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)

Applied Mathematics Seminar, University of Leicester
30 September 2021

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

Outline

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

Unconstrained nonlinear optimization: Given $f : \mathbb{R}^n \to \mathbb{R}$, solve

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

Nonlinear Optimization

Unconstrained nonlinear optimization: Given $f : \mathbb{R}^n \to \mathbb{R}$, solve

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

Meaning?

- Global minimizer: find $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathbb{R}^n$

Nonlinear Optimization

Unconstrained nonlinear optimization: Given $f : \mathbb{R}^n \to \mathbb{R}$, solve

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

Meaning?

- Global minimizer: find $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathbb{R}^n$
- Local minimizer: find $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in B(\boldsymbol{x}^*, \epsilon)$, some $\epsilon > 0$
  - Sufficient condition: $\nabla f(\boldsymbol{x}^*) = 0$ and $\nabla^2 f(\boldsymbol{x}^*)$ positive definite

Nonlinear Optimization

Unconstrained nonlinear optimization: Given $f : \mathbb{R}^n \to \mathbb{R}$, solve

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

Meaning?

- Global minimizer: find $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}^*) \le f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathbb{R}^n$
- Local minimizer: find $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}^*) \le f(\boldsymbol{x})$ for all $\boldsymbol{x} \in B(\boldsymbol{x}^*, \epsilon)$, some $\epsilon > 0$
  - Sufficient condition: $\nabla f(\boldsymbol{x}^*) = 0$ and $\nabla^2 f(\boldsymbol{x}^*)$ positive definite
- Stationary point: find $\boldsymbol{x}^*$ such that $\nabla f(\boldsymbol{x}^*) = 0$

If f is convex (e.g. $f(x) = x^2$) then all conditions above are equivalent (not today!).

Nonlinear Optimization: Motivation

Important problem across every quantative discipline!

Nonlinear Optimization: Motivation

Important problem across every quantative discipline!

Example application: least-squares parameter fitting

- Observations of some process: $(\boldsymbol{w}_1, y_1), \ldots, (\boldsymbol{w}_m, y_m)$
- Model for the process, parametrized by $\boldsymbol{x}$: $y \approx \text{model}(\boldsymbol{w}, \boldsymbol{x})$
    - e.g. Linear regression $\text{model}(\boldsymbol{w}, \boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$, PDE model, neural network, ...

Nonlinear Optimization: Motivation

Important problem across every quantative discipline!

Example application: least-squares parameter fitting

- Observations of some process: $(\boldsymbol{w}_1, y_1), \dots, (\boldsymbol{w}_m, y_m)$
- Model for the process, parametrized by $\boldsymbol{x}$: $y \approx \text{model}(\boldsymbol{w}, \boldsymbol{x})$
  - e.g. Linear regression $\text{model}(\boldsymbol{w}, \boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$, PDE model, neural network, ...
- Fitting/learning: find parameters which fit data

$$\min_{\boldsymbol{x}} \sum_{i=1}^{m} \|y_i - \text{model}(\boldsymbol{w}_i, \boldsymbol{x})\|^2.$$

- Final fitted model: $\boldsymbol{w} \to \text{model}(\boldsymbol{w}, \boldsymbol{x}^*)$

Other metrics ("losses") over y are possible: e.g. adjust for correlations, robust to outliers, ...

Basic trust-region method

Solve using trust-region methods *(alternatives: BFGS+linesearch, nonlinear CG, ...)*

Basic trust-region method

Solve using trust-region methods *(alternatives: BFGS+linesearch, nonlinear CG, ...)*

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

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

- Get step by minimizing model in a neighborhood

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

Basic trust-region method

Solve using trust-region methods *(alternatives: BFGS+linesearch, nonlinear CG, ...)*

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

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

- Get step by minimizing model in a neighborhood

$$\boldsymbol{s}_k = \operatorname*{arg\,min}_{\boldsymbol{s}\in\mathbb{R}^n} m_k(\boldsymbol{s}) \qquad \text{subject to } \|\boldsymbol{s}\|_2 \leq \Delta_k$$

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

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

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

[Conn, Gould & Toint, 2000]

Derivative-Free Optimization

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

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

Derivative-Free Optimization

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

- 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

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

- 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

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

- Instead, approximate

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

and find $\boldsymbol{g}_k$ and H_k without using derivatives

Model-Based DFO — Basic Ideas

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

- Instead, approximate

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

  and find $\boldsymbol{g}_k$ and H_k without using derivatives

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

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

Model-Based DFO — Basic Ideas

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

- Instead, approximate

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

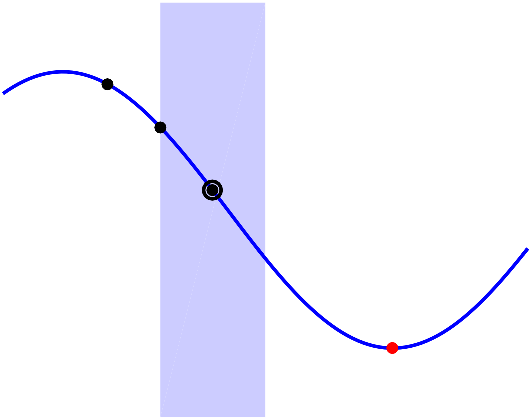
  and find $\boldsymbol{g}_k$ and H_k without using derivatives
- How? Interpolate f over a set of points — find $\boldsymbol{g}_k$, H_k such that

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

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

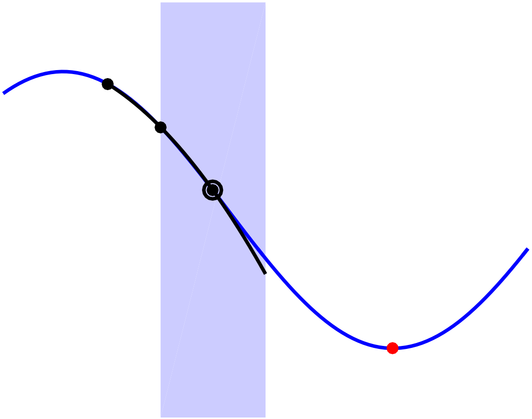
[Powell, 2003; Conn, Scheinberg & Vicente, 2009]

Example: Model-Based DFO



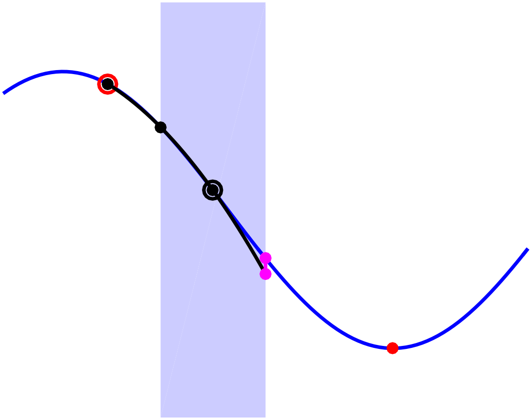
1. Choose interpolation set

Example: Model-Based DFO



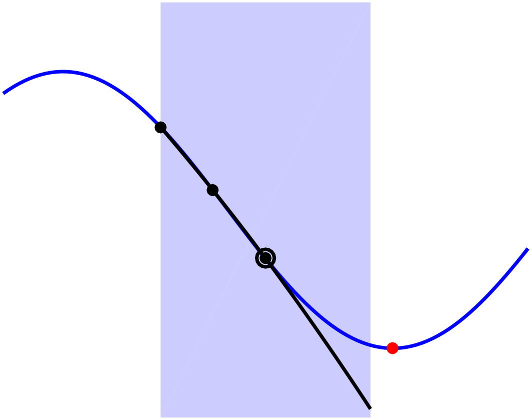
2. Interpolate & minimize...

Example: Model-Based DFO



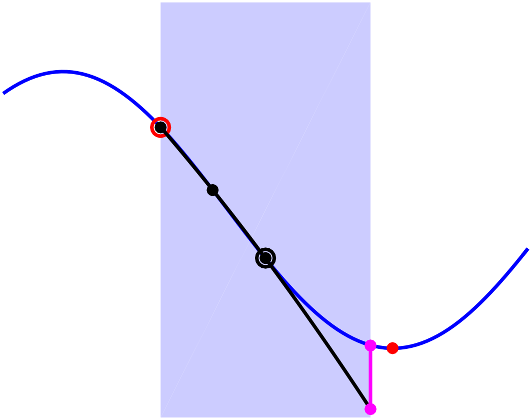
3. Add new point to interpolation set (replace a bad point)

Example: Model-Based DFO



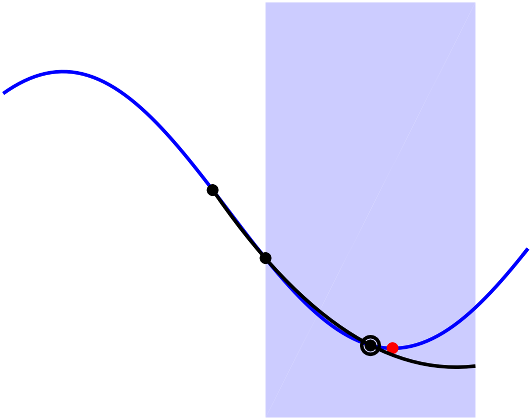
4. Repeat with new interpolation set & model

Example: Model-Based DFO



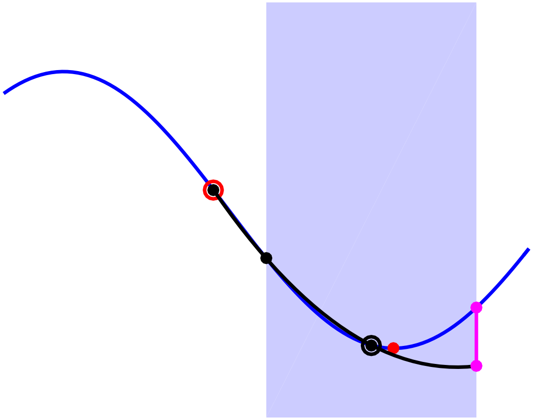
4. Repeat with new interpolation set & model

Example: Model-Based DFO



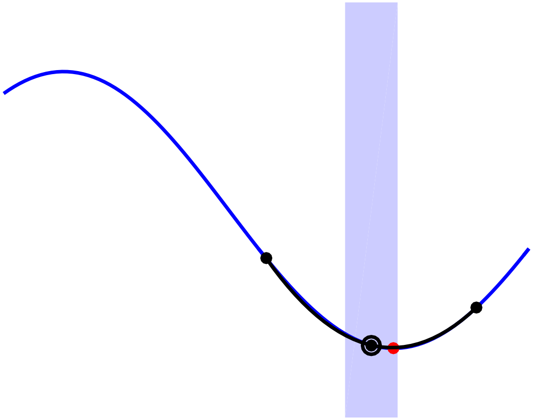
4. Repeat with new interpolation set & model

Example: Model-Based DFO



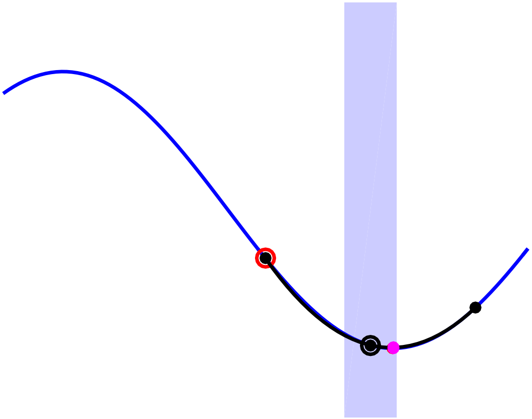
4. Repeat with new interpolation set & model

Example: Model-Based DFO



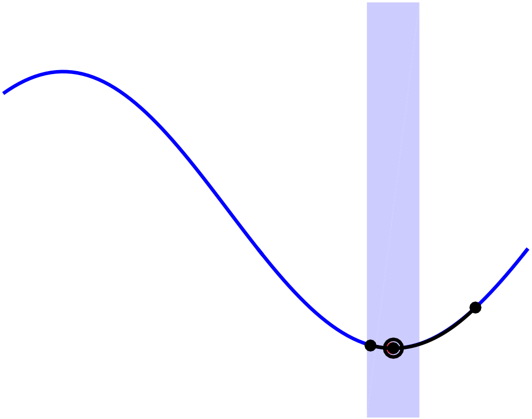
4. Repeat with new interpolation set & model

Example: Model-Based DFO



4. Repeat with new interpolation set & model

Example: Model-Based DFO



4. Repeat with new interpolation set & model

Model-Based DFO — Theory

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

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

Model-Based DFO — Theory

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: $\|\nabla f(\boldsymbol{x}_k)\|_2 \leq \epsilon$	$\mathcal{O}(n^2\epsilon^{-2})$	$\mathcal{O}(\epsilon^{-2})$
2nd: 1st & $\lambda_{\min}(\nabla^2 f(\boldsymbol{x}_k)) \geq -\epsilon$	$\mathcal{O}(n^9\epsilon^{-3})$	$\mathcal{O}(\epsilon^{-3})$

[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 DFO — Theory

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: $\|\nabla f(\boldsymbol{x}_k)\|_2 \leq \epsilon$	~~$\mathcal{O}(n^2\epsilon^{-2})$~~ $\mathcal{O}(\epsilon^{-2})$	$\mathcal{O}(\epsilon^{-2})$
2nd: 1st & $\lambda_{\min}(\nabla^2 f(\boldsymbol{x}_k)) \geq -\epsilon$	$\mathcal{O}(n^9\epsilon^{-3})$	$\mathcal{O}(\epsilon^{-3})$

[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?

Outline

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

Scalable DFO

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 $\hat{\boldsymbol{g}}_k \in \mathbb{R}^p$, $\hat{H}_k \in \mathbb{R}^{p \times p}$ to get

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

- Solve subspace trust-region subproblem: $\min_{\hat{\boldsymbol{s}} \in \mathbb{R}^p} \hat{m}_k(\hat{\boldsymbol{s}})$ s.t. $\|\hat{\boldsymbol{s}}\|_2 \leq \Delta_k$
- 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

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

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

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

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

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

Subspace DFO — Convergence

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 $\|\nabla f(\boldsymbol{x}_k)\|_2 \leq \epsilon$.

- Matches usual $\mathcal{O}(\epsilon^{-2})$ worst-case complexity bound with high probability
- Implies $\mathbb{E}\left[K_\epsilon\right] = \mathcal{O}(\epsilon^{-2})$ 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 $\|\nabla f(\boldsymbol{x}_k)\|_2 > \epsilon$, bound number of iterations across 6 cases.

Good subspace:

1. Δ_k large + successful: get $f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k+1}) \geq \Omega(\epsilon^2)$, 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

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

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

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

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

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

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	$\mathcal{O}(n^9\epsilon^{-3})$	??	$\mathcal{O}(\epsilon^{-3})$

Outline

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

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

Classical Gauss-Newton

Derivative-Free Gauss-Newton

DFO for Least-Squares — Basic Framework

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

Classical Gauss-Newton **Derivative-Free Gauss-Newton**

- Linearize $\boldsymbol{r}$ at $\boldsymbol{x}_k$ using Jacobian

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

DFO for Least-Squares — Basic Framework

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

Classical Gauss-Newton

- Linearize $\boldsymbol{r}$ at $\boldsymbol{x}_k$ using Jacobian

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

Derivative-Free Gauss-Newton

- Jacobian not available: use

$$\boldsymbol{m}_k(\boldsymbol{s}) = \boldsymbol{r}(\boldsymbol{x}_k) + J_k\boldsymbol{s}$$

- Find J_k using linear interpolation

DFO for Least-Squares — Basic Framework

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

Classical Gauss-Newton

- Linearize $\boldsymbol{r}$ at $\boldsymbol{x}_k$ using Jacobian

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

Derivative-Free Gauss-Newton

- Jacobian not available: use

$$\boldsymbol{m}_k(\boldsymbol{s}) = \boldsymbol{r}(\boldsymbol{x}_k) + J_k\boldsymbol{s}$$

- Find J_k using linear interpolation

In both cases, get a local quadratic model

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

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 $\boldsymbol{r}(\boldsymbol{x})$ 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 $\boldsymbol{r}(\boldsymbol{x})$?

Derivative-Free Block Gauss-Newton

Practical Challenge

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

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

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

- Same low linear algebra cost, but $\boldsymbol{s}_k \in \mathrm{col}(Q_k)$ — 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 $\boldsymbol{s}_k = Q_k \hat{\boldsymbol{s}}_k$
2. Evaluate $f(\boldsymbol{x}_k + \boldsymbol{s}_k)$, accept/reject step, and update Δ_k (as before)
3. Add $\boldsymbol{x}_k + \boldsymbol{s}_k$ to interpolation set
4. Remove $p_{drop} \geq 2$ points from the interpolation set
5. Add random orthogonal directions $\boldsymbol{x}_k + \Delta_k \boldsymbol{d}$ 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 $\boldsymbol{s}_k = Q_k \hat{\boldsymbol{s}}_k$
2. Evaluate $f(\boldsymbol{x}_k + \boldsymbol{s}_k)$, accept/reject step, and update Δ_k (as before)
3. Add $\boldsymbol{x}_k + \boldsymbol{s}_k$ to interpolation set
4. Remove $p_{drop} \geq 2$ points from the interpolation set
5. Add random orthogonal directions $\boldsymbol{x}_k + \Delta_k \boldsymbol{d}$ until $p+1$ interpolation points

Comments:

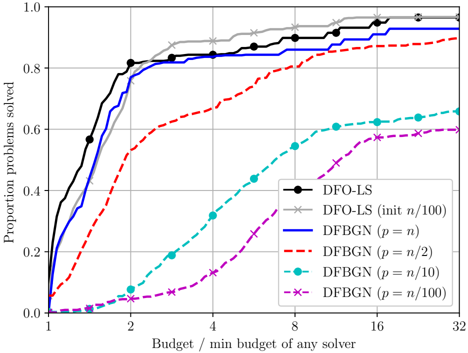
- $p_{drop} \geq 2$ ensures new direction(s) $\boldsymbol{d}$ added next iteration $\implies$ $Q_{k+1} \neq Q_k$.
    - Practical choice: $p_{\text{drop}} = 2$ on success, $p/10$ otherwise (geometry-aware removal)
- Linear algebra cost $\mathcal{O}(mp^2 + np^2)$ vs. standard method $\mathcal{O}(mn^2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

Outline

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]*

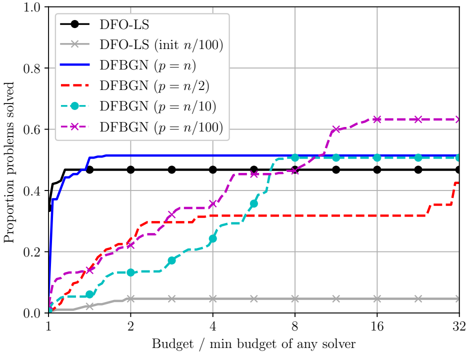


Medium-scale problems, $n \approx 100$

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...

Timeout Rate

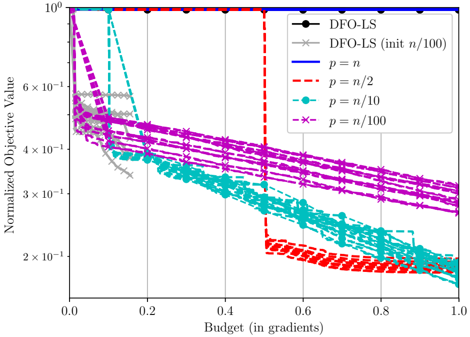
Proportion of problems where solver times out (before usual termination):

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%

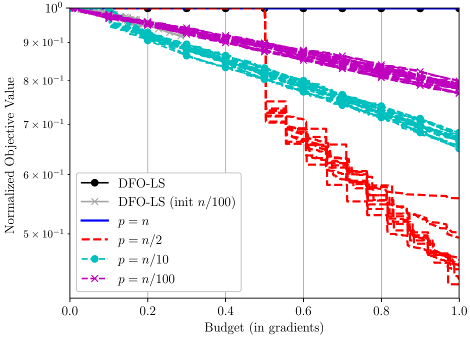
... because it doesn't time out

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 & Future Work

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
- DFBGN 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, Github: `numerical algorithms group/dfbgn`]

References i

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.

References ii

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.

References iii

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.

References iv

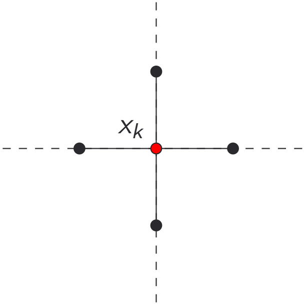
T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, *Evolution strategies as a scalable alternative to reinforcement learning*, arXiv preprint arXiv:1703.03864, (2017).

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).

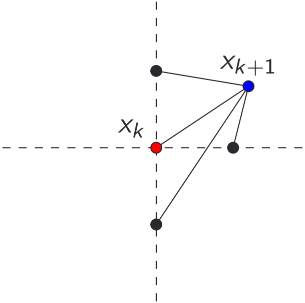
S. J. Wright, *Coordinate descent algorithms*, Mathematical Programming, 151 (2015), pp. 3–34.

General Objective Problems

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



2 points per subspace direction



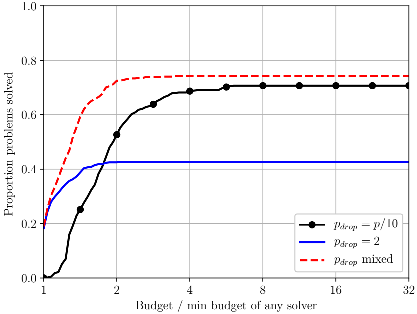
After step, how to rotate subspace?

Subspace dimensions decoupled from interpolation directions $\boldsymbol{y}_t - \boldsymbol{x}_k$

Choice of p_{drop}

How to choose p_{drop}?

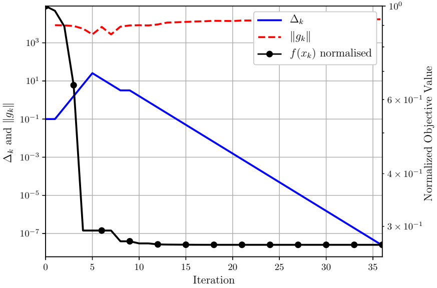
- Large changes to Q_k (e.g. $p_{drop} = p/10$) — explore whole space quickly
- Small changes to Q_k (e.g. $p_{drop} = 2$) — use few evaluations
- Compromise? ($p_{drop} = 2$ on successful iterations, $p/10$ on unsuccessful iterations)



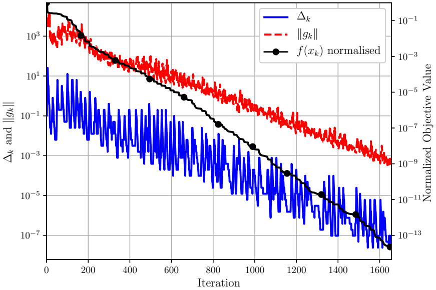
% problems solved vs. # objective evaluations (normalized)

Choice of p_{drop}

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



$\boldsymbol{p_{drop}} = 2$



$\boldsymbol{p_{drop}}$ **mixed**

(CUTEst problem LUKSAN13 with $n = 100$)