INTERNATIONAL CONFERENCE ON CONTINUOUS OPTIMIZATION

Stochastic Subspace Descent

Stochastic gradient-free* optimization,

with applications to PDE-constrained optimization

Stephen Becker

joint work with David Kozak, Alireza Doostan, and Luis Tenorio

University of Colorado **Boulder**

COLORADO SCHOOL OF MINES
EARTH • ENERGY • ENVIRONMENT

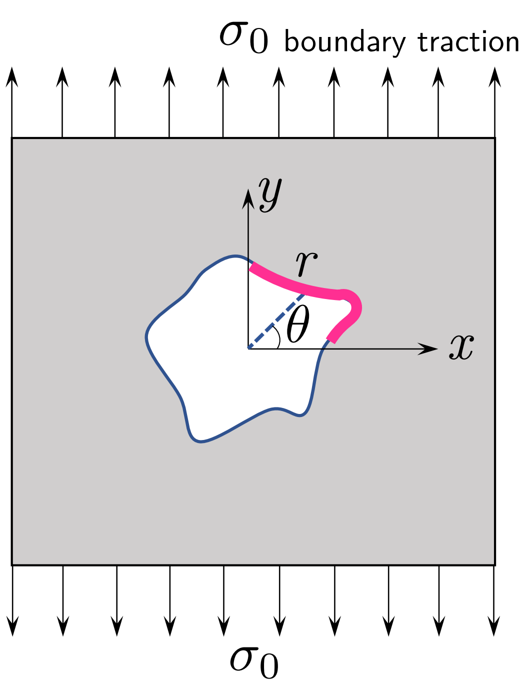
*(not *derivative* free)

Motivating Application: shape optimization

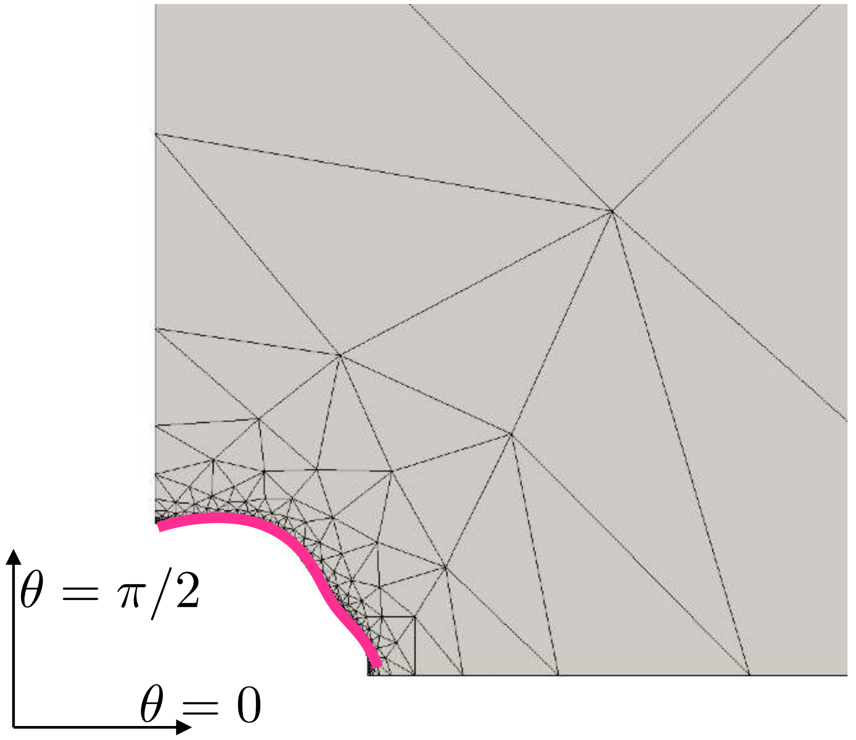
Forward problem: find vertical stress σ_y

linear elasticity PDE

(boundary conditions depend on shape of the hole)



conforming finite element mesh, used in FEniCS

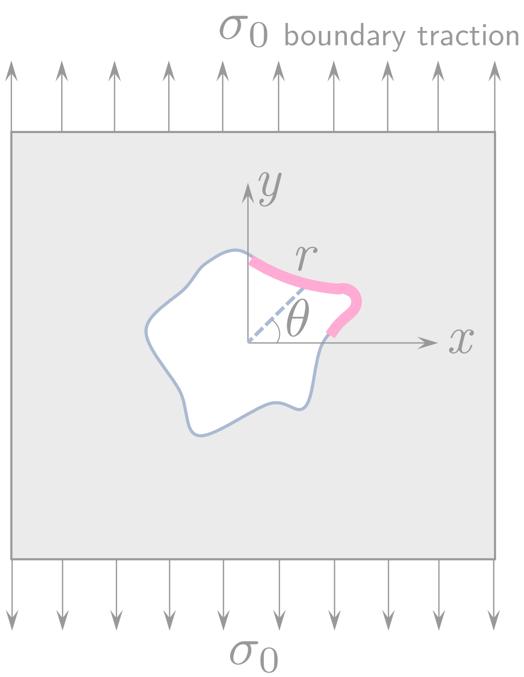


Motivating Application: shape optimization

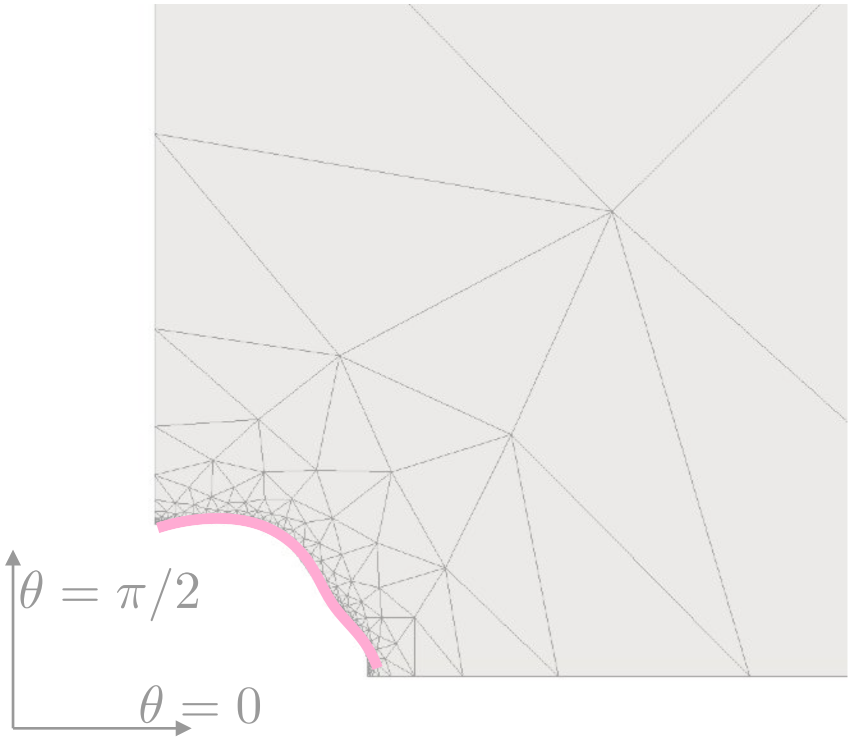
Forward problem: find vertical stress σ_y

linear elasticity PDE

(boundary conditions depend on shape of the hole)



conforming finite element mesh, used in FEniCS



Inverse problem: what shape minimizes the vertical stress?

parameterize the shape of the hole as follows, which automatically enforces a constant area constraint

$$r(\theta) = \frac{1}{2\pi} + \delta \sum_{k=0}^{d/2-1} \frac{1}{\sqrt{2k+1}} \left(x_{2k+1} \sin((2k+1)\cdot\theta) + x_{2k+2} \cos((2k+1)\cdot\theta) \right)$$

optimization variable:

$x \in \mathbb{R}^d$

Generic PDE-constrained optimization

implicitly saying that u solves the PDE

$$\boxed{\min_{u,x}\ \mathcal{L}(u) \quad \text{subject to} \quad \phi(u,x) = 0}$$

Examples:

$\dot{u} = \Delta u,\ u(0) = h$ "x" is the initial condition

$\ddot{u} = c^2 \Delta u,\ u(0) = h$ "x" is a parameter

$\Delta u = 0,\ u(\Gamma) = h$ "x" is the boundary condition

$\mathcal{L}(u)$ is the loss which penalizes something like:

- deviation from observations
- drag
- mass
- cost of materials
- compliance

etc.

Generic PDE-constrained optimization

implicitly saying that u solves the PDE

$$\boxed{\min_{u,x} \ \mathcal{L}(u) \quad \text{subject to} \quad \phi(u,x) = 0}$$

$$\phi(u,x) = 0 \quad \Longrightarrow \quad u = u(x)$$

Rewrite: $\boxed{\min_{x} \ f(x) \stackrel{\text{def}}{=} \mathcal{L}\left(u(x)\right)}$

… but finding the gradient is tricky:

$$\nabla f(x) = \frac{\partial \mathcal{L}}{\partial u} \cdot \frac{\partial u}{\partial x}$$

Why not just find gradients automatically?

✓The **adjoint state method** and **reverse-mode automatic differentiation** can automatically calculate gradients in about the same time (~4x) as a function evaluation

- ... so if we can evaluate $f(x)$ numerically, we can find the gradient
- (this applies if $f : \mathbb{R}^d \to \mathbb{R}$; $f : \mathbb{R}^d \to \mathbb{R}^q$ with $q \gg 1$ is another story)

Note: we're assuming derivative exists, just hard to actually calculate

This is *not* non-smooth optimization

Why not just find gradients automatically?

✓The **adjoint state method** and **reverse-mode automatic differentiation** can automatically calculate gradients in about the same time (~4x) as a function evaluation

- ... so if we can evaluate $f(x)$ numerically, we can find the gradient
- (this applies if $f : \mathbb{R}^d \to \mathbb{R}$; $f : \mathbb{R}^d \to \mathbb{R}^q$ with $q \gg 1$ is another story)

✗Requires specialized/restricted libraries/code (`dolfin-adjoint/FEniCS`, `autograd`)

Why not just find gradients automatically?

✓ The **adjoint state method** and **reverse-mode automatic differentiation** can automatically calculate gradients in about the same time (~4x) as a function evaluation

- … so if we can evaluate $f(x)$ numerically, we can find the gradient
- (this applies if $f : \mathbb{R}^d \to \mathbb{R}$; $f : \mathbb{R}^d \to \mathbb{R}^q$ with $q \gg 1$ is another story)

✗ Requires specialized/restricted libraries/code (`dolfin-adjoint/FEniCS`, `autograd`)

✗ Adjoint state method requires a method to solve adjoint PDE

- difficult to maintain in large code bases, e.g., 4D-var for weather codes (and have to parallelize for HPC)

Why not just find gradients automatically?

✓ The **adjoint state method** and **reverse-mode automatic differentiation** can automatically calculate gradients in about the same time (~4x) as a function evaluation

- ... so if we can evaluate $f(x)$ numerically, we can find the gradient
- (this applies if $f : \mathbb{R}^d \to \mathbb{R}$; $f : \mathbb{R}^d \to \mathbb{R}^q$ with $q \gg 1$ is another story)

✗ Requires specialized/restricted libraries/code (`dolfin-adjoint/FEniCS`, `autograd`)

✗ Adjoint state method requires a method to solve adjoint PDE

- difficult to maintain in large code bases, e.g., 4D-var for weather codes

✗ Slow if used for intermediate calculations involving some $f : \mathbb{R}^d \to \mathbb{R}^q$

- e.g., seismic inversion with many observations

Why not just find gradients automatically?

✓ The **adjoint state method** and **reverse-mode automatic differentiation** can automatically calculate gradients in about the same time (~4x) as a function evaluation

- ... so if we can evaluate $f(x)$ numerically, we can find the gradient
- (this applies if $f : \mathbb{R}^d \to \mathbb{R}$; $f : \mathbb{R}^d \to \mathbb{R}^q$ with $q \gg 1$ is another story)

✗ Requires specialized/restricted libraries/code (`dolfin-adjoint/FEniCS`, `autograd`)

✗ Adjoint state method requires a method to solve adjoint PDE

- difficult to maintain in large code bases, e.g., 4D-var for weather codes

✗ Slow if used for intermediate calculations involving some $f : \mathbb{R}^d \to \mathbb{R}^q$

- e.g., seismic inversion with many observations

✗ Possible memory explosion

- e.g., time-dependent problems. Check-pointing schemes somewhat helpful

Why not just find gradients automatically?

✓ The **adjoint state method** and **reverse-mode automatic differentiation** can automatically calculate gradients in about the same time (~4x) as a function evaluation

- … so if we can evaluate $f(x)$ numerically, we can find the gradient
- (this applies if $f : \mathbb{R}^d \to \mathbb{R}$; $f : \mathbb{R}^d \to \mathbb{R}^q$ with $q \gg 1$ is another story)

✗ Requires specialized/restricted libraries/code (`dolfin-adjoint/FEniCS`, `autograd`)

✗ Adjoint state method requires a method to solve adjoint PDE

- difficult to maintain in large code bases, e.g., 4D-var for weather codes

✗ Slow if used for intermediate calculations involving some $f : \mathbb{R}^d \to \mathbb{R}^q$

- e.g., seismic inversion with many observations

✗ Possible memory explosion

- e.g., time-dependent problems. Check-pointing schemes somewhat helpful

✗ Requires access to original source code

Why not just find gradients automatically?

✓ The **adjoint state method** and **reverse-mode automatic differentiation** can automatically calculate gradients in about the same time (~4x) as a function evaluation

- ... so if we can evaluate $f(x)$ numerically, we can find the gradient
- (this applies if $f : \mathbb{R}^d \to \mathbb{R}$; $f : \mathbb{R}^d \to \mathbb{R}^q$ with $q \gg 1$ is another story)

✗ Requires specialized/restricted libraries/code (`dolfin-adjoint/FEniCS`, `autograd`)

✗ Adjoint state method requires a method to solve adjoint PDE

- difficult to maintain in large code bases, e.g., 4D-var for weather codes

✗ Slow if used for intermediate calculations involving some $f : \mathbb{R}^d \to \mathbb{R}^q$

- e.g., seismic inversion with many observations

✗ Possible memory explosion

- e.g., time-dependent problems. Check-pointing schemes somewhat helpful

✗ Requires access to original source code

✗ Assumes a computational structure

- inapplicable for physical observations (wind farms; *rollout* in AI)

Baseline Algorithms (for comparison)

$f : \mathbb{R}^d \to \mathbb{R}$

Algorithm Gradient Descent via Finite Differences

1: **for** $k = 1, 2, \ldots$ **do**
2: Estimate $g_k \approx \nabla f(x_k)$ ▷ Use finite differences
3: $x_{k+1} \leftarrow x_k - \eta_k g_k$ ▷ For appropriate step-size η_k

✔ ignoring finite-difference error, enjoys well-understood convergence

✘ requires $d+1$ function evaluations per iter.

Baseline Algorithms (for comparison)

$f : \mathbb{R}^d \to \mathbb{R}$

Algorithm Gradient Descent via Finite Differences

1: **for** $k = 1, 2, \ldots$ **do**
2: Estimate $g_k \approx \nabla f(x_k)$ ▷ Use finite differences
3: $x_{k+1} \leftarrow x_k - \eta_k g_k$ ▷ For appropriate step-size η_k

✔ ignoring finite-difference error, enjoys well-understood convergence
✘ requires $d+1$ function evaluations per iter.

Algorithm Randomized Coordinate Descent (CD)

1: **for** $k = 1, 2, \ldots$ **do**
2: Choose $j \in \{1, 2, \ldots, d\}$ at random
3: $g_k = e_j e_j^T \nabla f(x_k)$
4: $x_{k+1} \leftarrow x_k - \eta_k g_k$ ▷ For appropriate step-size η_k (or exact minimization… depends on structure)

✔ just 1 function evaluation per iteration
✘ poor convergence properties, slow rates

Baseline Algorithms (for comparison)

$f : \mathbb{R}^d \to \mathbb{R}$

Algorithm Gradient Descent via Finite Differences

```
for k = 1, 2, ... do
    Estimate g_k ≈ ∇f(x_k)                ▷ Use finite differences
    x_{k+1} ← x_k − η_k g_k               ▷ For appropriate step-size η_k
```

✓ ignoring finite-difference error, enjoys well-understood convergence

✗ requires $d+1$ function evaluations per iter.

Algorithm Randomized Coordinate Descent (CD)

```
for k = 1, 2, ... do
    Choose j ∈ {1, 2, ..., d} at random
    g_k = e_j e_j^T ∇f(x_k)
    x_{k+1} ← x_k − η_k g_k               ▷ For appropriate step-size η_k (or exact minimization... depends on structure)
```

✓ just 1 function evaluation per iteration

✗ poor convergence properties, slow rates

Why not use traditional Derivative Free Optimization (DFO) methods?

Answer: most classical DFO methods don't scale well with dimension

Part I: A Simple Method

Stochastic Subspace Descent

directional derivative $qq^T\nabla f(x_k) = \left(\lim_{h\to 0}\frac{f(x_k+h\cdot q)-f(x_k)}{h}\right)q$

Assume we can compute this!

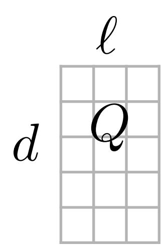
e.g.,

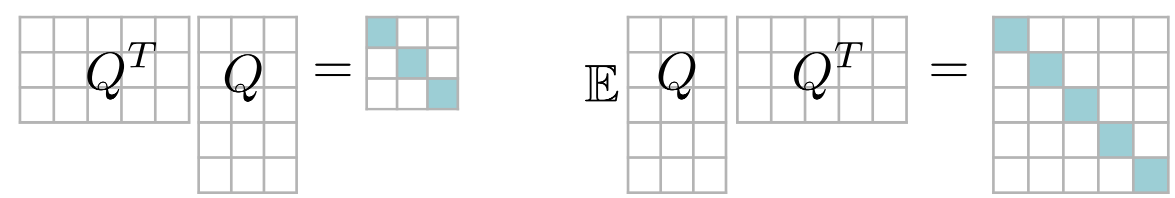
1) forward finite diff
2) forward-mode AD

Stochastic Subspace Descent

directional derivative $qq^T \nabla f(x_k) = \left(\lim_{h \to 0} \frac{f(x_k + h \cdot q) - f(x_k)}{h} \right) q$

$Q = [q_1, q_2, \ldots, q_\ell] \sim \mathrm{Haar}(d \times \ell)$ $\qquad Q^T Q = I_{\ell \times \ell}, \quad \mathbb{E}\left(\frac{d}{\ell} QQ^T \right) = I_{d \times d}$





One benefit: in the limit $\ell = d$, $QQ^T = I_{d \times d}$, and so we'll recover the full gradient (for Gaussians, this is only true in expectation)

Stochastic Subspace Descent

directional derivative $qq^T \nabla f(x_k) = \left(\lim_{h \to 0} \frac{f(x_k + h \cdot q) - f(x_k)}{h} \right) q$

$Q = [q_1, q_2, \ldots, q_\ell] \sim \mathrm{Haar}(d \times \ell)$ $\qquad Q^T Q = I_{\ell \times \ell}, \quad \mathbb{E}\left(\frac{d}{\ell} Q Q^T \right) = I_{d \times d}$

Algorithm "Stochastic Subspace Descent" (SSD)

1: **for** $k = 1, 2, \ldots$ **do**
2: Draw $Q \sim \mathrm{Haar}(d \times \ell)$ or *any* generic SSD
3: $x_{k+1} \leftarrow x_k - \eta_k \frac{d}{\ell} Q Q^T \nabla f(x_k)$

Generic SSD $\boxed{Q^T Q = I_{\ell \times \ell}, \quad \mathbb{E}\left(\frac{d}{\ell} Q Q^T \right) = I_{d \times d}}$

Both **Haar** and **Coordinate Descent** methods are valid generic SSD

Stochastic Subspace Descent

directional derivative $qq^T\nabla f(x_k) = \left(\lim_{h\to 0} \frac{f(x_k + h\cdot q) - f(x_k)}{h}\right) q$

$Q = [q_1, q_2, \dots, q_\ell] \sim \text{Haar}(d\times\ell)$ $\qquad Q^TQ = I_{\ell\times\ell}, \quad \mathbb{E}\left(\frac{d}{\ell}QQ^T\right) = I_{d\times d}$

Algorithm "Stochastic Subspace Descent" (SSD)

1: **for** $k = 1, 2, \dots$ **do**
2: Draw $Q \sim \text{Haar}(d\times\ell)$
3: $x_{k+1} \leftarrow x_k - \eta_k \frac{d}{\ell} QQ^T\nabla f(x_k)$

We call Q a "Haar" distributed r.v. (i.e., the Haar measure over orthogonal matrices), but really care about QQ^T which is a projection matrix (onto $\text{col}(Q)$).

We get Q via Gram-Schmidt (or appropriately modified `QR`) on a Gaussian G, and note $\text{col}(Q) = \text{col}(G)$ w.p. 1, so our update is equivalent to

$$x_{k+1} \leftarrow x_k - \eta_k \frac{d}{\ell}\, \mathcal{P}_{\text{col}(G)}\left(\nabla f(x_k)\right)$$

and hence the term "stochastic subspace".

Stochastic Subspace Descent

directional derivative $qq^T \nabla f(x_k) = \left(\lim_{h \to 0} \frac{f(x_k + h \cdot q) - f(x_k)}{h} \right) q$

$Q = [q_1, q_2, \ldots, q_\ell] \sim \mathrm{Haar}(d \times \ell) \qquad Q^T Q = I_{\ell \times \ell}, \quad \mathbb{E}\left(\frac{d}{\ell} QQ^T \right) = I_{d \times d}$

Algorithm "Stochastic Subspace Descent" (SSD)

1: **for** $k = 1, 2, \ldots$ **do**
2: Draw $Q \sim \mathrm{Haar}(d \times \ell)$
3: $x_{k+1} \leftarrow x_k - \eta_k \frac{d}{\ell} QQ^T \nabla f(x_k)$

Haar is a better choice than alternatives:

- canonical basis (coordinate descent)
- Gaussian sampling
- unit sphere sampling

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)*

Stochastic Subspace Descent

directional derivative $q^T \nabla f(x_k) = \lim_{h \to 0} \frac{f(x_k + h \cdot q) - f(x_k)}{h}$

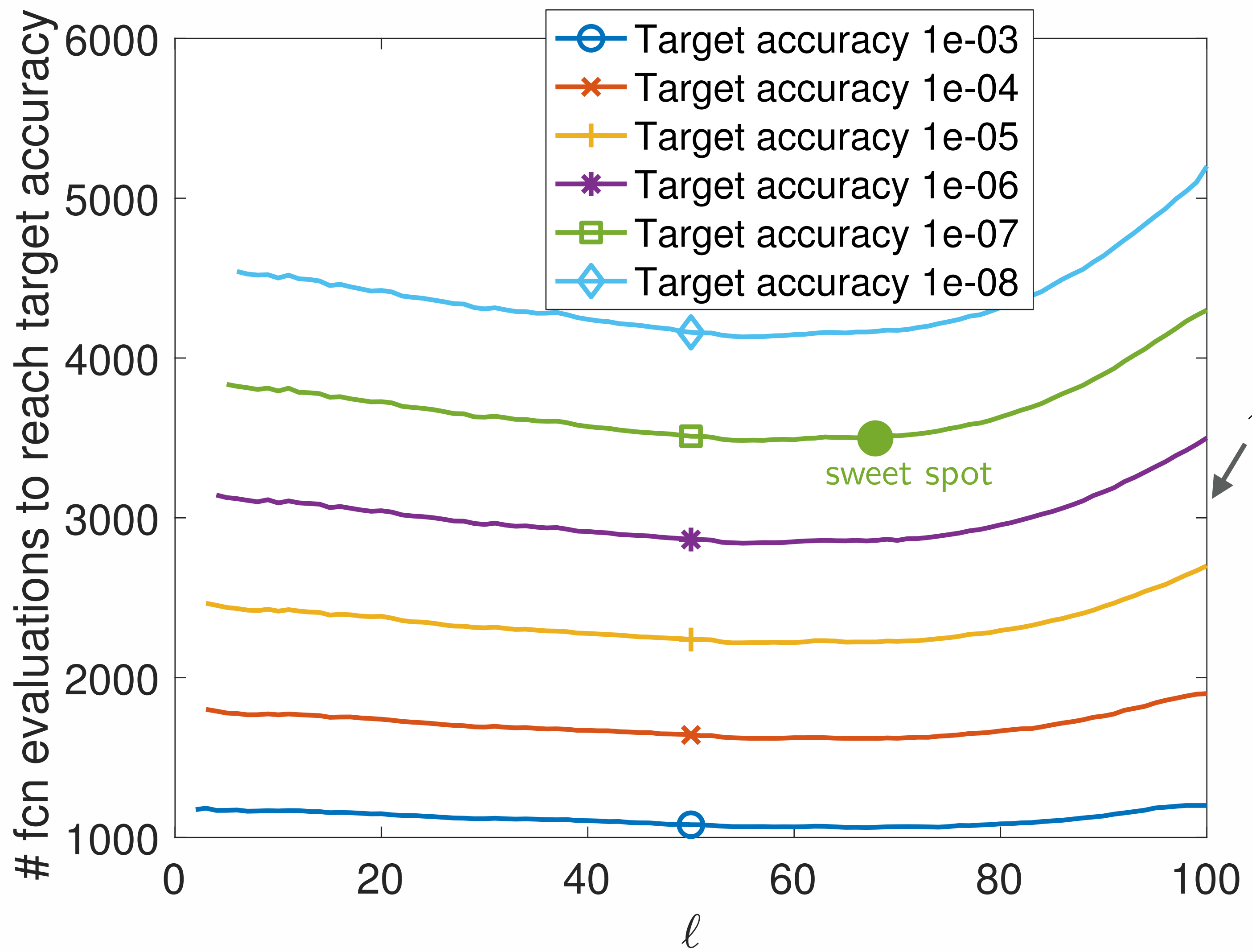
And much recent work on variants, 2011—2020 [and more since then!]

- 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)

Stochastic Subspace Descent

directional derivative $q^T \nabla f(x_k) = \lim_{h \to 0} \frac{f(x_k + h \cdot q) - f(x_k)}{h}$

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

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

First theory results (for *generic* SSD)

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

Theorem (Kozak, Becker, Tenorio, Doostan '20)

Assume: minimizer attained, gradient Lipschitz L, stepsize η_k chosen appropriately. $\eta = \frac{\ell}{d}\frac{1}{L}$

1. If f is **convex**,

$$\mathbb{E}f(x_k) - f^\star \le 2\frac{d}{\ell}\frac{L}{k}R^2 = \mathcal{O}(k^{-1})$$

2. If f is **not convex** but satisfies the μ **Polyak-Lojasiewicz** inequality,

$$\mathbb{E}f(x_k) - f^\star \le \rho^k(f(x_0) - f^\star) = \mathcal{O}(\rho^k) \quad \text{and} \quad f(x_k) \xrightarrow{\text{a.s.}} f^\star$$

3. If f is **strongly convex**, statements of 2 above hold, and also

$$x_k \xrightarrow{\text{a.s.}} \operatorname{argmin}_x f(x)$$

4. If f is **not convex** (nor PL),

$$\min_{k' \in \{0,\dots,k\}} \mathbb{E}\|\nabla f(x_{k'})\|^2 \le \frac{d}{\ell}\frac{2L(f(x_0) - f^\star)}{k+1}$$

$f^\star \stackrel{\text{def}}{=} \min_x f(x)$

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

$d =$ ambient dimension

$\ell = \#$ directional derivs

Generic SSD $\quad Q^TQ = I_{\ell\times\ell}, \quad \mathbb{E}\left(\frac{d}{\ell}QQ^T\right) = I_{d\times d}$

First theory results (for *generic* SSD)

Theorem (Kozak, Becker, Tenorio, Doostan '20)

Polyak-Lojasiewicz Condition $\frac{1}{2}\|\nabla f(x)\|^2 \geq \mu\,(f(x) - f^*)\,,\ \forall\, x$

Example: $f(x) = \frac{1}{2}\|Ax - b\|^2$ where A isn't injective

Acronym	Name	Some references
PL	Polyak-Lojasiewicz	Karimi, Nutini, Schmidt '16
SC	Strong Convexity	
EB	Error Bound	Luo and Tseng '93
ESC	Essential Strong Convexity	Liu et al. '14
WSC	Weak Strong Convexity	Necoara et al. '15
RSI	Restricted Secant Inequality	Zhang and Yin '13
RSC	Restricted Strong Convexity	$=$ RSI $+$ Convexity
QG	Quadratic Growth	Anitescu '00
OSC	Optimal Strong Convexity	$=$ QG $+$ Convexity
SSC	Semi-Strong Convexity	$=$ QG $+$ Convexity

Theorem 2. *For a function f with a Lipschitz-continuous gradient, the following implications hold:*

$$(SC) \rightarrow (ESC) \rightarrow (WSC) \rightarrow (RSI) \rightarrow (EB) \equiv (PL) \rightarrow (QG).$$

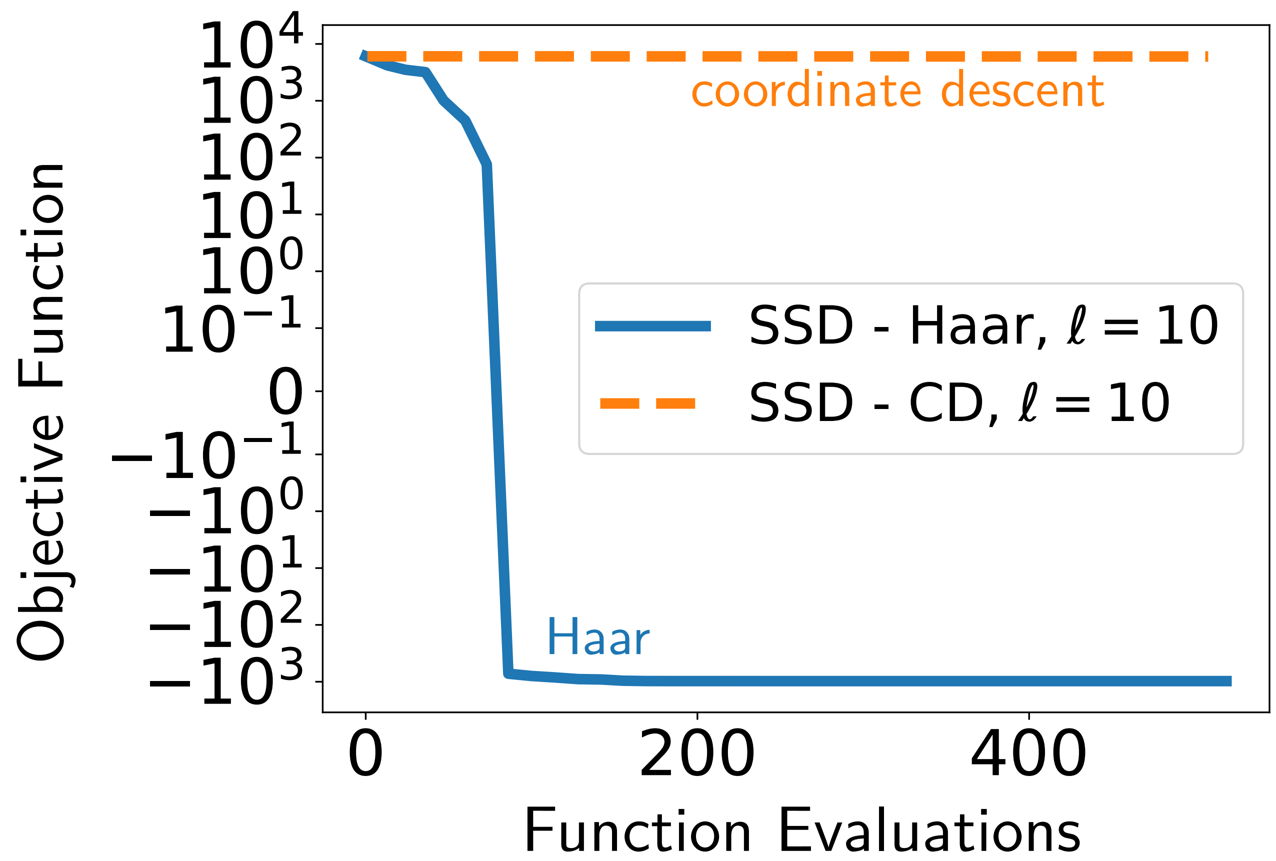
If we further assume that f is convex then we have

$$(RSI) \equiv (EB) \equiv (PL) \equiv (QG).$$

(Karimi, Nutini, Schmidt '16)

$\ell = \#$ directional derivs

Numerical Results: better than expected

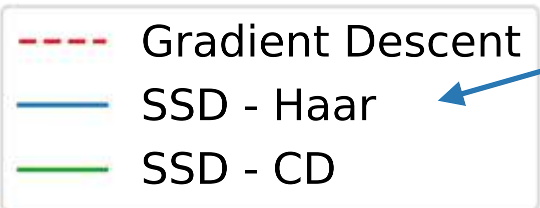


Observation: sometimes SSD (with Haar) drastically outperforms randomized coordinate descent (CD)

Generic SSD $\boxed{Q^T Q = I_{\ell\times\ell}, \quad \mathbb{E}\left(\frac{d}{\ell} Q Q^T\right) = I_{d\times d}}$

Numerical Results: better than expected

SSD drastically outperforms randomized coordinate descent (CD)



new method

$r = 20$
$\ell = 3$

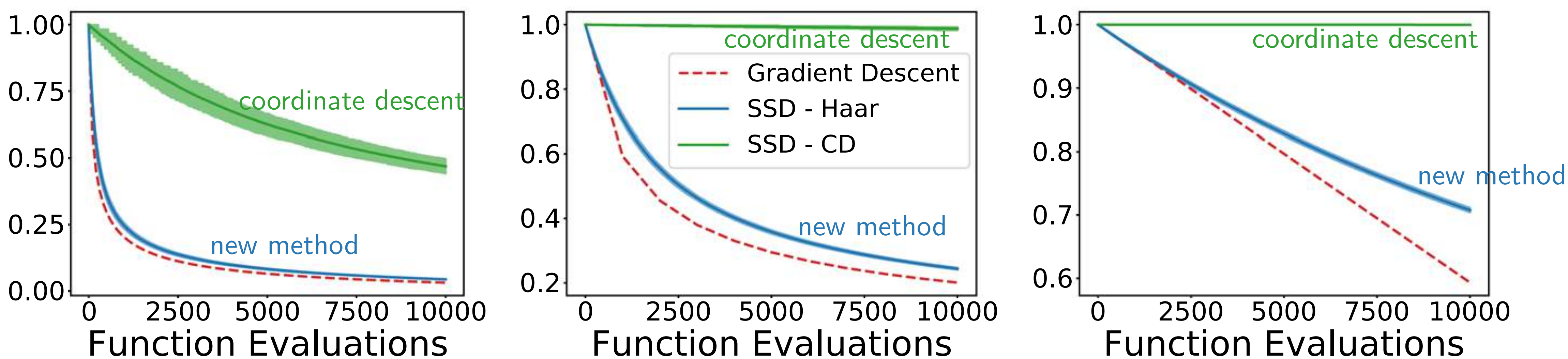
We can force it to happen by making a problem with low "intrinsic" dimension, e.g., Nesterov's "worst function in the world"

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

This has **intrinsic dimension** of r

fixed stepsize

dimension 100 **dimension 1,000** **dimension 10,000**



Numerical Results: better than expected

SSD drastically outperforms randomized coordinate descent (CD)

new method

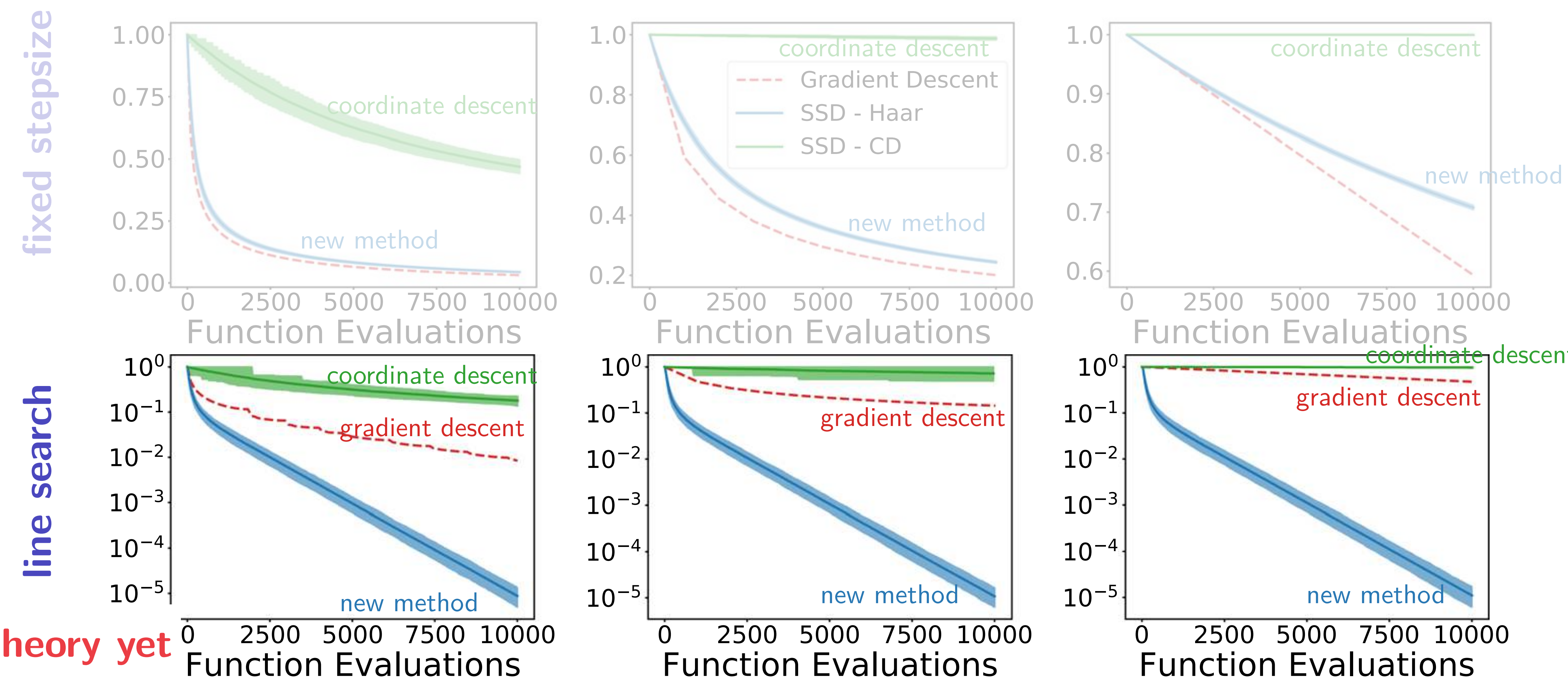
$r = 20$
$\ell = 3$

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

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

This has **intrinsic dimension** of r

dimension 100 **dimension 1,000** **dimension 10,000**



no theory yet

Theory: explain better-than-expected results

Previous theorem didn't actually rely on properties of Haar distribution, just generic Q:

$$\boxed{Q^T Q = I_{\ell\times\ell}, \quad \mathbb{E}\left(\frac{d}{\ell} QQ^T\right) = I_{d\times d}}$$

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* $\forall 0 \neq g \in \mathbb{R}^d$,

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

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

Theory: explain better-than-expected results

Previous theorem didn't actually rely on properties of Haar distribution, just

$$Q^T Q = I_{\ell\times\ell}, \quad \mathbb{E}\left(\frac{d}{\ell} QQ^T\right) = I_{d\times d}$$

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 $\forall 0 \neq g \in \mathbb{R}^d$,*

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

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*

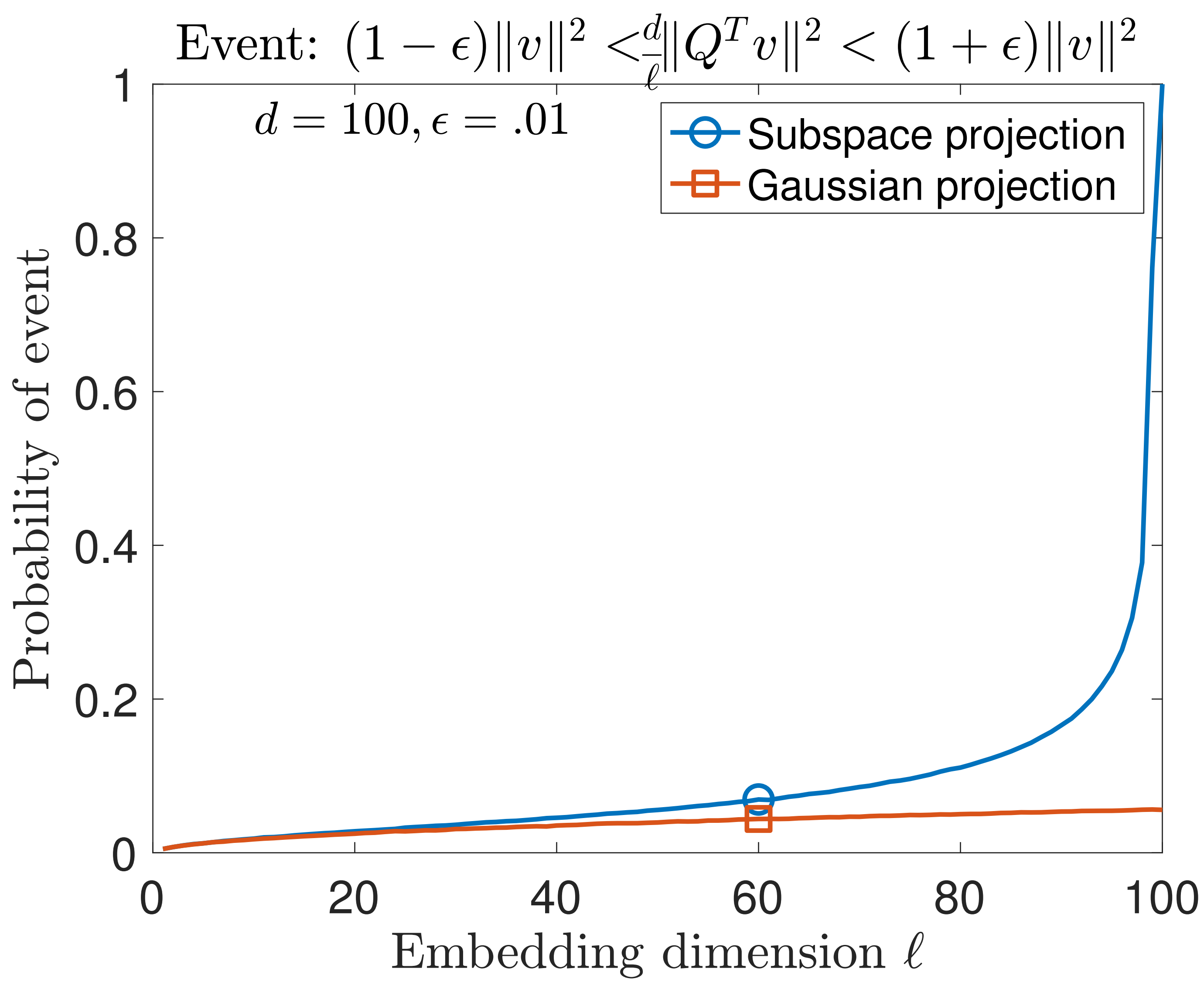
$$f(x_k) - f^\star \le (1 + (1-\epsilon)\rho)^{k/2} (f(x_0) - f^\star) \quad \textit{with probability } \ge 0.998,$$

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

Subspace/Haar outperforms Gaussian projection



Event means *successful* embedding (a good thing)

Tightening things up

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* $\forall 0 \neq g \in \mathbb{R}^d$,

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

... in fact, we can have tight (dimension-dependent) bounds:

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

The CDF of the Beta distribution can be stably computed via the regularized incomplete Beta function

Tightening things up

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

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

The CDF of the Beta distribution can be stably computed via the regularized incomplete Beta function

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

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

Easy to compute, e.g., MATLAB code

```
dist         = @( ell, d ) makedist('Beta','a',ell/2, 'b',(d-ell)/2 );
epsFromDelta = @( delta, ell, d ) 1-d/ell*icdf( dist(ell,d),1-delta );
deltaFromEps = @( eps,   ell, d ) 1-cdf( dist(ell,d), (1-eps)*ell/d );
```

Some numbers

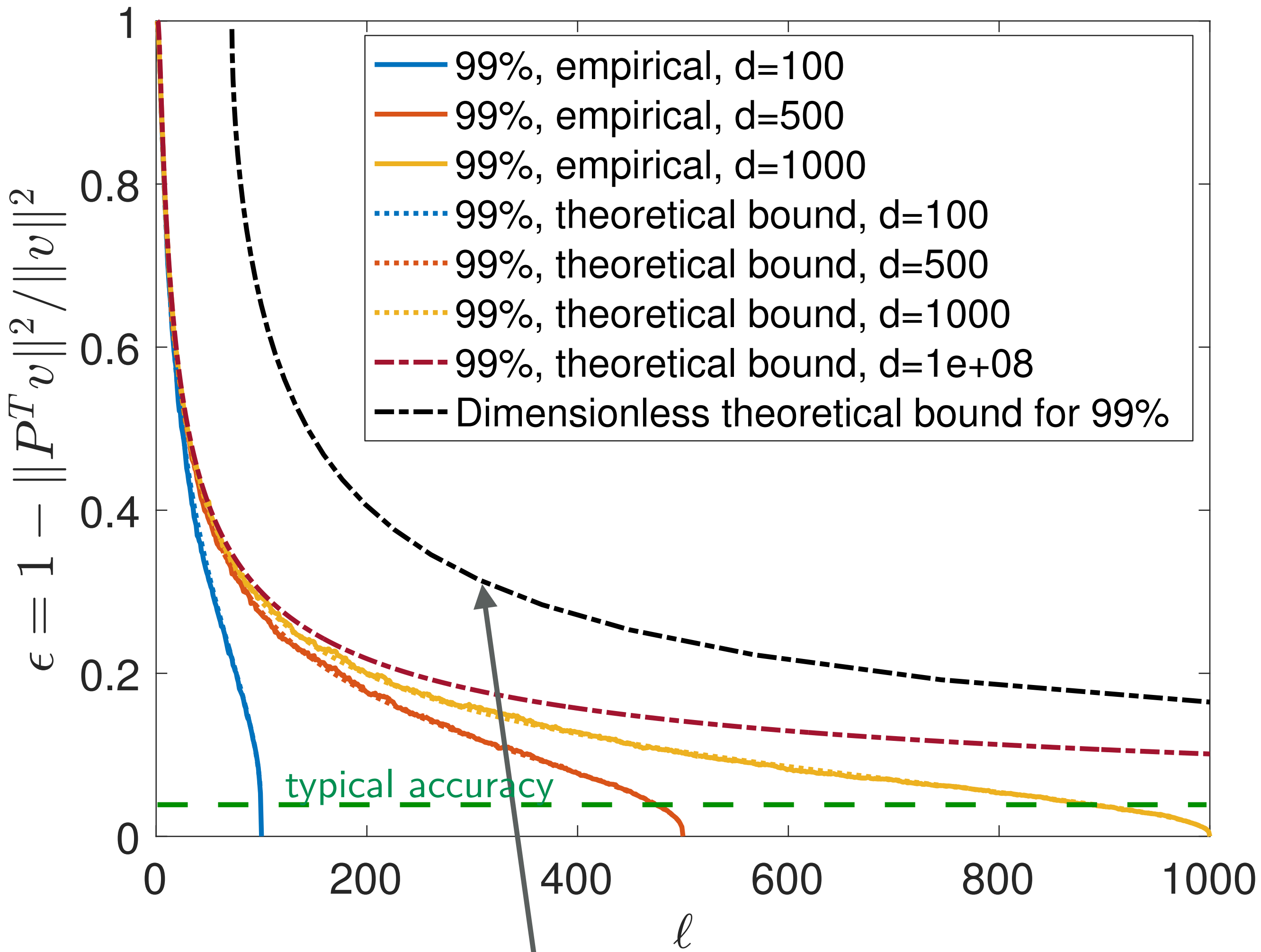
For an embedding of accuracy $\epsilon = 0.1$

Success probability
$\delta \approx 1$

	$\delta = 99\%$		$\delta = 99.99\%$	
Dimension 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%

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

Johnson-Lindenstrauss results are too loose



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

Part II: Variance Reduction

Inspiration

Due to machine learning applications, a lot of work exploits the Empirical Risk Minimization (ERM) structure:

$$f(x) = \frac{1}{N}\sum_{i=1}^{N} f_i(x)$$

control variate

Algorithm SVRG (Johnson, Zhang '13) for solving the ERM model

for $k = 1, 2, \ldots$ **do** $\qquad \triangleright$ k is the "epoch"

$\quad \bar{z} \leftarrow \frac{1}{N}\sum_{i=1}^{N} \nabla f_i(x_k)$

$\quad w_0 \leftarrow x_k$

$\quad$ **for** $t = 1, 2, \ldots, T$ **do** $\qquad \triangleright$ Typically $T = \mathcal{O}(N)$

$\quad\quad$ Draw $j \sim \text{Uniform}([1, \ldots, N])$ $\qquad$ (stable but stale)

$\quad\quad w_{t+1} \leftarrow w_t - \eta\left(\nabla f_j(w_t) - \nabla f_j(x_k) + \bar{z}\right)$

$\quad$ **Option I** $x_{k+1} \leftarrow w_T$ $\qquad \triangleright$ Obvious practical choice

$\quad$ **Option II** $x_{k+1} \leftarrow w_t$ for $t \sim \text{Uniform}([0, \ldots, T-1])$ $\qquad \triangleright$ Tighter analysis

SGD

we have new estimate of this component,
so subtract off the stale version

Inspiration

Due to machine learning applications, a lot of work exploits

the E

Control Variates

Goal: estimate mean $\mu = \mathbb{E}[x]$ of a random variable x

Suppose we have another r.v. y with $\mathbb{E}[y] = \nu$ (called the "control variate")

Form $z = x + c(y - \nu)$ which is an unbiased estimate of the mean: $\mathbb{E}[z] = \mu$

in practice, must estimate

Then for a good choice of c, $c = -\dfrac{\mathrm{Cov}(x, y)}{\mathrm{Var}(y)}$

we've reduced the variance of our estimate:

$$\mathrm{Var}(z) = (1 - \rho^2)\mathrm{Var}(x)$$

(so only advantageous if x and y are correlated)

where $\rho = \dfrac{\mathrm{Cov}(x, y)}{\sqrt{\mathrm{Var}(x)\mathrm{Var}(y)}}$ (Pearson correlation)

New variance-reduced SSD

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 $Q \sim \mathrm{Haar}(d \times \ell)$
  $w_{t+1} \leftarrow w_t - \eta \left(\underbrace{\tfrac{d}{\ell} QQ^T \nabla f(w_t)}_{\text{regular SSD term}} - \alpha_k \underbrace{\left(\tfrac{d}{\ell} QQ^T - I \right)}_{\text{orthogonal projection}} \bar{z} \right)$ ▷ α_k to be estimated
 $x_{k+1} \leftarrow w_T$

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!

SAGA

from the literature:

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

1: $\forall i = 1, \ldots .N$, $x^{(i)} \stackrel{\text{def}}{=} x_0$; store $\{\nabla f_i(x^{(i)})\}_{i=1}^N$ in table
2: **for** $k = 1, 2, \ldots$ **do**
3: Draw $j \sim \text{Uniform}([1, \ldots, N])$
4: $\bar{z} \leftarrow \frac{1}{N}\sum_{i=1}^N \nabla f_i(x^{(i)})$ ▷ From table
5: $x_{k+1} \leftarrow x_k - \eta\left(\nabla f_j(x_k) - \nabla f_j(x^{(j)}) + \bar{z}\right)$
6: 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

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

key: update control variate in the subspace

Other types of control variates

- algorithmic control variates (e.g., SVRG, SAGA, SARAH, etc.)
- approximate computer model of complicated phenomenon (reduced order model)
  - ex. Radio Frequency power amplifiers, where expensive simulation or laboratory measurement is true objective function, but can be approximated by closed-form equations (control variate)
- PDE-specific
  - coarse-grid approximation of a "ground-truth" fine-grid PDE solve
  - lower-order element approximation of main PDE solve
- sketching and other dimensionality reduction methods
  - artificially introduce randomness to compress dimensions
  - early stopping
  - or any other low-order model

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

Exploiting generic control variates

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

control variate, coarse approximation, cheap to evaluate

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

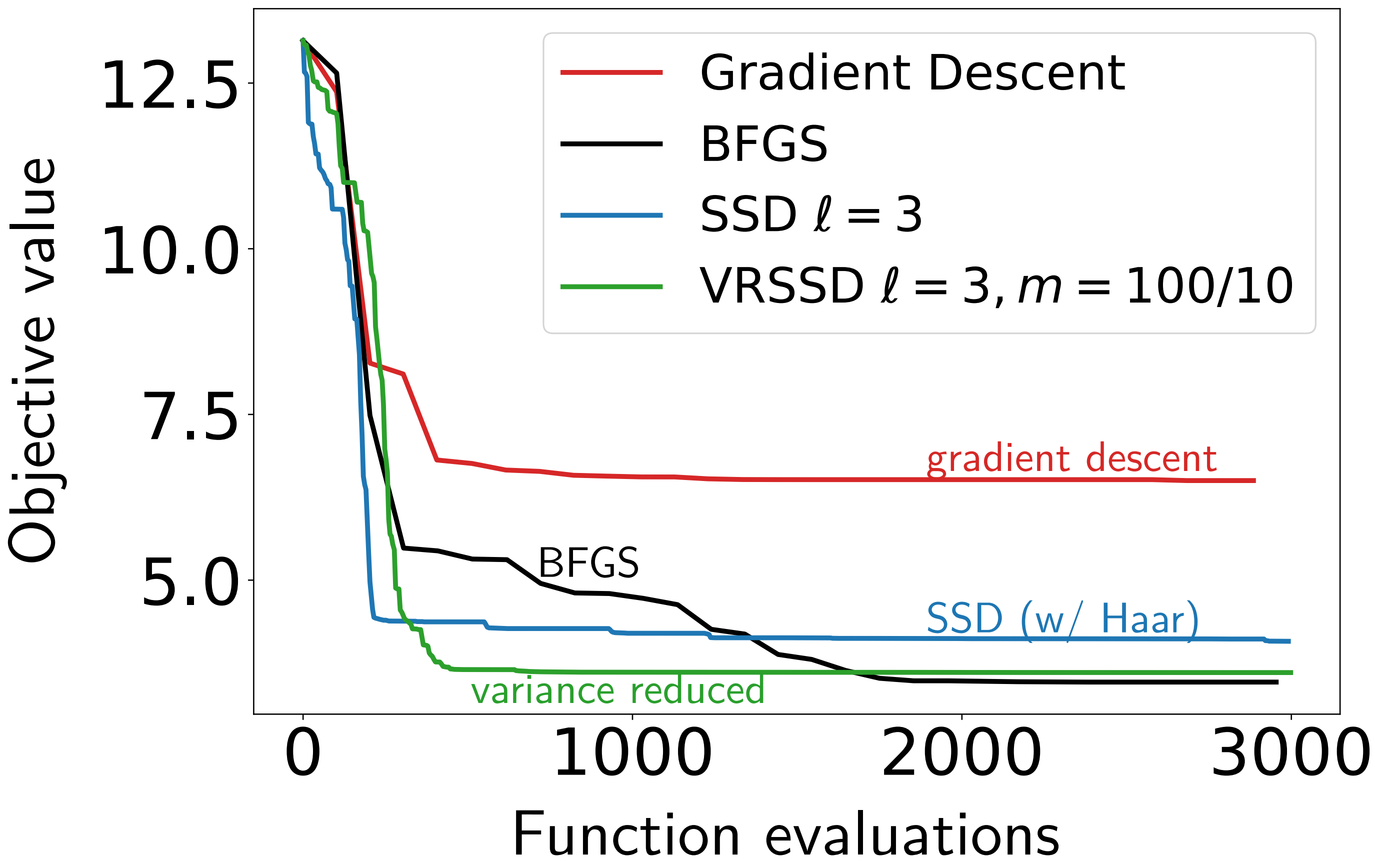
for $k = 1, 2, \ldots$ **do**
    $\bar{z} \leftarrow \nabla f_c(x_k)$ ▷ Full coarse-grid gradient
    Draw $Q \sim \text{Haar}(d \times \ell)$
    $x_{k+1} \leftarrow x_k - \eta_k \left(\frac{d}{\ell} QQ^T \nabla f(x_k) + \alpha_k \left(\frac{d}{\ell} QQ^T \bar{z} - \bar{z} \right) \right)$

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

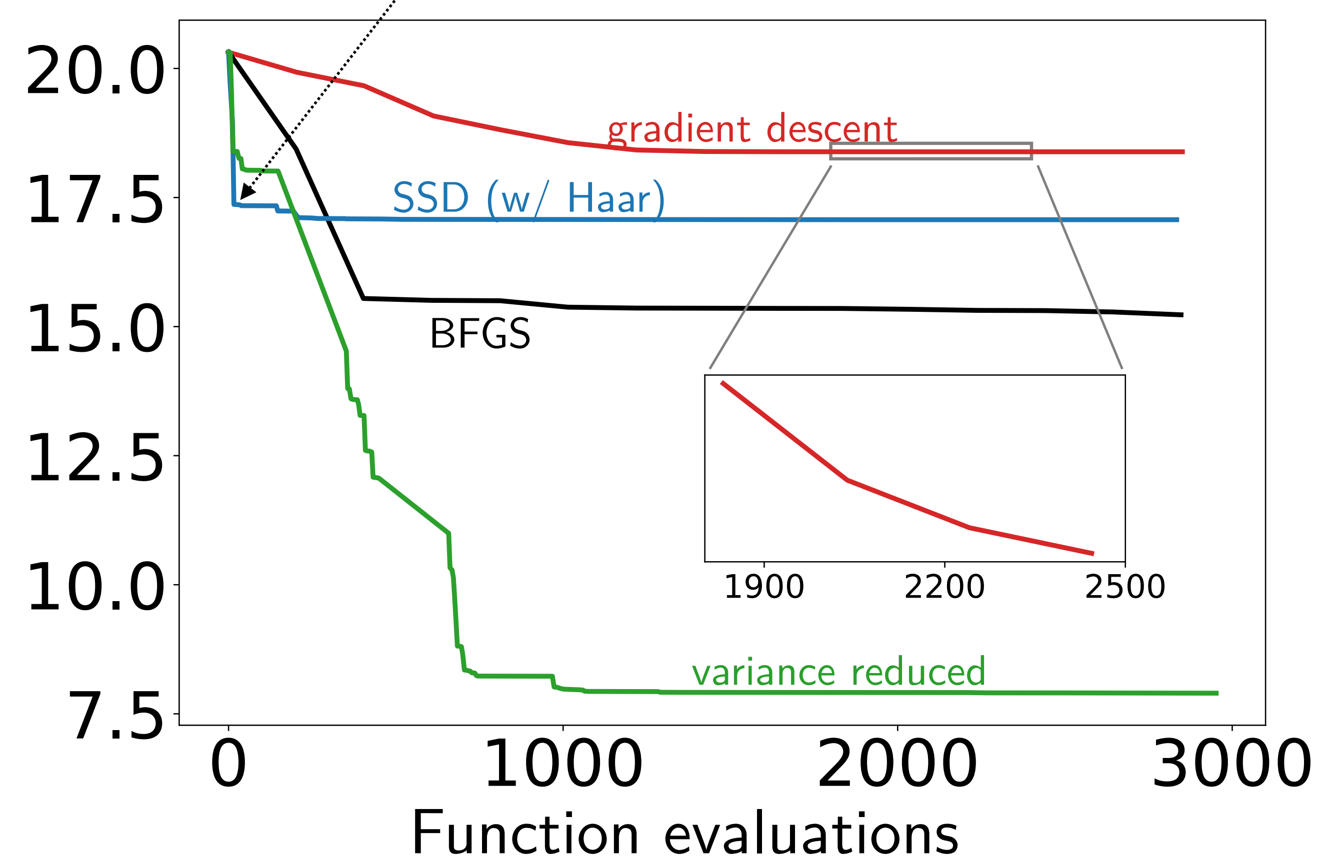
Part III: Numerical Examples

Shape Optimization Application

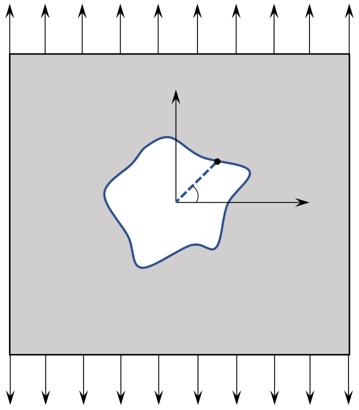
SSD outperform BFGS for low-accuracy solutions



100 dimensional



200 dimensional

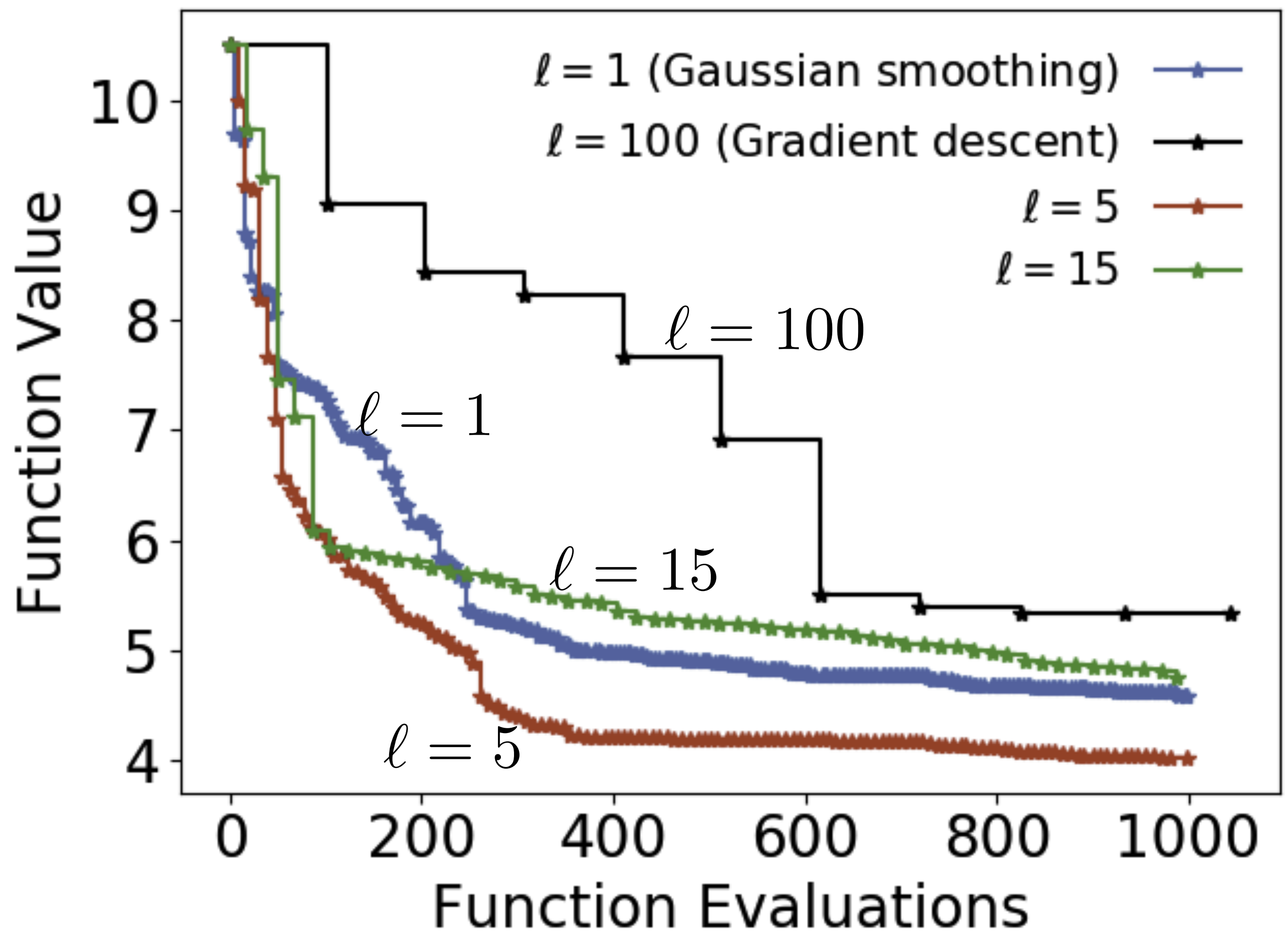


Variance reduction scheme can really work well
(current theory shows that *it converges* but not that it should *converge faster*)

Note: objective is *not* convex, and probably no PL nor Lipschitz gradient

Shape Optimization Application

100 dimensional example



Message: empirically, intermediate values $1 < \ell < d$ work best

Gaussian Process Application: setup

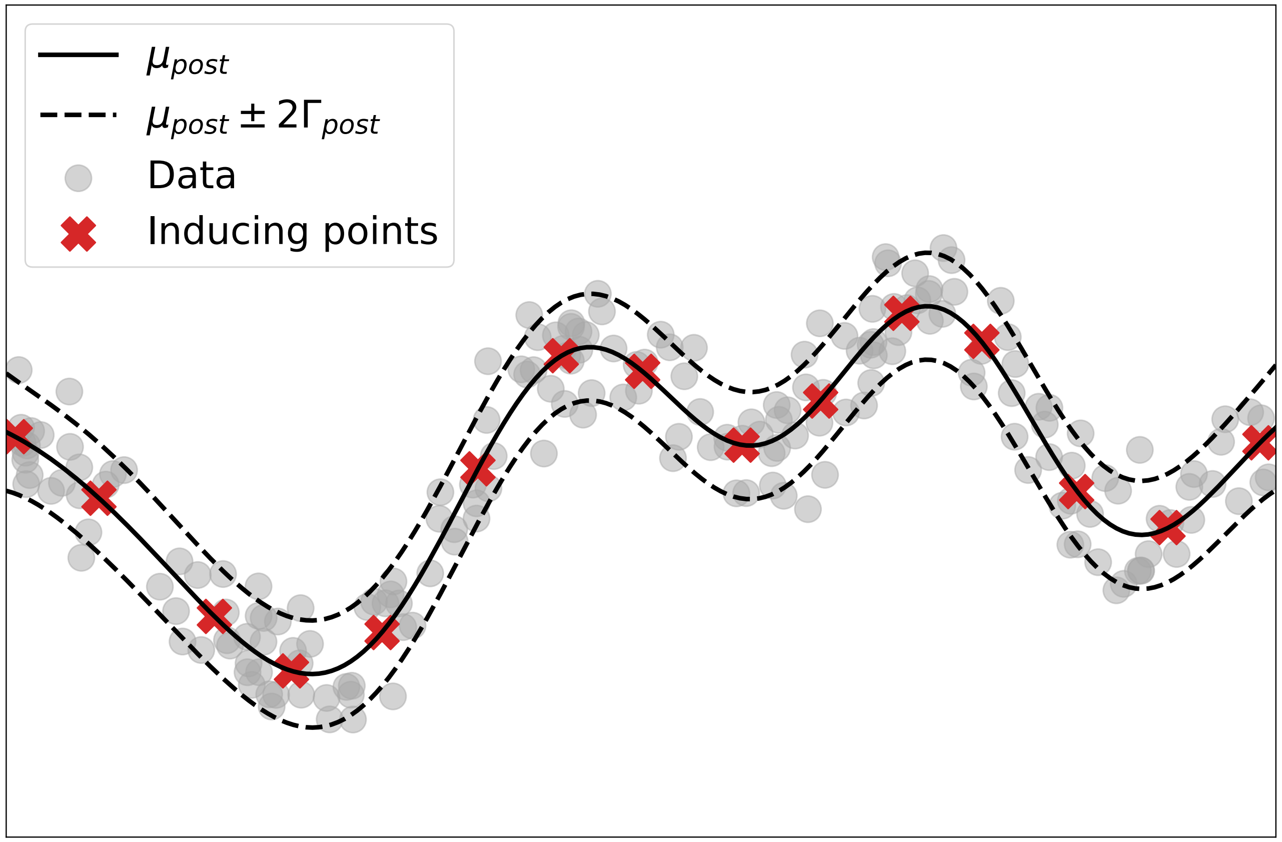
Observations:

$$y_i = \varphi(z_i) + \varepsilon_i,\ \varepsilon_i \sim N(0,\sigma^2)\ \text{ for }\ z_1,\ldots,z_m$$
$$\varphi : \mathbb{R}^p \to \mathbb{R}$$

For inference, assume it's a GP:

$$\mathbb{C}\mathrm{ov}(\varphi(z_i),\varphi(z_j)) = K(z_i,z_j)$$

(and iid errors)



Goal: (approximate) maximum-Likelihood estimation of parameters

but takes $\mathcal{O}(m^3)$ time complexity!

Find a few "inducing points" (Nyström method)

$\widetilde{z}_1,\ldots,\widetilde{z}_{\widetilde{m}}$ $\quad \widetilde{m} \ll m$ ref.: Titsias, *Variational learning of inducing variables in sparse Gaussian processes*, AISTATS '09

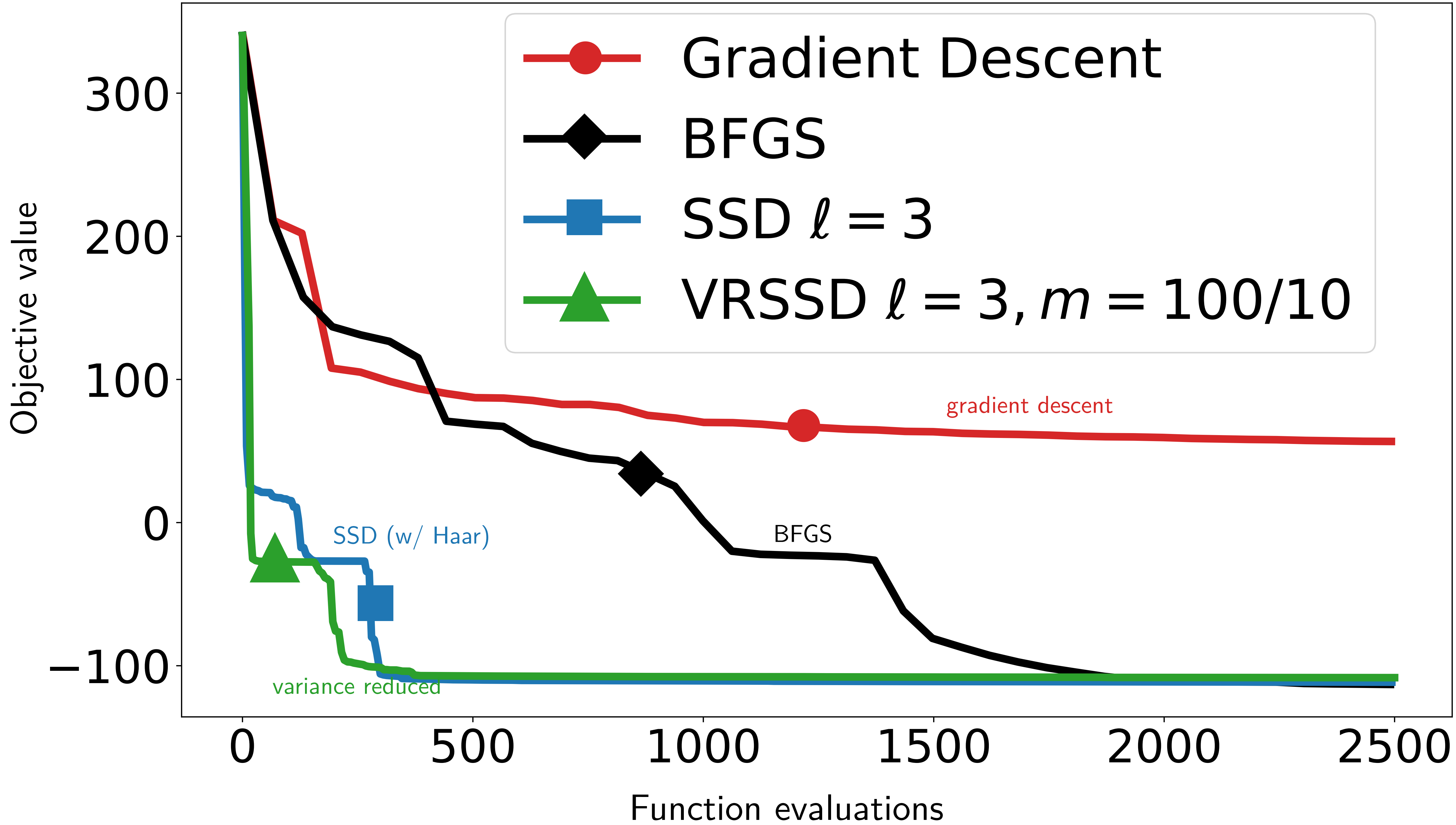
Result: high-dimensional, non-convex optimization problem

$$x \in \mathbb{R}^d,\ d = \widetilde{m}\cdot p + 2 + 1$$

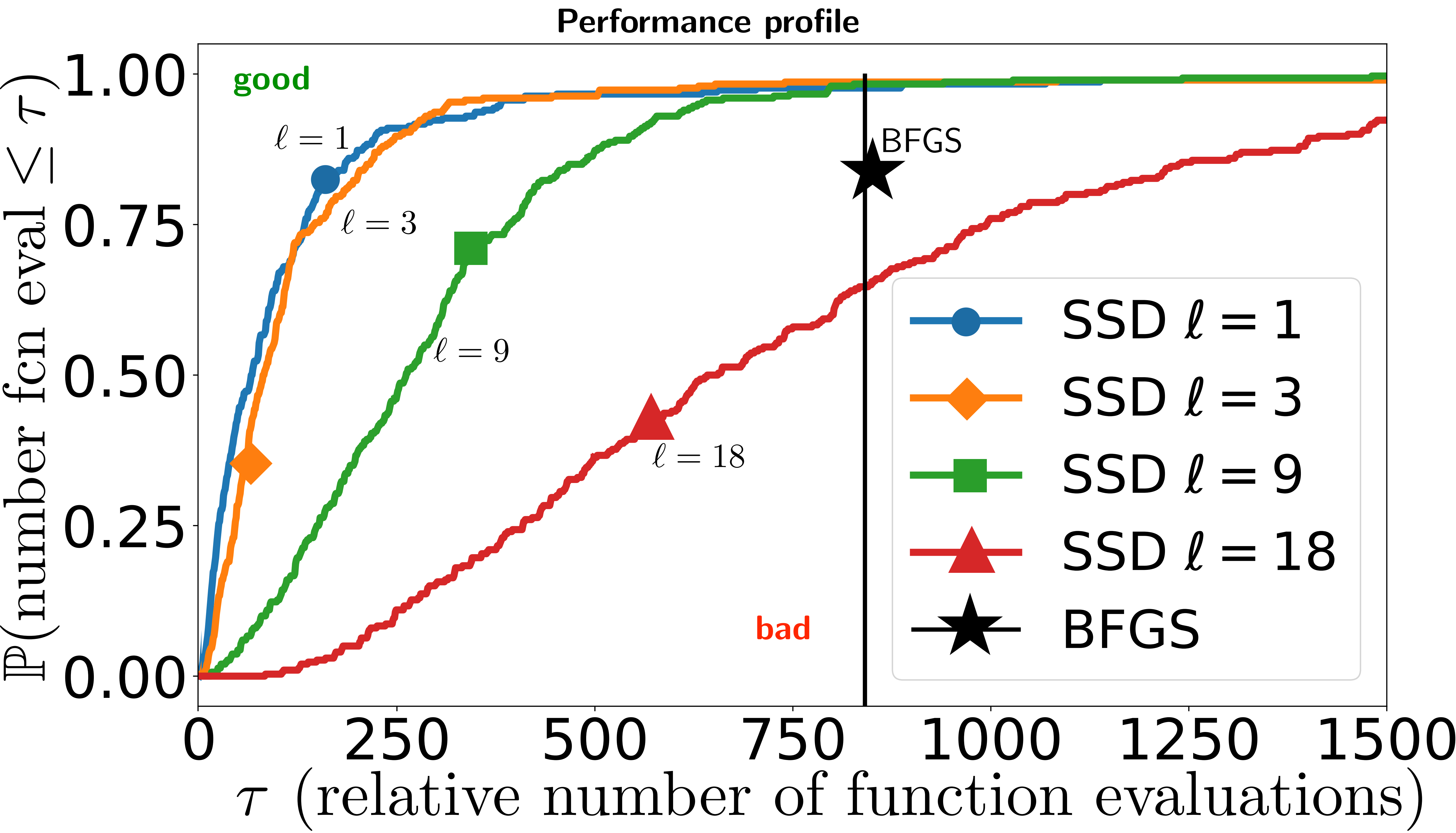
for σ

for kernel parameters (2 for Gaussian kernel: height and width)

Gaussian Process Application: results



Gaussian Process Application: results



60 dimensional; gradient descent has a line at 22,828 (not shown)

Part IV: Comparisons

Alternatives: Gaussian instead of Haar

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*

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

Haar
$1 \le \ell \le d$

where

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

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

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*

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

Gaussian
$\ell = 1$

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

(convex, not necessarily strongly convex)

Alternatives: Gaussian instead of Haar

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*

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

Haar
$1 \le \ell \le d$

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**)

Summary:

- Presented some of the only analysis of SSD (i.e., $Q \sim \text{Haar}$)
- Haar sampling is better than...
  - ...coordinate-wise sampling
  - ...Gaussian sampling (for Haar, $\ell = d$ turns into deterministic gradient descent)
- Exploit concentration-of-measure to get sharpened theorem
- First variance-reduced version of SSD
- Empirical evidence that SSD works fine on non-convex objectives
- High-dimensional "gradient-free" optimization has many applications

Going forward

- Seems to work well on low-effective dimension functions $f(x) = g(Ax), \quad A \in \mathbb{R}^{m \times n}, \; m \ll n$
- Co-author D. Kozak has results on finite difference error
  - very benign due to our randomized setting

arXiv:2107.03941

Zeroth order optimization with orthogonal random directions

David Kozak* Cesare Molinari† Lorenzo Rosasco‡ Luis Tenorio § Silvia Villa ¶

Nonlinear Optimization		Rauch 201
Session Title	:	Methods for Meta-Parameter Estimation in Complex Nonlinear Models
Organizer(s)	:	Aleksandr Aravkin
Chair(s)	:	Kevin Doherty
Speaker #1	:	**Kevin Doherty**, Derivative Free Optimization with Interpolation and Trust Regions: Efficient Use of Zeroth Order Information
Speaker #2	:	**Aleksei Sholokhov**, A Relaxation Approach to Feature Selection for Linear Mixed Effects Models
Speaker #3	:	**Kelsey Maass**, A Hyperparameter-Tuning Approach to Automated Radiotherapy Inverse Planning

Thanks!

Reference:

Stochastic Subspace Descent, David Kozak, Stephen Becker, Alireza Doostan, Luis Tenorio https://arxiv.org/abs/1904.01145 (version 1), https://arxiv.org/abs/2003.02684 (version 2, published in *Computational Optimization and Applications*, 2020)