

# Improving the scalability of model-based derivative-free optimization

*With Coralia Cartis (Oxford), Jan Fiala & Benjamin Marteau (NAG)*

---

Lindon Roberts, Mathematical Sciences Institute, ANU ([lindon.roberts@anu.edu.au](mailto:lindon.roberts@anu.edu.au))

Data Science Down Under (University of Newcastle)

11 December 2019

*Supported by EPSRC (EP/L015803/1) & NAG Ltd.*

1. Derivative-free optimization for least-squares problems
2. Scalability bottleneck
3. Model-based subspace method
4. Results
5. General objective case

# Derivative-Free Optimization

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

- Objective  $f$  nonlinear, nonconvex, structure unknown

# Derivative-Free Optimization

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

- Objective  $f$  nonlinear, nonconvex, structure unknown
- Standard methods locally approximate  $f$  by quadratic models (e.g. Taylor series)
- How to calculate derivatives of  $f$  to build model?
  - Write code by hand
  - Finite differences
  - Algorithmic differentiation (backprop)

# Derivative-Free Optimization

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

- Objective  $f$  nonlinear, nonconvex, structure unknown
- Standard methods locally approximate  $f$  by quadratic models (e.g. Taylor series)
- How to calculate derivatives of  $f$  to build model?
  - Write code by hand
  - Finite differences
  - Algorithmic differentiation (backprop)
- Difficulties when function evaluation is
  - ‘Black-box’
  - Noisy
  - Computationally expensive

# Derivative-Free Optimization

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

- Objective  $f$  nonlinear, nonconvex, structure unknown
- Standard methods locally approximate  $f$  by quadratic models (e.g. Taylor series)
- How to calculate derivatives of  $f$  to build model?
  - Write code by hand
  - Finite differences
  - Algorithmic differentiation (backprop)
- Difficulties when function evaluation is
  - ‘Black-box’
  - Noisy
  - Computationally expensive
- Alternative — **derivative-free optimization (DFO)** [aka “zero-order methods”]
  - Hyperparameter tuning, adversarial example generation, ... [Liu & Chen (2019)]
  - Plus applications in finance, climate, engineering, ...

## Model-Based DFO — Basic Ideas

Many approaches: [model-based](#), gradient sampling, Nelder-Mead, direct search, genetic algorithms, Bayesian optimization, ...

- Classically (e.g. Newton's method),

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

## Model-Based DFO — Basic Ideas

Many approaches: [model-based](#), gradient sampling, Nelder-Mead, direct search, genetic algorithms, Bayesian optimization, ...

- Classically (e.g. Newton's method),

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

- Instead, approximate

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



## Model-Based DFO — Basic Ideas

Many approaches: [model-based](#), gradient sampling, Nelder-Mead, direct search, genetic algorithms, Bayesian optimization, ...

- Classically (e.g. Newton's method),

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

- Instead, approximate

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

- Find  $\mathbf{g}_k$  and  $H_k$  without using derivatives: [interpolate](#)  $f$  over a set of points
- Geometry of points good  $\implies$  interpolation model accurate  $\implies$  convergence

*[Conn, Powell, Scheinberg, Vicente, ...]*

# 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

- 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

# 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  $\mathbf{J}_k$  by **interpolation** — maintain a cloud of points which moves towards solution (**with good geometry**)

# 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  $\mathbf{J}_k$  by **interpolation** — maintain a cloud of points which moves towards solution (**with good geometry**)

In both cases, get a local quadratic model (with approximate Hessian)

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

# DFO for Least-Squares — Algorithm

## Implement in trust-region method:

1. Build interpolation model

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

2. Minimize model inside trust region

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

3. Evaluate  $f(\mathbf{x}_k + \mathbf{s}_k)$ , check sufficient decrease, select  $\mathbf{x}_{k+1}$  and  $\Delta_{k+1}$
4. **Update interpolation set**: add  $\mathbf{x}_k + \mathbf{s}_k$  and move points to ensure good geometry (if needed)  
← requires calculation of Lagrange polynomials

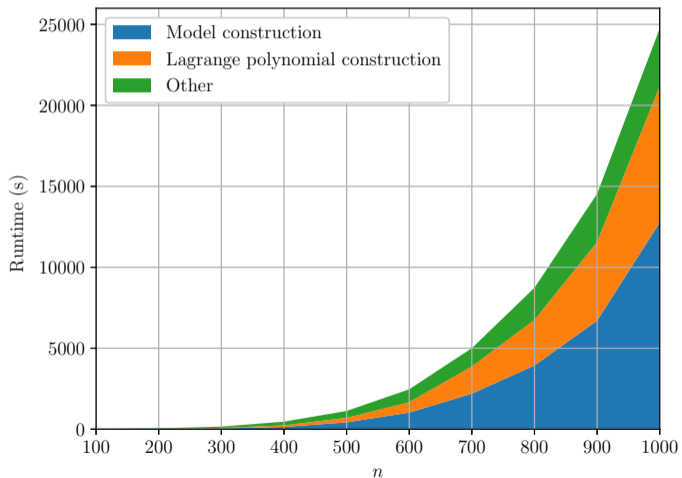
Implemented in *DFO-LS* package (*Github*: [numerical algorithms group/dfols](#))

Also have *Py-BOBYQA* for general objectives (*Github*), using quadratic interpolation

- DFO methods are well-known not to scale well (i.e.  $n$  large)
  - e.g. adversarial examples, weather forecasting/data assimilation, ...

**Where is the issue for model-based DFO?**

Runtime of DFO-LS on generalized Rosenbrock function:





# Interpolation

Interpolation linear system (for model construction):

$$\begin{bmatrix} (\mathbf{y}_1 - \mathbf{x}_k)^\top \\ \vdots \\ (\mathbf{y}_n - \mathbf{x}_k)^\top \end{bmatrix} \mathbf{g}_{k,i} = \begin{bmatrix} r_i(\mathbf{y}_1) - r_i(\mathbf{x}_k) \\ \vdots \\ r_i(\mathbf{y}_n) - r_i(\mathbf{x}_k) \end{bmatrix}, \quad \forall i = 1, \dots, m,$$

where  $J_k$  has rows  $\mathbf{g}_{k,i}^\top$ .

$$\text{Cost} = \text{factorization} + \text{solve} = \mathcal{O}(n^3) + \mathcal{O}(mn^2) \approx \mathcal{O}(mn^2)$$

# Interpolation

Interpolation linear system (for model construction):

$$\begin{bmatrix} (\mathbf{y}_1 - \mathbf{x}_k)^\top \\ \vdots \\ (\mathbf{y}_n - \mathbf{x}_k)^\top \end{bmatrix} \mathbf{g}_{k,i} = \begin{bmatrix} r_i(\mathbf{y}_1) - r_i(\mathbf{x}_k) \\ \vdots \\ r_i(\mathbf{y}_n) - r_i(\mathbf{x}_k) \end{bmatrix}, \quad \forall i = 1, \dots, m,$$

where  $J_k$  has rows  $\mathbf{g}_{k,i}^\top$ .

$$\text{Cost} = \text{factorization} + \text{solve} = \mathcal{O}(n^3) + \mathcal{O}(mn^2) \approx \mathcal{O}(mn^2)$$

## Goal

Can we construct a method with reduced interpolation cost, but still efficient in # evaluations required?

## Goal

Can we construct a method with reduced interpolation cost, but still efficient in # evaluations required?

## Key idea: dimensionality reduction in $n$ — existing approaches

- Block Coordinate Descent: perturb subset of variables each iteration  
[Xu & Yin (2017), Richtárik & Takáč (2014)]
- Block Coordinate Gauss-Newton: generalize BCD to least-squares  
[Cartis & Fowkes (2018)]
- Probabilistic direct search: random search direction at each iteration  
[Gratton, Royer, Vicente & Zhang (2015)]
- Projection DFO methods: optimize over random subspace with existing method  
[Qian, Hu & Yu (2016), Wang, Du, Balakrishnan & Singh (2018)]

# Subspace Methods

## Goal

Can we construct a method with reduced interpolation cost, but still efficient in # evaluations required?

**Key idea: dimensionality reduction in  $n$**

# Subspace Methods

## Goal

Can we construct a method with reduced interpolation cost, but still efficient in # evaluations required?

**Key idea: dimensionality reduction in  $n$**

Use interpolation set  $\{\mathbf{x}_k, \mathbf{y}_1, \dots, \mathbf{y}_p\}$  for  $p < n$ , then solve

$$\begin{bmatrix} (\mathbf{y}_1 - \mathbf{x}_k)^\top \\ \vdots \\ (\mathbf{y}_p - \mathbf{x}_k)^\top \end{bmatrix} \mathbf{g}_{k,i} = \begin{bmatrix} r_i(\mathbf{y}_1) - r_i(\mathbf{x}_k) \\ \vdots \\ r_i(\mathbf{y}_p) - r_i(\mathbf{x}_k) \end{bmatrix}, \quad \forall i = 1, \dots, m.$$

Underdetermined system  $\implies$  take minimal norm solution.

$$\text{Cost} = \text{factorization} + \text{solve} = \mathcal{O}(np^2) + \mathcal{O}(mp^2) \approx \mathcal{O}(mp^2)$$

Choose  $p$  based on computational resources

# Subspace Methods

- Model only varies in subspace  $\mathcal{Y}_k := \text{span}\{\mathbf{y}_1 - \mathbf{x}_k, \dots, \mathbf{y}_p - \mathbf{x}_k\}$ .

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

where  $\mathbf{Q}_k \in \mathbb{R}^{n \times p}$  is orthonormal basis for  $\mathcal{Y}_k$  (from QR factorization).

- Solve trust-region subproblem in subspace

$$\mathbf{s}_k = \mathbf{Q}_k \hat{\mathbf{s}}_k, \quad \text{where} \quad \hat{\mathbf{s}}_k = \arg \min_{\|\hat{\mathbf{s}}\|_2 \leq \Delta_k} \hat{\mathbf{m}}_k(\hat{\mathbf{s}}) = \frac{1}{2} \|\hat{\mathbf{m}}_k(\hat{\mathbf{s}})\|_2^2,$$

- Need a mechanism to explore whole space:

# Subspace Methods

- Model only varies in subspace  $\mathcal{Y}_k := \text{span}\{\mathbf{y}_1 - \mathbf{x}_k, \dots, \mathbf{y}_p - \mathbf{x}_k\}$ .

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

where  $\mathbf{Q}_k \in \mathbb{R}^{n \times p}$  is orthonormal basis for  $\mathcal{Y}_k$  (from QR factorization).

- Solve trust-region subproblem in subspace

$$\mathbf{s}_k = \mathbf{Q}_k \hat{\mathbf{s}}_k, \quad \text{where} \quad \hat{\mathbf{s}}_k = \arg \min_{\|\hat{\mathbf{s}}\|_2 \leq \Delta_k} \hat{\mathbf{m}}_k(\hat{\mathbf{s}}) = \frac{1}{2} \|\hat{\mathbf{m}}_k(\hat{\mathbf{s}})\|_2^2,$$

- Need a mechanism to explore whole space:
  - i.e. need to **change  $\mathcal{Y}_k$  on each iteration**
  - Replace interpolation points with **random directions** (orthogonal to  $\mathcal{Y}_k$ )
  - **No free lunch**: extra evaluations used to change  $\mathcal{Y}_k$  to save on linear algebra

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

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



## Algorithm DFBN (Derivative-Free Block Gauss-Newton):

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

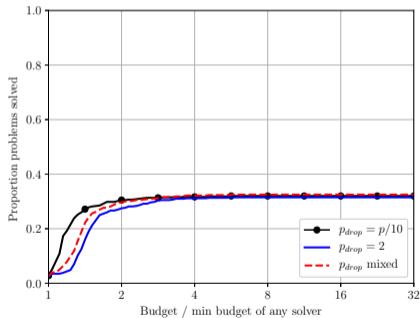
## Comments:

- $p_{drop} \geq 2$  ensures new direction(s)  $\mathbf{d}$  added next iteration  $\implies \mathcal{Y}_{k+1} \neq \mathcal{Y}_k$ .
- Linear algebra cost  $\mathcal{O}(mp^2 + np^2 + p^3)$  vs. full space method  $\mathcal{O}(mn^2 + n^3)$
- Choosing points to remove uses Lagrange polynomials (geometry-aware)

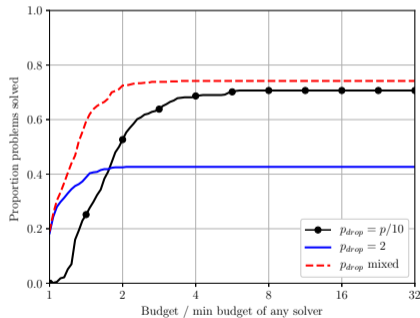
# Choice of $p_{drop}$

## How to choose $p_{drop}$ ?

- Large change to  $\mathcal{Y}_k$  each iteration (e.g.  $p_{drop} = p/10$ ) — explore whole space quickly
- Small change to  $\mathcal{Y}_k$  each iteration (e.g.  $p_{drop} = 2$ ) — use few evaluations
- Compromise? ( $p_{drop} = 2$  on successful iterations,  $p/10$  on unsuccessful iterations)



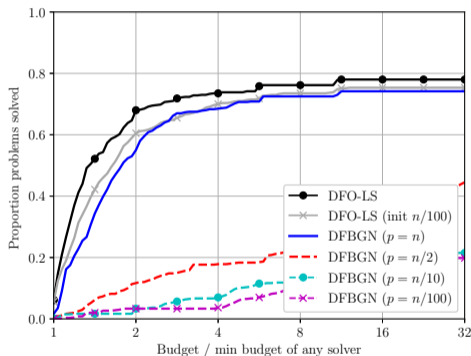
**DFBGN,  $p = n/4$**



**DFBGN,  $p = n$**

# Numerical Results — high accuracy

Compare DFBGN method to DFO-LS (high accuracy  $\tau = 10^{-5}$ )

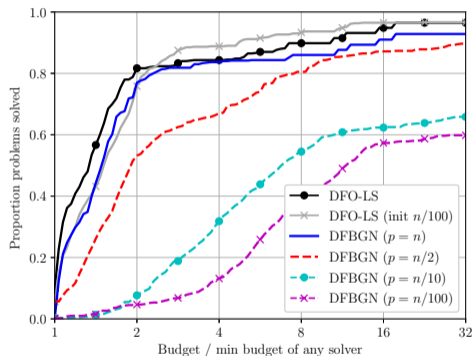


$n \approx 100$  [CUTEst]

Performance improves with increasing block size

## Numerical Results — low accuracy

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

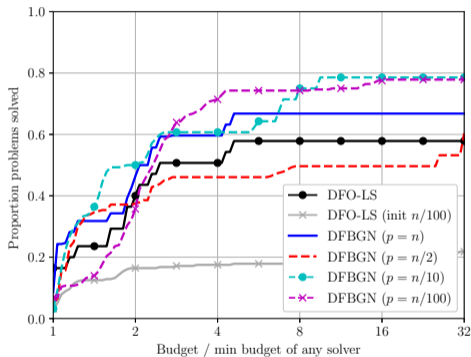


$n \approx 100$  [CUTEst]

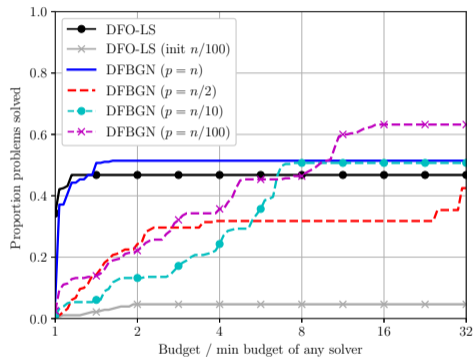
DFBGN is more suitable for low accuracy solutions

# Numerical Results — high dimensional problems

High-dimensional test set  $n \approx 1000$  [CUTEst], max 12hrs per problem



$\tau = 0.5$ , vs. budget



$\tau = 0.1$ , vs. budget

DFBGN outperforms DFO-LS for low accuracy solutions ...

# Timeout Rate

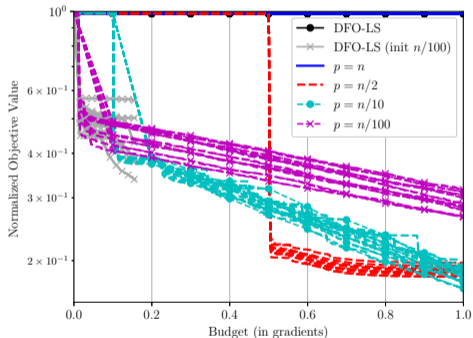
Proportion of problems where solver times out (before usual termination):

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

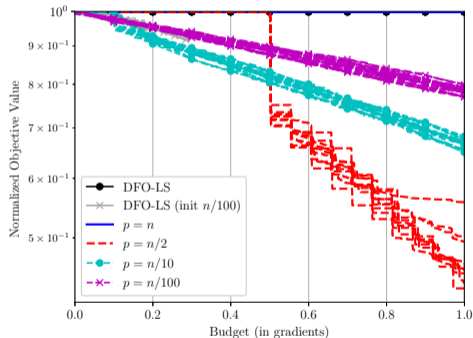
# Numerical Results — low budget

Other advantage: DFBGN make progress after  $p \ll n$  evaluations (especially important when  $n$  large)



**ARWHDNE,  $n = 2000$**

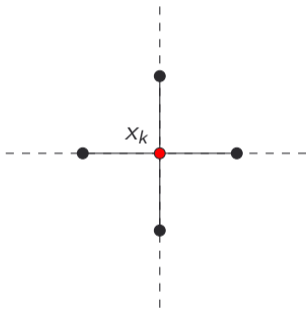
*(normalized objective reduction vs. # evaluations, 12hr timeout)*



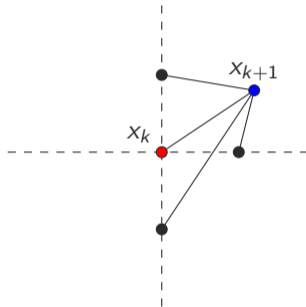
**CHANDHEQ,  $n = 2000$**

# General Objective Problems

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



*2 points per subspace direction*



*After step, how to rotate subspace?*

- **Subspace dimensions decoupled from interpolation directions**  $\mathbf{y}_t - \mathbf{x}_k$
- **Current idea:** linear model ( $p + 1$  points) + extra curvature-only points (not recycled)






# Conclusion & Future Work

## Conclusions

- Model construction cost a key barrier to scalability of model-based DFO
- Subspace method gives cheaper linear algebra:  $\mathcal{O}(n)$  vs.  $\mathcal{O}(n^3)$
- Useful for: low accuracy, small budget, and/or limited computational resources
- Extending to large-scale general objective problems

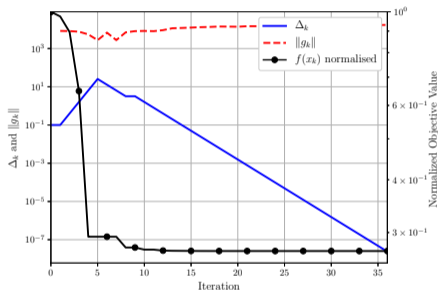
## Future Work

- Sketching for nonlinear least-squares
- Convergence theory
- Exploit Jacobian sparsity

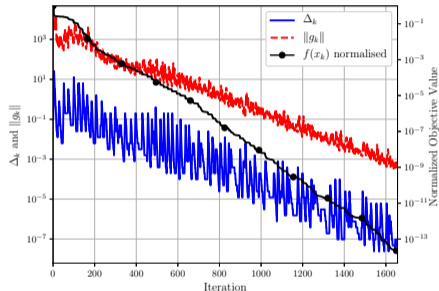
-  Coralia Cartis, Jan Fiala, Benjamin Marteau, and Lindon Roberts.  
**Improving the flexibility and robustness of model-based derivative-free optimization solvers.**  
*ACM Transactions on Mathematical Software*, 45(3):32:1–32:41, 2019.
-  Coralia Cartis and Lindon Roberts.  
**A derivative-free Gauss-Newton method.**  
*Mathematical Programming Computation*, 11(4):631–674, 2019.
-  Lindon Roberts.  
**Derivative-Free Algorithms for Nonlinear Optimisation Problems.**  
PhD thesis, University of Oxford, 2019.

# Choice of $p_{drop}$

Choice of  $p_{drop}$  prevents  $\Delta_k$  too small too soon (need for convergence)



$p_{drop} = 2$



$p_{drop}$  mixed

(CUTEst problem LUKSAN13 with  $n = 100$ )