University of Colorado **Boulder**

0th, ~~1st~~ and 2nd order optimization methods for learning problems

Stephen Becker, University of Colorado Boulder

Collaborators: Alireza Doostan, Nuojin Chen, Killian Wood, Drona Khurana (CU)
Cooper Simpson (U Washington)
David Kozak, Luis Tenorio (Colorado School of Mines)

Workshop: Optimization and learning: theory and applications
Centre de recherches mathématiques (CRM)
Montréal, May 27-30, 2025

Outline

Most machine learning (ML) algorithms use 1st order optimization like (stochastic) gradient descent

That's **generally** a good idea! But in some cases, 0th and 2nd order methods make sense

Outline

Most machine learning (ML) algorithms use 1st order optimization like (stochastic) gradient descent

That's **generally** a good idea! But in some cases, 0th and 2nd order methods make sense

Learning objectives of this talk

- 0th order optimization / "derivative-free optimization"
  - Introduce a class of 0th order optimization methods
  - Argue that **stepsize** selection is a key issue
  - Show some ML **examples** where these methods make sense

Outline

Most machine learning (ML) algorithms use 1st order optimization like (stochastic) gradient descent

That's **generally** a good idea! But in some cases, 0th and 2nd order methods make sense

Learning objectives of this talk

- 0th order optimization / "derivative-free optimization"
  - Introduce a class of 0th order optimization methods
  - Argue that **stepsize** selection is a key issue
  - Show some ML **examples** where these methods make sense

- 2nd order optimization
  - Introduce a variant of Newton's method
  - Demonstrate why non-convexity has to be taken more seriously
  - Argue that **linear algebra** is a key issue

Part 0: 0th order methods / derivative-free optimization

$\min_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x})$ where we do not have access to $\nabla f(\boldsymbol{x})$

Traditional applications: PDE constrained optimization, when the **adjoint state method** or **automatic differentiation** is inapplicable
- e.g., multiphysics codes with complicated adjoints (and hard to code in HPC); memory issues in AD for time-dependent problems, etc.

Part 0: 0th order methods / derivative-free optimization

$\min_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x})$ where we do not have access to $\nabla f(\boldsymbol{x})$

Traditional applications: PDE constrained optimization, when the **adjoint state method** or **automatic differentiation** is inapplicable

- e.g., multiphysics codes with complicated adjoints (and hard to code in HPC); memory issues in AD for time-dependent problems, etc.

Machine learning applications

- hyper parameter tuning
- black-box attacks, e.g., adversarial attacks on a model (attacker doesn't have access to source code)

Part 0: 0th order methods / derivative-free optimization

$$\min_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x}) \text{ where we do not have access to } \nabla f(\boldsymbol{x})$$

Traditional applications: PDE constrained optimization, when the **adjoint state method** or **automatic differentiation** is inapplicable

- e.g., multiphysics codes with complicated adjoints (and hard to code in HPC); memory issues in AD for time-dependent problems, etc.

Machine learning applications

- hyper parameter tuning
- black-box attacks, e.g., adversarial attacks on a model (attacker doesn't have access to source code)

Recent theme of my work: exploit **multi-fidelity** models

- in applied math, for PDE simulations, we often have several physics based models (with different approximations), different discretizations, different numerical precisions, reduced-order models, etc

.

- **machine learning** also has examples of multi-fidelity models

Context

High dimensions, low accuracy

Low dimensions, high accuracy

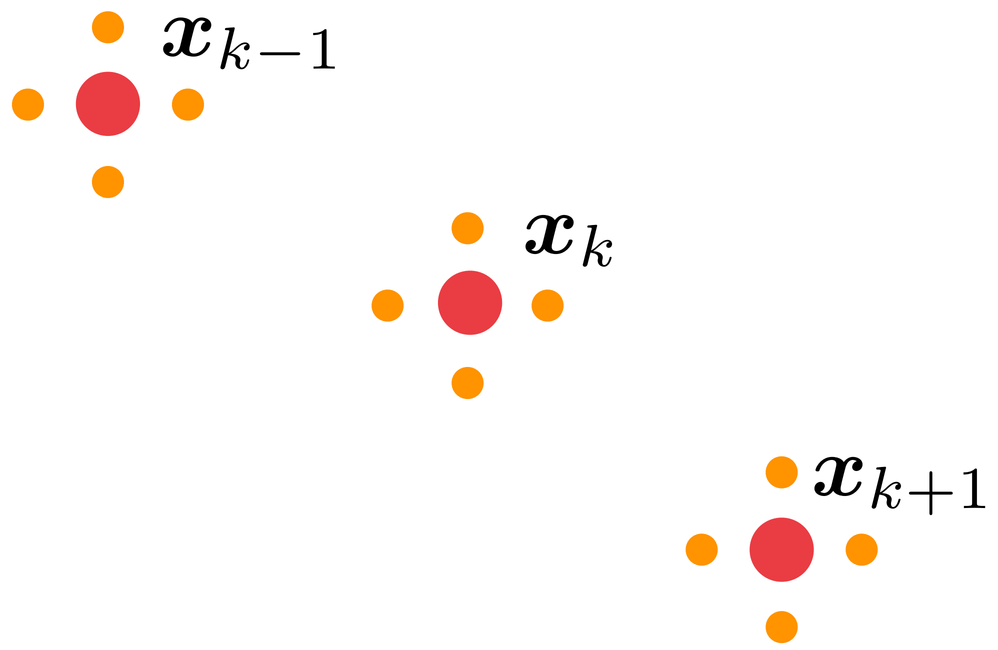
Local methods

"zeroth order": approximate $\nabla f(\boldsymbol{x})$ (e.g., w/ finite differences)

stochastic zeroth order: approximate $\boldsymbol{g}$ such that $\mathbb{E}[\boldsymbol{g}] = \nabla f(\boldsymbol{x})$

today's talk

polling methods: Nelder-Mead, coordinate descent, etc.

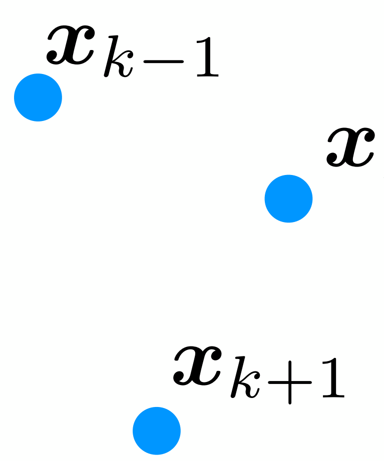


Global / model-based methods

polynomial model, minimize with a trust-region ("DFO-TR")

see our new paper "A Unified Framework for Entropy Search and Expected Improvement in Bayesian Optimization", N. Cheng et al. ICML '25

Gaussian process model, use acquisition function to tradeoff exploration and exploitation (Bayesian Optimization")



A guiding principle:

the time spent in the **optimization method** (creating and solving surrogate models)
should equal
the time spent in the **function evaluation** (e.g., solving the PDE, training neural net...)

... hence, the best method to use depends a lot on the problem.

Misc. / heuristics

genetic algorithms, particle swarm, CMA-ES, simulated annealing, etc.

A stochastic zeroth-order method: SSD

Assume we can compute/approximate $\boldsymbol{q}\boldsymbol{q}^\top \nabla f(\boldsymbol{x}) = \left(\lim_{h \to 0} \frac{f(\boldsymbol{x} + h\boldsymbol{q}) - f(\boldsymbol{x})}{h} \right) \boldsymbol{q}$ e.g., finite differences, or *forward-mode* AutoDiff

directional derivative, “two-point” estimator in ZO literature

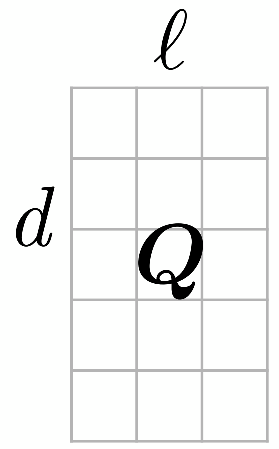
$$\boldsymbol{q}^\top \nabla f(\boldsymbol{x}) = \frac{d}{dt}\varphi(t)\Big|_{t=0} \quad \text{where} \quad \varphi(t) = f(\boldsymbol{x} + t\boldsymbol{q})$$

A stochastic zeroth-order method: SSD

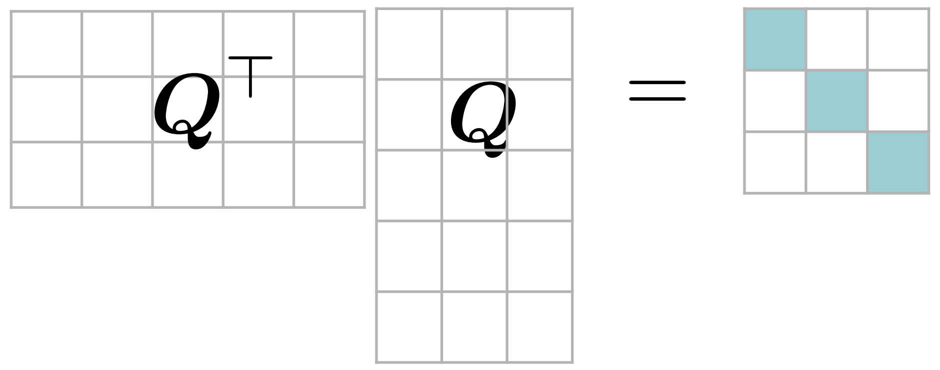
Assume we can compute/approximate $\boldsymbol{q}\boldsymbol{q}^\top \nabla f(\boldsymbol{x}) = \left(\lim_{h \to 0} \frac{f(\boldsymbol{x} + h\boldsymbol{q}) - f(\boldsymbol{x})}{h} \right) \boldsymbol{q}$

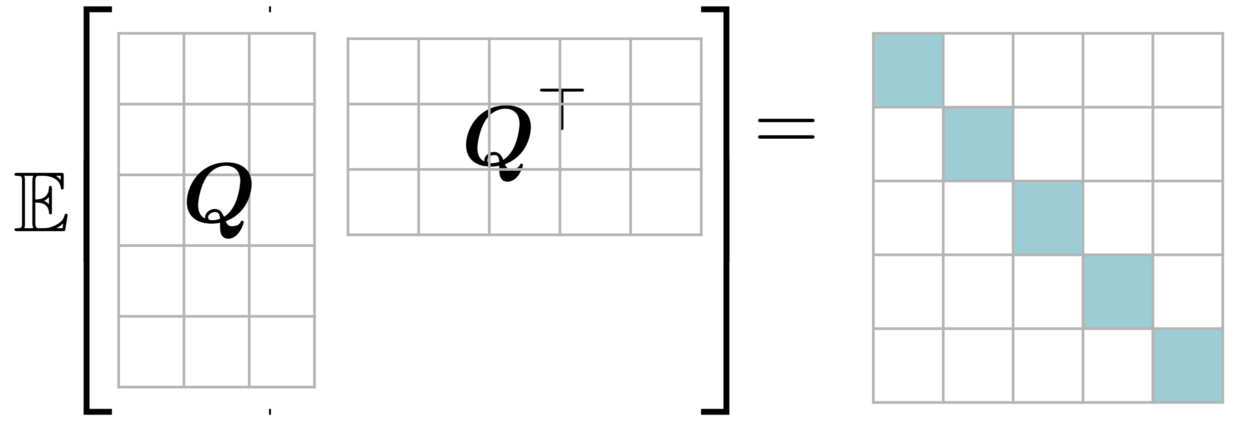
Even better, average a few copies to reduce variance

$$\boldsymbol{Q}\boldsymbol{Q}^\top \nabla f(\boldsymbol{x}) = \sum_{i=1}^{\ell} \boldsymbol{q}_i \boldsymbol{q}_i^\top \nabla f(\boldsymbol{x}) \qquad \boldsymbol{Q} = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_\ell] \qquad \boldsymbol{Q}^\top \boldsymbol{Q} = \boldsymbol{I}_{\ell \times \ell}, \quad \mathbb{E}\left(\frac{d}{\ell} \boldsymbol{Q}\boldsymbol{Q}^\top \right) = \boldsymbol{I}_{d \times d}$$



columns **not** independent!



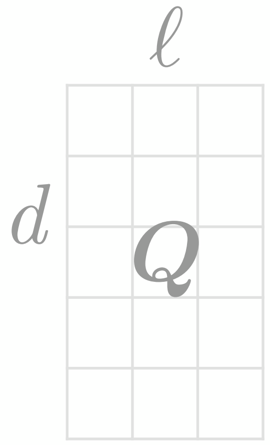


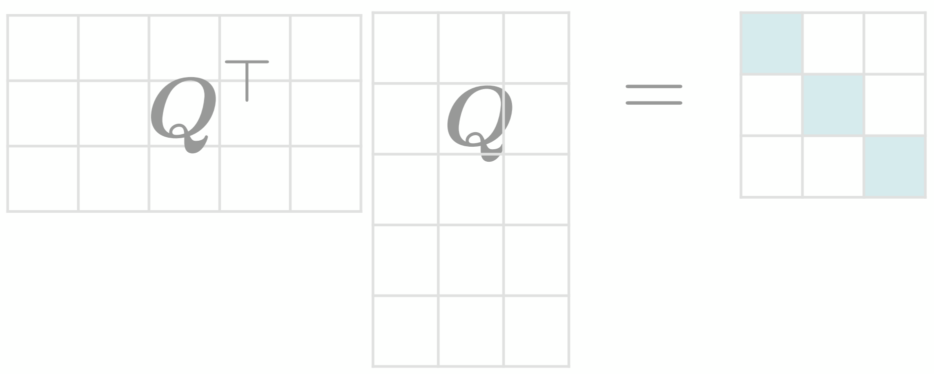
A stochastic zeroth-order method: SSD

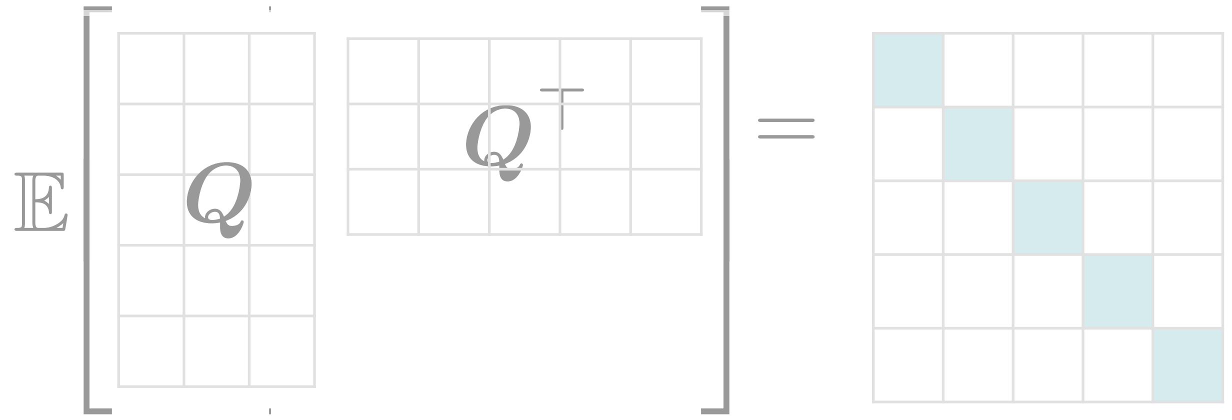
Assume we can compute/approximate $\boldsymbol{q}\boldsymbol{q}^\top \nabla f(\boldsymbol{x}) = \left(\lim_{h \to 0} \frac{f(\boldsymbol{x} + h\boldsymbol{q}) - f(\boldsymbol{x})}{h} \right) \boldsymbol{q}$

Even better, average a few copies to reduce variance

$$\boldsymbol{Q}\boldsymbol{Q}^\top \nabla f(\boldsymbol{x}) = \sum_{i=1}^{\ell} \boldsymbol{q}_i \boldsymbol{q}_i^\top \nabla f(\boldsymbol{x}) \qquad \boldsymbol{Q} = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_\ell] \qquad \boldsymbol{Q}^\top \boldsymbol{Q} = \boldsymbol{I}_{\ell \times \ell}, \quad \mathbb{E}\left(\frac{d}{\ell} \boldsymbol{Q}\boldsymbol{Q}^\top \right) = \boldsymbol{I}_{d \times d}$$







Algorithm: stochastic subspace descent (SSD)

Repeat:

Draw a $\boldsymbol{Q}$

$\boldsymbol{x} \leftarrow \boldsymbol{x} - \eta \frac{d}{\ell} \boldsymbol{Q}\boldsymbol{Q}^\top \nabla f(\boldsymbol{X})$

$\mathcal{O}(\ell)$ oracle calls

stepsize

Important: draw new (independent) $\boldsymbol{Q}$ every iteration

It's a type of "stochastic gradient method" (direction is unbiased) but has much stronger guarantees due to its structure

e.g., the direction is **descent direction** with prob. 1 if $\boldsymbol{Q}$ is continuous

A stochastic zeroth-order method: SSD

Assume we can compute/approximate $qq^\top \nabla f(x) = \left(\lim_{h \to 0} \frac{f(x + hq) - f(x)}{h} \right) q$

e.g., finite differences, or *forward-mode* AutoDiff

directional derivative, "two-point" estimator in ZO literature

Even better, average a few copies to reduce variance

$$QQ^\top \nabla f(x) = \sum_{i=1}^{\ell} q_i q_i^\top \nabla f(x) \qquad Q = [q_1, \ldots, q_\ell] \qquad Q^\top Q = I_{\ell \times \ell}, \quad \mathbb{E}\left(\frac{d}{\ell} QQ^\top \right) = I_{d \times d}$$

Algorithm: stochastic subspace descent (SSD)

Repeat:

Draw a Q

$x \leftarrow x - \eta \frac{d}{\ell} QQ^\top \nabla f(X)$

$\mathcal{O}(\ell)$ oracle calls

stepsize

Equivalent formulation... and why we call it "SSD"

Columns of Q form a basis for $\mathcal{V}$

Algorithm 1 Stochastic Subspace Descent (SSD)

```
Require: $\eta$                                              ▷ Stepsize
Require: $x_0 \in \mathbb{R}^d$                              ▷ Initial point
for $k = 0, 1, 2, \ldots$ do
    Choose subspace $\mathcal{V}_k$ of dimension $\ell \le d$
    $g_k \leftarrow \mathrm{proj}_{\mathcal{V}_k}(\nabla f(x_k))$   ▷ Project onto subspace
    $x_{k+1} \leftarrow x_k - \eta g_k$
end for
```

SSD: context

Variants have been investigated for a long time...

- "random gradient", "random pursuit", "directional search", "random search"
- ch 6, Yu. Ermoliev and R.J.-B. Wets, *Numerical techniques for stochastic optimization*, Springer-Verlag, **1988**.
- M. Gaviano, *Some general results on convergence of random search algorithms in minimization problems*, Towards Global Optimisation, **1975**.
- F.J. Solis and R. J-B. Wets, *Minimization by random search techniques*, Math. of Operations Research 6 (**1981**), no. 1, 19–30. *(no rate)*
- Matyas **1965**, Polyak **1987**

much recent work on variants, 2011—2020 [and more since then!]

- D. Leventhal and A.S. Lewis, *Randomized Hessian estimation and directional search*, Optimization (2011)
- S. U. Stich, C. Muller, and B. Gartner, *Optimization of convex functions with random pursuit*, SIAM J. Opt. (2013)
- Yu. Nesterov, *Random gradient-free minimization of convex functions*, '11 / Yu. Nesterov and V. Spokoiny, FoCM 2017
- P. Dvurechensky, A. Gasnikov, and A. Tiurin, *Randomized similar triangles method: A unifying framework for accelerated randomized optimization methods (coordinate descent, directional search, derivative-free method)*, arXiv:1707.08486
- P. Dvurechensky, A. Gasnikov, and E. Gorbunov, *An accelerated directional derivative method for smooth stochastic convex optimization*; arXiv:1804.02394
- S. Ghadimi and G. Lan, *Stochastic first- and zeroth-order methods for nonconvex stochastic programming*, SIAM J. Opt. (2013)
- R. Chen and S. Wild, *Randomized derivative-free optimization of noisy convex functions*, arXiv:1507.03332 (2015).
- K. Choromanski, M. Rowland, V. Sindhwani, R. E. Turner, and A. Weller, *Structured evolution with compact architectures for scalable policy optimization*, ICML, 2018.
- T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, *Evolution strategies as a scalable alternative to reinforcement learning*, arXiv:1703.03864 (2017).
- J. Duchi, M. Jordan, M. Wainwright, A. Wibisono, *Optimal Rates for Zero-Order Convex Optimization: The Power of Two Function Evaluations*, IEEE Trans Info Theory (2015)
- A. S. Berahas, L. Cao, K. Choromanski, K. Scheinberg, *A Theoretical and Empirical Comparison of Gradient Approximations in Derivative-Free Optimization*, arXiv 1905.01332 (2019)
- F. Hanzely, K. Mishchenko, P. Richtarik, *SEGA: Variance Reduction via Gradient Sketching*, NeurIPS 2018
- *cousin of "direct search" methods, cf.* S. Gratton, C. W. Royer, L. N. Vicente, Z. Zhang, *Direct Search Based on Probabilistic Descent*, SIAM J. Opt. (2015)

For recent results (2015-2025) on zeroth-order ML, see:

- "Zeroth-order Machine Learning" AAAI tutorial 2024
  - Wotao Yin, Sijia Liu, Pin-Yu Chen
  - https://sites.google.com/view/zo-tutorial-aaai-2024/

SSD: context

Variants have been investigated for a long time...

- “random gradient”, “random pursuit”, “directional search”, “random search”
- ch 6, Yu. Ermoliev and R.J.-B. Wets, *Numerical techniques for stochastic optimization*, Springer-Verlag, **1988**.
- M. Gaviano, *Some general results on convergence of random search algorithms in minimization problems*, Towards Global Optimisation, **1975**.
- F.J. Solis and R. J-B. Wets, *Minimization by random search techniques*, Math. of Operations Research 6 (**1981**), no. 1, 19–30. *(no rate)*
- Matyas **1965**, Polyak **1987**

Most literature focuses on $\ell = 1$

How is $\boldsymbol{q}$ typically chosen?

$$\|\boldsymbol{q}\|_2 = 1$$
$$\mathbb{E}[\boldsymbol{q}\boldsymbol{q}^\top] = \frac{1}{d}\boldsymbol{I}_{d\times d}$$

- spherical (or Gaussian, scaled appropriately)

or

- canonical basis vector, $\boldsymbol{q} \sim \mathrm{uniform}[\boldsymbol{e}_1, \ldots, \boldsymbol{e}_d]$

(hence SSD reduces to randomized coordinate descent)

much recent work on variants, 2011—2020 [and more since then!]

- D. Leventhal and A.S. Lewis, *Randomized Hessian estimation and directional search*, Optimization (2011)
- S. U. Stich, C. Muller, and B. Gartner, *Optimization of convex functions with random pursuit*, SIAM J. Opt. (2013)
- Yu. Nesterov, *Random gradient-free minimization of convex functions*, ’11 / Yu. Nesterov and V. Spokoiny, FoCM 2017
- P. Dvurechensky, A. Gasnikov, and A. Tiurin, *Randomized similar triangles method: A unifying framework for accelerated randomized optimization methods (coordinate descent, directional search, derivative-free method)*, arXiv:1707.08486
- P. Dvurechensky, A. Gasnikov, and E. Gorbunov, *An accelerated directional derivative method for smooth stochastic convex optimization*; arXiv:1804.02394
- S. Ghadimi and G. Lan, *Stochastic first- and zeroth-order methods for nonconvex stochastic programming*, SIAM J. Opt. (2013)
- R. Chen and S. Wild, *Randomized derivative-free optimization of noisy convex functions*, arXiv:1507.03332 (2015).
- K. Choromanski, M. Rowland, V. Sindhwani, R. E. Turner, and A. Weller, *Structured evolution with compact architectures for scalable policy optimization*, ICML, 2018.
- T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, *Evolution strategies as a scalable alternative to reinforcement learning*, arXiv:1703.03864 (2017).
- J. Duchi, M. Jordan, M. Wainwright, A. Wibisono, *Optimal Rates for Zero-Order Convex Optimization: The Power of Two Function Evaluations*, IEEE Trans Info Theory (2015)
- A. S. Berahas, L. Cao, K. Choromanski, K. Scheinberg, *A Theoretical and Empirical Comparison of Gradient Approximations in Derivative-Free Optimization*, arXiv 1905.01332 (2019)
- F. Hanzely, K. Mishchenko, P. Richtarik, *SEGA: Variance Reduction via Gradient Sketching*, NeurIPS 2018
- *cousin of “direct search” methods, cf.* S. Gratton, C. W. Royer, L. N. Vicente, Z. Zhang, *Direct Search Based on Probabilistic Descent*, SIAM J. Opt. (2015)

For recent results (2015-2025) on zeroth-order ML, see:

- “Zeroth-order Machine Learning” AAAI tutorial 2024
  - Wotao Yin, Sijia Liu, Pin-Yu Chen
  - https://sites.google.com/view/zo-tutorial-aaai-2024/

SSD: context

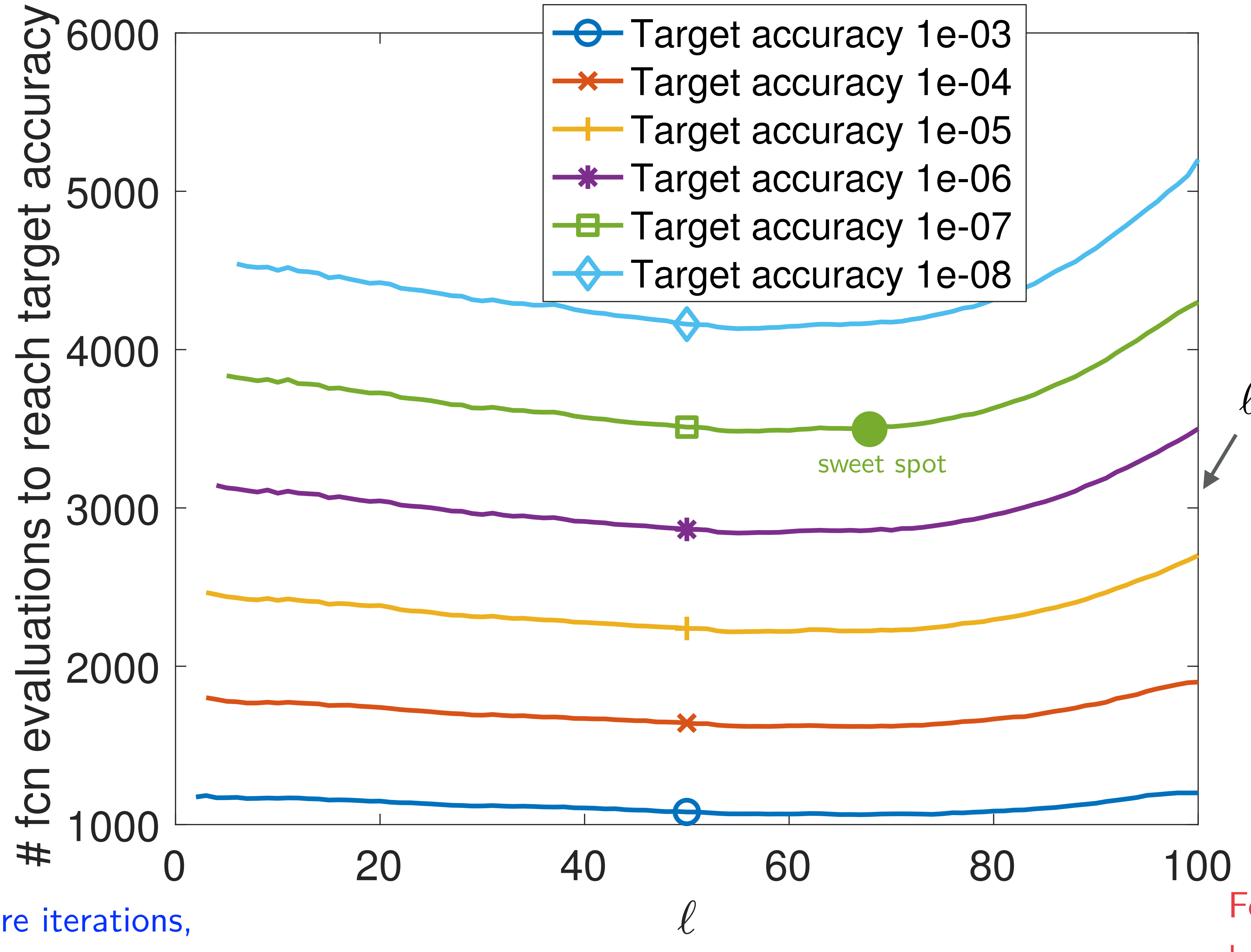
Variants have been investigated for a long time...

- "random gradient", "
  "directional searc
- ch 6, Yu. Ermoliev and
  optimization, Springer-
- M. Gaviano, *Some ger*
  minimization problems
- F.J. Solis and R. J-B.
  Operations Research 6
- Matyas **1965**, Polyak

much recent work on variants, 2011—2020 [and more since then!]

- D. Leventhal and A.S. Lewis, *Randomized Hessian estimation and directional search*, Optimization (2011)
- *ith random pursuit*, SIAM J. Opt. (2013)
- 'u. Nesterov and V. Spokoiny, FoCM 2017
- *hod: A unifying framework for accelerated*
  tive-free method), arXiv:1707.08486
- *vative method for smooth stochastic convex*
- *x stochastic programming*, SIAM J. Opt.
- *unctions*, arXiv:1507.03332 (2015).
- *tured evolution with compact architectures for*
- *a scalable alternative to reinforcement learning*,
- *r Convex Optimization: The Power of Two*
- *rical Comparison of Gradient Approximations*
- *t Sketching*, NeurIPS 2018
- '. Zhang, *Direct Search Based on Probabilistic*

Most work has focused on a single directional derivative, $\ell = 1$



100 dimensional quadratic test problem, using exact linesearch, averaged over 200 experiments

$\ell = d$ is gradient descent, and non-stochastic

More iterations, each one cheap

Fewer iterations, but each one costly

... but the optimal choice may be $1 < \ell < d$

Most literature foc

How is q typically

- spherical (or Ga

or

- canonical basis vector, $q \sim \text{uniform}[e_1, \ldots, e_d]$

(hence SSD reduces to randomized coordinate descent)

rder ML, see:

https://sites.google.com/view/zo-tutorial-aaai-2024/

Our first analysis

Theorem (Kozak, Becker, Tenorio, Doostan '20)
Assume: minimizer attained, gradient Lipschitz, stepsize η_k chosen appropriately.

1. If f is **convex**,
$$\mathbb{E}f(\boldsymbol{x}_k) - f^\star \leq 2\frac{d}{\ell}\frac{L}{k}R^2 = \mathcal{O}(k^{-1})$$

2. If f is **not necessarily convex** but satisfies the **Polyak-Lojasiewicz** inequality,
$$\mathbb{E}f(\boldsymbol{x}_k) - f^\star \leq \rho^k(f(\boldsymbol{x}_0) - f^\star) = \mathcal{O}(\rho^k) \quad \text{and} \quad f(\boldsymbol{x}_k) \xrightarrow{\text{a.s.}} f^\star$$

3. If f is **strongly convex**, statements of 2 above hold, and also
$$\boldsymbol{x}_k \xrightarrow{\text{a.s.}} \operatorname{argmin}_{\boldsymbol{x}} f(\boldsymbol{x})$$

4. If f is **not convex** (nor PL),
$$\min_{k' \in \{0,\ldots,k\}} \mathbb{E}\|\nabla f(\boldsymbol{x}_{k'})\|^2 \leq \frac{d}{\ell}\frac{2L(f(\boldsymbol{x}_0) - f^\star)}{k+1}$$

Generic SSD

$$\boldsymbol{Q}^\top\boldsymbol{Q} = \boldsymbol{I}_{\ell\times\ell}, \quad \mathbb{E}\left(\frac{d}{\ell}\boldsymbol{Q}\boldsymbol{Q}^\top\right) = \boldsymbol{I}_{d\times d}$$

David Kozak

$f^\star \overset{\text{def}}{=} \min_{\boldsymbol{x}} f(\boldsymbol{x})$

$\rho = 1 - \frac{\mu}{L}\frac{\ell}{d}$

d = ambient dimension

ℓ = # directional derivs

$\frac{d}{\ell} = 1$ is gradient descent

μ is PL constant

L is gradient Lipschitz constant

$\eta = \frac{\ell}{d}\frac{1}{L}$ is stepsize

Our first analysis

Our analysis is comparable to analysis of similar algorithms

Assume f obtains its minimum and ∇f is L-Lipschitz continuous.

Theorem 1 (Kozak, Becker, Tenorio, Doostan '19, Thm. 2.4)**.** *The SSD algorithm with stepsize* $\eta = \frac{1}{L}\frac{\ell}{d}$ *gives*

$$\mathbb{E}\, f(x_k) - f^\star \le \boxed{2\frac{d}{\ell}\frac{L}{k}R^2}$$

where

$$R = \sup_{x | f(x) \le f(x_0)} \; \inf_{x^\star \in \operatorname{argmin} f} \|x - x^*\|$$

(e.g., f is coercive $\implies R < \infty$).

SSD

$1 \le \ell \le d$

Theorem 2 (Nesterov, Spokoiny '17, Thm. 8)**.** *Take stepsize* $\eta = \frac{1}{4(d+4)L}$*, then the random gradient method with a Gaussian direction converges as*

$$\frac{1}{k}\sum_{i=0}^{k-1} \mathbb{E}\, f(x_i) - f^\star \le \boxed{\frac{4(d+4)L}{k}\|x_0 - x^\star\|^2}$$

where $x^\star$ is any optimal solution.

Gaussian

$\ell = 1$

convex, not necessarily strongly convex, scenario

Our first analysis

Our analysis is comparable to analysis of similar algorithms

Assume f obtains its minimum, ∇f is L-Lipschitz continuous, and f is μ PL or strongly convex.

Theorem 3 (Kozak, Becker, Tenorio, Doostan '19, Cor. 2.3)**.** *The SSD algorithm with stepsize* $\eta = \frac{1}{L}\frac{\ell}{d}$ *gives*

$$\mathbb{E}\, f(x_k) - f^\star \le \rho^k \left(f(x_0) - f^\star\right) \quad \textit{with} \quad \rho = \boxed{1 - \frac{\mu}{L}\frac{\ell}{d}}.$$

SSD

$1 \le \ell \le d$

Theorem 4 (Nesterov, Spokoiny '17, Thm. 8)**.** *Take stepsize* $\eta = \frac{1}{4(d+4)L}$*, then the random gradient method with a Gaussian direction converges as*

$$\mathbb{E}\, f(x_k) - f^\star \le \frac{L}{2}\rho^k \|x_0 - x^*\|^2 \quad \textit{with} \quad \rho = \boxed{1 - \frac{\mu}{L}\frac{1}{8(d+4)}}$$

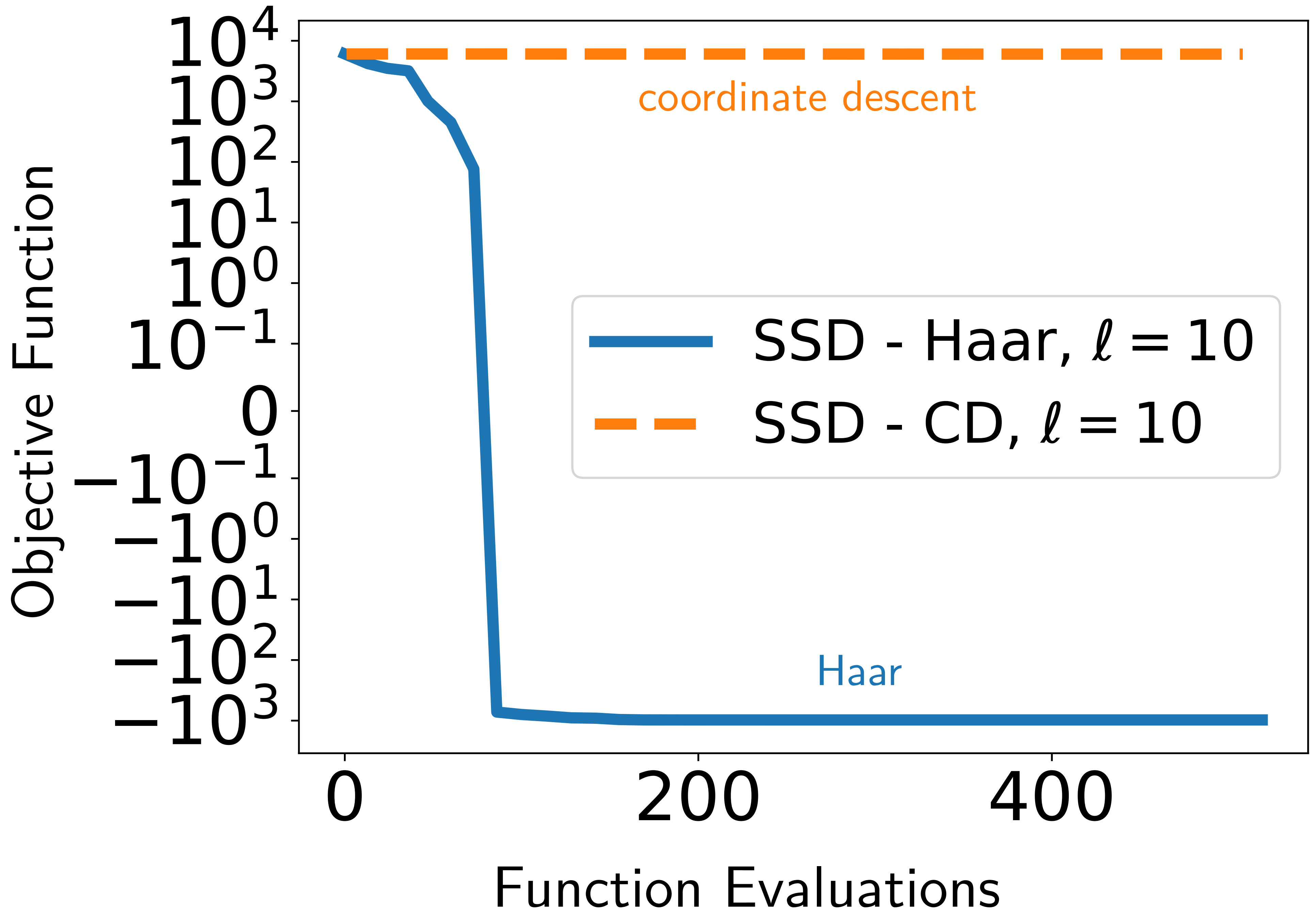
where $x^\star$ *is any optimal solution.*

Gaussian

$\ell = 1$

strongly convex or **PL scenario**

... but that theory doesn't capture the full story



Observation: sometimes SSD (with Haar) drastically outperforms randomized coordinate descent (CD)

Both of them are valid instantiations of SSD

So, the details of $\boldsymbol{Q}$ matter!

Algorithm: "Haar" SSD

Draw the random matrix $\boldsymbol{Q}$ as follows:

$\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_\ell\}$ a basis for $\operatorname{span}\{\tilde{\boldsymbol{q}}_1, \ldots, \tilde{\boldsymbol{q}}_\ell\}$ $\quad \tilde{\boldsymbol{q}}_i \overset{\text{iid}}{\sim} \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

(or any uniformly random subspace)

Equivalently, draw from the Haar distribution

Satisfies

$$\boldsymbol{Q}^\top \boldsymbol{Q} = \boldsymbol{I}_{\ell\times\ell}, \quad \mathbb{E}\left(\frac{d}{\ell}\boldsymbol{Q}\boldsymbol{Q}^\top\right) = \boldsymbol{I}_{d\times d}$$

but goes even further.

… but that theory doesn't capture the full story

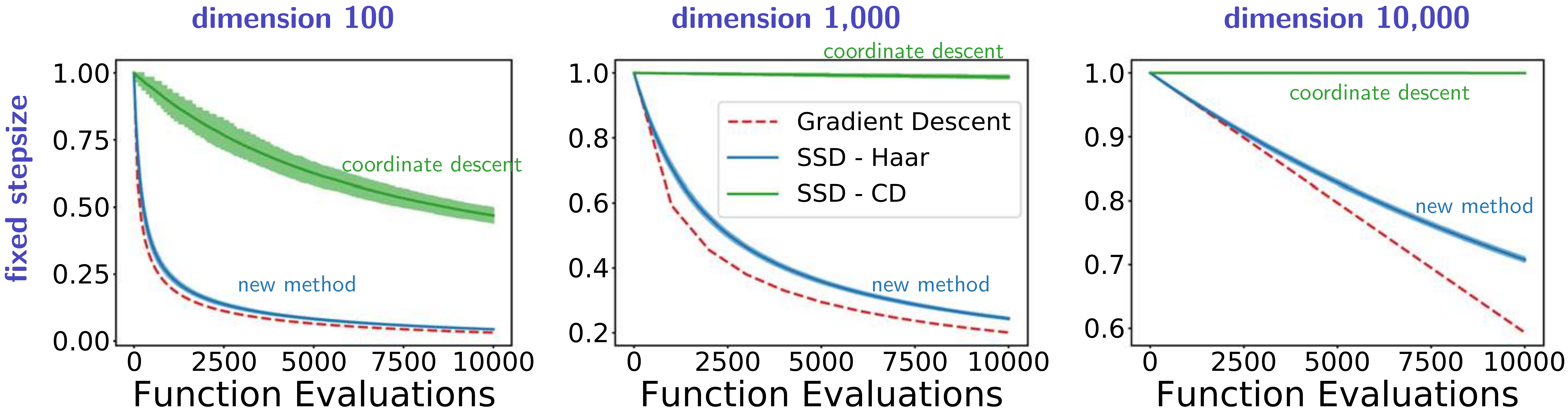
SSD drastically outperforms randomized coordinate descent (CD)

We can force it to happen by making a problem with low "intrinsic" dimension, e.g., Nesterov's "worst function in the world"

$$f_{\lambda,r}(\boldsymbol{x}) = \lambda((x_1^2 + \sum_{i=1}^{r-1}(x_i - x_{i+1})^2 + x_r^2)/2 - x_1)/4,$$

This has **intrinsic dimension** of r

$r = 20,\ \ell = 3$



Improved theory (specialized to SSD-Haar)

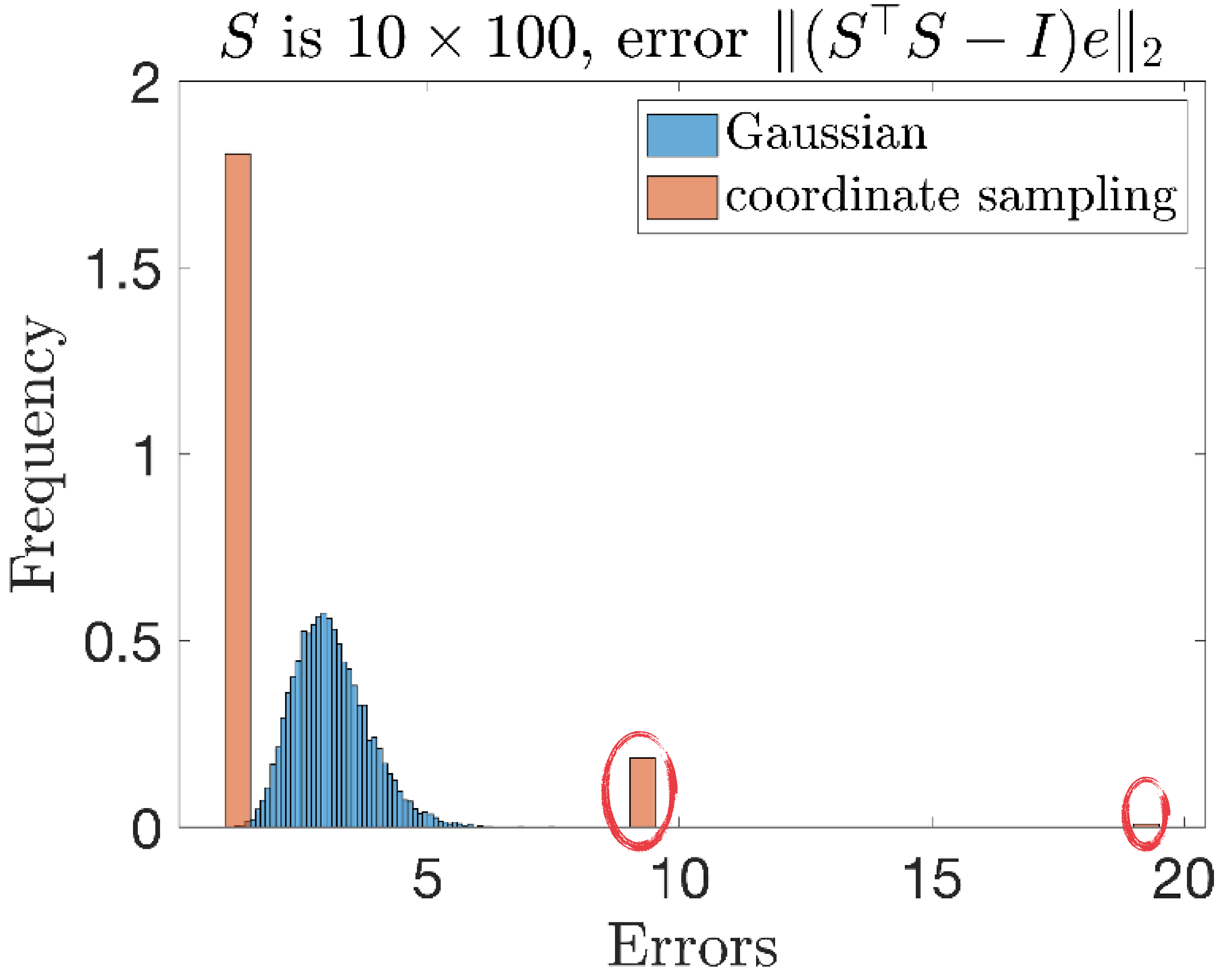
Recall...
$d =$ ambient dimension
$\ell = \#$ directional derivs

Tighter analysis using concentration-of-measure:

Lemma 2 (Johnson-Lindenstrauss style embedding, from Kozak, Becker, Tenorio '19, Lemma 1)**.**
$\forall \epsilon \in (0,1)$, if $\ell \gtrsim \epsilon^{-2}$, $Q \sim \mathrm{Haar}(d \times \ell)$, then $\forall 0 \neq g \in \mathbb{R}^d$,

$$1 - \epsilon \le \frac{d}{\ell}\frac{\|Q^T g\|^2}{\|g\|^2} \le 1 + \epsilon \quad \textit{w/ prob. } \delta \ge 0.8$$

Note: *coordinate descent* style projections
do **not** have similar nice embedding properties



Improved theory (specialized to SSD-Haar)

Recall...
$d =$ ambient dimension
$\ell =$ # directional derivs

Tighter analysis using concentration-of-measure:

Lemma 2 (Johnson-Lindenstrauss style embedding, from Kozak, Becker, Tenorio '19, Lemma 1)**.** $\forall \epsilon \in (0,1)$, *if* $\ell \gtrsim \epsilon^{-2}$, $Q \sim \mathrm{Haar}(d \times \ell)$, *then* $\forall 0 \neq g \in \mathbb{R}^d$,

$$1 - \epsilon \leq \frac{d}{\ell} \frac{\|Q^T g\|^2}{\|g\|^2} \leq 1 + \epsilon \quad \textit{w/ prob. } \delta \geq 0.8$$

... in fact, we can have tight (dimension-dependent) bounds via standard probability facts

Lemma Let $\boldsymbol{Q} \sim \mathrm{Haar}(d \times \ell)$, then $\forall \boldsymbol{g} \in \mathbb{R}^d$, $\quad \dfrac{d}{\ell} \dfrac{\|\boldsymbol{Q}^\top \boldsymbol{g}\|^2}{\|\boldsymbol{g}\|^2} \sim \mathcal{B}\mathrm{eta}\left(\dfrac{\ell}{2}, \dfrac{d-\ell}{2}\right)$

and the CDF of the Beta distribution can be stably computed via the Beta function

Example

Define $\quad \delta = \mathbb{P}\left(\frac{d}{\ell}\|\boldsymbol{Q}^\top \boldsymbol{g}\|^2 > (1-\epsilon)\|\boldsymbol{g}\|^2\right)$

For an embedding of accuracy $\epsilon = 0.1$

	$\delta = 99\%$		$\delta = 99.99\%$	
d	ℓ	ℓ/d	ℓ	ℓ/d
1000	520	51.98%	755	75.47%
10,000	933	9.32%	2086	20.86%
100,000	1013	1.01%	2532	2.53%
1,000,000	1022	0.10%	2587	0.26%
10,000,000	1023	0.01%	2593	0.03%

Improved theory (specialized to SSD-Haar)

Recall…
$d =$ ambient dimension
$\ell = \#$ directional derivs

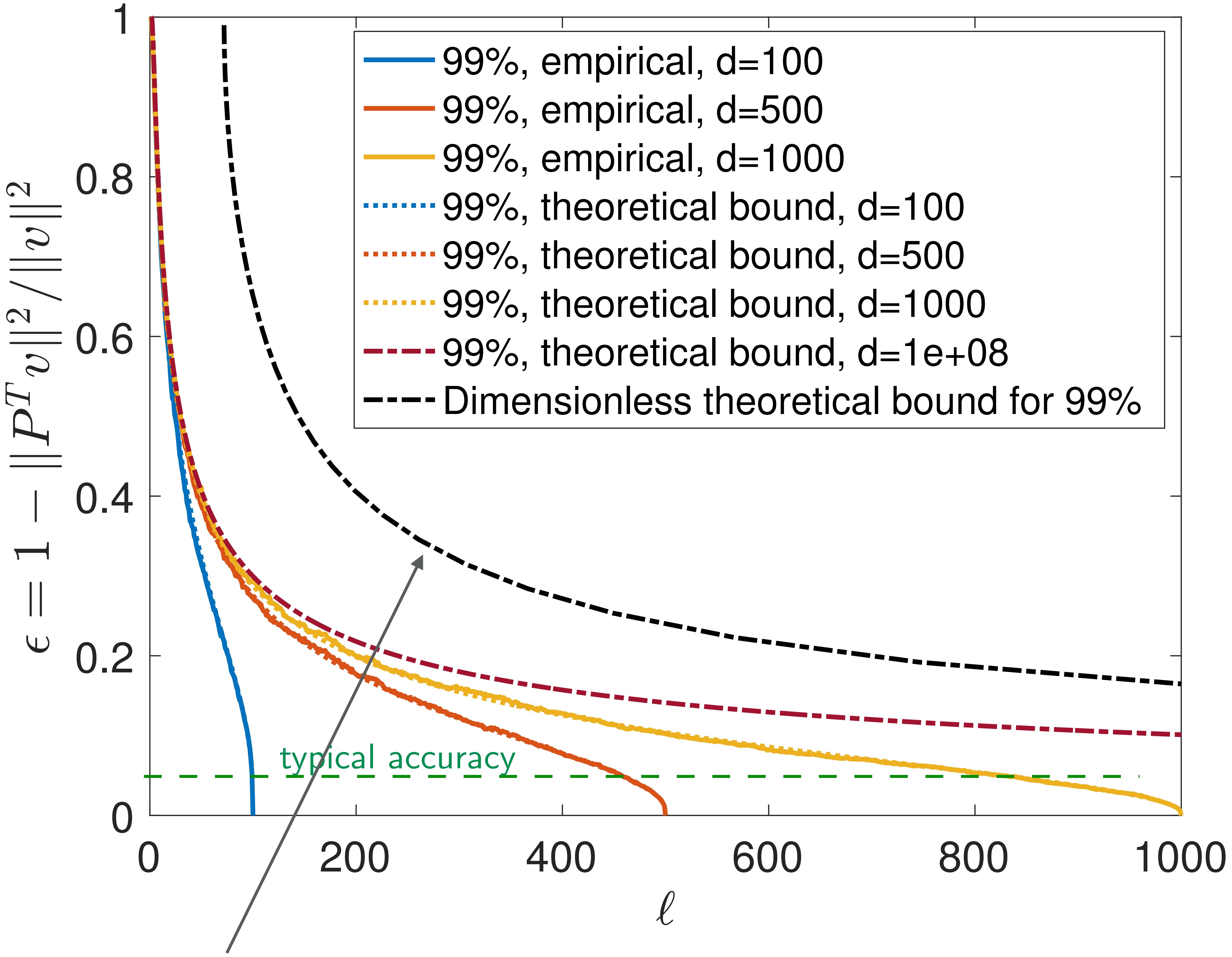
Tighter analysis using conc

Lemma 2 (Johnson-Li
$\forall \epsilon \in (0,1)$, *if* $\ell \gtrsim \epsilon^{-2}$, (

… in fact, we can have

Lemma Let $\boldsymbol{Q} \sim$ Ha

and the CDF



Message: usual dimensionless Johnson-Lindenstrauss style results are far from sharp in low dimensions (in fact, so loose that they can be meaningless)

Only downside of tighter analysis is that we can't write down a pretty formula

Improved theory (specialized to SSD-Haar)

Recall...
$d =$ ambient dimension
$\ell = \#$ directional derivs

Tighter analysis using concentration-of-measure:

Lemma 2 (Johnson-Lindenstrauss style embedding, from Kozak, Becker, Tenorio '19, Lemma 1). *$\forall \epsilon \in (0,1)$, if $\ell \gtrsim \epsilon^{-2}$, $Q \sim \mathrm{Haar}(d \times \ell)$, then $\forall 0 \neq g \in \mathbb{R}^d$,*

$$1 - \epsilon \leq \frac{d}{\ell} \frac{\|Q^T g\|^2}{\|g\|^2} \leq 1 + \epsilon \quad \textit{w/ prob. } \delta \geq 0.8$$

... in fact, we can have tight (dimension-dependent) bounds via standard probability facts

Lemma Let $\boldsymbol{Q} \sim \mathrm{Haar}(d \times \ell)$, then $\forall \boldsymbol{g} \in \mathbb{R}^d$, $\quad \dfrac{d}{\ell} \dfrac{\|\boldsymbol{Q}^\top \boldsymbol{g}\|^2}{\|\boldsymbol{g}\|^2} \sim \mathcal{B}\mathrm{eta}\left(\dfrac{\ell}{2}, \dfrac{d-\ell}{2}\right)$

and the CDF of the Beta distribution can be stably computed via the Beta function

... and putting it all together

Theorem 3 (Kozak, Becker, Tenorio '19, Thm. 1). *If f is strongly convex and ∇f is Lipschitz continuous, then for an appropriate stepsize η_k, the sequence (x_k) generated by SSD (with $Q \sim$ Haar), for $k > 100$, satisfies*

$$f(x_k) - f^\star \leq (1 + (1-\epsilon)\rho)^{k/2} \left(f(x_0) - f^\star\right) \quad \textit{with probability } \geq 0.998,$$

where $\rho < 1$ depends on ℓ, d and the Lipschitz and strong convexity parameters.

due to possibility of failure of JL

error in JL embedding

Extension: Variance Reduction

control variate

Algorithm SVRG-style Variance Reduced SSD method, “VRSSD”

```
for $k = 1, 2, \ldots$ do                                          ▷ $k$ is the “epoch”
    $\bar z \leftarrow \nabla f(x_k)$                              ▷ Expensive, but not done often
    $w_0 \leftarrow x_k$
    for $t = 1, 2, \ldots, T$ do                                   ▷ Typically $T = \mathcal{O}(d)$
        Draw $Q \sim \text{Haar}(d \times \ell)$
        $w_{t+1} \leftarrow w_t - \eta\left(\frac{d}{\ell}QQ^T\nabla f(w_t) - \alpha_k \underbrace{\left(\frac{d}{\ell}QQ^T - I\right)}_{\text{orthogonal projection}} \bar z\right)$    ▷ $\alpha_k$ to be estimated
    $x_{k+1} \leftarrow w_T$
```

regular SSD term

only use control variate in orthogonal subspace

(since we know gradient in main subspace)

Theorem 4 (Kozak, Becker, Tenorio, Doostan 2019; Thm. 2.7)**.** *If f is strongly convex and ∇f is Lipschitz continuous, then for an appropriate stepsize η_k, the sequence (x_k) generated by* VRSSD *converges almost surely to the (unique) minimizer of f and at a linear rate (the rate depends on η_k and α_k).*

⚠ We do not require the ERM structure!

$f_c(x) \approx f(x)$

generic (non-algorithmic) control variates

control variate, **c**oarse approximation, **c**heap to evaluate

Algorithm Proposed coarse-model variance reduced SSD/Random-Gradient

```
for $k = 1, 2, \ldots$ do
    $\bar z \leftarrow \nabla f_c(x_k)$                            ▷ Full coarse-grid gradient
    Draw $Q \sim \text{Haar}(d \times \ell)$
    $x_{k+1} \leftarrow x_k - \eta_k\left(\frac{d}{\ell}QQ^T\nabla f(x_k) + \alpha_k\left(\frac{d}{\ell}QQ^T\bar z - \bar z\right)\right)$
```

Key idea: easy to do **orthogonal projection**

from the literature:

Algorithm SAGA (Defazio, Bach, Lacoste-Julien ’14) for solving the ERM model

```
$\forall i = 1, \ldots .N$, $x^{(i)} \overset{\text{def}}{=} x_0$; store $\{\nabla f_i(x^{(i)})\}_{i=1}^N$ in table
for $k = 1, 2, \ldots$ do
    Draw $j \sim \text{Uniform}([1, \ldots, N])$
    $\bar z \leftarrow \frac{1}{N}\sum_{i=1}^N \nabla f_i(x^{(i)})$          ▷ From table
    $x_{k+1} \leftarrow x_k - \eta\left(\nabla f_j(x_k) - \nabla f_j(x^{(j)}) + \bar z\right)$
    Re-define $x^{(j)} \leftarrow x_k$ and update table with $\nabla f_j(x^{(j)})$
```

our variant:

Algorithm SAGA-style Variance Reduced SSD method

```
Pre-compute $\bar z \leftarrow \nabla f(x_0)$
for $k = 1, 2, \ldots$ do
    Draw $Q \sim \text{Haar}(p \times r)$
    $x_{k+1} \leftarrow x_k - \eta\left(\frac{d}{\ell}QQ^T\nabla f(x_k) - \frac{d}{\ell}QQ^T\bar z + \bar z\right)$
    $\bar z \leftarrow \bar z + QQ^T(\nabla f(x_k) - \bar z)$    ▷ Update of $\bar z$ is low-memory, unlike original SAGA
```

key: update control variate in the subspace

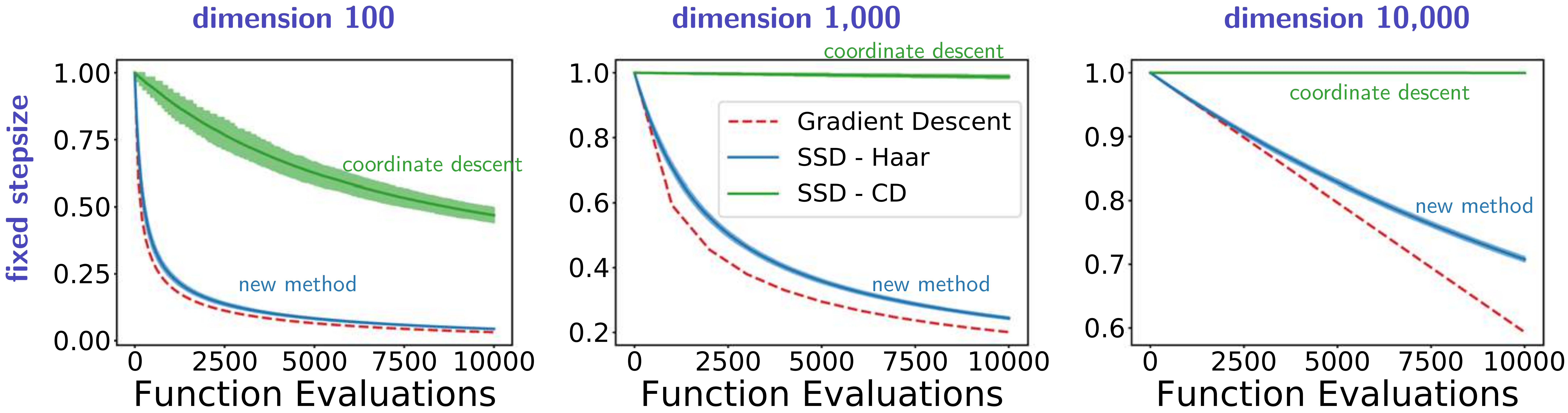
Extension: stepsize selection

$r = 20,\ \ell = 3$

Nesterov's "worst function in the world"

$$f_{\lambda,r}(\boldsymbol{x}) = \lambda((x_1^2 + \sum_{i=1}^{r-1}(x_i - x_{i+1})^2 + x_r^2)/2 - x_1)/4,$$

This has **intrinsic dimension** of r



Recall previous example

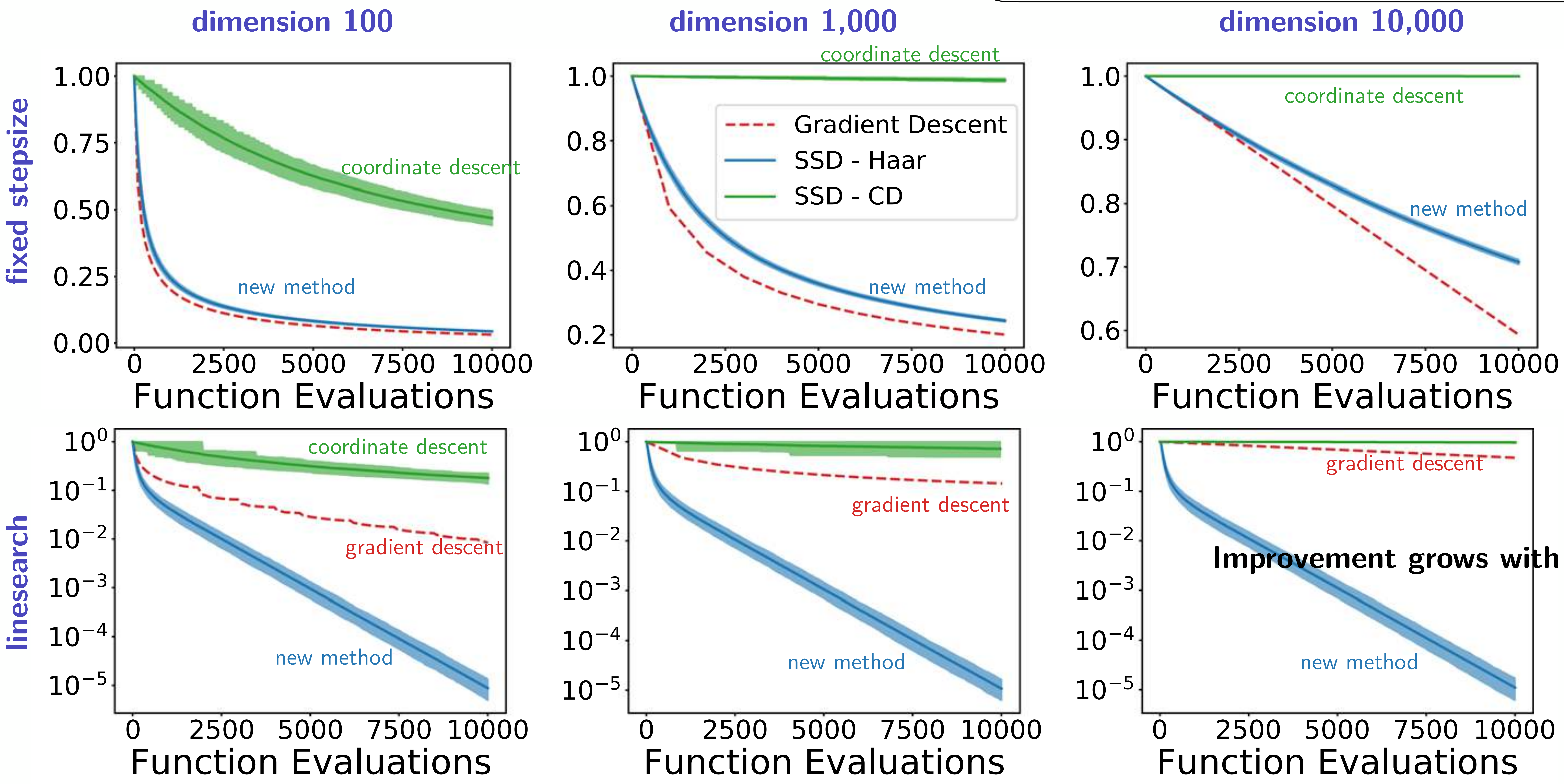
Extension: stepsize selection

$r = 20,\ \ell = 3$

Nesterov's "worst function in the world"

$$f_{\lambda,r}(\boldsymbol{x}) = \lambda((x_1^2 + \sum_{i=1}^{r-1} (x_i - x_{i+1})^2 + x_r^2)/2 - x_1)/4,$$

This has **intrinsic dimension** of r



Improvement grows with dimension!

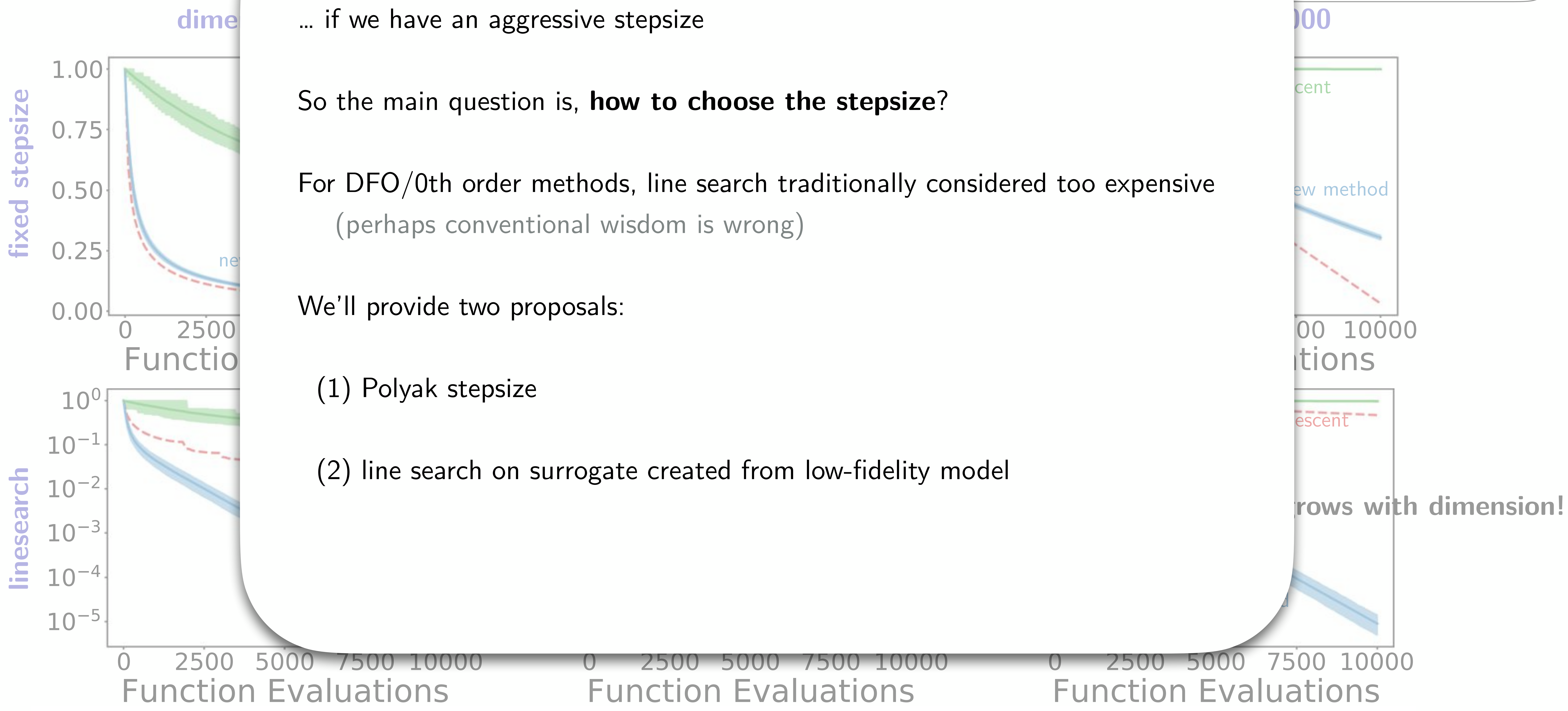
Extension: stepsize selection

$r = 20,\ \ell = 3$

Nesterov's "worst function in the world"

$r-1$

$_{+1})^2 + x_r^2)/2 - x_1)/4,$



Working hypothesis: SSD-Haar nicely exploits low-dimensional structure…

… if we have an aggressive stepsize

So the main question is, **how to choose the stepsize**?

For DFO/0th order methods, line search traditionally considered too expensive

(perhaps conventional wisdom is wrong)

We'll provide two proposals:

(1) Polyak stepsize

(2) line search on surrogate created from low-fidelity model

Stepsize selection: Polyak stepsize

Joint project with Killian Wood,
Drona Khurana

Polyak stepsize for gradient descent (1983)

Recently revisited a lot in literature

$$\eta_k^{\text{Polyak}} = \frac{f(\boldsymbol{x}_k) - f^\star}{\|\nabla f(\boldsymbol{x}_k)\|^2} \qquad f^\star = \min_{\boldsymbol{x}} f(\boldsymbol{x})$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \eta_k^{\text{Polyak}} \nabla f(\boldsymbol{x}_k)$$

Stepsize selection: Polyak stepsize

Inspired by recent analysis in literature

Polyak stepsize for gradient descent (1983)

Recently revisited a lot in literature

$$\eta_k^{\text{Polyak}} = \frac{f(\boldsymbol{x}_k) - f^\star}{\|\nabla f(\boldsymbol{x}_k)\|^2} \qquad f^\star = \min_{\boldsymbol{x}} f(\boldsymbol{x})$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \eta_k^{\text{Polyak}} \nabla f(\boldsymbol{x}_k)$$

Our extension to SSD case:

$$\eta_k^{\text{Polyak-SSD}} = \frac{f(\boldsymbol{x}_k) - f_k^\star}{\|\boldsymbol{Q}^\top \nabla f(\boldsymbol{x}_k)\|^2} \qquad f_k^\star = \min_{\boldsymbol{x} \in \{\boldsymbol{x}_k\} + \text{col}\boldsymbol{Q}} f(\boldsymbol{x})$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \eta_k^{\text{Polyak-SSD}} \boldsymbol{Q}\boldsymbol{Q}^\top \nabla f(\boldsymbol{x}_k)$$

Stepsize selection: Polyak stepsize

Inspired by recent analysis in literature

Polyak stepsize for gradient descent (1983)

Recently revisited a lot in literature

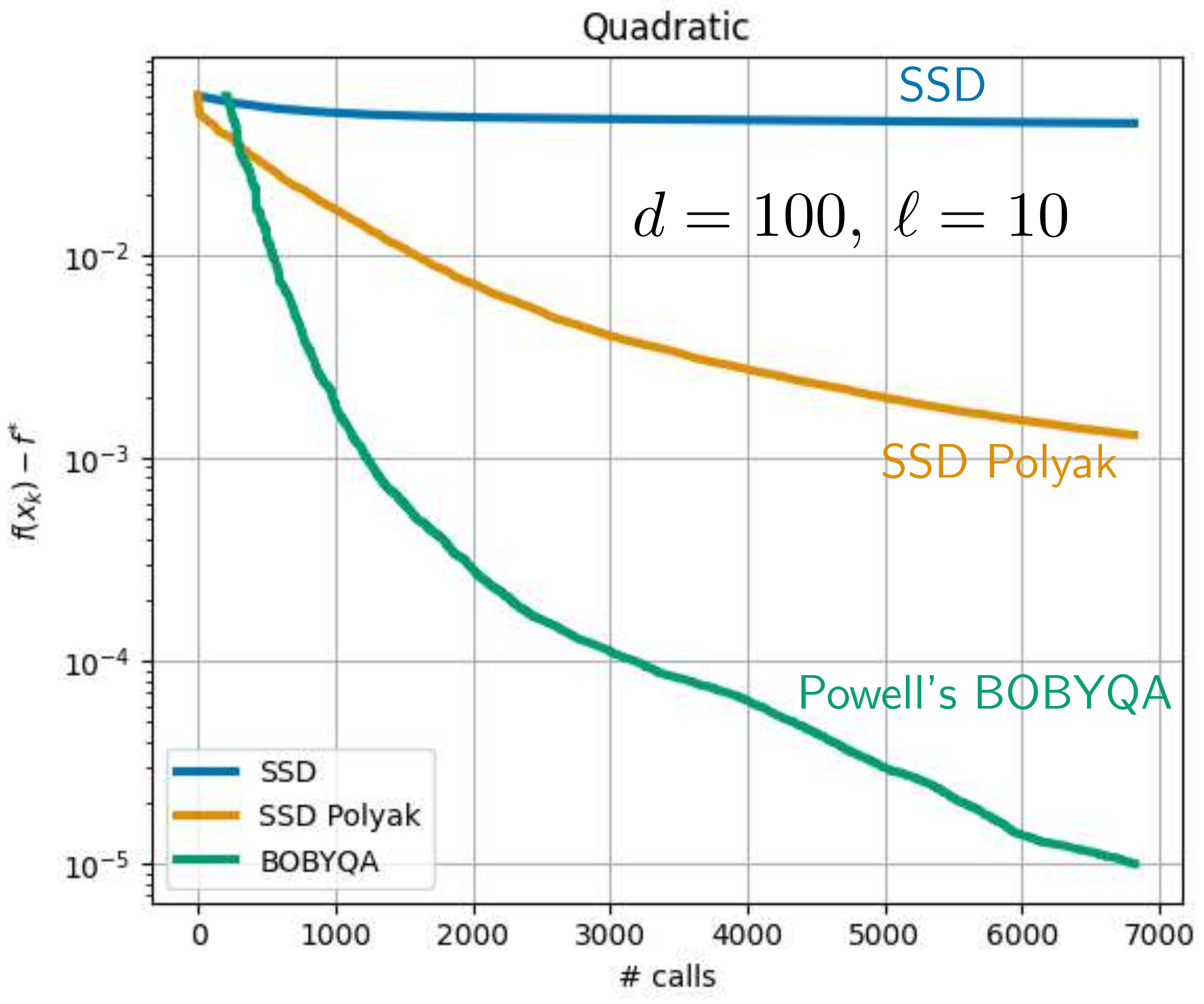
$$\eta_k^{\text{Polyak}} = \frac{f(\boldsymbol{x}_k) - f^\star}{\|\nabla f(\boldsymbol{x}_k)\|^2} \qquad f^\star = \min_{\boldsymbol{x}} f(\boldsymbol{x})$$

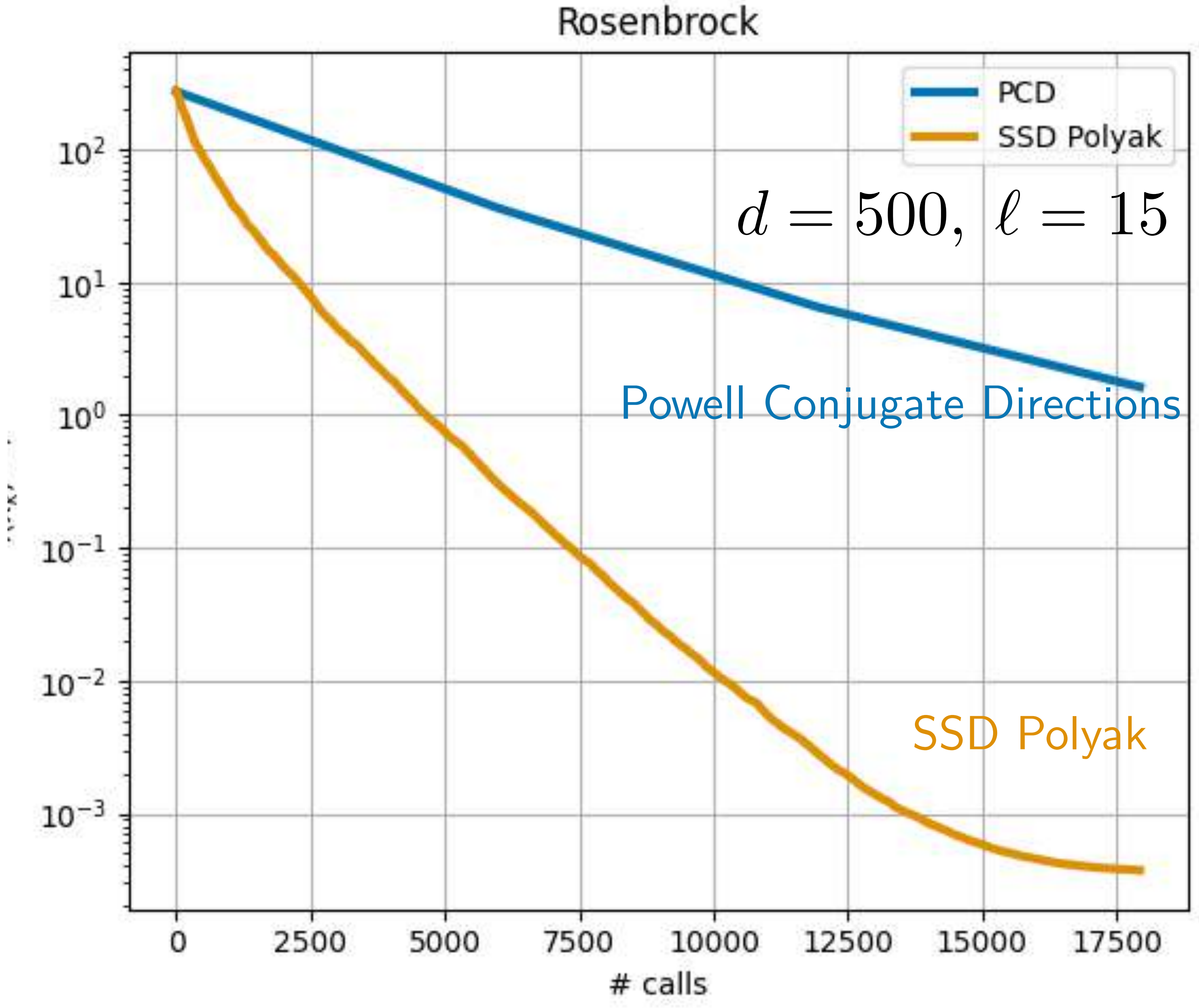
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \eta_k^{\text{Polyak}} \nabla f(\boldsymbol{x}_k)$$

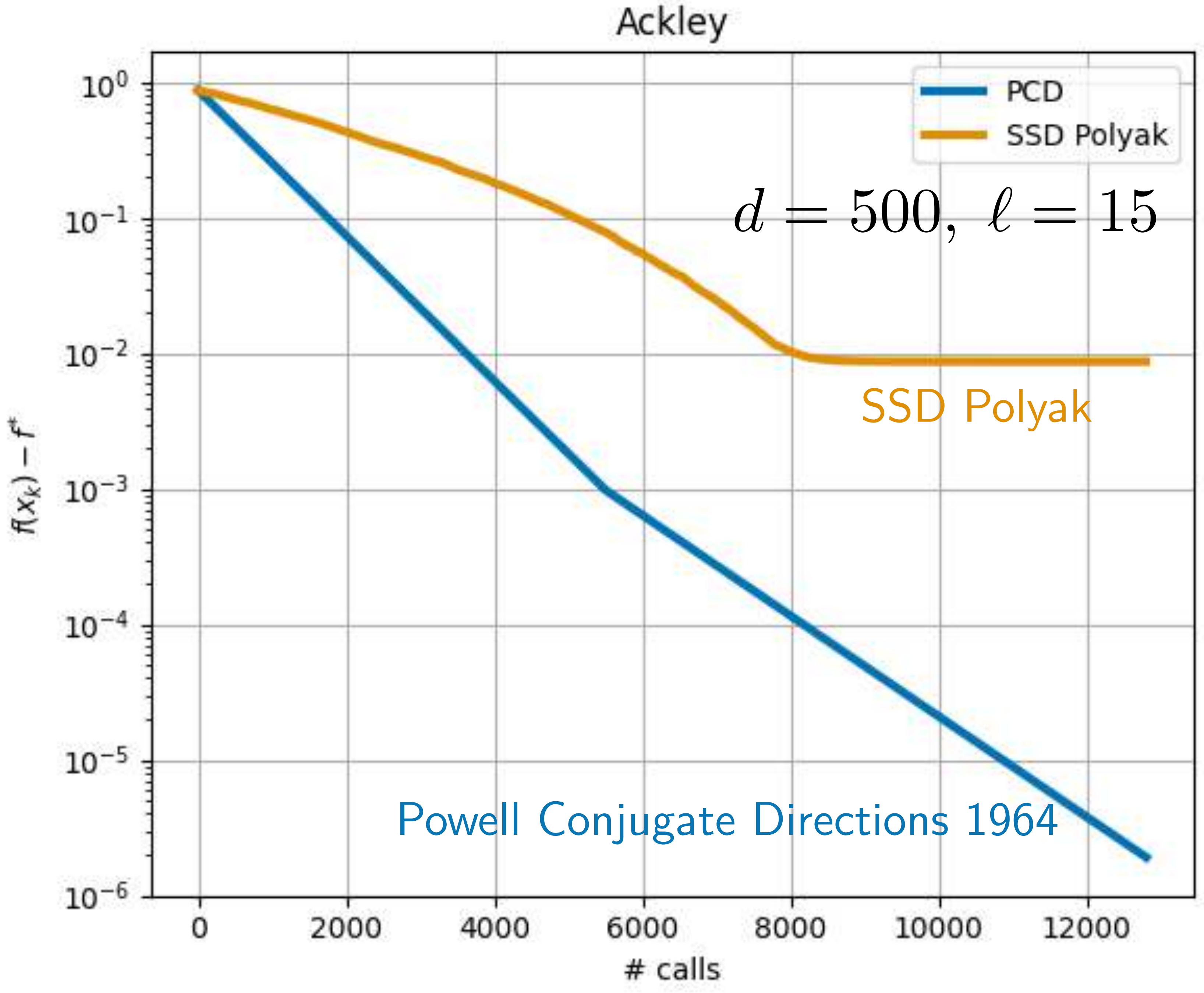
Our extension to SSD case:

$$\eta_k^{\text{Polyak-SSD}} = \frac{f(\boldsymbol{x}_k) - f_k^\star}{\|\boldsymbol{Q}^\top \nabla f(\boldsymbol{x}_k)\|^2} \qquad f_k^\star = \min_{\boldsymbol{x} \in \{\boldsymbol{x}_k\} + \text{col}\boldsymbol{Q}} f(\boldsymbol{x})$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \eta_k^{\text{Polyak-SSD}} \boldsymbol{Q}\boldsymbol{Q}^\top \nabla f(\boldsymbol{x}_k)$$







Works fine in practice, our analysis is ongoing

Stepsize selection: bifidelity surrogate

Joint project with Nuojin (Noki) Cheng (Google)

Classic exact linesearch

$$g_k = QQ^\top \nabla f(x_k)$$

$$\eta^\star = \operatorname{argmin} \varphi(\eta) \qquad \varphi(\eta) \overset{\text{def}}{=} f(x_k - \eta g_k)$$

$$x_{k+1} = x_k - \eta^\star g_k$$

ideally would do line search on this but too expensive

Premise: suppose we have a cheap, inaccurate approximation f^{low}

data (function evaluations)

surrogate model

Expensive $\varphi(\eta) \overset{\text{def}}{=} f(x_k - \eta g_k)$ $\{\varphi(0), \varphi(\eta_{\max})\}$

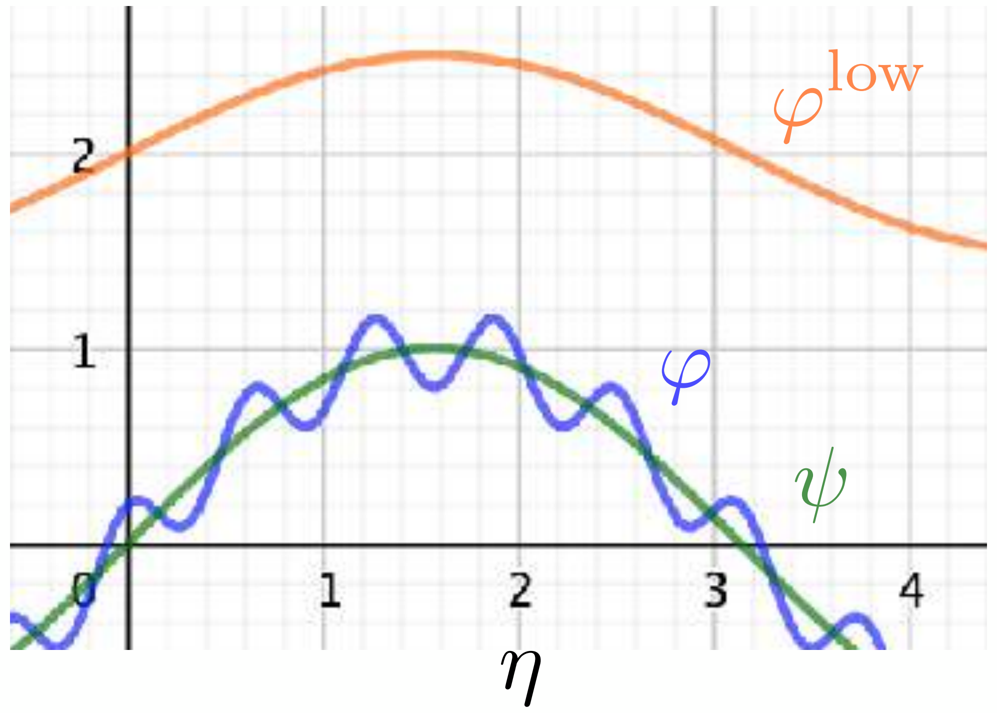
Inaccurate $\varphi^{\text{low}}(\eta) \overset{\text{def}}{=} f^{\text{low}}(x_k - \eta g_k)$ $\{\varphi^{\text{low}}(\eta_i)\}_{i=1}^{20}$

$\}$ co-kriging (1D) ⇨ $\psi(\eta)$ ⇨ $\eta^\star = \operatorname{argmin} \psi(\eta)$

traditional line search on surrogate model

(computationally "free")

e.g., calibrate low-fidelity model



Convergence analysis in our preprint "Stochastic Subspace Descent Accelerated via Bi-fidelity Line Search" arxiv.org/abs/2505.00162, Nuojin Chen, Alireza Doostan, Stephen Becker

ML bifidelity example 1

Context:

- black-box model
- high-dimensional, low accuracy

Premise: suppose we have a cheap, inaccurate approximation f^{low}

Example: black-box adversarial attack Carlini & Wagner '17, black-box extension Chen et al. '17

For a given sample, find a small perturbation such that the machine learning algorithm misclassifies it

switching to ML notation!

particular training example (features and true label)

$$\boxed{\min_{\boldsymbol{\epsilon}} -f_{\text{cross-entropy}}\left(g(\boldsymbol{x}^\dagger + \boldsymbol{\epsilon}), y^\dagger\right) + \tau \|\boldsymbol{\epsilon}\|^2}$$

encourages small perturbation

model output: vector with probability of different classes

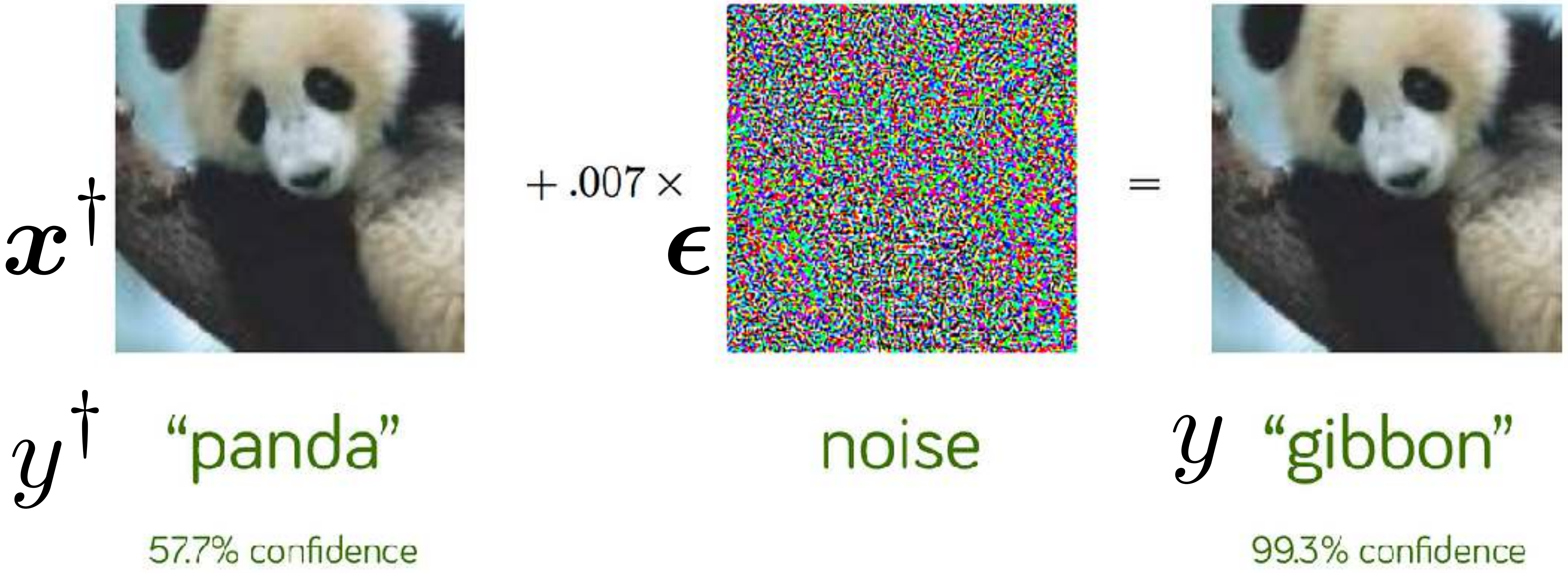


Image source: Explaining and Harnessing Adversarial Examples, Goodfellow et al, ICLR 2015

ML bifidelity example 1

Context:

- black-box model
- high-dimensional, low accuracy

Premise: suppose we have a cheap, inaccurate approximation f^{low}

Example: black-box adversarial attack Carlini & Wagner '17, black-box extension Chen et al. '17

For a given sample, find a small perturbation such that the machine learning algorithm misclassifies it

particular training example (features and true label)

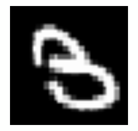
$$\min_{\epsilon} -f_{\text{cross-entropy}}(g(x^\dagger + \epsilon), y^\dagger) + \tau\|\epsilon\|^2$$

encourages small perturbation

model output: vector with probability of different classes

switching to ML notation!

MNIST is 28 x 28 images so $d = 784$



Train two models on MNIST data: (60k training, 10k test)

f is output of **large model**, trained conventionally

convolution (32 filters) -> convolution (64 filters) -> max-pooling/flatten, fully connected (1024 neurons) -> 10 class output. ReLU activation, 5x5 kernels

f^{low} is output of **small model**

trained not on MNIST but on output of large model (knowledge distillation), 1000 samples

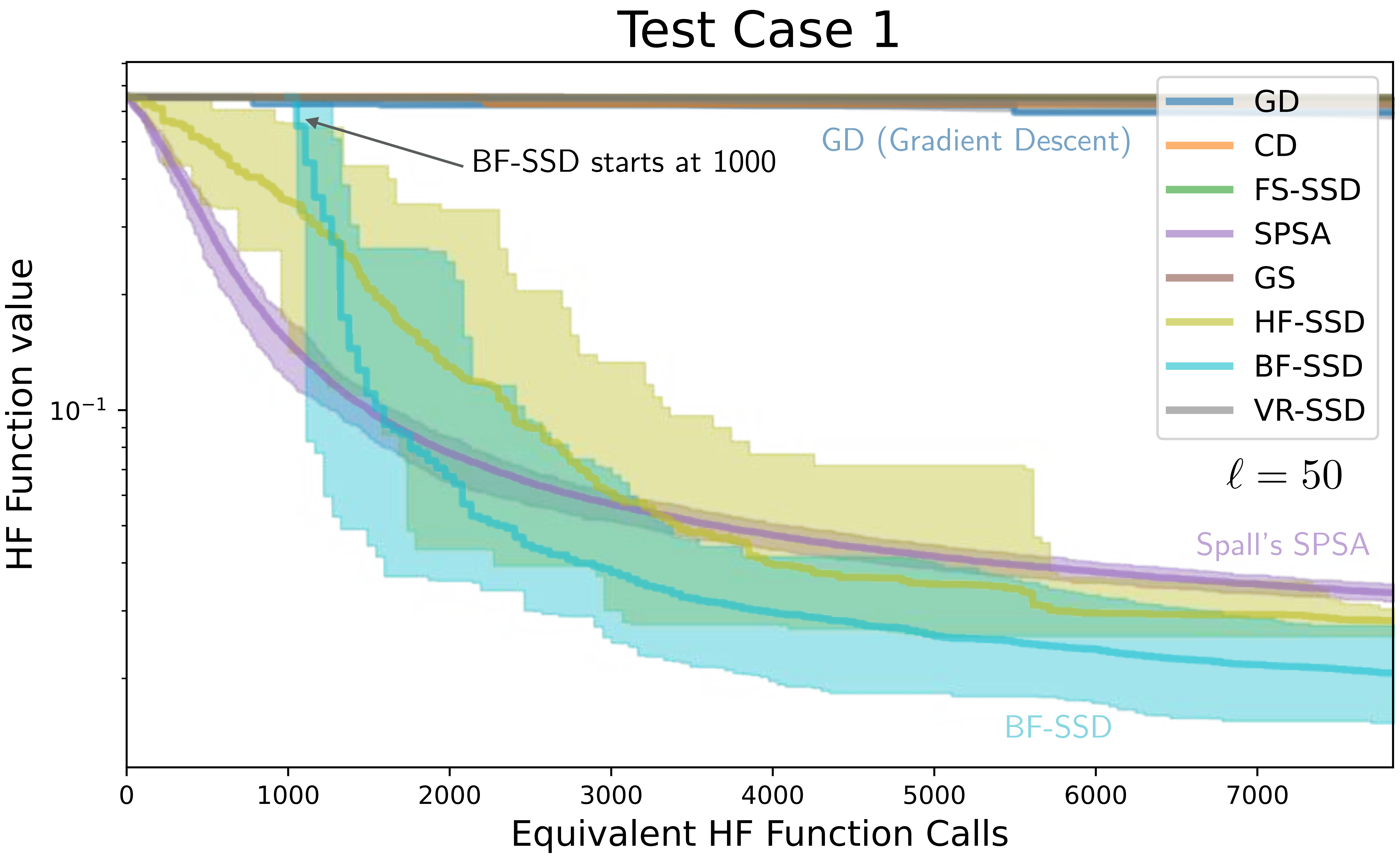
convolution (2 filters) -> max-pooling/flatten, fully connected (16 neurons) -> 10 class output. ReLU activation, 2x3 kernels

119x larger

	Large model	Small model
# parameters	3,274,634	27,562
Test Accuracy	99.02%	82.21%

In some scenarios, small model is not just cheap but "free"

ML bifidelity example 1: Results



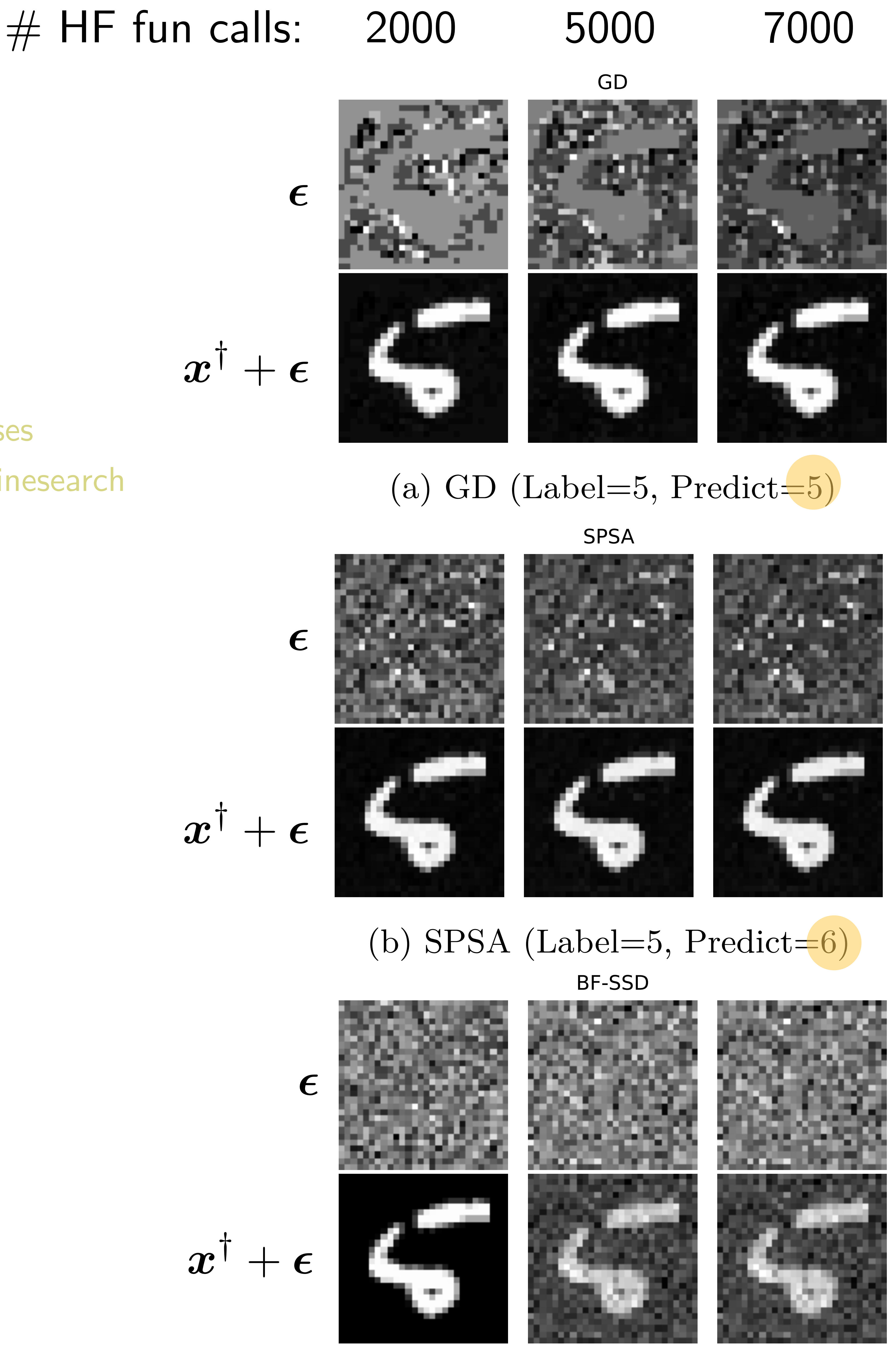
BF-SSD starts at 1000

GD (Gradient Descent)

HF-SSD uses "perfect" linesearch

Spall's SPSA

BF-SSD



(a) GD (Label=5, Predict=5)

(b) SPSA (Label=5, Predict=6)

(d) BF-SSD (Label=5, Predict=6)

ML bifidelity example 2

Context:

- black-box model
- high-dimensional, low accuracy

Premise: suppose we have a cheap, inaccurate approximation f^{low}

Example: soft prompting black-box LLM

We want to fine-tune a LLM like BERT or GPT

Instead of modifying network, lightweight alternative is to learn embeddings that are prepended to input sequence

ML bifidelity example 2

Context:

- black-box model
- high-dimensional, low accuracy

Premise: suppose we have a cheap, inaccurate approximation f^{low}

Example: soft prompting black-box LLM

We want to fine-tune a LLM like BERT or GPT

Instead of modifying network, lightweight alternative is to learn embeddings that are prepended to input sequence

Task: binary sentiment analysis (*classify a movie review as positive or negative*)

$$\text{Pretrained: } \begin{cases} f_{\text{token}} : \texttt{str} \to \mathbb{R}^{L_t \times d} & \text{tokenizer converts strings of any length to an embedding} \\ f_c : \mathbb{R}^{L_t \times d} \to [0,1] & \text{classifier (we use small } \texttt{DistilBERT}\text{, small version of } \texttt{BERT}\text{)} \\ (\boldsymbol{z}, y) \in \texttt{str} \times \{0,1\} & \text{data from } \texttt{aclIMDB} \text{ database} \end{cases}$$

transformer

$d = 784$

ML bifidelity example 2

Context:

- black-box model
- high-dimensional, low accuracy

Premise: suppose we have a cheap, inaccurate approximation f^{low}

Example: soft prompting black-box LLM

We want to fine-tune a LLM like BERT or GPT

Instead of modifying network, lightweight alternative is to learn embeddings that are prepended to input sequence

Task: binary sentiment analysis (*classify a movie review as positive or negative*)

Pretrained:

- $f_{\text{token}} : \texttt{str} \to \mathbb{R}^{L_t \times d}$ tokenizer converts strings of any length to an embedding
- $f_c : \mathbb{R}^{L_t \times d} \to [0,1]$ classifier (we use small DistilBERT, small version of BERT) transformer
- $(\boldsymbol{z}, y) \in \texttt{str} \times \{0,1\}$ data from aclIMDB database

$d = 784$

cross-entropy loss

$$\boxed{\boldsymbol{x}^\star \in \underset{\boldsymbol{x}\in\mathbb{R}^d}{\operatorname{argmin}} \ \mathbb{E}_{(\boldsymbol{z},y)} \left[\operatorname{CE}(f_c(\operatorname{cat}[\boldsymbol{x}, f_{\text{token}}(\boldsymbol{z})]), y)\right]}$$

risk, replaced by empirical risk for training

$$f(\boldsymbol{x}) = \frac{1}{10}\sum_{i=1}^{10} \operatorname{CE}(f_c(\operatorname{cat}[\boldsymbol{x}, f_{\text{token}}(\boldsymbol{z}_i)]), y_i)$$

ML bifidelity example 2

Context:

- black-box model
- high-dimensional, low accuracy

Premise: suppose we have a cheap, inaccurate approximation f^{low}

Example: soft prompting black-box LLM

We want to fine-tune a LLM like BERT or GPT

Instead of modifying network, lightweight alternative is to learn embeddings that are prepended to input sequence

Task: binary sentiment analysis (*classify a movie review as positive or negative*)

$$\text{Pretrained: } \begin{cases} f_{\text{token}} : \texttt{str} \to \mathbb{R}^{L_t \times d} & \text{tokenizer converts strings of any length to an embedding} \\ f_c : \mathbb{R}^{L_t \times d} \to [0,1] & \text{classifier (we use small } \texttt{DistilBERT}\text{, small version of } \texttt{BERT}\text{)} \\ (\boldsymbol{z}, y) \in \texttt{str} \times \{0,1\} & \text{data from } \texttt{aclIMDB} \text{ database} \end{cases}$$

transformer

$d = 784$

cross-entropy loss

$$\boxed{\boldsymbol{x}^\star \in \underset{\boldsymbol{x}\in\mathbb{R}^d}{\operatorname{argmin}} \ \mathbb{E}_{(\boldsymbol{z},y)} \left[\operatorname{CE}(f_c(\operatorname{cat}[\boldsymbol{x}, f_{\text{token}}(\boldsymbol{z})]), y)\right]}$$

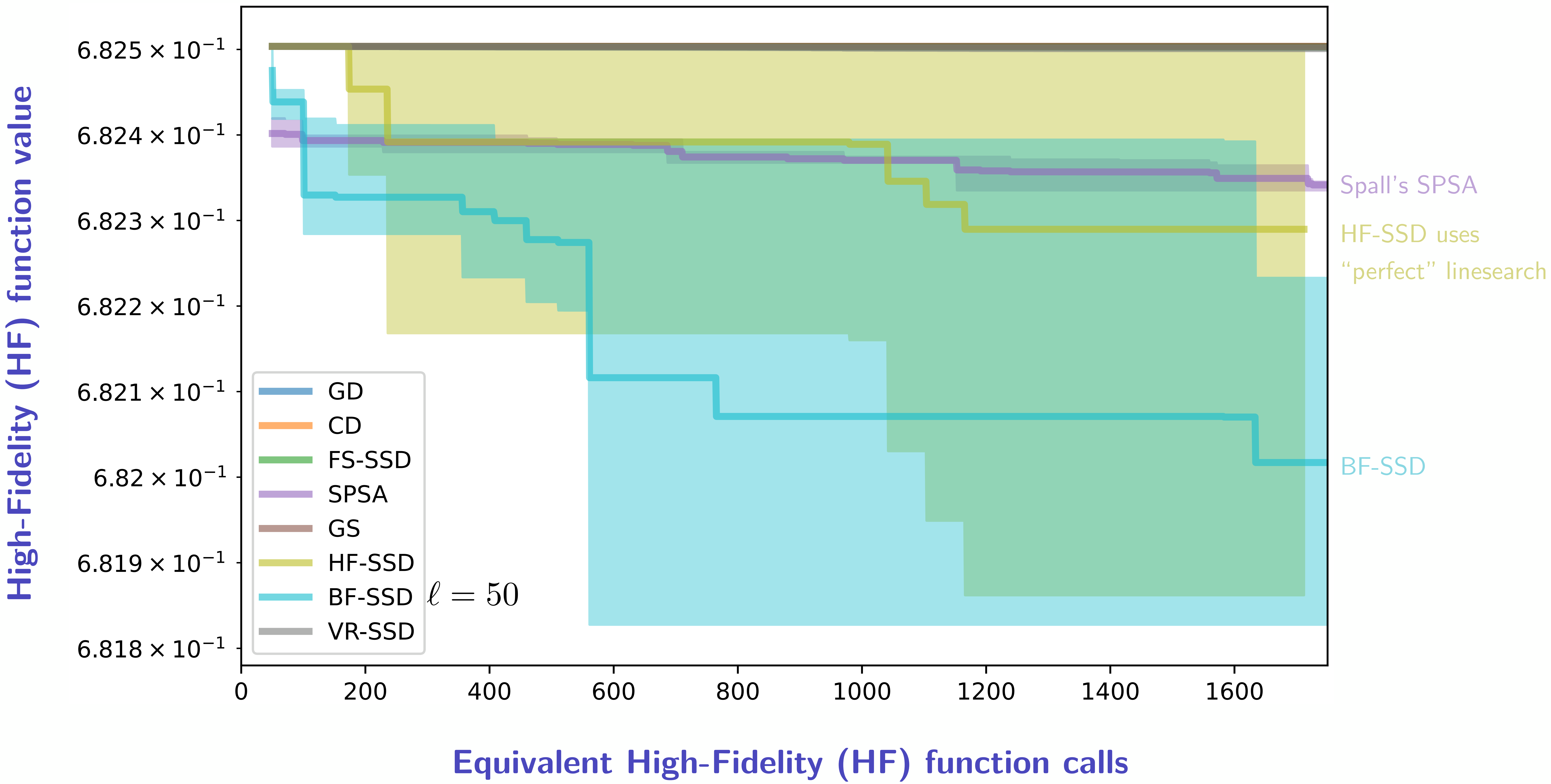
risk, replaced by empirical risk for training

$$f(\boldsymbol{x}) = \frac{1}{10} \sum_{i=1}^{10} \operatorname{CE}(f_c(\operatorname{cat}[\boldsymbol{x}, f_{\text{token}}(\boldsymbol{z}_i)]), y_i)$$

f uses a sample size of 10 High-Fidelity

f^{low} uses a sample size of 2 Low-Fidelity

ML bifidelity example 2: Results



Part 2: 2nd order methods

Joint project with Cooper Simpson (U Washington)

Learning objectives of this talk

- 0th order optimization / “derivative-free optimization”
  - Introduce a class of 0th order optimization methods
  - Argue that **stepsize** selection is a key issue
  - Show some ML **examples** where these methods make sense

- 2nd order optimization
  - Introduce a variant of Newton’s method
  - Demonstrate why non-convexity has to be taken more seriously
  - Argue that **linear algebra** is a key issue

2nd order methods for machine learning

Traditionally, 2nd order methods not considered for ML tasks like training neural nets:

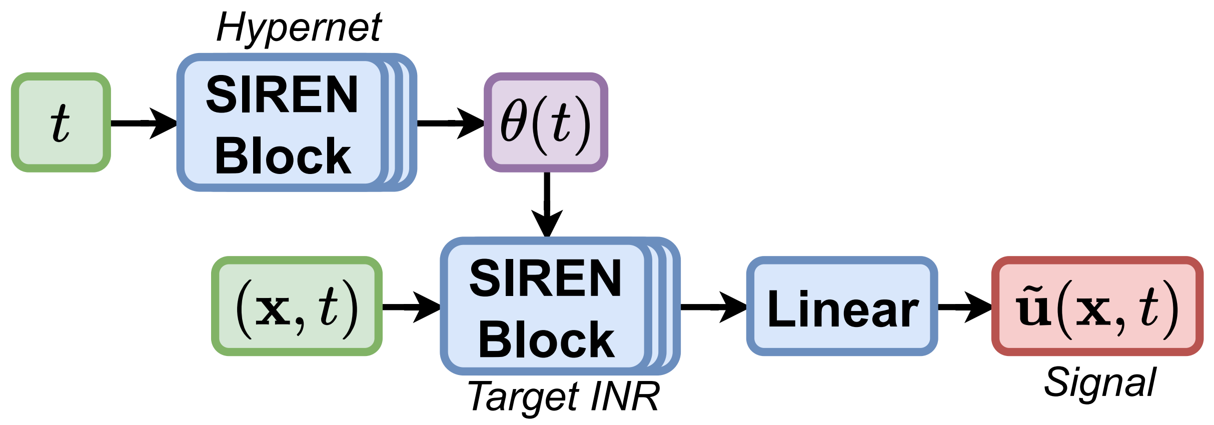
- for models with billions of parameters, the linear algebra per step is too expensive
- harder (not impossible) to do standard ML tricks like mini batch sampling
- sometimes perceived as converging to less desirable solutions (less generalizability??)

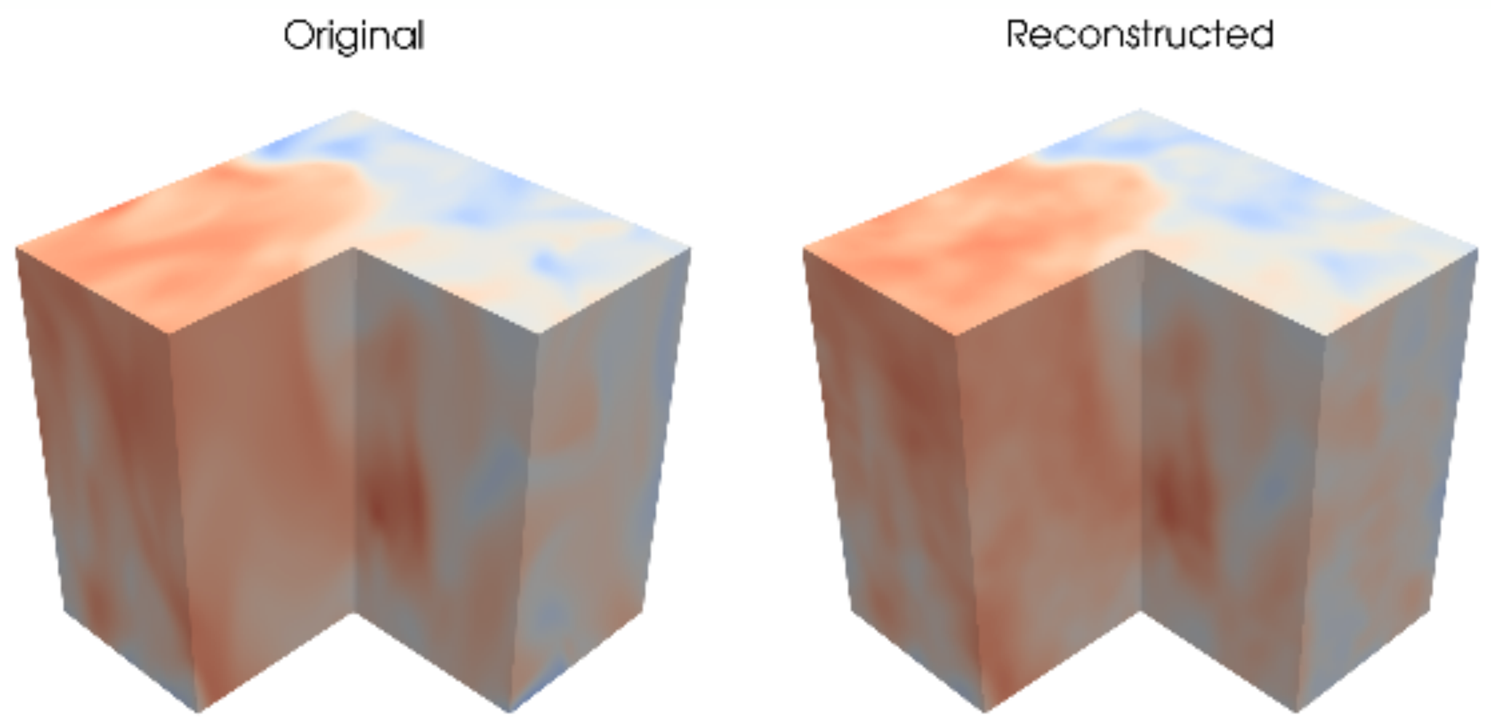
... but they have their place:

- quasi-Newton variants are common for training physics-informed neural networks
- there are plenty of tasks that are smaller
  - e.g., knowledge distillation of many small networks
  - for compression tasks, neural networks **must** be small (by design)

Example: knowledge distillation
or sketching
(to enable streaming training)
with Implicit Neural Representation
like SIRENs, NeRF...

joint with C. Simpson, A. Doostan





(a) Streamwise velocity INR reconstruction at 0.94% relative error and 42.1dB PSNR at 1,982x compression rate.

2nd order

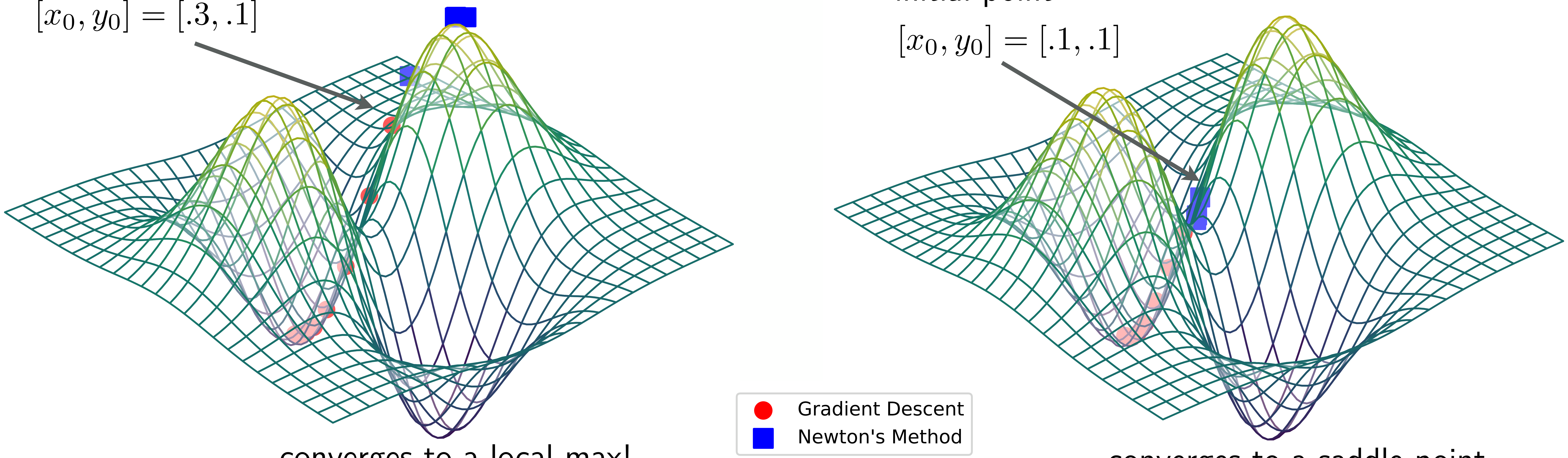
$$\min_{\boldsymbol{x}\in\mathbb{R}^d} \; f(\boldsymbol{x})$$

Prototype: **Newton's method**

$\boldsymbol{x} \leftarrow \boldsymbol{x} - \eta \boldsymbol{H}^{-1}\nabla f(\boldsymbol{x})$ where $\boldsymbol{H} = \nabla^2 f(\boldsymbol{x})$

... but Newton's method need not converge, or for non convex problems, may converge to the **wrong point**

non convex example $f(x,y) = \frac{1}{2}\left(x^2 - y^2\right)\cdot e^{-(x^2+y^2)/\sigma^2}$



initial point $[x_0, y_0] = [.3, .1]$

... converges to a local max!

initial point $[x_0, y_0] = [.1, .1]$

... converges to a saddle point

thanks to Michael McCabe for assistance with graphics

2nd order method: Saddle-Free Newton

Prototype: **Newton's method**

$\boldsymbol{x} \leftarrow \boldsymbol{x} - \eta \boldsymbol{H}^{-1}\nabla f(\boldsymbol{x})$ where $\boldsymbol{H} = \nabla^2 f(\boldsymbol{x})$

$$\boldsymbol{H} = \boldsymbol{V} \begin{bmatrix} \lambda_1 & 0 & & \cdots \\ 0 & \lambda_2 & & \\ \vdots & & \ddots & 0 \\ & & 0 & \lambda_n \end{bmatrix} \boldsymbol{V}^\top$$

How to prevent convergence to bad stationary points?

In addition to a line search, a common strategy is to $\boxed{\text{replace } \boldsymbol{H} \text{ with } |\boldsymbol{H}|}$

$$|\boldsymbol{H}| = \boldsymbol{V} \begin{bmatrix} |\lambda_1| & 0 & & \cdots \\ 0 & |\lambda_2| & & \\ \vdots & & \ddots & 0 \\ & & 0 & |\lambda_n| \end{bmatrix} \boldsymbol{V}^\top$$

Sometimes called "**Saddle-Free Newton**" (SFN)

It's a heuristic, and simpler/cheaper than a proper* treatment of nonconvexity

* e.g., trust-region methods or cubic regularization

Research question: can we modify SFN to make it work well,

and analyze rigorously as well? Can we quantify how it avoids saddle points?

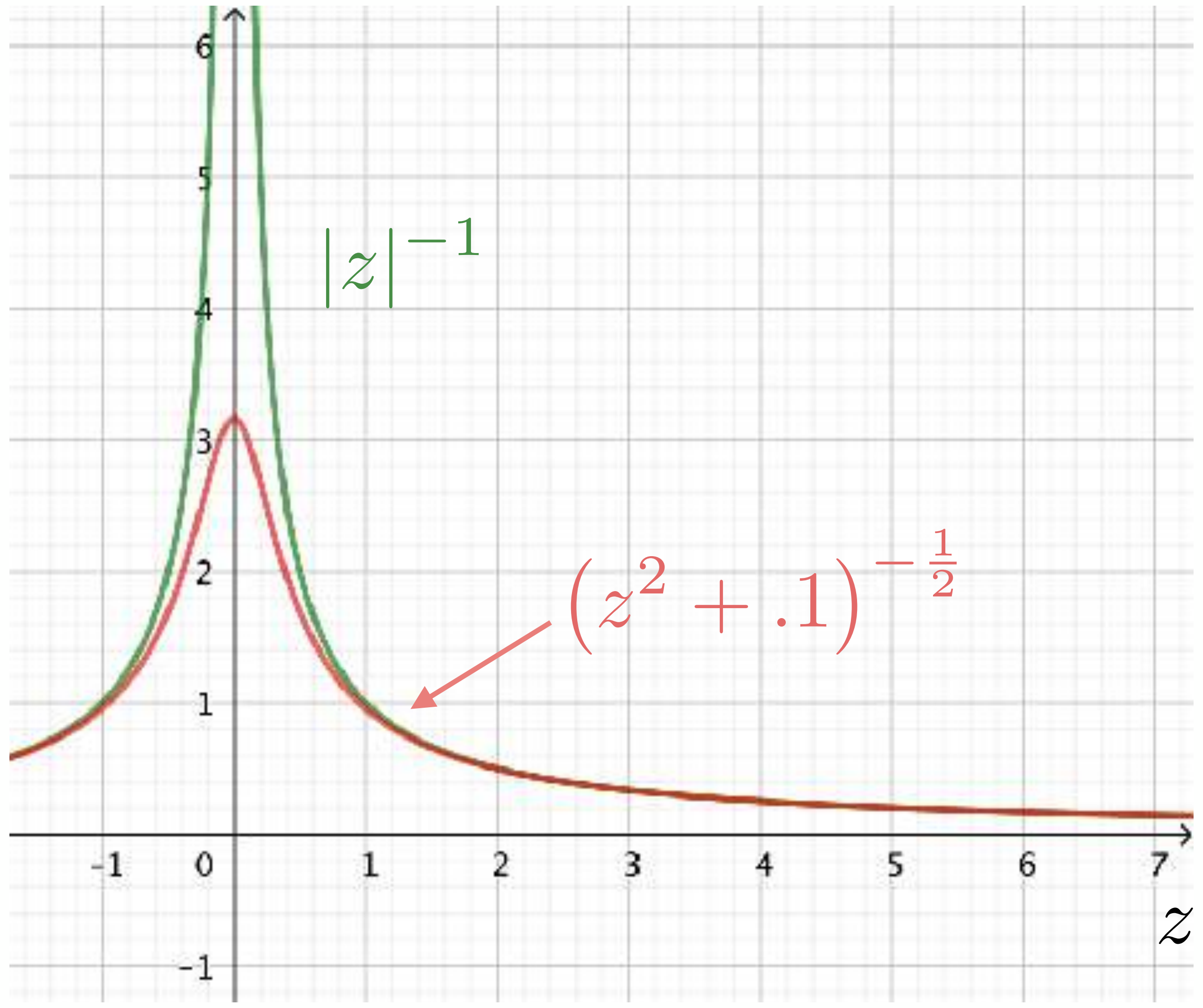
See "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization", by Dauphin, Pascanu, Gulcehre, Cho, Ganguli, Bengio (NIPS 2014) and Nocedal and Wright (2004)

Our proposed method: Regularized Saddle-Free Newton (RSFN)

$$\boldsymbol{H} = \nabla^2 f(\boldsymbol{x})$$

Newton $\boldsymbol{x} \leftarrow \boldsymbol{x} - \eta \boldsymbol{H}^{-1} \nabla f(\boldsymbol{x})$

Saddle-Free Newton $\boldsymbol{x} \leftarrow \boldsymbol{x} - \eta \left|\boldsymbol{H}\right|^{-1} \nabla f(\boldsymbol{x})$



Regularized Saddle-Free Newton $\boldsymbol{x} \leftarrow \boldsymbol{x} - \eta \left(\boldsymbol{H}^2 + \lambda \boldsymbol{I}\right)^{-\frac{1}{2}} \nabla f(\boldsymbol{x})$ since $\left|\boldsymbol{H}\right| = \lim_{\lambda \to 0} \left(\boldsymbol{H}^2 + \lambda \boldsymbol{I}\right)^{\frac{1}{2}}$

why regularize? easier linear algebra (**smoother**) and needed for analysis

Our method (and context)

Method	$\widehat{\boldsymbol{H}}$	η	λ	Global Convergence	Non-convex	Fast Impl.	Comments
Newton	$\nabla^2 f(\boldsymbol{x})$	1	0	✗	—	✓	—
Reg. Newton [3], [5]	$\nabla^2 f(\boldsymbol{x}) + \lambda \mathbf{I}$	1	$\sqrt{M\\|\nabla f(\boldsymbol{x})\\|}$	✓	✗	✓	—
AICN [4]	$\nabla^2 f(\boldsymbol{x})$	$\frac{-1+\sqrt{1+2G}}{G}$	0	✓	✗	✓	G is a local smoothness constant
SFN [2]	$\left\lvert\nabla^2 f(\boldsymbol{x})\right\rvert$	$(0,1]$	0	✗	✓	✗	—
LRSFN [7]	$\left\lvert\nabla^2 f(\boldsymbol{x})\right\rvert_r + \lambda \mathbf{I}$	$(0,1]$	$(0,1]$	✗	✓	✓	Rank-r approximation
Cubic Newton [6, 1]	$\nabla^2 f(\boldsymbol{x}) + \lambda \mathbf{I}$	1	$M\\|\boldsymbol{x} - \boldsymbol{x}_k\\|$	✓	✓	✓	Requires solving complicated sub-problem
NCN [8]	$\left\lvert\nabla^2 f(\boldsymbol{x})\right\rvert_m$	1	0	✓	✓	✗	Small eigenvalues replaced by m, requires complex perturbations
RSFN (Ours)	$\left(\left(\nabla^2 f(\boldsymbol{x})\right)^2 + \lambda \mathbf{I}\right)^{1/2}$	$(0,\infty)$	$M\\|\nabla f(\boldsymbol{x})\\|$	✓	✓	✓	Linesearch when M is unknown

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - \eta \widehat{\boldsymbol{H}}^{-1} \nabla f(\boldsymbol{x})$$

M is the Hessian Lipschitz constant

[1] Coralia Cartis, Nicholas IM Gould, and Philippe L Toint. "Adaptive cubic regularisation methods for unconstrained optimization. Part I: motivation, convergence and numerical results". In: *Mathematical Programming* 127.2 (2011), pp. 245–295.

[2] Yann N Dauphin et al. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization". In: *Advances in neural information processing systems* 27 (2014).

[3] Nikita Doikov and Yurii Nesterov. "Gradient regularization of Newton method with Bregman distances". In: *Mathematical Programming* (2023), pp. 1–25.

[4] Slavomír Hanzely et al. "A Damped Newton Method Achieves Global $\mathcal{O}(1/k^2)$ and Local Quadratic Convergence Rate". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 25320–25334.

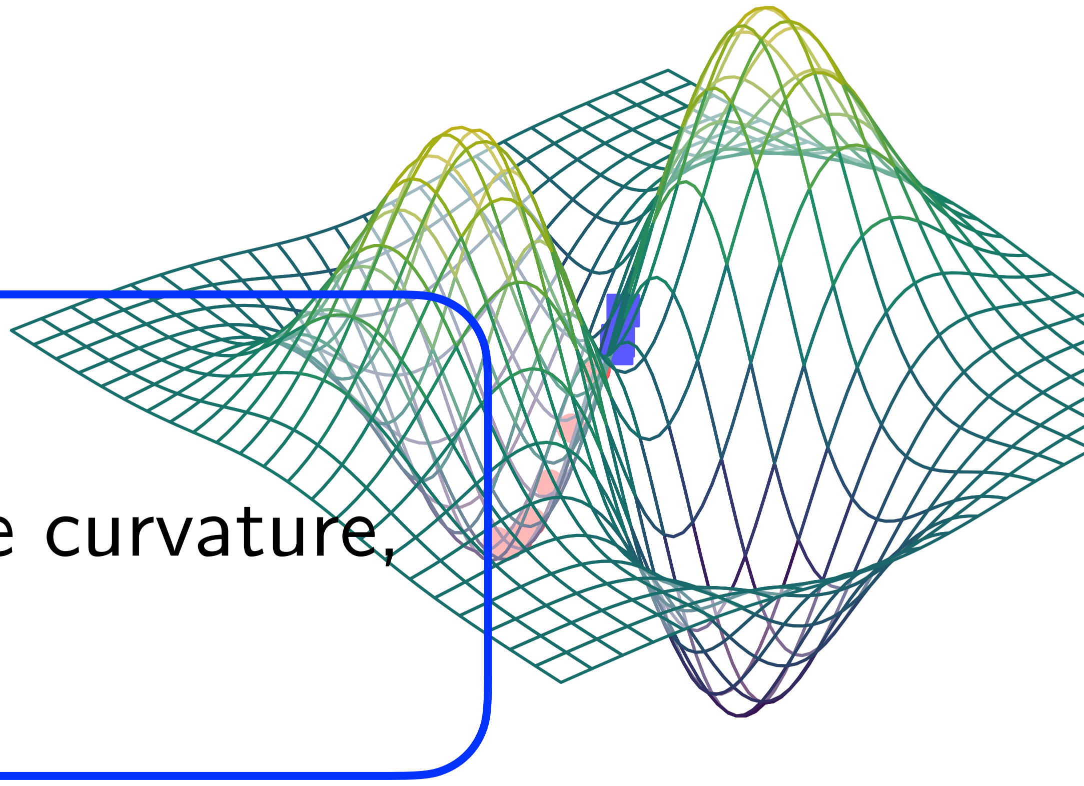
[5] Konstantin Mishchenko. "Regularized Newton Method with Global $\mathcal{O}(1/k^2)$ Convergence". In: *SIAM Journal on Optimization* 33.3 (2023), pp. 1440–1462.

[6] Yurii Nesterov and B.T. Polyak. "Cubic Regularization of Newton Method and its Globabl Performance". In: *Mathematical Programming* 108.1 (2006), pp. 177–205. DOI: 10.1007/s10107-006-0706-8.

[7] Thomas O'Leary-Roseberry, Nick Alger, and Omar Ghattas. "Low rank saddle free Newton: A scalable method for stochastic nonconvex optimization". In: *arXiv* (2020). DOI: 10.48550/arXiv.2002.02881.

[8] Santiago Paternain, Aryan Mokhtari, and Alejandro Ribeiro. "A Newton-based method for nonconvex optimization with fast evasion of saddle points". In: *SIAM Journal on Optimization* 29.1 (2019), pp. 343–368.

Theory results: it avoids saddle points



Definition: **Strict Saddle Point**

A strict saddle point $\boldsymbol{x}$ is a critical point, i.e., $\nabla f(\boldsymbol{x}) = \mathbf{0}$, where there is at least one direction of negative curvature, so the smallest eigenvalue of $\nabla f^2(\boldsymbol{x})$ is strictly less than 0.

If all saddle points are strict, then all second-order stationary points are local minima

Theorem: **Saddle Avoidance**

Under some assumptions, the RSFN iteration **avoids strict saddle points** with probability 1, assuming the initial point is chosen randomly according to any absolutely continuous distribution

Proof sketch:

At a strict saddle point, fixed point map is a local diffeomorphism and has at least one eigenvalue strictly larger than 1

Use **stable manifold theorem** to guarantee not converging to any particular strict saddle (i.e., probability 0)

Use Lindelöf's lemma to guarantee countable set of manifolds covering all such manifolds

(and measure of countable union of measure zero sets is zero)

Theory results: it converges

Theorem: **Global Convergence**

Under some assumptions, the RSFN iteration converges to a first-order stationary point

Theorem: **Convex Convergence Rate**

Under some assumptions, the RSFN iteration converges to an ϵ-optimal point in $\mathcal{O}(\sqrt{\epsilon})$ iterations

Theorem: **Local Super-Linear Convergence**

Under some assumptions, the RSFN iteration will converge super-linearly in the neighborhood of a second-order stationary point

Computing the Newton step

Solve $\left(\boldsymbol{H}^2 + \lambda \mathbf{I}\right)^{-1/2} \boldsymbol{g}$ where $\boldsymbol{H} = \nabla^2 f(\boldsymbol{x})$

We do this "matrix free"

Oracle: Hessian Vector Product (HVP)

$$\boldsymbol{v} \mapsto \boldsymbol{H}\boldsymbol{v}$$

i.e., $\nabla^2 f(\boldsymbol{x})\boldsymbol{v} = \frac{d}{dt}\varphi(t)\Big|_{t=0}$ where $\varphi : \mathbb{R}^1 \to \mathbb{R}^n$, $\varphi(t) = \nabla f(\boldsymbol{x} + t\boldsymbol{v})$ so this can be done efficiently via **automatic differentiation**

Long history of "matrix-free" Newton, "Newton-CG", "Newton-Krylov", "inexact Newton", etc.

Dagréou, Ablin, Vaiter, Moreau '24: https://iclr-blogposts.github.io/2024/blog/bench-hvp/
Forward-over-reverse, reverse-over-reverse, reverse-over-forward
Good (and constantly improving) implementations exist in Python (PyTorch/jax), julia, ...

Method 1: Shifted Lanczos Quadrature

$$\left(\boldsymbol{H}^2 + \lambda \mathbf{I}\right)^{-1/2} \boldsymbol{g} = \frac{2}{\pi} \int_0^\infty \left(\left(t^2 + \lambda\right) \mathbf{I} + \boldsymbol{H}^2\right)^{-1} \boldsymbol{g} \; dt$$

via Cauchy integral representation

cf. N. Higham, *Functions of matrices: theory and computation* (SIAM, 2008)

comment: Newton-Schulz (cf. N. Higham) doesn't seem to be applicable

Method 1: Shifted Lanczos Quadrature

$$\left(\boldsymbol{H}^2 + \lambda \mathbf{I}\right)^{-1/2} \boldsymbol{g} = \frac{2}{\pi} \int_0^\infty \left(\left(t^2+\lambda\right)\mathbf{I} + \boldsymbol{H}^2\right)^{-1} \boldsymbol{g}\; dt$$

via Cauchy integral representation

cf. N. Higham, *Functions of matrices: theory and computation* (SIAM, 2008)

Solve via quadrature rule, like Gauss-Laguerre; or, make Cayley transformation $s = -\beta \dfrac{1-t}{1+t}$

$$\left(\boldsymbol{H}^2 + \lambda \mathbf{I}\right)^{-1/2} \boldsymbol{g} = \frac{2\beta^{1/2}}{\pi} \int_{-1}^{1} \underbrace{(1-s)^{-1/2}(1+s)^{-1/2}}_{\omega(s)} \left(\left((\lambda-\beta)s + (\lambda+\beta)\right)\mathbf{I} + (1+s)\boldsymbol{H}^2\right)^{-1} \boldsymbol{g}\; ds$$

and use Gauss-Chebyshev weights (of the first kind), with standard nodes and weights $s_i, \omega_i \quad \forall i = 1, \ldots, N$

(want weights non-negative to guarantee descent direction)

Method 1: Shifted Lanczos Quadrature

$$\left(\boldsymbol{H}^2 + \lambda \mathbf{I}\right)^{-1/2} \boldsymbol{g} = \frac{2}{\pi} \int_0^\infty \left(\left(t^2 + \lambda\right) \mathbf{I} + \boldsymbol{H}^2\right)^{-1} \boldsymbol{g} \; dt$$

via Cauchy integral representation

cf. N. Higham, *Functions of matrices: theory and computation* (SIAM, 2008)

Solve via quadrature rule, like Gauss-Laguerre; or, make Cayley transformation $s = -\beta \dfrac{1-t}{1+t}$

$$\left(\boldsymbol{H}^2 + \lambda \mathbf{I}\right)^{-1/2} \boldsymbol{g} = \frac{2\beta^{1/2}}{\pi} \int_{-1}^{1} \underbrace{(1-s)^{-1/2}(1+s)^{-1/2}}_{\omega(s)} \left(\left((\lambda-\beta)s + (\lambda+\beta)\right) \mathbf{I} + (1+s)\boldsymbol{H}^2\right)^{-1} \boldsymbol{g} \; ds$$

and use Gauss-Chebyshev weights (of the first kind), with standard nodes and weights $s_i, \omega_i \quad \forall i = 1, \ldots, N$

(want weights non-negative to guarantee descent direction)

Setting $\mu_i = (\lambda - \beta)s_i + (\lambda + \beta)$ gives

$$\boxed{\left(\boldsymbol{H}^2 + \lambda \mathbf{I}\right)^{-1/2} \boldsymbol{g} \approx \frac{2\beta^{1/2}}{\pi} \sum_{i=1}^{N} \omega_i \left(\mu_i \mathbf{I} + (1+s_i)\boldsymbol{H}^2\right)^{-1} \boldsymbol{g}}$$

solve with Krylov-subspace method

We have a sequence of **shifted** matrices, so can re-use computations!

$$\mathcal{K}_k(\boldsymbol{H}, \boldsymbol{v}) = \operatorname{span}\left\{\boldsymbol{v}, \boldsymbol{H}\boldsymbol{v}, \boldsymbol{H}^2\boldsymbol{v}, \ldots, \boldsymbol{H}^{k-1}\boldsymbol{v}\right\}$$

Shift invariance of subspaces:

$$\mathcal{K}_k(s\boldsymbol{H} + \mu\mathbf{I}, \boldsymbol{v}) = \mathcal{K}_k(\boldsymbol{H}, \boldsymbol{v})$$

Use `Krylov.jl` (Montoison and Orban), inspired by Dussault, Migot and Orban's ARC code

Method 2: Nystrom sketching / randomized linear algebra

Step 1: draw random matrix $\boldsymbol{\Omega} \in \mathbb{R}^{d \times p},\ p \geq r$ orthonormal columns, isotropic column space

Method 2: Nystrom sketching / randomized linear algebra

Step 1: draw random matrix $\boldsymbol{\Omega} \in \mathbb{R}^{d \times p},\ p \geq r$ orthonormal columns, isotropic column space

Step 2: compute sketch $\boldsymbol{Y} = \boldsymbol{H}\boldsymbol{\Omega}$ via p HVPs

Step 3: linear algebra postprocessing

$$\widehat{\boldsymbol{H}}^{\mathrm{Nys}} = \boldsymbol{Y}\left(\boldsymbol{\Omega}^\top \boldsymbol{Y}\right)^\dagger \boldsymbol{Y}$$

$$\widehat{\boldsymbol{H}}_r = [[\widehat{\boldsymbol{H}}^{\mathrm{Nys}}]]_r$$

(careful numerical implementation not shown)

$$\left(\boldsymbol{H}^2 + \lambda \mathbf{I}\right)^{-1/2} \boldsymbol{g} \approx \underbrace{\left(\widehat{\boldsymbol{H}}_r^2 + \lambda \mathbf{I}\right)}_{\text{cheap}}{}^{-1/2} \boldsymbol{g}$$

cf. Tropp, Yurtsever, Udell, Cevher '17
(and Frangella, Rathore, Zhao, Martinsson...)

Downside: approximates Hessian only, ignores RHS

Hybrid variant: use this as preconditioner for Krylov subspace based method

`github.com/tjdiamandis/RandomizedPreconditioners.jl`

Method 3: Lanczos Function Approximation

The Lanczos algorithm builds a tridiagonal approximation of a symmetric matrix using matrix-vector multiplies

$$\boldsymbol{H} \approx \boldsymbol{Q}_k \boldsymbol{T}_k \boldsymbol{Q}_k^\top$$

most commonly used to find eigenvalue decompositions

closely related to conjugate gradient

Method 3: Lanczos Function Approximation

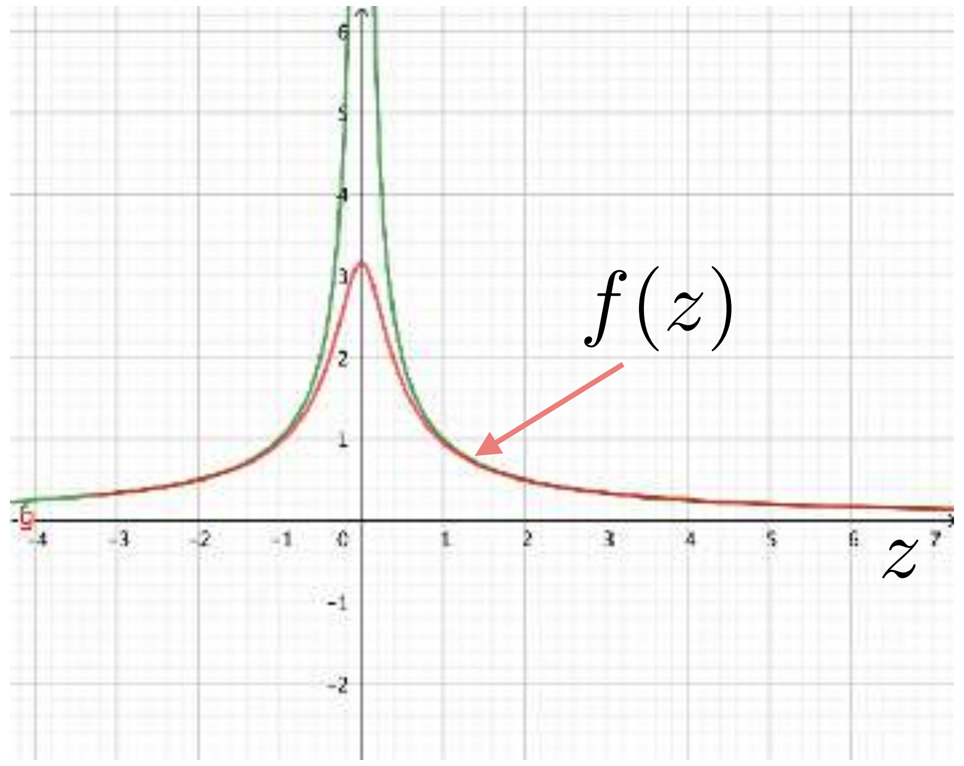
The Lanczos algorithm builds a tridiagonal approximation of a symmetric matrix using matrix-vector multiplies

$$\boldsymbol{H} \approx \boldsymbol{Q}_k \boldsymbol{T}_k \boldsymbol{Q}_k^\top$$

most commonly used to find eigenvalue decompositions

closely related to conjugate gradient

We want to apply this function spectrally $f(z) = (z^2 + \lambda)^{-1/2}$



So our approximation is $\left(\boldsymbol{H}^2 + \lambda \mathbf{I}\right)^{-1/2} \boldsymbol{g} \approx \boldsymbol{Q}_k f(\boldsymbol{T}_k) \boldsymbol{Q}_k^\top \boldsymbol{g}$

$f(\boldsymbol{T}_k)$ done via eigenvalue decomposition of $\boldsymbol{T}_k$(cheap, since tri-diagonal)

Of all 3 methods, this gives us the best results

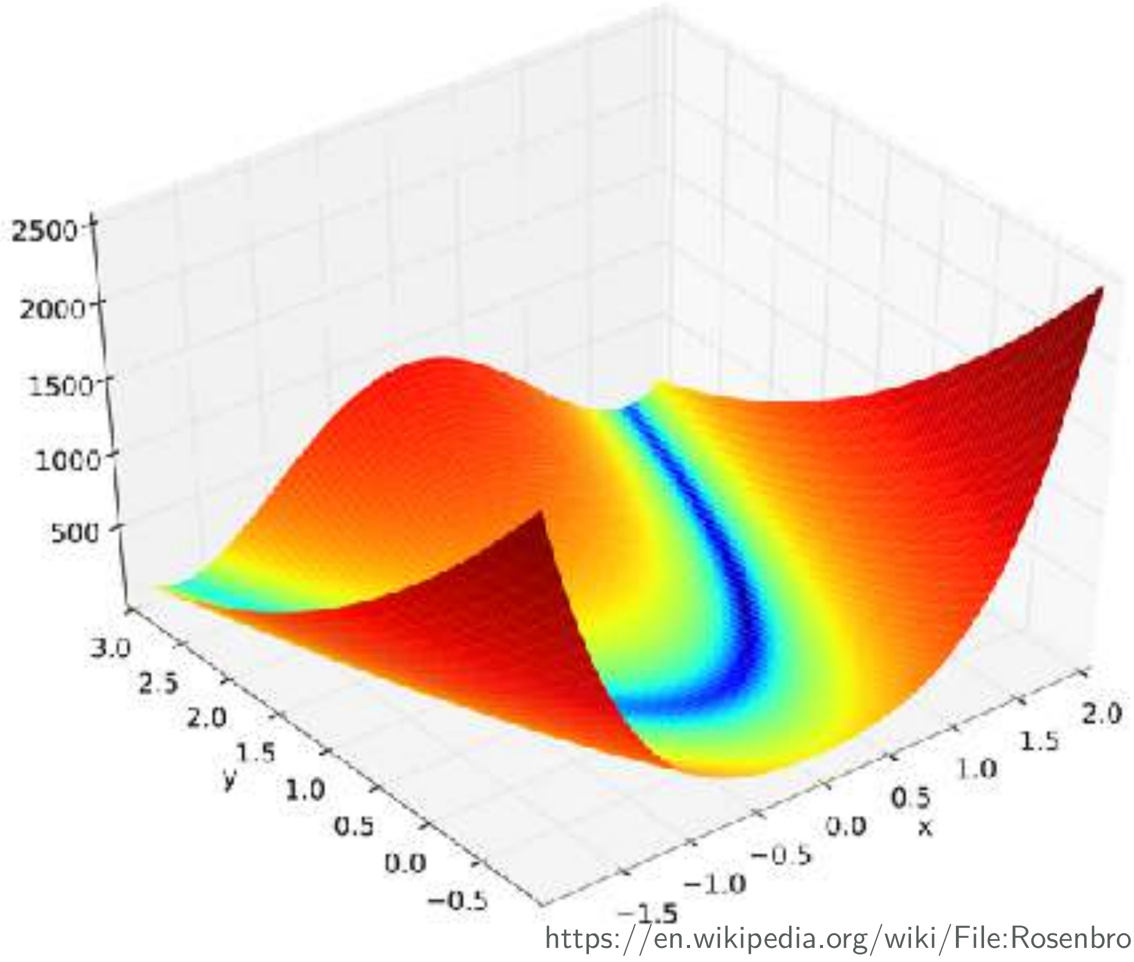
— possibly related to avoiding squaring the condition number?

We have some preliminary theory to suggest accuracy is similar to that of solving equation (with Lanczos)

cf. recent work by Chen, Tyler, Amsel, Greenbaum, Musco, Musco

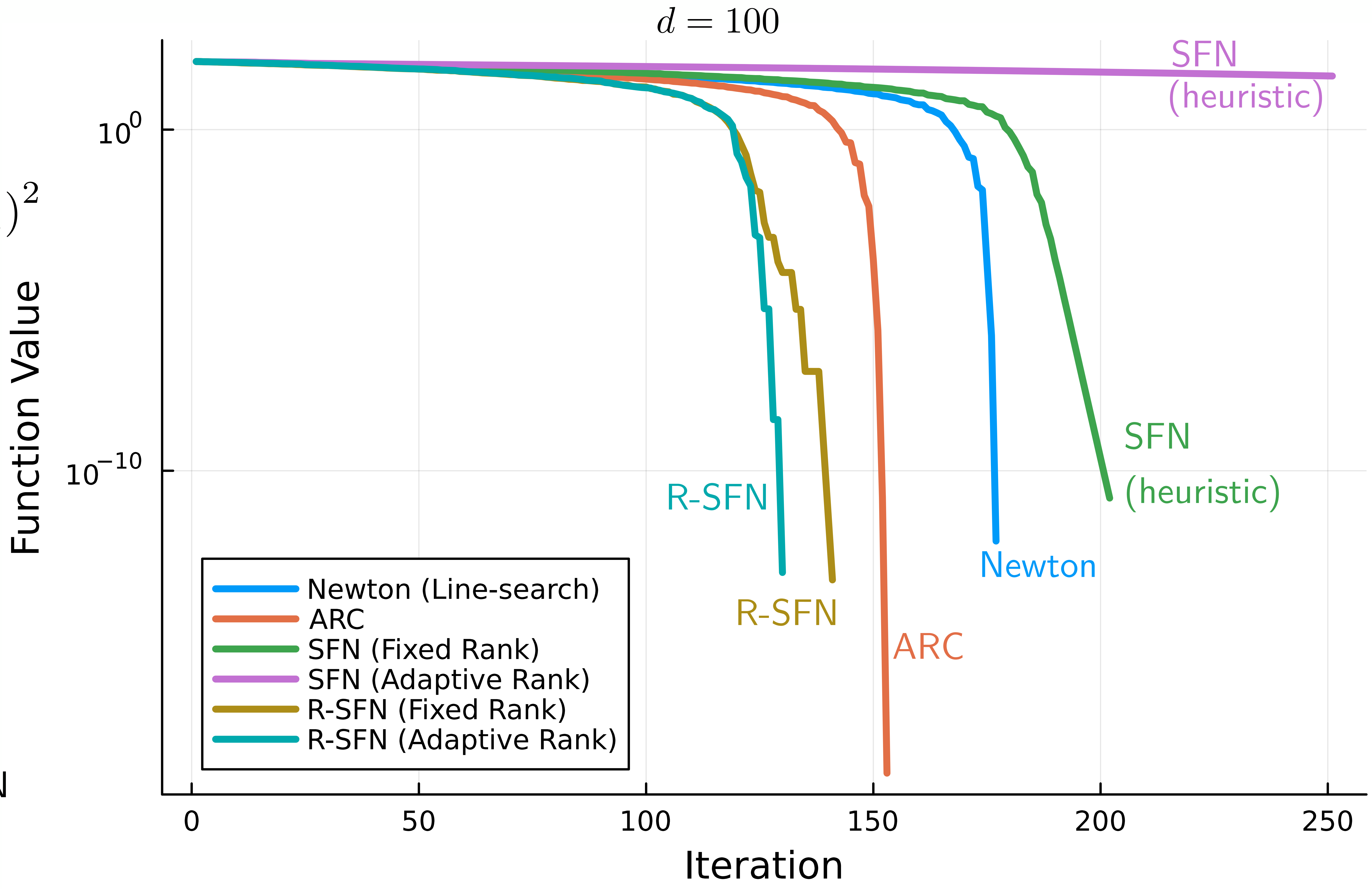
Experiments: Rosenbrock

$$f(\boldsymbol{x}) = \sum_{i=1}^{d-1} 100\left(\boldsymbol{x}_{i+1} - \boldsymbol{x}_i^2\right)^2 + (1 - \boldsymbol{x}_i)^2$$



https://en.wikipedia.org/wiki/File:Rosenbrock_function.svg

Results: RSFN usually similar to ARC and SFN

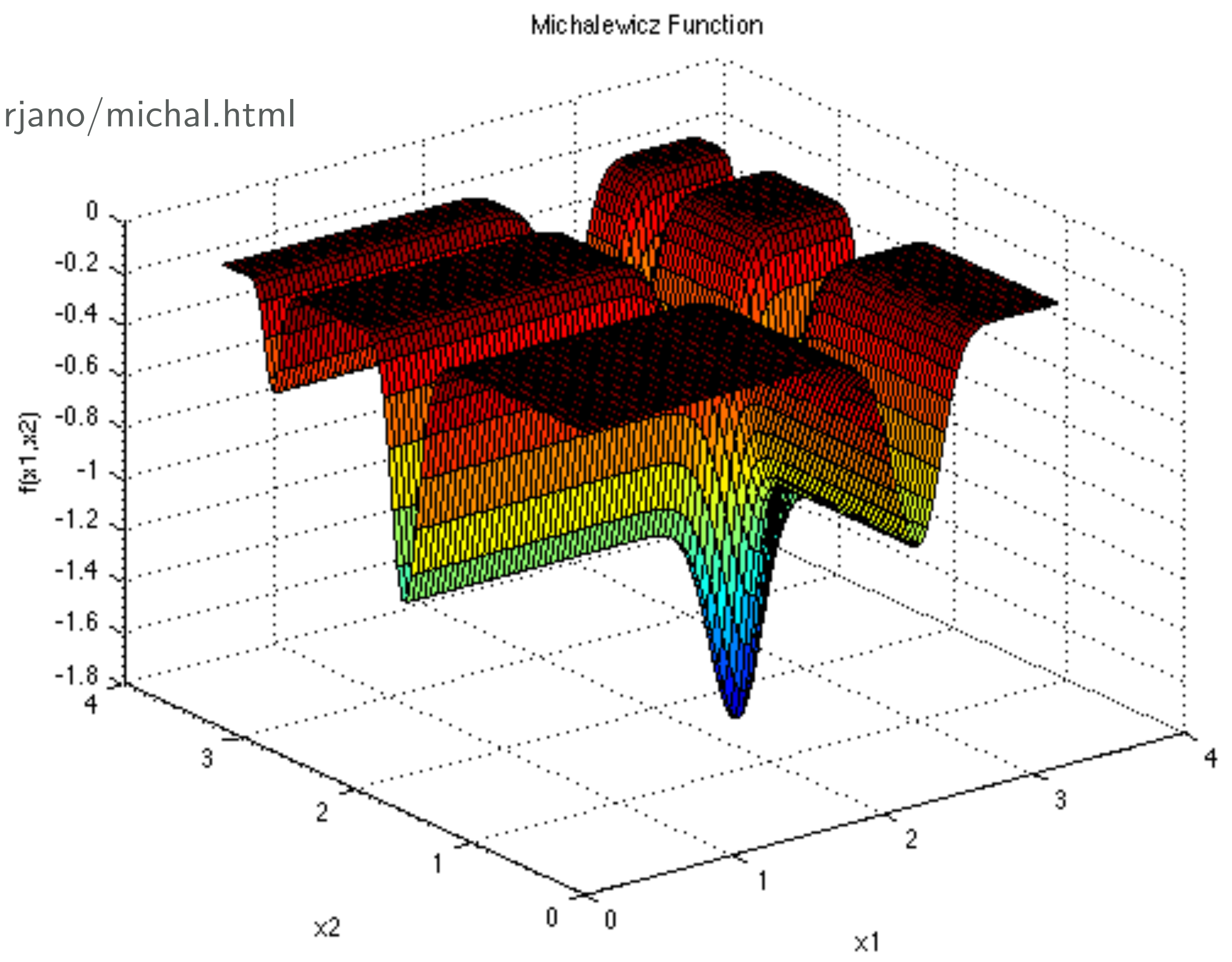


Code: Julia package
https://github.com/rs-coop/QuasiNewton.jl
and experiment code at .../rs-coop/R-SFN

ARC implementation modified from:

Jean-Pierre Dussault, Tangi Migot, and Dominique Orban. "Scalable adaptive cubic regularization methods". In: *Mathematical Programming* (2023), pp. 1–35.

Experiments: Michalewicz function



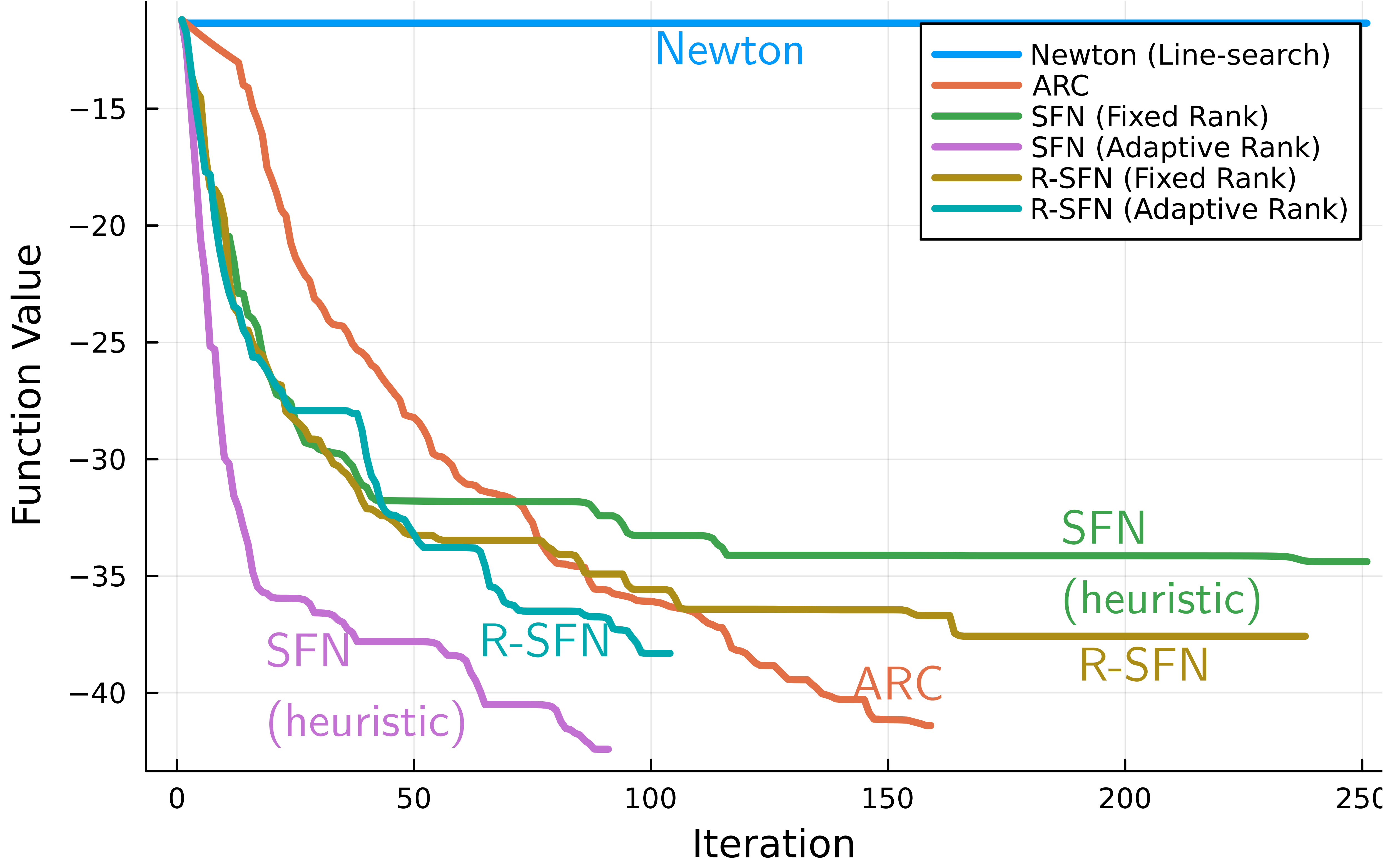
https://www.sfu.ca/~ssurjano/michal.html

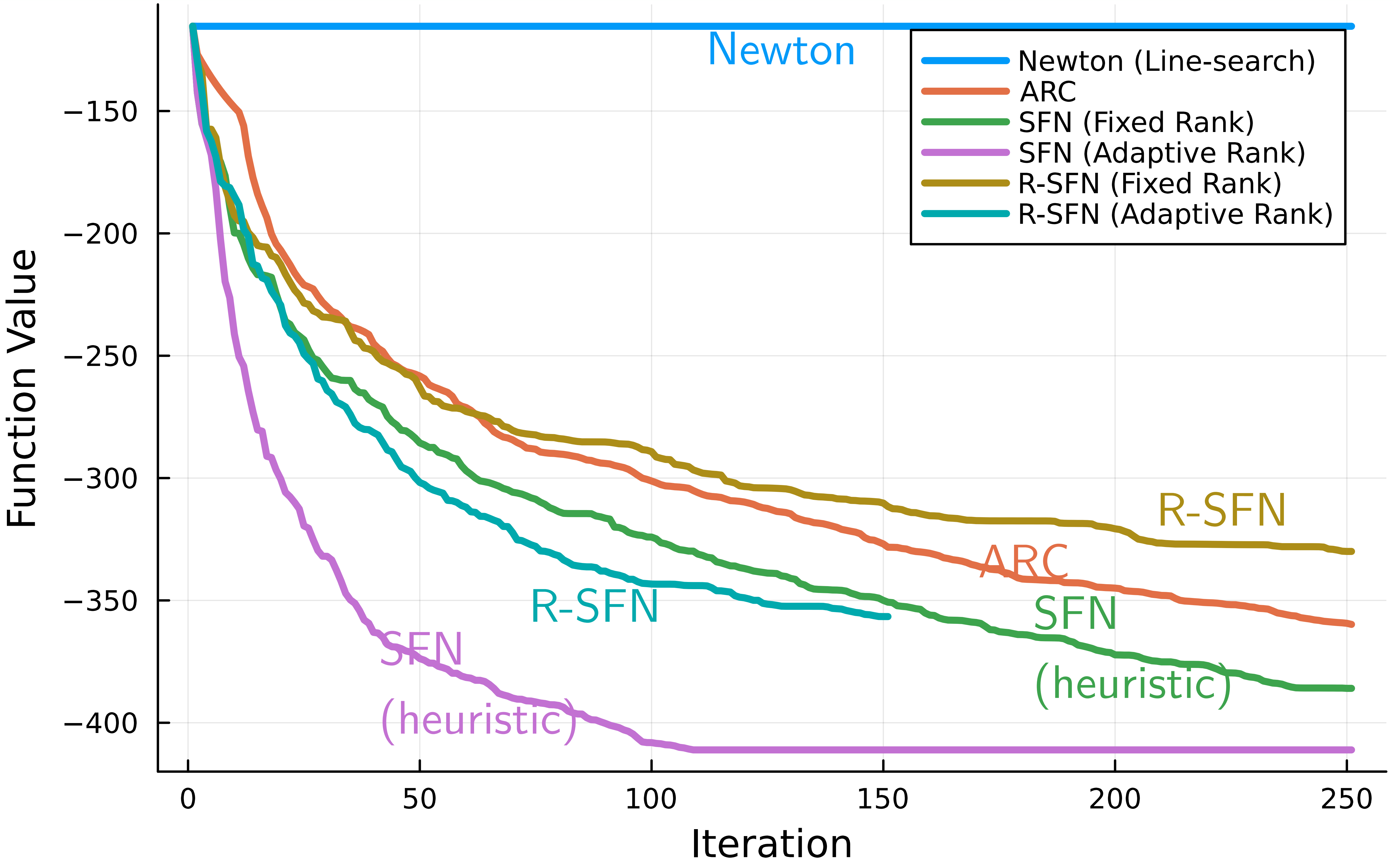
$$f(\boldsymbol{x}) = -\sum_{i=1}^{d} \sin(x_i)\sin^{20}\left(\frac{i x_i^2}{\pi}\right)$$

Very non convex, $d!$ local minima

Sometimes we do much better than ARC

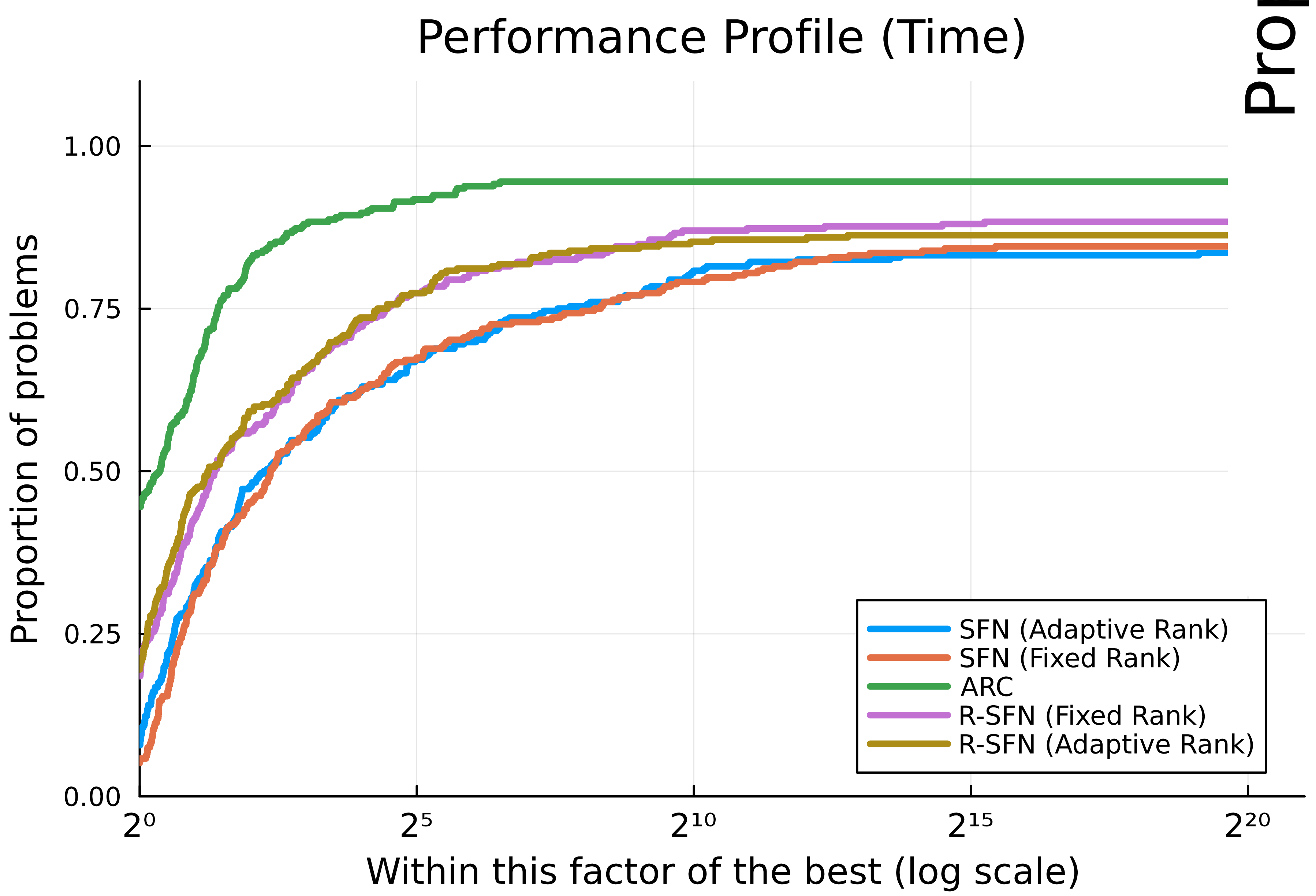
$d = 1000$

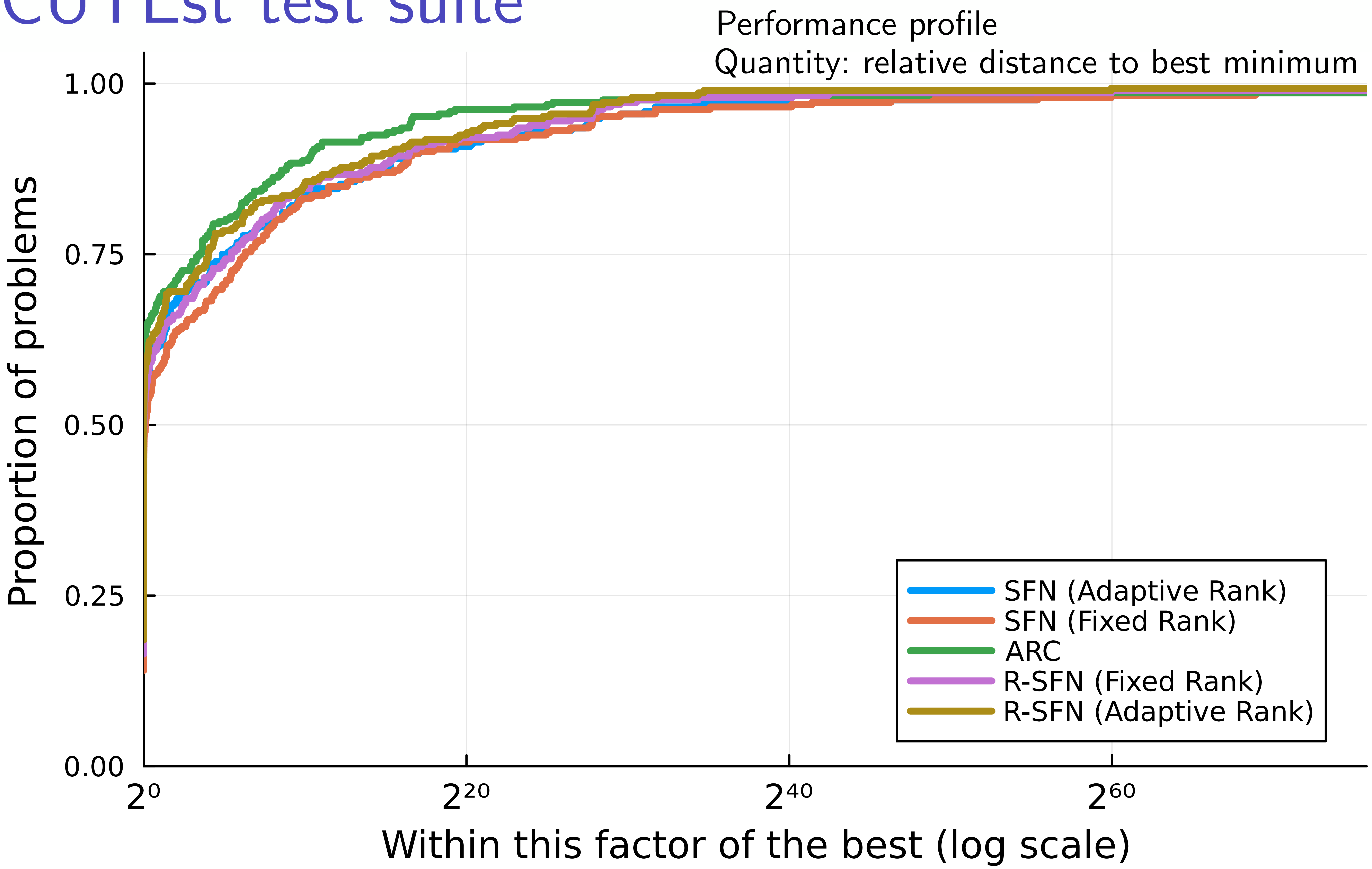




Appropriate subset of CUTEst test suite

Results: ARC is usually a little better (depending on the exact metric) but RSFN is comparable





CUTEst.jl: Julia's CUTEst interface. https://github.com/JuliaSmoothOptimizers/CUTEst.jl

Conclusion

- 0th and 2nd order methods have their roles
- stepsize selection (and/or line search) is important
- multi fidelity is useful

Code: Julia package
https://github.com/rs-coop/QuasiNewton.jl
and experiment code at .../rs-coop/R-SFN