymbol{x})\|^2 = \frac{1}{2}\sum_{i=1}^{m} r_i(\boldsymbol{x})^2$$

for a (derivative-free) residual function $\boldsymbol{r} : \mathbb{R}^n \to \mathbb{R}^m$. Often, including the climate application, we naturally get all residuals $\boldsymbol{r}(\boldsymbol{x})$ for similar cost as getting just one $r_i(\boldsymbol{x})$.

Nonlinear Least-Squares

For NLLS problems, we can compute

$$\nabla f(\boldsymbol{x}) = J(\boldsymbol{x})^T \boldsymbol{r}(\boldsymbol{x}) \qquad \text{and} \qquad \nabla^2 f(\boldsymbol{x}) = J(\boldsymbol{x})^T J(\boldsymbol{x}) + \sum_{i=1}^{m} r_i(\boldsymbol{x}) \nabla^2 r_i(\boldsymbol{x})$$

where $J(\boldsymbol{x}) \in \mathbb{R}^{m \times n}$ has rows $\nabla r_i(\boldsymbol{x})^T$.

Nonlinear Least-Squares

For NLLS problems, we can compute

$$\nabla f(\boldsymbol{x}) = J(\boldsymbol{x})^T \boldsymbol{r}(\boldsymbol{x}) \qquad \text{and} \qquad \nabla^2 f(\boldsymbol{x}) = J(\boldsymbol{x})^T J(\boldsymbol{x}) + \sum_{i=1}^{m} r_i(\boldsymbol{x}) \nabla^2 r_i(\boldsymbol{x})$$

where $J(\boldsymbol{x}) \in \mathbb{R}^{m \times n}$ has rows $\nabla r_i(\boldsymbol{x})^T$. In MBDFO, we can get a practical algorithm by building a linear interpolation model for the full vector of residuals

$$\boldsymbol{r}(\boldsymbol{y}) \approx \boldsymbol{m}_k(\boldsymbol{y}) := \boldsymbol{r}(\boldsymbol{x}_k) + J_k(\boldsymbol{y} - \boldsymbol{x}_k) \qquad \text{where} \quad J_k \approx J(\boldsymbol{x}_k)$$

Nonlinear Least-Squares

For NLLS problems, we can compute

$$\nabla f(\boldsymbol{x}) = J(\boldsymbol{x})^T \boldsymbol{r}(\boldsymbol{x}) \qquad \text{and} \qquad \nabla^2 f(\boldsymbol{x}) = J(\boldsymbol{x})^T J(\boldsymbol{x}) + \sum_{i=1}^{m} r_i(\boldsymbol{x}) \nabla^2 r_i(\boldsymbol{x})$$

where $J(\boldsymbol{x}) \in \mathbb{R}^{m \times n}$ has rows $\nabla r_i(\boldsymbol{x})^T$. In MBDFO, we can get a practical algorithm by building a linear interpolation model for the full vector of residuals

$$\boldsymbol{r}(\boldsymbol{y}) \approx \boldsymbol{m}_k(\boldsymbol{y}) := \boldsymbol{r}(\boldsymbol{x}_k) + J_k(\boldsymbol{y} - \boldsymbol{x}_k) \qquad \text{where} \quad J_k \approx J(\boldsymbol{x}_k)$$

This gives a natural quadratic model for the objective (c.f. Gauss–Newton method)

$$f(\boldsymbol{y}) \approx m_k(\boldsymbol{y}) := \frac{1}{2} \|\boldsymbol{m}_k(\boldsymbol{y})\|^2$$

Nonlinear Least-Squares

For NLLS problems, we can compute

$$\nabla f(\boldsymbol{x}) = J(\boldsymbol{x})^T \boldsymbol{r}(\boldsymbol{x}) \qquad \text{and} \qquad \nabla^2 f(\boldsymbol{x}) = J(\boldsymbol{x})^T J(\boldsymbol{x}) + \sum_{i=1}^{m} r_i(\boldsymbol{x}) \nabla^2 r_i(\boldsymbol{x})$$

where $J(\boldsymbol{x}) \in \mathbb{R}^{m \times n}$ has rows $\nabla r_i(\boldsymbol{x})^T$. In MBDFO, we can get a practical algorithm by building a linear interpolation model for the full vector of residuals

$$\boldsymbol{r}(\boldsymbol{y}) \approx \boldsymbol{m}_k(\boldsymbol{y}) := \boldsymbol{r}(\boldsymbol{x}_k) + J_k(\boldsymbol{y} - \boldsymbol{x}_k) \qquad \text{where} \quad J_k \approx J(\boldsymbol{x}_k)$$

This gives a natural quadratic model for the objective (c.f. Gauss–Newton method)

$$f(\boldsymbol{y}) \approx m_k(\boldsymbol{y}) := \frac{1}{2} \|\boldsymbol{m}_k(\boldsymbol{y})\|^2$$

In practice, this model performs comparably to a full quadratic model for $\boldsymbol{r}$, and significantly outperforms MBDFO applied to f. [Cartis & LR, 2019]

Nonsmooth Compositions

This idea readily generalizes to other smooth h, but h can be nonsmooth (i.e. non-differentiable) in some applications, e.g. $h(\boldsymbol{r}) = \|\boldsymbol{r}\|_1$. [Larson et al., 2016]

Nonsmooth Compositions

This idea readily generalizes to other smooth h, but h can be nonsmooth (i.e. non-differentiable) in some applications, e.g. $h(\boldsymbol{r}) = \|\boldsymbol{r}\|_1$. [Larson et al., 2016]

For these functions, a more complex approach is needed:

Nonsmooth Compositions

This idea readily generalizes to other smooth h, but h can be nonsmooth (i.e. non-differentiable) in some applications, e.g. $h(\boldsymbol{r}) = \|\boldsymbol{r}\|_1$. [Larson et al., 2016]

For these functions, a more complex approach is needed:

- Build a linear model for $\boldsymbol{r}(\boldsymbol{y})$

Nonsmooth Compositions

This idea readily generalizes to other smooth h, but h can be nonsmooth (i.e. non-differentiable) in some applications, e.g. $h(\boldsymbol{r}) = \|\boldsymbol{r}\|_1$. [Larson et al., 2016]

For these functions, a more complex approach is needed:

- Build a linear model for $\boldsymbol{r}(\boldsymbol{y})$
- Identify approximate generators for the subdifferential $\partial f(\boldsymbol{x}_k) \approx \text{conv}(\mathbb{G}_k)$, based on the current structure of $h(\boldsymbol{r}(\boldsymbol{x}_k))$ (e.g. current sign pattern of $\boldsymbol{r}$)

Nonsmooth Compositions

This idea readily generalizes to other smooth h, but h can be nonsmooth (i.e. non-differentiable) in some applications, e.g. $h(\boldsymbol{r}) = \|\boldsymbol{r}\|_1$. [Larson et al., 2016]

For these functions, a more complex approach is needed:

- Build a linear model for $\boldsymbol{r}(\boldsymbol{y})$
- Identify approximate generators for the subdifferential $\partial f(\boldsymbol{x}_k) \approx \text{conv}(\mathbb{G}_k)$, based on the current structure of $h(\boldsymbol{r}(\boldsymbol{x}_k))$ (e.g. current sign pattern of $\boldsymbol{r}$)
- Build a smooth quadratic model as a weighted combination of linear models (weights depend on $\mathbb{G}_k$ and linear model for $\boldsymbol{r}$)

Nonsmooth Compositions

This idea readily generalizes to other smooth h, but h can be nonsmooth (i.e. non-differentiable) in some applications, e.g. $h(\boldsymbol{r}) = \|\boldsymbol{r}\|_1$. [Larson et al., 2016]

For these functions, a more complex approach is needed:

- Build a linear model for $\boldsymbol{r}(\boldsymbol{y})$
- Identify approximate generators for the subdifferential $\partial f(\boldsymbol{x}_k) \approx \text{conv}(\mathbb{G}_k)$, based on the current structure of $h(\boldsymbol{r}(\boldsymbol{x}_k))$ (e.g. current sign pattern of $\boldsymbol{r}$)
- Build a smooth quadratic model as a weighted combination of linear models (weights depend on $\mathbb{G}_k$ and linear model for $\boldsymbol{r}$)
- If $\boldsymbol{x}_k + \boldsymbol{s}_k$ identifies a new active manifold (e.g. new sign pattern), augment $\mathbb{G}_k$ and repeat

Nonsmooth Compositions

This idea readily generalizes to other smooth h, but h can be nonsmooth (i.e. non-differentiable) in some applications, e.g. $h(\boldsymbol{r}) = \|\boldsymbol{r}\|_1$. [Larson et al., 2016]

For these functions, a more complex approach is needed:

- Build a linear model for $\boldsymbol{r}(\boldsymbol{y})$
- Identify approximate generators for the subdifferential $\partial f(\boldsymbol{x}_k) \approx \text{conv}(\mathbb{G}_k)$, based on the current structure of $h(\boldsymbol{r}(\boldsymbol{x}_k))$ (e.g. current sign pattern of $\boldsymbol{r}$)
- Build a smooth quadratic model as a weighted combination of linear models (weights depend on $\mathbb{G}_k$ and linear model for $\boldsymbol{r}$)
- If $\boldsymbol{x}_k + \boldsymbol{s}_k$ identifies a new active manifold (e.g. new sign pattern), augment $\mathbb{G}_k$ and repeat

Limit points of this algorithm are (Clarke) stationary. [Larson et al., 2019]

Exploiting Structure

Question

Why does exploiting available structure help MBDFO methods?

Exploiting Structure

Question

Why does exploiting available structure help MBDFO methods?

At least three reasons:

- Model adapts to known problem structure

Exploiting Structure

Question

Why does exploiting available structure help MBDFO methods?

At least three reasons:

- Model adapts to known problem structure
- Get more information per evaluation ($\boldsymbol{r}(\boldsymbol{x}) \in \mathbb{R}^m$ vs. $f(\boldsymbol{x}) \in \mathbb{R}$)

Exploiting Structure

Question

Why does exploiting available structure help MBDFO methods?

At least three reasons:

- Model adapts to known problem structure
- Get more information per evaluation ($\boldsymbol{r}(\boldsymbol{x}) \in \mathbb{R}^m$ vs. $f(\boldsymbol{x}) \in \mathbb{R}$)
- Often linear interpolation is practically sufficient (faster & easier to implement)

Exploiting Structure

Question

Why does exploiting available structure help MBDFO methods?

At least three reasons:

- Model adapts to known problem structure
- Get more information per evaluation ($\boldsymbol{r}(\boldsymbol{x}) \in \mathbb{R}^m$ vs. $f(\boldsymbol{x}) \in \mathbb{R}$)
- Often linear interpolation is practically sufficient (faster & easier to implement)

Can get other benefits too, e.g. known f_{low} (for termination) and convex trust-region subproblems for NLLS.

Outline

1. Incremental Geometry Improvement
2. Constrained Optimization
3. Noisy Problems
4. Structured Problems
5. **Software & Outlook**

Powell's Software

The most widely used (and effective) MBDFO software is a collection of routines developed by Mike Powell, see Zaikun Zhang's PRIMA on Github:

- 1994: COBYLA — linear interpolation, nonlinear constraints
- 2002: UOBYQA — fully quadratic interpolation, unconstrained
- 2006: NEWUOA — MFN interpolation, unconstrained
- 2009: BOBYQA — MFN interpolation, bound constraints
- 2015: LINCOA — MFN interpolation, linear constraints

Powell's Software

The most widely used (and effective) MBDFO software is a collection of routines developed by Mike Powell, see Zaikun Zhang's PRIMA on Github:

- 1994: COBYLA — linear interpolation, nonlinear constraints
- 2002: UOBYQA — fully quadratic interpolation, unconstrained
- 2006: NEWUOA — MFN interpolation, unconstrained
- 2009: BOBYQA — MFN interpolation, bound constraints
- 2015: LINCOA — MFN interpolation, linear constraints

Uses incremental geometry updating based on Lagrange polynomials, extremely sophisticated trust-region management, and lots of consideration of linear algebra costs. [Ragonneau & Zhang, 2024]

Powell's Software

The most widely used (and effective) MBDFO software is a collection of routines developed by Mike Powell, see Zaikun Zhang's PRIMA on Github:

- 1994: COBYLA — linear interpolation, nonlinear constraints
- 2002: UOBYQA — fully quadratic interpolation, unconstrained
- 2006: NEWUOA — MFN interpolation, unconstrained
- 2009: BOBYQA — MFN interpolation, bound constraints
- 2015: LINCOA — MFN interpolation, linear constraints

Uses incremental geometry updating based on Lagrange polynomials, extremely sophisticated trust-region management, and lots of consideration of linear algebra costs. [Ragonneau & Zhang, 2024]

Limited convergence theory, but very successful in practice. [Han & Liu, 2004]

Other Software

Other useful open source packages include:

- COBYQA — nonlinear constraints in style of Powell's codes [Ragonneau, 2022]

Other Software

Other useful open source packages include:

- COBYQA — nonlinear constraints in style of Powell's codes [Ragonneau, 2022]
- Py-BOBYQA (mine) — Python version of BOBYQA with improved heuristics for noisy problems [Cartis et al., 2019]

Other Software

Other useful open source packages include:

- COBYQA — nonlinear constraints in style of Powell's codes [Ragonneau, 2022]
- Py-BOBYQA (mine) — Python version of BOBYQA with improved heuristics for noisy problems [Cartis et al., 2019]
- NOMAD — widely used Mesh Adaptive Direct Search code, accelerated using model-based ideas. Can handle: mixed integer, arbitrary constraints, nonsmooth objectives, multi-objective, ... [Le Digabel, 2011; Audet et al., 2022]

Other Software

Other useful open source packages include:

- COBYQA — nonlinear constraints in style of Powell's codes [Ragonneau, 2022]
- Py-BOBYQA (mine) — Python version of BOBYQA with improved heuristics for noisy problems [Cartis et al., 2019]
- NOMAD — widely used Mesh Adaptive Direct Search code, accelerated using model-based ideas. Can handle: mixed integer, arbitrary constraints, nonsmooth objectives, multi-objective, ... [Le Digabel, 2011; Audet et al., 2022]
- ASTRO — for stochastic problems [Shashaani et al., 2018]

Other Software

Other useful open source packages include:

- COBYQA — nonlinear constraints in style of Powell's codes [Ragonneau, 2022]
- Py-BOBYQA (mine) — Python version of BOBYQA with improved heuristics for noisy problems [Cartis et al., 2019]
- NOMAD — widely used Mesh Adaptive Direct Search code, accelerated using model-based ideas. Can handle: mixed integer, arbitrary constraints, nonsmooth objectives, multi-objective, ... [Le Digabel, 2011; Audet et al., 2022]
- ASTRO — for stochastic problems [Shashaani et al., 2018]

And other packages that tackle problems with more structure: DFO-LS (mine) and IBCDFO for nonlinear least-squares & nonsmooth compositions. [Cartis et al., 2019; Wild, 2017; Larson & Menickelly, 2024]

Outlook

These lectures have covered the basic ideas of MBDFO, and some recent extensions.

Outlook

These lectures have covered the basic ideas of MBDFO, and some recent extensions.

There are many active/open research directions in MBDFO, including:

- Improving complexity analysis for incremental geometry methods

Outlook

These lectures have covered the basic ideas of MBDFO, and some recent extensions.

There are many active/open research directions in MBDFO, including:

- Improving complexity analysis for incremental geometry methods
- Better handling of nonlinear constraints

Outlook

These lectures have covered the basic ideas of MBDFO, and some recent extensions.

There are many active/open research directions in MBDFO, including:

- Improving complexity analysis for incremental geometry methods
- Better handling of nonlinear constraints
- Extensions to more complex problem types

Outlook

These lectures have covered the basic ideas of MBDFO, and some recent extensions.

There are many active/open research directions in MBDFO, including:

- Improving complexity analysis for incremental geometry methods
- Better handling of nonlinear constraints
- Extensions to more complex problem types
- Incorporating partial derivative information

Outlook

These lectures have covered the basic ideas of MBDFO, and some recent extensions.

There are many active/open research directions in MBDFO, including:

- Improving complexity analysis for incremental geometry methods
- Better handling of nonlinear constraints
- Extensions to more complex problem types
- Incorporating partial derivative information
- Aligning theory & software

Outlook

These lectures have covered the basic ideas of MBDFO, and some recent extensions.

There are many active/open research directions in MBDFO, including:

- Improving complexity analysis for incremental geometry methods
- Better handling of nonlinear constraints
- Extensions to more complex problem types
- Incorporating partial derivative information
- Aligning theory & software
- AI-driven model selection

Outlook

These lectures have covered the basic ideas of MBDFO, and some recent extensions.

There are many active/open research directions in MBDFO, including:

- Improving complexity analysis for incremental geometry methods
- Better handling of nonlinear constraints
- Extensions to more complex problem types
- Incorporating partial derivative information
- Aligning theory & software
- AI-driven model selection
- Exploit parallel evaluations

Outlook

For the rest of this week:

- **Zaikun Zhang (Thurs)** will talk about a key current research direction: improving the scalability of MBDFO (i.e. solve problems with larger dimension n) using dimensionality reduction techniques

Outlook

For the rest of this week:

- **Zaikun Zhang (Thurs)** will talk about a key current research direction: improving the scalability of MBDFO (i.e. solve problems with larger dimension n) using dimensionality reduction techniques
- **Sara Shashaani (Fri)** will look at more sophisticated algorithms and theory for stochastic problems

Outlook

For the rest of this week:

- **Zaikun Zhang (Thurs)** will talk about a key current research direction: improving the scalability of MBDFO (i.e. solve problems with larger dimension n) using dimensionality reduction techniques
- **Sara Shashaani (Fri)** will look at more sophisticated algorithms and theory for stochastic problems

Want more details? arXiv:2510.04473

Outlook

For the rest of this week:

- **Zaikun Zhang (Thurs)** will talk about a key current research direction: improving the scalability of MBDFO (i.e. solve problems with larger dimension n) using dimensionality reduction techniques
- **Sara Shashaani (Fri)** will look at more sophisticated algorithms and theory for stochastic problems

Want more details? arXiv:2510.04473

Thank you!

References i

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.

F. Augustin and Y. M. Marzouk, *NOWPAC: A provably convergent derivative-free nonlinear optimizer with path-augmented constraints*.
arXiv preprint arXiv:1403.1931, 2014.

I. Bajaj, S. S. Iyer, and M. Faruque Hasan, *A trust region-based two phase algorithm for constrained black-box and grey-box optimization with infeasible initial point*, Computers & Chemical Engineering, 116 (2018), pp. 306–321.

J. Blanchet, C. Cartis, M. Menickelly, and K. Scheinberg, *Convergence rate analysis of a stochastic trust-region method via supermartingales*, INFORMS Journal on Optimization, 1 (2019), pp. 92–119.

S. Boucheron, G. Lugosi, and P. Massart, *Concentration Inequalities: A Nonasymptotic Theory of Independence*, Oxford University Press, Oxford, 2013.

References ii

C. Cartis, J. Fiala, B. Marteau, and L. Roberts, *Improving the flexibility and robustness of model-based derivative-free optimization solvers*, ACM Transactions on Mathematical Software, 45 (2019), pp. 32:1–32:41.

C. Cartis and L. Roberts, *A derivative-free Gauss–Newton method*, Mathematical Programming Computation, 11 (2019), pp. 631–674.

P. Conejo, E. Karas, and L. Pedroso, *A trust-region derivative-free algorithm for constrained optimization*, Optimization Methods and Software, 30 (2015), pp. 1126–1145.

A. Conn, K. Scheinberg, and Ph. Toint, *A derivative free optimization algorithm in practice*, in 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis,MO,U.S.A., 1998, American Institute of Aeronautics and Astronautics.

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, *Geometry of interpolation sets in derivative free optimization*, Mathematical Programming, 111 (2007), pp. 141–172.

References iii

A. R. Conn, K. Scheinberg, and L. N. Vicente, *Geometry of sample sets in derivative-free optimization: Polynomial regression and underdetermined interpolation*, IMA Journal of Numerical Analysis, 28 (2008), pp. 721–748.

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.

L. Han and G. Liu, *On the convergence of the UOBYQA method*, Journal of Applied Mathematics and Computing, 16 (2004), pp. 125–142.

M. Hough and L. Roberts, *Model-based derivative-free methods for convex-constrained optimization*, SIAM Journal on Optimization, 32 (2022), pp. 2552–2579.

J. Larson and M. Menickelly, *Structure-aware methods for expensive derivative-free nonsmooth composite optimization*, Mathematical Programming Computation, 16 (2024), pp. 1–36.

J. Larson, M. Menickelly, and S. M. Wild, *Manifold sampling for ℓ_1 nonconvex optimization*, SIAM Journal on Optimization, 26 (2016), pp. 2540–2563.

References iv

J. Larson, M. Menickelly, and B. Zhou, *Manifold sampling for optimizing nonsmooth nonconvex compositions*, SIAM Journal on Optimization, 31 (2021), pp. 2638–2664.

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

M. J. D. Powell, *A direct search optimization method that models the objective and constraint functions by linear interpolation*, in Advances in Optimization and Numerical Analysis, S. Gomez and J.-P. Hennart, eds., Springer Netherlands, Dordrecht, 1994, pp. 51–67.

———, *UOBYQA: Unconstrained optimization by quadratic approximation*, Mathematical Programming, 92 (2002), pp. 555–582.

———, *The NEWUOA software for unconstrained optimization without derivatives*, in Large-Scale Nonlinear Optimization, P. Pardalos, G. Di Pillo, and M. Roma, eds., vol. 83, Springer US, Boston, MA, 2006, pp. 255–297.

———, *The BOBYQA algorithm for bound constrained optimization without derivatives*, Tech. Rep. DAMTP 2009/NA06, University of Cambridge, 2009.

References v

———, *On fast trust region methods for quadratic models with linear constraints*, Mathematical Programming Computation, 7 (2015), pp. 237–267.

T. M. Ragonneau, *Model-Based Derivative-Free Optimization Methods and Software*, PhD thesis, Hong Kong Polytechnic University, 2022.

T. M. Ragonneau and Z. Zhang, *PDFO: A cross-platform package for Powell's derivative-free optimization solvers*, Mathematical Programming Computation, 16 (2024), pp. 535–559.

R. G. Regis and S. M. Wild, *CONORBIT: Constrained optimization by radial basis function interpolation in trust regions*, Optimization Methods and Software, 32 (2017), pp. 552–580.

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

———, *Model construction for convex-constrained derivative-free optimization*, SIAM Journal on Optimization, 35 (2025), pp. 622–650.

References vi

K. Scheinberg and A. Chaudhry, *On complexity of model-based derivative-free methods*, in International Congress of Mathematicians, Volume 7: Invited Lectures: Sections 15–20, USA, 2026, Society for Industrial and Applied Mathematics, pp. 208–228.

K. Scheinberg and Ph. L. Toint, *Self-correcting geometry in model-based algorithms for derivative-free unconstrained optimization*, SIAM Journal on Optimization, 20 (2010), pp. 3512–3532.

S. Shashaani, F. S. Hashemi, and R. Pasupathy, *ASTRO-DF: A class of adaptive sampling trust-region algorithms for derivative-free stochastic optimization*, SIAM Journal on Optimization, 28 (2018), pp. 3145–3176.

H.-J. M. Shi, Y. Xie, M. Q. Xuan, and J. Nocedal, *Adaptive finite-difference interval estimation for noisy derivative-free optimization*, SIAM Journal on Scientific Computing, 44 (2022), pp. A2302–A2321.

L. N. Trefethen, *Approximation Theory and Approximation Practice*, SIAM, Philadelphia, 2020.

A. Tröltzsch, *A sequential quadratic programming algorithm for equality-constrained optimization without derivatives*, Optimization Letters, 10 (2016), pp. 383–399.

References vii

S. M. Wild, *POUNDERS in TAO: Solving derivative-free nonlinear least-squares problems with POUNDERS*, in Advances and Trends in Optimization with Engineering Applications, T. Terlaky, M. F. Anjos, and S. Ahmed, eds., vol. 24 of MOS-SIAM Book Series on Optimization, MOS/SIAM, Philadelphia, 2017, pp. 529–539.