Block Methods for Scalable Derivative-Free Optimisation

With Coralia Cartis (Oxford)

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

64th Annual Meeting of the Australian Mathematical Society (University of New England)
9 December 2020

Outline

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

Derivative-Free Optimization

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

- Objective f nonlinear, nonconvex, structure unknown

Derivative-Free Optimization

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

- 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 [aka backpropagation]

Derivative-Free Optimization

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

- 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 [aka backpropagation]
- Difficulties when function evaluation is
  - 'Black-box'
  - Noisy
  - Computationally expensive

Derivative-Free Optimization

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

- 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 [aka backpropagation]
- Difficulties when function evaluation is
  - 'Black-box'
  - Noisy
  - Computationally expensive
- Alternative — derivative-free optimization (DFO) [aka "zero-order methods"]
  - Applications in finance, climate, image analysis, data science, engineering, ...

Model-Based DFO — Basic Ideas

Many approaches: model-based, direct search, Nelder-Mead, ...

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

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

Model-Based DFO — Basic Ideas

Many approaches: model-based, direct search, Nelder-Mead, ...

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

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

- Instead, approximate

$$f(x_k + s) \approx m_k(s) = f(x_k) + {g_k}^T s + \frac{1}{2} s^T H_k s$$

Model-Based DFO — Basic Ideas

Many approaches: model-based, direct search, Nelder-Mead, ...

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

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

- Instead, approximate

$$f(x_k + s) \approx m_k(s) = f(x_k) + g_k{}^T s + \frac{1}{2} s^T H_k s$$

- Find 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

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

Classical Gauss-Newton

Derivative-Free Gauss-Newton

DFO for Least-Squares — Basic Framework

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

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

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

$$r(x_k + s) \approx M_k(s) = r(x_k) + J(x_k)s$$

DFO for Least-Squares — Basic Framework

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

Classical Gauss-Newton

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

$$r(x_k + s) \approx M_k(s) = r(x_k) + J(x_k)s$$

Derivative-Free Gauss-Newton

- Jacobian not available: use

$$M_k(s) = r(x_k) + J_k s$$

- Find J_k by interpolation — maintain a cloud of points which moves towards solution (with good geometry)

DFO for Least-Squares — Basic Framework

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

Classical Gauss-Newton

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

$$r(x_k + s) \approx M_k(s) = r(x_k) + J(x_k)s$$

Derivative-Free Gauss-Newton

- Jacobian not available: use

$$M_k(s) = r(x_k) + J_k s$$

- Find 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)

$$f(x_k + s) \approx m_k(s) = \frac{1}{2}\|M_k(s)\|_2^2$$

DFO for Least-Squares — Algorithm

Implement in trust-region method:

1. Build interpolation model

$$f(x_k + s) \approx m_k(s) := \frac{1}{2}\|M_k(s)\|_2^2.$$

2. Minimize model inside trust region

$$s_k = \underset{s \in \mathbb{R}^n}{\arg\min}\, m_k(s) \quad \text{s.t.} \quad \|s\|_2 \leq \Delta_k.$$

3. Evaluate $f(x_k + s_k)$, check sufficient decrease, select x_{k+1} and Δ_{k+1}
4. Update interpolation set: add $x_k + s_k$ and move points to ensure good geometry (if needed)
   $\leftarrow$ requires calculation of Lagrange polynomials

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

(Also have software for general objectives using quadratic interpolation)

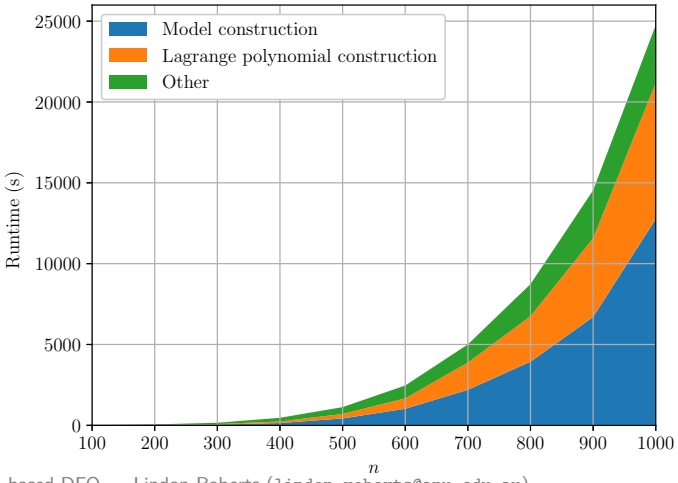
Scalability

- 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?

Scalability

Runtime of DFO-LS on generalized Rosenbrock function:



Interpolation

Interpolation linear system (for model construction):

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

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

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

Interpolation

Interpolation linear system (for model construction):

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

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

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

Goal

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

Subspace Methods

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 $\{\mathbf{x}_k, \mathbf{y}_1, \ldots, \mathbf{y}_p\}$ for $p < n$, then solve

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

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 $\mathcal{Y}_k := \text{span}\{\mathbf{y}_1 - \mathbf{x}_k, \ldots, \mathbf{y}_p - \mathbf{x}_k\}$.

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

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

- Solve trust-region subproblem in subspace

$$\mathbf{s}_k = Q_k\hat{\mathbf{s}}_k, \qquad \text{where} \qquad \hat{\mathbf{s}}_k = \underset{\|\hat{\mathbf{s}}\|_2 \leq \Delta_k}{\arg\min}\, \hat{m}_k(\hat{\mathbf{s}}) = \frac{1}{2}\|\hat{\mathbf{m}}_k(\hat{\mathbf{s}})\|_2^2,$$

- Need a mechanism to explore whole space:

Subspace Methods

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

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

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

- Solve trust-region subproblem in subspace

$$\mathbf{s}_k = Q_k\hat{\mathbf{s}}_k, \qquad \text{where} \qquad \hat{\mathbf{s}}_k = \underset{\|\hat{\mathbf{s}}\|_2 \le \Delta_k}{\arg\min}\, \hat{m}_k(\hat{\mathbf{s}}) = \frac{1}{2}\|\hat{\mathbf{m}}_k(\hat{\mathbf{s}})\|_2^2,$$

- 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

Changing $\mathcal{Y}_k$

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 Δ_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

Changing $\mathcal{Y}_k$

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 Δ_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

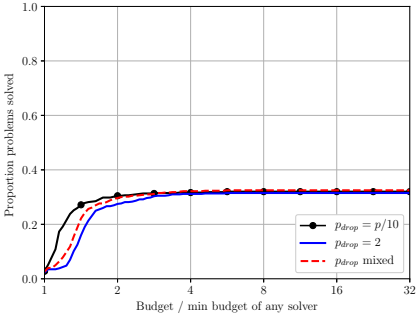
Comments:

- $p_{drop} \geq 2$ 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)

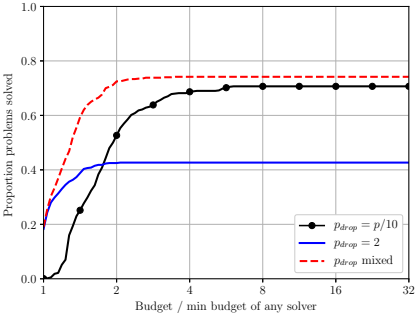
Choice of p_{drop}

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)



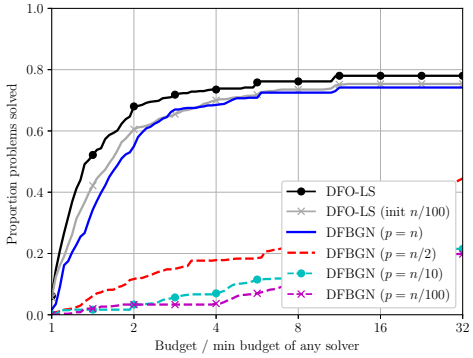
DFBGN, $p = n/4$



DFBGN, $p = n$

Numerical Results — high accuracy

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

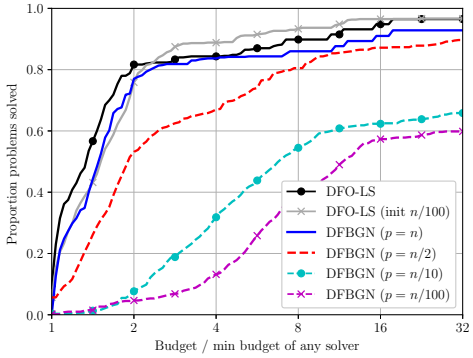


$n \approx 100$ [CUTEst]

Performance improves with increasing block size

Numerical Results — low accuracy

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

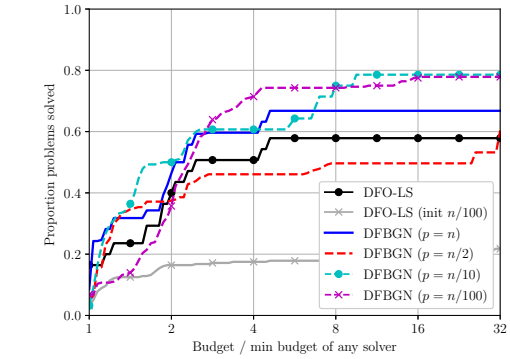


$n \approx 100$ [CUTEst]

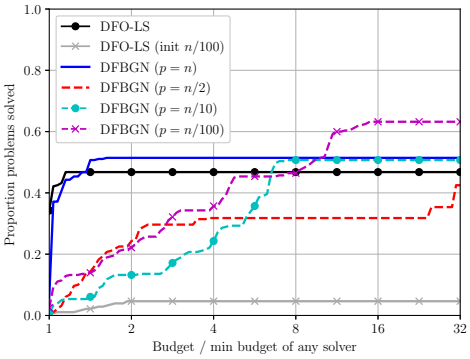
DFBGN is more suitable for low accuracy solutions

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

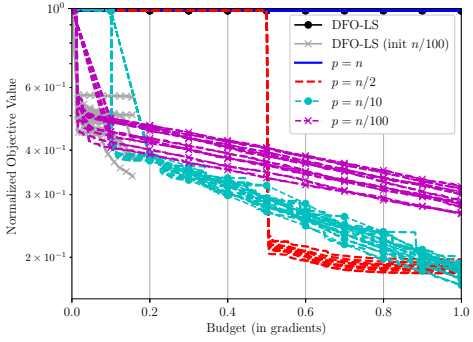
Proportion of problems where solver times out (before usual termination):

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%

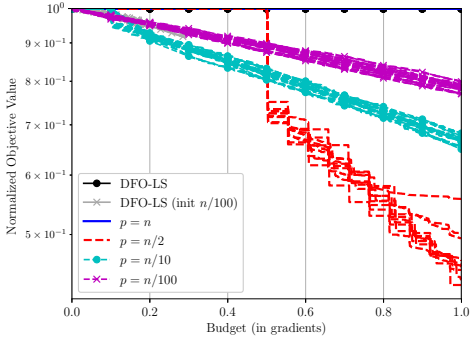
... because it doesn't time out

Numerical Results — low budget

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



ARWHDNE, $n = 2000$



CHANDHEQ, $n = 2000$

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

Convergence Guarantees

Convergence guarantees are available for a generic block method:

- Randomly select Q_k independently at each iteration
- With probability $1 - \delta$, we have $\|Q_k^T \nabla f(\mathbf{x}_k)\| \geq \alpha \|\nabla f(\mathbf{x}_k)\|$, some $\alpha \in (0, 1)$
- Do not need to assume least-squares structure

Theorem

For k sufficiently large,

$$\mathbb{P}\left[\min_{j=0,\ldots,k} \|\nabla f(\mathbf{x}_j)\| \leq \frac{C}{\sqrt{k}}\right] \geq 1 - e^{-ck},$$

for some $c, C > 0$.

Matches standard $\mathcal{O}(k^{-1/2})$ convergence rate for nonconvex problems with high probability.

(Proof another time)

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

Future Work

- Sketching (dimensionality reduction in # residuals; WoMBaT talk)
- Extend to general objective problems (quadratic interpolation models)

References

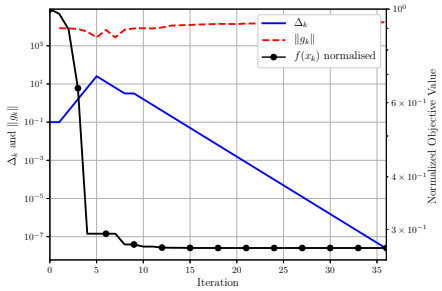

Coralia Cartis, Jan Fiala, Benjamin Marteau, and Lindon Roberts.
Improving the flexibility and robustness of model-based derivative-free optimization solvers.
ACM Transactions on Mathematical Software, 45(3):32:1–32:41, 2019.


Coralia Cartis and Lindon Roberts.
A derivative-free Gauss-Newton method.
Mathematical Programming Computation, 11(4):631–674, 2019.

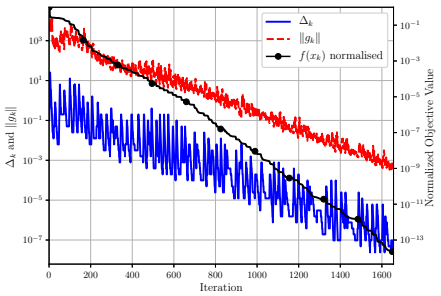

Coralia Cartis and Lindon Roberts.
Scalable block methods for derivative-free nonlinear least-squares optimization.
in preparation, 2020.

Choice of p_{drop}

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



$p_{drop} = 2$



p_{drop} **mixed**

(CUTEst problem LUKSAN13 with $n = 100$)