

Scalable Subspace Methods for Derivative-Free Nonlinear Least-Squares Optimization

Joint work with Coralia Cartis (Oxford)

Lindon Roberts, Australian National University (lindon.roberts@anu.edu.au)

20th IMA Leslie Fox Prize Event

21 June 2021

Partially supported by EPSRC (EP/L015803/1) & NAG Ltd.

1. **Introduction to derivative-free optimization (DFO)**
2. Subspace DFO methods: algorithm & theory
3. Specialization to least-squares: theory & practice
4. Numerical results

Nonlinear Optimization

Interested in **unconstrained nonlinear optimization**

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

where the objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is smooth.

- f is possibly nonconvex and/or 'black-box'
 - In practice, allow inaccurate evaluations of f , e.g. noise, outcome of iterative process
- Seek **local minimizer** (actually, approximate stationary point: $\|\nabla f(\mathbf{x})\|_2 \leq \epsilon$)

Solve using **trust-region methods** (adapt well to derivative-free setting).

(alternatives: BFGS+linesearch, cubic regularization, ...)

Basic trust-region method

- Approximate f near \mathbf{x}_k with a **local quadratic (Taylor) model**

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

- Get step by minimizing model in a neighborhood

$$\mathbf{s}_k = \arg \min_{\mathbf{s} \in \mathbb{R}^n} m_k(\mathbf{s}) \quad \text{subject to } \|\mathbf{s}\|_2 \leq \Delta_k$$

- Accept/reject step and adjust Δ_k based on quality of new point $f(\mathbf{x}_k + \mathbf{s}_k)$

$$\mathbf{x}_{k+1} = \begin{cases} \mathbf{x}_k + \mathbf{s}_k, & \text{if sufficient decrease,} & \longleftarrow (\text{maybe increase } \Delta_k) \\ \mathbf{x}_k, & \text{otherwise.} & \longleftarrow (\text{decrease } \Delta_k) \end{cases}$$

State-of-the-art algorithm with theoretical guarantees (e.g. $\lim_{k \rightarrow \infty} \|\nabla f(\mathbf{x}_k)\|_2 = 0$).

[Conn, Gould & Toint, 2000]

Derivative-Free Optimization

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

- How to calculate derivatives of f in practice?
 - Write code by hand
 - Finite differences
 - Algorithmic differentiation

Derivative-Free Optimization

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

- How to calculate derivatives of f in practice?
 - Write code by hand
 - Finite differences
 - Algorithmic differentiation
- Difficulties when function evaluation is
 - Black-box
 - Noisy
 - Computationally expensive

Derivative-Free Optimization

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

- How to calculate derivatives of f in practice?
 - Write code by hand
 - Finite differences
 - Algorithmic differentiation
- Difficulties when function evaluation is
 - Black-box
 - Noisy
 - Computationally expensive
- Alternative — **derivative-free optimization (DFO)**
- Many applications: climate, experimental design, machine learning, ...
- Several approaches: **model-based**, Nelder-Mead, direct search, ...

Model-Based DFO — Basic Ideas

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

- Instead, approximate

$$f(\mathbf{x}_k + \mathbf{s}) \approx m_k(\mathbf{s}) = f(\mathbf{x}_k) + \mathbf{g}_k^T \mathbf{s} + \frac{1}{2} \mathbf{s}^T \mathbf{H}_k \mathbf{s}$$

and find \mathbf{g}_k and \mathbf{H}_k without using derivatives

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

- Instead, approximate

$$f(\mathbf{x}_k + \mathbf{s}) \approx m_k(\mathbf{s}) = f(\mathbf{x}_k) + \mathbf{g}_k^T \mathbf{s} + \frac{1}{2} \mathbf{s}^T \mathbf{H}_k \mathbf{s}$$

and find \mathbf{g}_k and \mathbf{H}_k without using derivatives

- How? **Interpolate f over a set of points** — find $\mathbf{g}_k, \mathbf{H}_k$ such that

$$m_k(\mathbf{y} - \mathbf{x}_k) = f(\mathbf{y}), \quad \forall \mathbf{y} \in \mathcal{Y}$$

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

- Instead, approximate

$$f(\mathbf{x}_k + \mathbf{s}) \approx m_k(\mathbf{s}) = f(\mathbf{x}_k) + \mathbf{g}_k^T \mathbf{s} + \frac{1}{2} \mathbf{s}^T \mathbf{H}_k \mathbf{s}$$

and find \mathbf{g}_k and \mathbf{H}_k without using derivatives

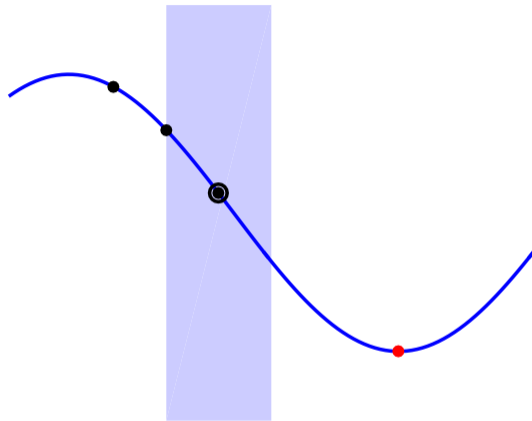
- How? **Interpolate f over a set of points** — find $\mathbf{g}_k, \mathbf{H}_k$ such that

$$m_k(\mathbf{y} - \mathbf{x}_k) = f(\mathbf{y}), \quad \forall \mathbf{y} \in \mathcal{Y}$$

- Use modified trust region method: shrink Δ_k or fix bad model?
- Ensure $\Delta_k \sim \|\nabla f(\mathbf{x}_k)\|_2$ to measure progress
- Geometry of points good \implies interpolation model Taylor-accurate \implies convergence

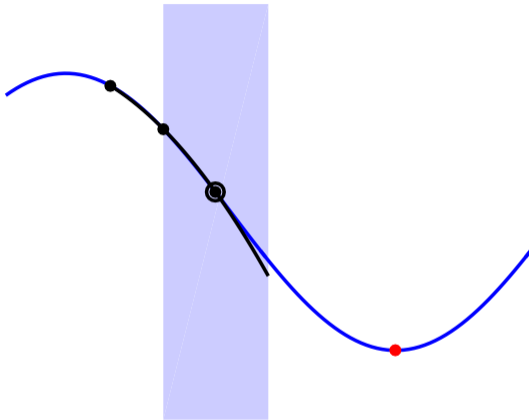
[Powell, 2003; Conn, Scheinberg & Vicente, 2009]

Example: Model-Based DFO



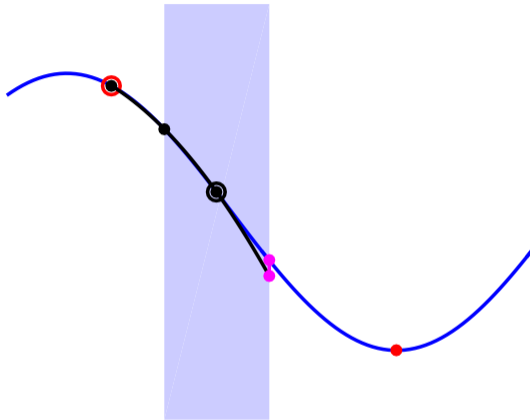
1. Choose interpolation set

Example: Model-Based DFO



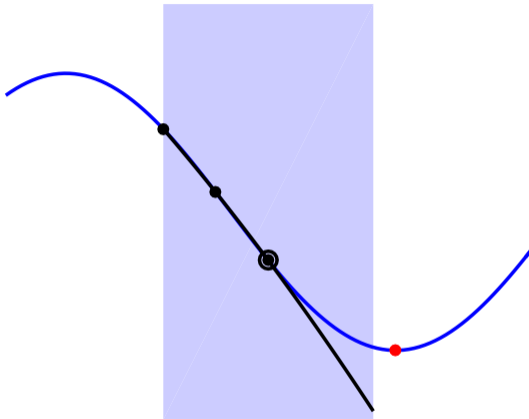
2. Interpolate & minimize...

Example: Model-Based DFO



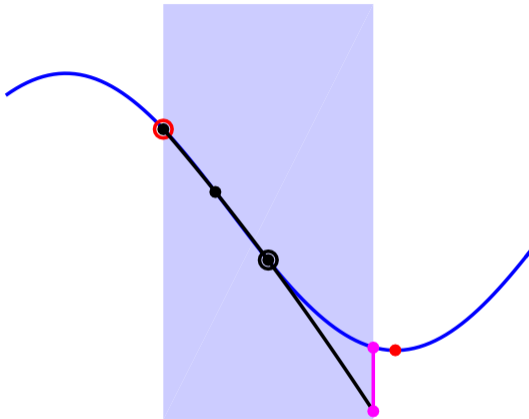
3. Add new point to interpolation set (replace a bad point)

Example: Model-Based DFO



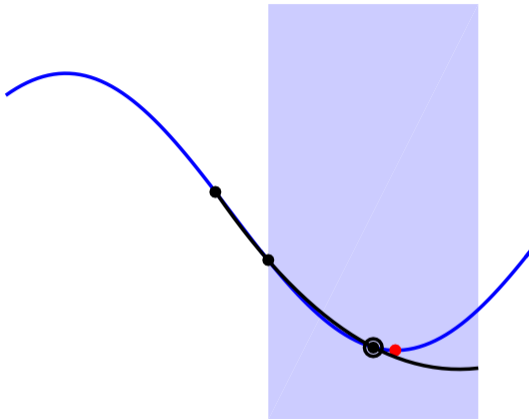
4. Repeat with new interpolation set & model

Example: Model-Based DFO



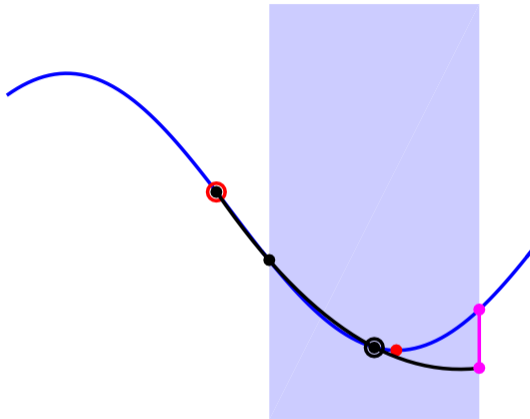
4. Repeat with new interpolation set & model

Example: Model-Based DFO



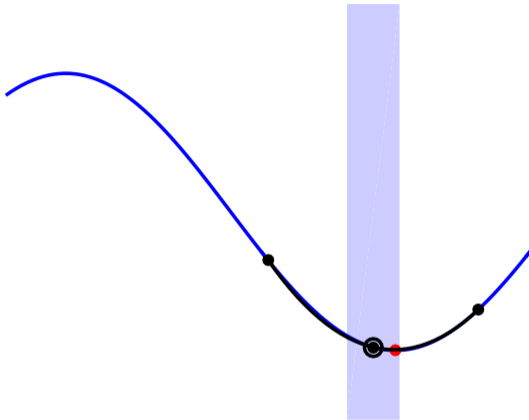
4. Repeat with new interpolation set & model

Example: Model-Based DFO



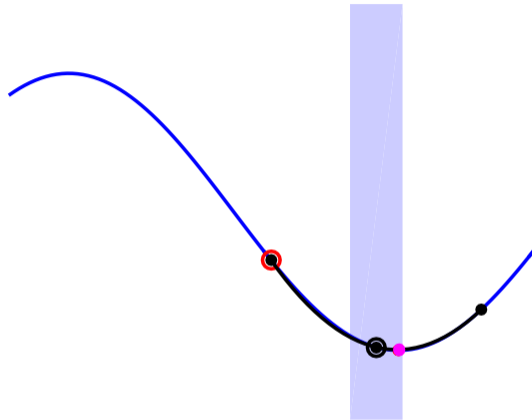
4. Repeat with new interpolation set & model

Example: Model-Based DFO



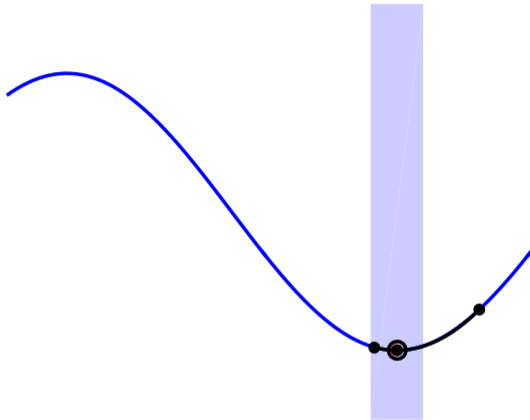
4. Repeat with new interpolation set & model

Example: Model-Based DFO



4. Repeat with new interpolation set & model

Example: Model-Based DFO



4. Repeat with new interpolation set & model

Model-based methods have similar convergence results to derivative-based methods.

Worst-case complexity: how many iterations before ϵ accuracy guaranteed?

Accuracy order	Model-based DFO	Taylor models
1st: $\ \nabla f(\mathbf{x}_k)\ _2 \leq \epsilon$	$\mathcal{O}(n^2\epsilon^{-2})$	$\mathcal{O}(\epsilon^{-2})$
2nd: 1st & $\lambda_{\min}(\nabla^2 f(\mathbf{x}_k)) \geq -\epsilon$	$\mathcal{O}(n^9\epsilon^{-3})$	$\mathcal{O}(\epsilon^{-3})$

[Cartis, Gould & Toint, 2010; Garmanjani, Júdice & Vicente, 2016]

- Same ϵ dependency as derivative-based, but **scales badly with problem dimension n**
- Substantial linear algebra work for interpolation and geometry management:
 - $\mathcal{O}(n^3)$ flops per iteration for linear models, $\mathcal{O}(n^6)$ for quadratic models.

Challenge

How can DFO methods be made scalable?

Model-based methods have similar convergence results to derivative-based methods.

Worst-case complexity: how many iterations before ϵ accuracy guaranteed?

Accuracy order	Model-based DFO	Taylor models
1st: $\ \nabla f(\mathbf{x}_k)\ _2 \leq \epsilon$	$\mathcal{O}(n^2\epsilon^{-2})$ $\mathcal{O}(\epsilon^{-2})$	$\mathcal{O}(\epsilon^{-2})$
2nd: 1st & $\lambda_{\min}(\nabla^2 f(\mathbf{x}_k)) \geq -\epsilon$	$\mathcal{O}(n^9\epsilon^{-3})$	$\mathcal{O}(\epsilon^{-3})$

[Cartis, Gould & Toint, 2010; Garmanjani, Júdice & Vicente, 2016]

- Same ϵ dependency as derivative-based, but ~~scales badly with problem dimension n~~
- Substantial linear algebra work for interpolation and geometry management:
 - $\mathcal{O}(n^3)$ $\mathcal{O}(n)$ flops per iteration for linear models, $\mathcal{O}(n^6)$ for quadratic models.

Challenge

How can DFO methods be made scalable?

1. Introduction to derivative-free optimization (DFO)
2. **Subspace DFO methods: algorithm & theory**
3. Specialization to least-squares: theory & practice
4. Numerical results

Challenge

How can DFO methods be made scalable?

- Exploit known problem structure [Porcelli & Toint, 2020; Bandeira et al., 2012]
- Randomized finite differencing ('gradient sampling') [Nesterov & Spokoiny, 2017]
- Randomized direct search: sample a subset of search directions — improves complexity from $\mathcal{O}(n^2\epsilon^{-2})$ to $\mathcal{O}(n\epsilon^{-2})$ [Gratton et al., 2015; Bergou et al., 2020]

Applications for scalable DFO methods include:

- Machine learning [Salimans et al., 2017; Ughi et al., 2020]
- Image analysis [Ehrhardt & R., 2021]
- Proxy for global optimization methods [Cartis, R. & Sheridan-Methven, 2021]

Subspace DFO

We use a subspace method: only search in **low-dimensional subspaces** of \mathbb{R}^n

- Related to coordinate descent methods [Wright, 2015; Patrascu & Necoara, 2015]
- Some implementations exist, but no theory [Gross & Parks, 2020; Neumaier et al., 2011]
- **Build on recent derivative-based analysis** [Cartis, Fowkes & Shao, 2020]

Subspace DFO

We use a subspace method: only search in **low-dimensional subspaces** of \mathbb{R}^n

- Related to coordinate descent methods [Wright, 2015; Patrascu & Necoara, 2015]
- Some implementations exist, but no theory [Gross & Parks, 2020; Neumaier et al., 2011]
- **Build on recent derivative-based analysis** [Cartis, Fowkes & Shao, 2020]

Subspace DFO framework:

- Generate subspace of dimension $p \ll n$ given by $\text{col}(Q_k)$ for random $Q_k \in \mathbb{R}^{n \times p}$
- **Build a low-dimensional model**: find $\hat{\mathbf{g}}_k \in \mathbb{R}^p$, $\hat{H}_k \in \mathbb{R}^{p \times p}$ to get

$$f(\mathbf{x}_k + Q_k \hat{\mathbf{s}}) \approx \hat{m}_k(\hat{\mathbf{s}}) = f(\mathbf{x}_k) + \hat{\mathbf{g}}_k^T \hat{\mathbf{s}} + \frac{1}{2} \hat{\mathbf{s}}^T \hat{H}_k \hat{\mathbf{s}},$$

- Solve subspace trust-region subproblem: $\min_{\hat{\mathbf{s}} \in \mathbb{R}^p} \hat{m}_k(\hat{\mathbf{s}})$ s.t. $\|\hat{\mathbf{s}}\|_2 \leq \Delta_k$
- **Benefits**: fewer interpolation points needed, cheap linear algebra (everything in \mathbb{R}^p).

Subspace DFO — Subspace Quality

Choice of subspace: we need to make sure we search in ‘good’ subspaces (where there is potential to decrease f sufficiently).

The subspace at iteration k is **well-aligned** if

$$\|Q_k^T \nabla f(\mathbf{x}_k)\|_2 \geq \alpha \|\nabla f(\mathbf{x}_k)\|_2, \quad \text{for some } \alpha > 0.$$

Subspace DFO — Subspace Quality

Choice of subspace: we need to make sure we search in ‘good’ subspaces (where there is potential to decrease f sufficiently).

The subspace at iteration k is **well-aligned** if

$$\|Q_k^T \nabla f(\mathbf{x}_k)\|_2 \geq \alpha \|\nabla f(\mathbf{x}_k)\|_2, \quad \text{for some } \alpha > 0.$$

Key Assumption

The subspace Q_k is well-aligned with probability $1 - \delta$ (whenever Q_k is resampled, independent of history), and $\|Q_k\|_2 \leq Q_{\max}$.

Why? If $\|\nabla f(\mathbf{x}_k)\|_2 \geq \epsilon$, Q_k well-aligned and \hat{m}_k fully linear, then $\|\hat{\mathbf{g}}_k\|_2 \geq \Omega(\epsilon)$

– If there is still work to do, then the algorithm (probably) knows it

Subspace DFO Algorithm

RSDFO (Random Subspace DFO):

[model-based DFO, RSDFO-specific]

1. If FLAG, use previous $Q_k = Q_{k-1}$ and construct fully linear subspace model \hat{m}_k .
2. Otherwise, generate random Q_k and construct subspace model \hat{m}_k .
3. If $\|\hat{\mathbf{g}}_k\|_2$ small, ensure model fully linear and $\Delta_k \sim \|\nabla f(\mathbf{x}_k)\|_2$. [criticality]
4. Minimize model to get $\mathbf{s}_k = Q_k \hat{\mathbf{s}}_k$, evaluate $f(\mathbf{x}_k + \mathbf{s}_k)$.
5. Check sufficient decrease, then accept/reject step and update Δ_k :
 - If decrease: $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$ and $\Delta_{k+1} = \gamma_{\text{inc}} \Delta_k$, add \mathbf{x}_{k+1} to model. [successful]
 - If no decrease and model not fully linear: $\mathbf{x}_{k+1} = \mathbf{x}_k$ and $\Delta_{k+1} = \Delta_k$, make model fully linear. Set FLAG=TRUE. [model-improving]
 - If no decrease and model fully linear: $\mathbf{x}_{k+1} = \mathbf{x}_k$ and $\Delta_{k+1} = \gamma_{\text{dec}} \Delta_k$. [unsuccessful]

Theorem (Cartis & R., 2021)

If f is sufficiently smooth and bounded below, $\gamma_{dec} > \gamma_{inc}^{-1/2}$ and ϵ sufficiently small, then for some $c, C > 0$,

$$\mathbb{P} \left[K_\epsilon \leq \frac{C}{\alpha^2(1-\delta)\epsilon^2} \right] \geq 1 - e^{-c\epsilon^{-2}},$$

where K_ϵ is the first iteration with $\|\nabla f(\mathbf{x}_k)\|_2 \leq \epsilon$.

- Matches usual $\mathcal{O}(\epsilon^{-2})$ worst-case complexity bound with high probability
- Implies $\mathbb{E}[K_\epsilon] = \mathcal{O}(\epsilon^{-2})$ and almost-sure convergence
- Constant C depends on p (from fully linear error bounds), c depends on p and δ

Convergence Proof — Sketch

Proof sketch: while $\|\nabla f(\mathbf{x}_k)\|_2 > \epsilon$, bound number of iterations across 6 cases.

Good subspace:

1. Δ_k large + successful: get $f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}) \geq \Omega(\epsilon^2)$, so happens $\mathcal{O}(\epsilon^{-2})$ times.
2. Δ_k large + unsuccessful: bounded by case #1 from Δ_k management.
3. Δ_k small + unsuccessful + good model: doesn't happen (Taylor accuracy)
4. Δ_k small + successful: bounded by cases #3 and #5 from Δ_k management
5. Δ_k small + bad model: keep Q_k and Δ_k , build good model (next time #3 or #4)

(extra difficulties: different Δ_k large/small thresholds, $4 \leftrightarrow 5$, criticality steps, ...)

Bad subspace:

6. Happens with small probability δ . Need $\gamma_{\text{dec}} > \gamma_{\text{inc}}^{-1/2}$ to ensure Δ_k not decreased too quickly in these iterations.

Generating Q_k

For RSDFO to work, need to be able to generate Q_k such that

$$\|Q_k^T \nabla f(\mathbf{x}_k)\|_2 \geq \alpha \|\nabla f(\mathbf{x}_k)\|_2 \quad \text{with probability } \geq 1 - \delta.$$

If Q_k is a random orthonormal set (e.g. block coordinates), need $p \sim \alpha n$.

Generating Q_k

For RSDFO to work, need to be able to generate Q_k such that

$$\|Q_k^T \nabla f(\mathbf{x}_k)\|_2 \geq \alpha \|\nabla f(\mathbf{x}_k)\|_2 \quad \text{with probability } \geq 1 - \delta.$$

If Q_k is a random orthonormal set (e.g. block coordinates), need $p \sim \alpha n$.

Instead, make Q_k a **Johnson-Lindenstrauss embedding**, such as

- Q_k has i.i.d. Gaussian entries $\mathcal{N}(0, 1/p)$
- Q_k has s random nonzero entries per row, value $\pm 1/\sqrt{s}$ with probability $1/2$

Then, only need $p \sim (1 - \alpha)^{-2} |\log \delta|$, **independent of n** .

Generating Q_k

For RSDFO to work, need to be able to generate Q_k such that

$$\|Q_k^T \nabla f(\mathbf{x}_k)\|_2 \geq \alpha \|\nabla f(\mathbf{x}_k)\|_2 \quad \text{with probability } \geq 1 - \delta.$$

If Q_k is a random orthonormal set (e.g. block coordinates), need $p \sim \alpha n$.

Instead, make Q_k a **Johnson-Lindenstrauss embedding**, such as

- Q_k has i.i.d. Gaussian entries $\mathcal{N}(0, 1/p)$
- Q_k has s random nonzero entries per row, value $\pm 1/\sqrt{s}$ with probability $1/2$

Then, only need $p \sim (1 - \alpha)^{-2} |\log \delta|$, **independent of n** .

Accuracy order	Model-based DFO	RSDFO	Taylor models
1st	$\mathcal{O}(n^2 \epsilon^{-2})$	$\mathcal{O}(\epsilon^{-2})$	$\mathcal{O}(\epsilon^{-2})$
2nd	$\mathcal{O}(n^9 \epsilon^{-3})$??	$\mathcal{O}(\epsilon^{-3})$

1. Introduction to derivative-free optimization (DFO)
2. Subspace DFO methods: algorithm & theory
3. **Specialization to least-squares: theory & practice**
4. Numerical results

DFO for Least-Squares — Basic Framework

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) = \frac{1}{2} \|\mathbf{r}(\mathbf{x})\|_2^2, \quad \mathbf{r}(\mathbf{x}) \in \mathbb{R}^m$$

Classical Gauss-Newton

Derivative-Free Gauss-Newton

DFO for Least-Squares — Basic Framework

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) = \frac{1}{2} \|\mathbf{r}(\mathbf{x})\|_2^2, \quad \mathbf{r}(\mathbf{x}) \in \mathbb{R}^m$$

Classical Gauss-Newton

Derivative-Free Gauss-Newton

- Linearize \mathbf{r} at \mathbf{x}_k using Jacobian

$$\mathbf{r}(\mathbf{x}_k + \mathbf{s}) \approx \mathbf{m}_k(\mathbf{s}) = \mathbf{r}(\mathbf{x}_k) + \mathbf{J}(\mathbf{x}_k)\mathbf{s}$$

DFO for Least-Squares — Basic Framework

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) = \frac{1}{2} \|\mathbf{r}(\mathbf{x})\|_2^2, \quad \mathbf{r}(\mathbf{x}) \in \mathbb{R}^m$$

Classical Gauss-Newton

- Linearize \mathbf{r} at \mathbf{x}_k using Jacobian

$$\mathbf{r}(\mathbf{x}_k + \mathbf{s}) \approx \mathbf{m}_k(\mathbf{s}) = \mathbf{r}(\mathbf{x}_k) + \mathbf{J}(\mathbf{x}_k)\mathbf{s}$$

Derivative-Free Gauss-Newton

- Jacobian not available: use

$$\mathbf{m}_k(\mathbf{s}) = \mathbf{r}(\mathbf{x}_k) + \mathbf{J}_k\mathbf{s}$$

- Find \mathbf{J}_k using linear interpolation

DFO for Least-Squares — Basic Framework

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) = \frac{1}{2} \|\mathbf{r}(\mathbf{x})\|_2^2, \quad \mathbf{r}(\mathbf{x}) \in \mathbb{R}^m$$

Classical Gauss-Newton

- Linearize \mathbf{r} at \mathbf{x}_k using Jacobian

$$\mathbf{r}(\mathbf{x}_k + \mathbf{s}) \approx \mathbf{m}_k(\mathbf{s}) = \mathbf{r}(\mathbf{x}_k) + \mathbf{J}(\mathbf{x}_k)\mathbf{s}$$

Derivative-Free Gauss-Newton

- Jacobian not available: use

$$\mathbf{m}_k(\mathbf{s}) = \mathbf{r}(\mathbf{x}_k) + \mathbf{J}_k\mathbf{s}$$

- Find \mathbf{J}_k using **linear interpolation**

In both cases, get a local quadratic model

$$f(\mathbf{x}_k + \mathbf{s}) \approx m_k(\mathbf{s}) = \frac{1}{2} \|\mathbf{m}_k(\mathbf{s})\|_2^2$$

Implemented in state-of-the-art solver **DFO-LS** (+ NAG Library)

[Cartis et al., 2019]

DFO for Least-Squares

Standard method has first-order complexity $\mathcal{O}(n^6\epsilon^{-2})$: dependency on n between first & second order methods. [Cartis & R., 2019]

RSDFO with Gauss-Newton models gets **dimension-independent $\mathcal{O}(\epsilon^{-2})$ bound**.

DFO for Least-Squares

Standard method has first-order complexity $\mathcal{O}(n^6\epsilon^{-2})$: dependency on n between first & second order methods. [Cartis & R., 2019]

RSDFO with Gauss-Newton models gets **dimension-independent $\mathcal{O}(\epsilon^{-2})$ bound**.

Practical considerations:

- Linear algebra cost of standard method is $\mathcal{O}(mn^2 + n^3)$ flops per iteration from linear interpolation, RSDFO only needs $\mathcal{O}(mp^2 + np^2)$
- Standard method reuses (possibly expensive) evaluations of $\mathbf{r}(\mathbf{x})$ across iterations, RSDFO has to resample all points from new subspace

Practical Challenge

Can we construct a method with reduced interpolation cost, but still efficient in # evaluations of $\mathbf{r}(\mathbf{x})$?

Practical Challenge

Can we construct a method with reduced interpolation cost, but still efficient in # evaluations of $\mathbf{r}(\mathbf{x})$?

The key idea here is to **use the locations of interpolation points to define the subspace**.

If we have $p + 1$ interpolation points $\{\mathbf{x}_k, \mathbf{y}_1, \dots, \mathbf{y}_p\}$, then make Q_k an orthonormal basis for $\{\mathbf{y}_1 - \mathbf{x}_k, \dots, \mathbf{y}_p - \mathbf{x}_k\}$ (from QR factorization).

- Same low linear algebra cost, but $\mathbf{s}_k \in \text{col}(Q_k)$ — only explore initial subspace!
- Need a mechanism to explore the whole space:
 - i.e. need to **change Q_k on each iteration**
 - Replace some interpolation points with **random directions** (orthogonal to Q_k)
 - **No free lunch**: more new subspace directions requires more new evaluations

Derivative-Free Block Gauss-Newton

Algorithm DFBGN (Derivative-Free Block Gauss-Newton):

1. Build low-dimensional model and calculate trust-region step $\mathbf{s}_k = Q_k \hat{\mathbf{s}}_k$
2. Evaluate $f(\mathbf{x}_k + \mathbf{s}_k)$, accept/reject step, and update Δ_k (as before)
3. Add $\mathbf{x}_k + \mathbf{s}_k$ to interpolation set
4. Remove $p_{drop} \geq 2$ points from the interpolation set
5. Add random orthogonal directions $\mathbf{x}_k + \Delta_k \mathbf{d}$ until $p + 1$ interpolation points

Derivative-Free Block Gauss-Newton

Algorithm DFBGN (Derivative-Free Block Gauss-Newton):

1. Build low-dimensional model and calculate trust-region step $\mathbf{s}_k = Q_k \hat{\mathbf{s}}_k$
2. Evaluate $f(\mathbf{x}_k + \mathbf{s}_k)$, accept/reject step, and update Δ_k (as before)
3. Add $\mathbf{x}_k + \mathbf{s}_k$ to interpolation set
4. Remove $p_{drop} \geq 2$ points from the interpolation set
5. Add random orthogonal directions $\mathbf{x}_k + \Delta_k \mathbf{d}$ until $p + 1$ interpolation points

Comments:

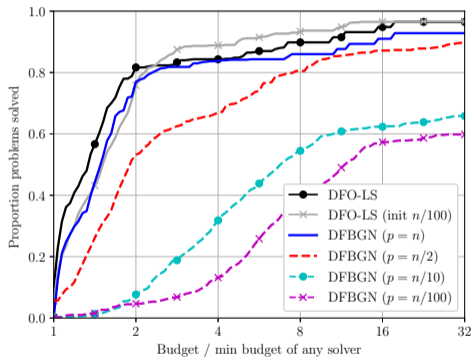
- $p_{drop} \geq 2$ ensures new direction(s) \mathbf{d} added next iteration $\implies Q_{k+1} \neq Q_k$.
 - Practical choice: $p_{drop} = 2$ on success, $p/10$ otherwise (geometry-aware removal)
- Linear algebra cost $\mathcal{O}(mp^2 + np^2)$ vs. standard method $\mathcal{O}(mn^2 + n^3)$
- Package on Github: [numericalalgorithmsgroup/dfbgn](https://github.com/numericalalgorithmsgroup/dfbgn)

1. Introduction to derivative-free optimization (DFO)
2. Subspace DFO methods: algorithm & theory
3. Specialization to least-squares: theory & practice
4. **Numerical results**

Numerical Results — low accuracy

DFBGN vs. DFO-LS (low accuracy $\tau = 10^{-1}$)

[% problems solved vs. # evals]

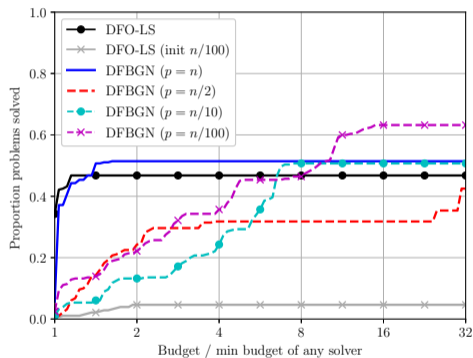


Medium-scale problems, $n \approx 100$

DFBGN is more suitable for low accuracy solutions, performance improves with larger p

Numerical Results — high dimensional problems

Compare DFBGN method to DFO-LS (low accuracy $\tau = 10^{-1}$)



Large problems $n \approx 1000$, 12hr timeout

DFBGN outperforms DFO-LS for low accuracy solutions on large-scale problems...

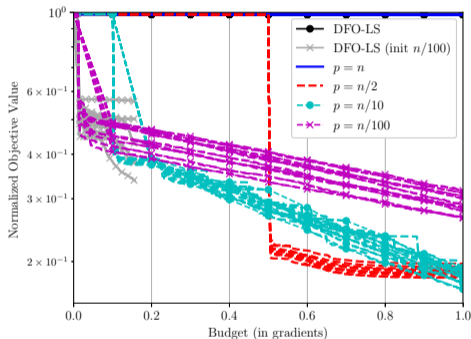
Proportion of problems where solver times out (before usual termination):

Solver	Timeout
DFO-LS	93%
DFO-LS (init $n/100$)	98%
DFBGN ($p = n/100$)	35%
DFBGN ($p = n/10$)	74%
DFBGN ($p = n/2$)	82%
DFBGN ($p = n$)	66%

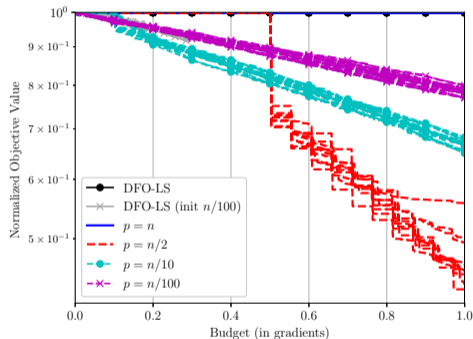
... because it doesn't time out

Numerical Results — low budget

Other advantage: **DFBGN** progresses after $p \ll n$ evaluations (important when n large)



ARWHDNE, $n = 2000$



CHANDHEQ, $n = 2000$

(normalized objective reduction vs. # evaluations, 12hr timeout)

Conclusions

- Scalability of model-based DFO is currently limited (in theory & practice)
- New algorithms reduce linear algebra cost and iteration complexity
- Novel complexity analysis with dimension-independent bounds
- DFBN outperforms state-of-the-art code on large-scale problems

Future Work

- Second-order complexity analysis
- Efficient implementation of subspace quadratic models
- Similar strategies for direct search DFO

[[arXiv:2102.12016](https://arxiv.org/abs/2102.12016), Github: [numericalalgorithmsgroup/dfbgn](https://github.com/numericalalgorithmsgroup/dfbgn)]

- A. S. BANDEIRA, K. SCHEINBERG, AND L. N. VICENTE, *Computation of sparse low degree interpolating polynomials and their application to derivative-free optimization*, *Mathematical Programming*, 134 (2012), pp. 223–257.
- E. H. BERGOU, E. GORBUNOV, AND P. RICHTÁRIK, *Stochastic three points method for unconstrained smooth minimization*, *SIAM Journal on Optimization*, (2020).
- 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, J. FOWKES, AND Z. SHAO, *A randomised subspace Gauss-Newton method for nonlinear least-squares*, in *Workshop on “Beyond first-order methods in ML systems” at the 37th International Conference on Machine Learning*, Vienna, Austria, 2020.
- C. CARTIS, N. I. M. GOULD, AND P. L. TOINT, *On the complexity of steepest descent, Newton’s and regularized Newton’s methods for nonconvex unconstrained optimization problems*, *SIAM Journal on Optimization*, 20 (2010), pp. 2833–2852.

C. CARTIS AND L. ROBERTS, *A derivative-free Gauss-Newton method*, Mathematical Programming Computation, 11 (2019), pp. 631–674.

———, *Scalable subspace methods for derivative-free nonlinear least-squares optimization*, arXiv preprint arXiv:2102.12016, (2021).

C. CARTIS, L. ROBERTS, AND O. SHERIDAN-METHVEN, *Escaping local minima with local derivative-free methods: a numerical investigation*, Optimization, to appear (2021).

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.

M. J. EHRHARDT AND L. ROBERTS, *Inexact derivative-free optimization for bilevel learning*, Journal of Mathematical Imaging and Vision, 63 (2020), pp. 580–600.

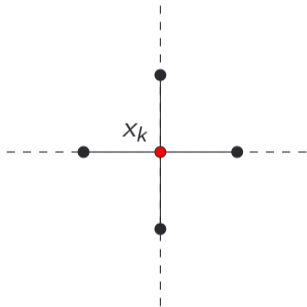
R. GARMANJANI, D. JÚDICE, AND L. N. VICENTE, *Trust-region methods without using derivatives: Worst case complexity and the nonsmooth case*, SIAM Journal on Optimization, 26 (2016), pp. 1987–2011.

- 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.
- J. C. GROSS AND G. T. PARKS, *Optimization by moving ridge functions: Derivative-free optimization for computationally intensive functions*, arXiv preprint arXiv:2007.04893, (2020).
- Y. NESTEROV AND V. SPOKOINY, *Random gradient-free minimization of convex functions*, Foundations of Computational Mathematics, 17 (2017), pp. 527–566.
- A. NEUMAIER, H. FENDL, H. SCHILLY, AND T. LEITNER, *VXQR: Derivative-free unconstrained optimization based on QR factorizations*, Soft Computing, 15 (2011), pp. 2287–2298.
- A. PATRASCU AND I. NECOARA, *Efficient random coordinate descent algorithms for large-scale structured nonconvex optimization*, Journal of Global Optimization, 61 (2015), pp. 19–46.
- M. PORCELLI AND P. L. TOINT, *Global and local information in structured derivative free optimization with BFO*, arXiv preprint arXiv:2001.04801, (2020).
- M. J. D. POWELL, *On trust region methods for unconstrained minimization without derivatives*, Mathematical Programming, 97 (2003), pp. 605–623.

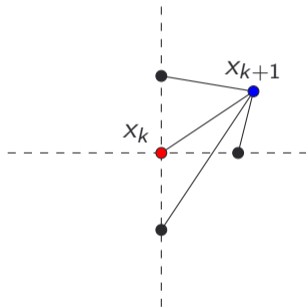
- T. SALIMANS, J. HO, X. CHEN, S. SIDOR, AND I. SUTSKEVER, *Evolution strategies as a scalable alternative to reinforcement learning*, arXiv preprint arXiv:1703.03864, (2017).
- G. UGHI, V. ABROL, AND J. TANNER, *An empirical study of derivative-free-optimization algorithms for targeted black-box attacks in deep neural networks*, arXiv preprint arXiv:2012.01901, (2020).
- S. J. WRIGHT, *Coordinate descent algorithms*, *Mathematical Programming*, 151 (2015), pp. 3–34.

General Objective Problems

General objective case is much harder — rely on **quadratic** interpolation models.



2 points per subspace direction

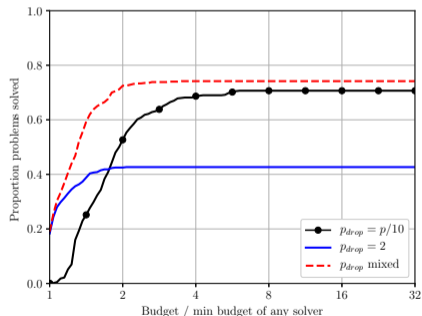


After step, how to rotate subspace?

Subspace dimensions decoupled from interpolation directions $y_t - x_k$

How to choose p_{drop} ?

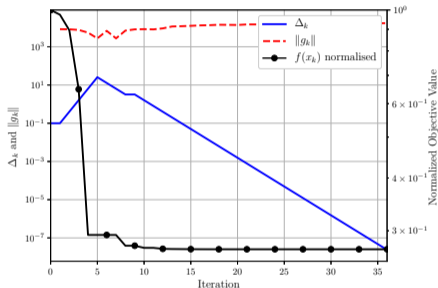
- Large changes to Q_k (e.g. $p_{drop} = p/10$) — explore whole space quickly
- Small changes to Q_k (e.g. $p_{drop} = 2$) — use few evaluations
- Compromise? ($p_{drop} = 2$ on successful iterations, $p/10$ on unsuccessful iterations)



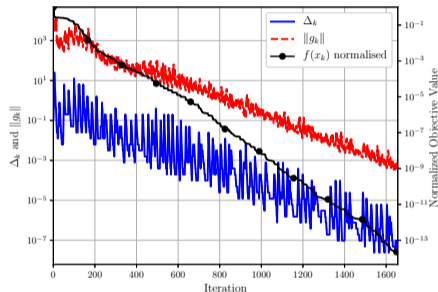
% problems solved vs. # objective evaluations (normalized)

Choice of p_{drop}

Choice of p_{drop} prevents Δ_k too small too soon (needed for convergence)



$p_{drop} = 2$



p_{drop} mixed

(CUTEst problem LUKSAN13 with $n = 100$)