Scalable Subspace Methods for Unconstrained Optimisation

Gabriel Berk Pereira

St Cross College

University of Oxford

A thesis submitted for the degree of

Master of Science in
Mathematical Modelling and Scientific Computing

Trinity Term 2024

Acknowledgements

First of all, I would like to acknowledge the help and inspiration of my supervisor, Prof. Coralia Cartis. Her guidance allowed me to push through this challenge while doing my best and most rewarding academic work.

I must also thank Dr. Kathryn Gillow for running the MMSC programme so smoothly and for seeking the best for and from our cohort.

A deep word of gratitude goes to my parents, Paulo and Rüveyda, without whose love and support I could never have made it to Oxford.

Lastly, I thank my lovely girlfriend Maria and my Pusey Corner friends for furnishing this incredible year with incredible memories.

Abstract

Large-scale unconstrained optimisation problems abound in applications such as training machine learning models and data assimilation in weather forecasting. Evaluations of the objective or its derivatives—for example, computing a full gradient vector—may be extremely computationally expensive. This issue has recently revived research interest in *subspace methods* for optimisation, which shrink the dimension of the subproblems solved by typical numerical optimisation strategies at each iterate. In addition to making the subproblem cheaper to solve, these algorithms require only reduced problem information, such as a subset of partial derivatives as opposed to the full gradient.

Aside from (deterministic) cyclic block-coordinate algorithms, we can divide existing subspace methods into two classes: *random*, where subspaces are drawn randomly, and *deterministic*, where full gradient information informs the subspace construction. In this dissertation we propose *hybrid* subspace methods, where the chosen subspace is a function of randomly reduced problem gradient information. The algorithms we present sit between the two aforementioned classes, novelly allowing the use of only partial gradient information in designing the chosen subspace.

We describe a class of **l**imited-memory **h**ybrid **s**ubspace line search algorithms, `L-HS`, and derive high-probability global worst-case convergence and complexity bounds for it. In Python, we implement `L-HS-SD` and `L-HS-N`, first- and second-order variants of `L-HS`, and, following a thorough discussion of their per-iteration costs, carry out extensive numerical tests on problems from the CUTEst suite.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this dissertation we are interested in the nonconvex unconstrained minimisation problem,

$$\min_{x\in\mathbb{R}^n} f(x), \quad f\colon \mathbb{R}^n \to \mathbb{R}, \tag{1.1}$$

where the *ambient dimension* n is large. We assume throughout that f is of class C^1 and bounded below. Such a setting is common in applications such as training large machine learning models [1] or data assimilation in weather forecasting [2].

Our attention is principally dedicated to *subspace methods*. Broadly speaking, we use this term to refer to numerical optimisation algorithms in which each iterate update is **explicitly** calculated in a lower-dimensional subspace.[1]

Suppose that an algorithm applied to (1.1) produces a sequence of (decision variable) iterates $\{x_k\}_{k\in\mathbb{N}}$ with updates

$$x_{k+1} = x_k + s_k. \tag{1.2}$$

In a subspace method, we can codify each subspace by the range of a matrix $P_k \in \mathbb{R}^{n\times m_p}$, with subspace dimension $m_p < n$. Thus,

$$s_k = P_k \hat{s}_k, \quad \text{for some} \quad \hat{s}_k \in \mathbb{R}^{m_p}. \tag{1.3}$$

Compared to full-space counterparts, this can bring about computational savings on two fronts: for one, the subproblem at each iterate is of lower dimension and therefore likely to be cheaper to solve; moreover, the method may only require incomplete problem information to form the subproblem in the first place. In large-scale set-

[1] Strictly speaking and depending on the frame of reference, one could call these "affine subspace methods". In any case, "subspace methods" seems more common.

tings, where computing and storing a single full-dimensional gradient vector may be prohibitively expensive, this is highly appealing.

Let us now trace a path to the state of the art in subspace methods, beginning with related principles found in well-known algorithms. Note that we use the notation $\nabla f_k := \nabla f(x_k)$ and $\langle \cdot \rangle := \operatorname{span}\{\cdot\}$.

We can start by pointing out that, under certain conditions, commonly used **quasi-Newton** methods—using trust region or line search safeguards—generate iterate steps contained in the nested subspaces $\langle \nabla f_0, \ldots, \nabla f_k \rangle$ [3, Lemma 2.3]. Popular nonlinear **conjugate gradient** (CG) algorithms for optimisation, such as the Fletcher-Reeves and the Polak-Ribière methods, take steps $s_k \in \langle \nabla f_k, s_{k-1} \rangle$ [4, sec. 5.2].

Going a level deeper, one can use Krylov subspace methods to inexactly solve (quasi-)Newton systems [5]. An example is the **Newton-CG** method, where the Newton system is solved approximately by a linear CG iteration truncated to ensure the generation of descent search directions [4, sec. 6.2].[2]

Newton-CG reduces the cost of solving the subproblem at each iterate, and it even does away with requiring the full Hessian matrix—only *Hessian actions* (Hessian-vector products) are required.[3] However, it still requires the full gradient; by contrast, subspace methods can avoid this too.

The simplest examples of subspace methods are perhaps **block-coordinate descent** algorithms, where the subspaces are aligned with coordinate directions. Thus each iterate step changes only a subset of its entries, and the first-order derivative information of f in such a subspace is similarly determined by just a subset of m_p partial derivatives of f. *Deterministic* (cyclic) [6] as well as *random* [7, 8] coordinate block selection rules have been proposed and analysed, and are very popular [9].

Unsurprisingly, subspace methods with arbitrarily aligned subspaces have also been proposed. Here, too, deterministic [3, 10–13] as well as random [14–19] approaches have been studied. As in the block-coordinate case, these methods do not require the full objective gradient at each iteration, relying instead on a small number of (scalar) *directional derivatives* of f.

Novelly, in this dissertation we propose what we call *hybrid* subspace methods, which exploit (randomly) reduced problem information in subspace construction. Our methods sit, in some sense, between the deterministic subspace methods of [3,

[2]Note that this still falls **outside** our definition of subspace methods, since the subproblem is indirectly—not explicitly—restricted to a subspace.

[3]Provided f is twice continuously differentiable, the Hessian is the $n \times n$ symmetric matrix function of second-order partial derivatives, denoted by $\nabla^2 f$.

10–13]—in which full-dimensional gradient information is used in constructing the subspace—and the random subspace methods of [14–19]—where no problem information is used at all in constructing the subspace. While aiming to reduce our problem evaluation expenditure, we still want to exploit whatever reduced information we *do* compute in constructing subspaces.

The elementary unit of evaluation cost we use is therefore the consultation of a *directional derivative* oracle which provides gradient-vector inner products. In practice this role may be achieved through automatic differentiation [20] or, approximately, through finite differences [4, sec. 7.1]; for our purposes, this occurs in a black box.

1.1 Notation and Terminology

We take this opportunity to (re)state some key notation. We use $\mathbb{N} := \{0, 1, \ldots\}$ and $\mathbb{N}_+ := \mathbb{N} \setminus \{0\}$. The symbols ∇f_k and $\nabla^2 f_k$ stand for the gradient vector and Hessian matrix, respectively, of f at x_k, and we also use $\langle \cdot \rangle = \operatorname{span}\{\cdot\}$. Given a real symmetric matrix A, its (real) eigenvalues are $\lambda_i(A)$, with the smallest one denoted by $\lambda_{\min}(A)$. Standard vector and matrix norm notations are used.

At points, we may slightly abuse the symbols used in previous statements. As an example, we may write "S_k satisfies the bound (x)" when what we mean precisely is "S_k satisfies the bound obtained from (x) by replacing P_k with S_k". We do this in the hope that the gaps are easy for the reader to fill in through context, and that the resulting statements are more concise as well as easier to understand.

As a final word of caution, we note our frequent non-standard use of the terms "first-order method" and "second-order method". Within our framework, this relates only to the way in which search directions are computed, so while "first-order" variants never use second-order information, "second-order" ones may or may not do so.

1.2 Organisation

In Chapter 2 we describe our basic theory on hybrid subspace methods using line search. Then, in Chapter 3, we describe `L-HS`, a class of **l**imited-memory **h**ybrid **s**ubspace line search methods, with first- and second-order variants `L-HS-SD` and `L-HS-N` respectively, and show how they fit within the theory of Chapter 2. After thoroughly discussing these methods' per-iteration costs, we carry out extensive numerical studies and benchmarks in Chapter 4. We finally conclude in Chapter 5.

Chapter 2

Algorithm Framework and Theory

In this chapter we introduce the algorithmic/theoretical framework to be considered for the remainder of this thesis. We proceed to state and prove conditions under which global convergence guarantees and complexity results can be provided for random/hybrid subspace methods using line searches—these conditions and assumptions will inform the practical method, `L-HS`, that we propose in Chapter 3.

2.1 Classical Line Search Algorithms

We begin our discussion of line search methods with a brief presentation of basic theory. For the remainder of this dissertation, we assume that the objective function satisfies the standard Assumption 2.1, and the symbol L is reserved accordingly.

Assumption 2.1 (Smoothness)**.** *The objective function $f\colon \mathbb{R}^n \to \mathbb{R}$ is of class C^1 and bounded below. Moreover, its gradient ∇f is Lipschitz-continuous with Lipschitz constant L over $\mathbb{R}^n$.*

We also need the notion of *convergence to ϵ-first-order optimality* of an algorithm for (1.1). This is defined on the basis that the algorithm in question finds an iterate x_k such that $\epsilon \geq \|\nabla f_k\|_2 \geq 0$.

Now, under Assumption 2.1, global first-order convergence guarantees of deterministic line search methods are classically reliant on three ingredients:

1. Search directions $p_k \in \mathbb{R}^n$ of descent (i.e. where $p_k^\top \nabla f_k < 0$) and uniformly bounded away from orthogonal to ∇f_k.
2. A line search procedure ensuring sufficient decrease in f relative to α_k.
3. A relative lower bound on the step size α_k.

A simple way to secure ingredients 2 and 3 is to perform a so-called *backtracking Armijo* line search [4, sec. 3.1]. Thus, at iteration k, given a descent direction p_k and (fixed) parameters $\alpha^{(0)} > 0, \tau \in (0,1), \beta \in (0,1)$, we successively try $\alpha_k = \alpha^{(0)}\tau^0, \alpha^{(0)}\tau^1, \ldots$ **until** the *Armijo condition* for sufficient decrease

$$f(x_k) - f(x_k + \alpha_k p_k) \geq \beta\alpha_k \left|\nabla f_k^\top p_k\right| \tag{2.1}$$

is satisfied. We then settle on that value for α_k. This ensures that the function decrease is, at worst, proportional to that of the first-order Taylor approximant of f along p_k. Further, one can derive a (relative) lower bound on the resulting α_k [21, Lemma 3].

Theorem 2.1.1 follows from the use of a generic backtracking Armijo line search method for (1.1).

Theorem 2.1.1 ([21, Theorem 4])**.** *Suppose f satisfies* Assumption 2.1. *Apply a generic backtracking Armijo line search method—along descent directions—to its minimisation over infinitely many iterations, with updates $x_{k+1} = x_k + \alpha_k p_k$. Let θ_k denote the angle between $-\nabla f_k$ and p_k. Then* either $\exists\, l \geq 0$ *such that* $\nabla f_l = 0$, or

$$\lim_{k\to\infty} \|\nabla f_k\| \cdot \cos(\theta_k) \cdot \min\{1, \|p_k\|\} = 0. \tag{2.2}$$

This result makes clear that using descent directions is not enough. We must generally ensure that search directions are uniformly bounded away from orthogonal to the gradient—that is, there is an iteration-independent constant $\cos_{\min} > 0$ such that $\cos(\theta_k) \geq \cos_{\min}$ for all k—to secure global convergence $\left(\lim_{k\to\infty} \|\nabla f_k\|_2 = 0\right)$.

2.2 Generic Subspace Method: CFS Framework

While the discussion in Section 2.1 gives us a basic intuition for classical line search methods, our consideration of non-deterministic subspace matrices P_k entails more care. In deriving conditions for global convergence (in a probabilistic sense), we make use of the framework introduced in [18, sec. 2], which we shall refer to as the (Cartis-Fowkes-Shao) *CFS framework*.

The authors of [18] propose a generic algorithm based on (generally) random models of f centred on the current iterate and (generally) of reduced dimension. We present an adapted version in Algorithm 2.1.

The generality of the CFS framework makes the theoretical results derived in [18]

Algorithm 2.1 CFS: Generic Optimisation Framework Based on Random Reduced Models for (1.1) [18, Algorithm 1]

Initialisation:

Choose a subspace dimension $m_p \leq n$ and a class of (generally random) models $m_k(w_k(\hat{s})) = \hat{m}_k(\hat{s})$. $\hat{s} \in \mathbb{R}^{m_p}$ is the *reduced step* and w_k is the *prolongation function*, which maps $\hat{s} \in \mathbb{R}^{m_p}$ to $s \in \mathbb{R}^n$. Choose constants $\tau \in (0,1)$, $c \in \mathbb{N}_+$, $\beta \in (0,1)$, and $\alpha_{\max} > 0$. Define $\nu = \tau^{-c}$. Initialise by setting $k = 0$, $x_0 \in \mathbb{R}^n$, and $\alpha_0 = \alpha_{\max}\tau^p$ for some $p \in \mathbb{N}_+$.

1. Compute a reduced model and a potential step:

Compute a local (generally) random model $\hat{m}_k(\hat{s})$ with $\hat{m}_k(0) = f(x_k)$.
Compute a reduced step $\hat{s}_k(\alpha_k)$, where α_k influences either the model $\hat{m}_k$ or the step computation.
Compute a potential step $s_k = w_k(\hat{s}_k)$.

2. Check sufficient decrease

Compute $f(x_k + s_k)$ and check whether a sufficient decrease in f (parameterised by β) is achieved.

3. Update α_k and possibly take step s_k

[Successful iteration] If sufficient decrease is achieved, set $x_{k+1} = x_k + s_k$ and $\alpha_{k+1} = \min\{\alpha_{\max}, \nu\alpha_k\}$.
[Unsuccessful iteration] Otherwise set $x_{k+1} = x_k$ and $\alpha_{k+1} = \tau\alpha_k$.
In both cases set $k = k + 1$ and go to step **1.**

widely applicable. However, whereas the authors there particularise Algorithm 2.1 and—crucially—the assumptions required from it for trust region and quadratic regularisation variants, here we do so for line search methods. Further, we examine the exploitation of problem information in constructing subspaces, while in [18] this is not considered.

2.3 A Generic CFS-Based Line Search Method

The notation we chose to adopt in Algorithm 2.1 is suggestive of how some of those generic elements can be transposed to a line search algorithm. Namely, the parameters α_k, τ, and β will play the same roles as they did in Section 2.1.

On the other hand, the notions of successful and unsuccessful iterations will translate a (modified) backtracking procedure—thus a series of unsuccessful iterations, where the Armijo condition (2.1) does *not* hold and α_k is successively reduced by a factor τ, will be followed by a successful iteration where it *does* hold; a step is taken; and α_{k+1} is increased to $\min\{\alpha_{\max}, \nu\alpha_k\}$.

Our proposed CFS-based line search method is given in Algorithm 2.2. We will

proceed to motivate some of our choices therein by walking through the assumptions required by the CFS framework for the global convergence/complexity theorem of [18] to apply.

Algorithm 2.2 Random Subspace Line Search Method for (1.1)

Initialisation: Similar to that of Algorithm 2.1, with two differences: forgo the definition of a prolongation function; choose an additional constant $\#_{\text{try}} \in \mathbb{N}_+$.

$P_0 \leftarrow$ `ConstructSubspace(problem, ensemble, ·)`
$k \leftarrow 0, j_{\text{try}} \leftarrow 0$
while not terminated **do**
  $p_k \leftarrow$ `ComputeDirection(`P_k`, problem)` ▷ $p_k \in \mathbb{R}^n$ is a descent direction.
  Compute trial step $s_k = \alpha_k p_k$
  $j_{\text{try}} \leftarrow j_{\text{try}} + 1$
  if trial step satisfies Armijo condition (2.1) **then** ▷ Successful iteration.
    $P_{k+1} \leftarrow$ `ConstructSubspace(problem, ensemble, history)`
    $x_{k+1} \leftarrow x_k + s_k$
    $\alpha_{k+1} \leftarrow \min\{\alpha_{\max}, \nu\alpha_k\}$
    $j_{\text{try}} \leftarrow 0$
  else ▷ Unsuccessful iteration.
    $x_{k+1} \leftarrow x_k$
    $\alpha_{k+1} \leftarrow \tau\alpha_k$
    if $j_{\text{try}} = \#_{\text{try}}$ **then** ▷ Hit backtracking limit in the current subspace.
      $P_{k+1} \leftarrow$ `ConstructSubspace(problem, ensemble, history)`
      $j_{\text{try}} \leftarrow 0$
    end if
  end if
  $k \leftarrow k + 1$
end while

2.4 Global Convergence and Complexity of Algorithm 2.2

The main global worst-case convergence and complexity result from [18] is its Theorem 2.1, which is applicable to the very generic Algorithm 2.1. This relies on a set of **four assumptions**, which we will transpose to our line search particularisation, Algorithm 2.2.

It is essential to note that, for our Algorithm 2.2 to fit within the CFS framework, Algorithm 2.1, and its associated theory from [18], **it must be run with $\#_{\text{try}} = 1$.** This is so that the local random model is re-computed/re-drawn after every iteration,

regardless of whether it was successful or not. The reason we leave open the option to have $\#_{\text{try}} > 1$ is that it leads to better performance on our metrics of problem information expenditure. We elaborate on this matter in Chapters 3 and 4.

Since Algorithm 2.2 is generally random, the mathematical objects involved—x_k, s_k, P_k, and so on—are random variables. Wherever context so requires, in this chapter we use overbars to distinguish these variables from their realisations—respectively $\bar{x}_k$, $\bar{s}_k$, $\bar{P}_k$, and so on. We also define (first-order) convergence on the basis of the random variable

$$N_\epsilon = \min \{k : \|\nabla f_k\|_2 \leq \epsilon\}. \tag{2.3}$$

Assumption 2.2 introduces the notion of a *true iteration*, which will be used throughout the discussions in this chapter as well as in Chapter 3.[1] It is adapted from its original statement in [18]—this enables our framework to include hybrid constructions where the probability of truthfulness is conditionally dependent on more than just the current iterate.

Assumption 2.2 (Adapted from [18, Assumption 1])**.** *There exist $\delta_S \in (0, 1)$ and $p \in \mathbb{N}_+$ such that, for any set of iterates $\{\bar{x}_l, \ldots, \bar{x}_{k-1}, \bar{x}_k\}$ and $k \in \mathbb{N}_+$,*

$$\Pr \{T_k = 1 \mid x_l = \bar{x}_l, \ldots, x_k = \bar{x}_k\} \geq 1 - \delta_S, \tag{2.4}$$

where $l = \max\{0, k - p + 1\}$ and T_k is defined as

$$T_k = \begin{cases} 1, & \text{if iteration } k \text{ is true} \\ 0, & \text{otherwise.} \end{cases} \tag{2.5}$$

Moreover, $\Pr \{T_0\} \geq 1 - \delta_S$, and T_k is conditionally independent of $T_0, T_1, \ldots, T_{l-1}$ given $x_l = \bar{x}_l, \ldots, x_k = \bar{x}_k$.[2]

Assumption 2.3 states that before convergence and for α_k small enough, a **true** iteration is guaranteed to also be **successful**.

Assumption 2.3 ([18, Assumption 2])**.** *For any $\epsilon > 0$, there exists an iteration-independent constant α_{low} such that if iteration k is true, $k < N_\epsilon$, and $\alpha_k < \alpha_{\text{low}}$, then iteration k is successful.*

[1] N.B.: Generally an iteration's truthfulness has no implication on its success, and vice-versa.

[2] Note that the set $\{\bar{x}_l, \ldots, \bar{x}_k\}$ comprises p "past" iterations including the kth ("current") one; the exception is when $k < p - 1$, in which case this set comprises the $k + 1$ iterations $\{\bar{x}_0, \ldots, \bar{x}_k\}$.

Assumption 2.4 provides a condition of sufficient decrease at each true and successful iteration before convergence. In the case of Algorithm 2.2, this assumption is fulfilled in part by the Armijo condition (2.1) for iteration success.

Assumption 2.4 ([18, Assumption 3])**.** *There exists a non-negative, non-decreasing (in both arguments) function $h(z_1, z_2)$ such that, for any $\epsilon > 0$, if iteration k is true and successful with $k < N_\epsilon$, then*

$$f(x_k) - f(x_k + s_k) \geq h(\epsilon, \alpha_k), \tag{2.6}$$

where s_k is computed in Step 1 of Algorithm 2.1 *or* Line 5 *of* Algorithm 2.2*. Moreover, $h(z_1, z_2) > 0$ provided $z_1 > 0$ and $z_2 > 0$.*

Assumption 2.5, last in this foursome, requires simply that objective values are monotonically decreasing throughout the algorithm. This is clearly fulfilled by our use of the Armijo condition (2.1) for iteration success.

Assumption 2.5 ([18, Assumption 4])**.** *For any $k \in \mathbb{N}$, we have*

$$f(x_k) \geq f(x_{k+1}). \tag{2.7}$$

Having listed the assumptions we require Algorithm 2.2 to satisfy, we move to particularise it further.

2.4.1 Search Directions and True Iterations

The notion of a true iteration plays a significant role in Assumptions 2.2 to 2.4. However, we will see that it may be sensible to define it differently depending on how we compute p_k—that is, how we specify `ComputeDirection` in Algorithm 2.2.

Here we propose classes of well-known search directions, motivated by the minimisation of **(a) linear** or **(b) strictly convex quadratic** local models. We do this by setting $p_k = P_k \hat{p}_k$ in a typical derivative-based local model construction,[3] leading to the subproblem

$$\hat{p}_k = \underset{\hat{p} \in \mathbb{R}^{m_p}}{\arg\min}\, \hat{p}^\top \left(P_k^\top \nabla f_k\right) + \frac{1}{2} \hat{p}^\top \left(P_k^\top B_k P_k\right) \hat{p}, \tag{2.8}$$

where $B_k \in \mathbb{S}^n$ is user-chosen (often the Hessian $\nabla^2 f_k$ or an approximation to it).[4]

[3] The construction of local models incorporating derivative information is a ubiquitous technique in nonlinear optimisation. Many examples can be found in [4].

[4] We use $\mathbb{S}^n$ to denote the set of real symmetric $n \times n$ matrices.

2.4.1.1 First-Order Local Models

If in (2.8) we have $B_k = 0$, a linear model is recovered and a constraint must be added for the subproblem to be well-posed. A well-known approach is to force $\|\hat{p}\|_2 = 1$, which can easily be shown to lead to the **steepest descent** (SD) direction in the subspace, and a resulting full-space search direction

$$p_k = -P_k P_k^\top \nabla f_k. \tag{2.9}$$

When using this steepest descent-like search direction—which we call *SD-like*—we may define true iterations as in Definition 2.1.

Definition 2.1 (True iteration in SD-like search [18, Def. 3.1])**.** *In* Algorithm 2.2*, let* $P_{\max} > 0$ *and* $\epsilon_S \in (0, 1)$ *be iteration-independent constants. Iteration* k *is true if and only if*

$$\left\|P_k^\top \nabla f_k\right\|_2 \geq \epsilon_S \left\|\nabla f_k\right\|_2 \text{ and} \tag{2.10}$$

$$\left\|P_k\right\|_2 \leq P_{\max}. \tag{2.11}$$

It remains for us to prove that this setup satisfies Assumptions 2.3 and 2.4. The first step—bounding below the cosine of the angle between p_k and ∇f_k—is familiar to us from Section 2.1.

Lemma 2.1. *In* Algorithm 2.2*, for all* $k \in \mathbb{N}$*, let* $p_k = -P_k P_k^\top \nabla f_k$*. If iteration* k *is true* (Definition 2.1)*, then*

$$\frac{\left|\nabla f_k^\top p_k\right|}{\left\|\nabla f_k\right\|_2 \left\|p_k\right\|_2} \geq \frac{\epsilon_S}{P_{\max}} > 0. \tag{2.12}$$

Proof. Since $p_k = -P_k P_k^\top \nabla f_k$, we can write

$$\begin{aligned}\frac{\left|\nabla f_k^\top p_k\right|}{\left\|\nabla f_k\right\|_2 \left\|p_k\right\|_2} &= \frac{\left\|P_k^\top \nabla f_k\right\|_2^2}{\left\|\nabla f_k\right\|_2 \left\|P_k P_k^\top \nabla f_k\right\|_2} \\ &\geq \frac{\left\|P_k^\top \nabla f_k\right\|_2^2}{\left\|\nabla f_k\right\|_2 \left\|P_k\right\|_2 \left\|P_k^\top \nabla f_k\right\|_2} \geq \frac{\left\|P_k^\top \nabla f_k\right\|_2}{\left\|\nabla f_k\right\|_2 P_{\max}} \geq \frac{\epsilon_S}{P_{\max}} > 0.\end{aligned}$$

Note that the first inequality follows from the submultiplicativity of the spectral matrix norm, while the second and third follow from k being a true iteration. ■

The next step is to prove the existence of α_{low} in Assumption 2.3. We also provide

a bound on the norm of p_k, to be used in proving Lemma 2.3.

Lemma 2.2. *In* Algorithm 2.2*, for all* $k \in \mathbb{N}$*, let* $p_k = -P_k P_k^\top \nabla f_k$*. If iteration* k *is true* (Definition 2.1)*, then*

$$\|p_k\|_2 \geq \epsilon_S^2 \|\nabla f_k\|_2 . \tag{2.13}$$

Furthermore, the Armijo condition (2.1) *is then satisfied for all* $\alpha_k \in (0, \alpha_{\text{low}})$*, where*

$$\alpha_{\text{low}} = \frac{(1-\beta)\epsilon_S}{P_{\max}^3 L}. \tag{2.14}$$

Proof. Start with the bound on $\|p_k\|_2$. Let us momentarily define $A_k = P_k P_k^\top$. Since A_k is symmetric, it can be expressed as $A_k = \sum_{i=1}^n \lambda_i \left(u_i u_i^\top\right)$, where $\{\lambda_i, u_i\}$ are pairs of (real) eigenvalues and unit eigenvectors of A_k. Further, the set $\{u_i\}_{i=1}^n$ is an orthogonal eigenbasis.

Now, since iteration k satisfies Definition 2.1, we have $\left\|P_k^\top \nabla f_k\right\|_2^2 \geq \epsilon_S^2 \|\nabla f_k\|_2^2$, which is the same as $\nabla f_k^\top A_k \nabla f_k \geq \epsilon_S^2 \|\nabla f_k^2\|_2$. On the other hand,

$$\nabla f_k^\top A_k \nabla f_k = \sum_{i=1}^n \lambda_i \left(u_i^\top \nabla f_k\right)^2 = \|\nabla f_k\|_2^2 \sum_{i=1}^n \lambda_i \cos^2 \eta_i, \tag{2.15}$$

where η_i is the angle between ∇f_k and u_i. Therefore $\sum_{i=1}^n \lambda_i \cos^2 \eta_i \geq \epsilon_S^2$.

Now, recall that we are interested in $\|p_k\|_2^2 = \left\|P_k P_k^\top \nabla f_k\right\|_2^2 = \nabla f_k^\top A_k^2 \nabla f_k = \|\nabla f_k\|_2^2 \sum_{i=1}^n \lambda_i^2 \cos^2 \eta_i$. By the Cauchy-Schwarz inequality,

$$\left(\|\nabla f_k\|_2^2 \epsilon_S^4 \leq\right) \|\nabla f_k\|_2^2 \left|\sum_{i=1}^n \lambda_i \cos^2 \eta_i\right|^2 \leq \overbrace{\|\nabla f_k\|_2^2 \left|\sum_{i=1}^n \lambda_i^2 \cos^2 \eta_i\right|}^{=\|p_k\|_2^2} \cdot \overbrace{\left|\sum_{i=1}^n \cos^2 \eta_i\right|}^{=1}, \tag{2.16}$$

where the latter term's being equal to 1 follows from $\{u_i\}_{i=1}^n$ being an orthogonal basis of $\mathbb{R}^n$. Taking square roots yields the bound on $\|p_k\|_2$.

Now we turn to the α_{low} result. We know from [21, Lemma 2] that the Armijo condition (2.1) is satisfied for all $\alpha_k \in (0, \alpha_{\text{Arm}})$, where

$$\begin{aligned}\alpha_{\text{Arm}} &= \frac{(1-\beta)\left|\nabla f_k^\top p_k\right|}{L\|p_k\|_2^2} = \frac{(1-\beta)\left|\nabla f_k^\top p_k\right|}{L\|p_k\|_2 \|\nabla f_k\|_2} \cdot \frac{\|\nabla f_k\|_2}{\|p_k\|_2} \\ &\geq \frac{(1-\beta)\epsilon_S}{P_{\max} L} \cdot \frac{\|\nabla f_k\|_2}{\left\|P_k P_k^\top \nabla f_k\right\|_2} \geq \frac{(1-\beta)\epsilon_S}{P_{\max}^3 L}.\end{aligned}$$

Note that the first inequality follows from Lemma 2.1, while the second follows from $\|P_k\|_2 \leq P_{\max}$ in true iterations (Definition 2.1) and the submultiplicativity of the

spectral matrix norm. The result then follows. ■

We are now only left with specifying the sufficient decrease function h for Assumption 2.4.

Lemma 2.3. *Suppose iteration k is true as per* Definition 2.1. *Let $p_k = -P_k P_k^\top \nabla f_k$, with a corresponding trial iterate update $s_k = \alpha_k p_k$ for some $\alpha_k > 0$. Then* Assumption 2.4 *holds with*

$$h(\epsilon, \alpha_k) = \frac{\beta \epsilon_S^3}{P_{\max}} \alpha_k \epsilon^2. \tag{2.17}$$

Proof. The Armijo condition (2.1) implies that, in true successful iterations, $f(x_k) - f(x_k + s_k) \geq \left|\beta \alpha_k p_k^\top \nabla f_k\right|$. On the other hand, we have from Lemma 2.1 that $\left|p_k^\top \nabla f_k\right| \geq \epsilon_S \left\|\nabla f_k\right\|_2 \left\|p_k\right\|_2 / P_{\max}$. The bounds (2.13) and $\left\|\nabla f_k\right\|_2 > \epsilon$ (before convergence) also hold, and plugging them into this leads to the stated result. ■

We have thus shown that under this setup—defining true iterations as Definition 2.1, using search directions $p_k = -P_k P_k^\top \nabla f_k$, and defining iteration success using the Armijo condition (2.1)—Assumptions 2.3 to 2.5 are satisfied.

2.4.1.2 Second-Order Local Models

We now turn to cases where $B_k \neq 0$. This could encompass—for example—Newton, quasi-Newton, and Gauss-Newton methods [4], so we refer to this case as a *Newton-like* search. We use Assumption 2.6, adapted from common conditions in practical second-order methods [4, sec. 6.3].

Assumption 2.6 (Spectrum of B_k)**.** *$P_k \in \mathbb{R}^{n \times m_p}$ has full (column) rank for all $k \in \mathbb{N}$. Moreover, there are iteration-independent constants $M_1 \geq M_2 > 0$ such that, for all $k \in \mathbb{N}$, $M_1 I_n \succeq B_k$ and $\left(P_k \hat{p}_k\right)^\top B_k \left(P_k \hat{p}_k\right) \geq M_2 \left\|P_k \hat{p}_k\right\|_2^2$ for all $\hat{p}_k \in \mathbb{R}^{m_p} \setminus \{0\}$.*

In these cases we may define true iterations differently to when $B_k = 0$. Namely, we take an iteration to be true provided that at least one of the columns of P_k is uniformly bounded away from orthogonal to ∇f_k.

Definition 2.2 (True iteration in Newton-like search)**.** *Let $\left\{P_k^1, \ldots, P_k^{m_p}\right\}$ denote the set of columns of $P_k \in \mathbb{R}^{n \times m_p}$ in* Algorithm 2.2. *Iteration k is true if and only if*

$$\frac{\left|\nabla f_k^\top P_k^j\right|}{\left\|\nabla f_k\right\|_2 \left\|P_k^j\right\|_2} \geq \cos_{\min} > 0 \quad \text{for some} \quad j \in \{1, \ldots, m_p\}, \tag{2.18}$$

where $\cos_{\min} \in (0, 1)$ is an iteration-independent constant.

We now turn, once more, to finding α_{low} for Assumption 2.3, as well as bounding the norm of p_k relative to ∇f_k.

Lemma 2.4 (Partially from [13])**.** *In* Algorithm 2.2*, suppose* Assumption 2.6 *holds. Suppose, in addition, that P_k also satisfies* (2.18)*. Then* (2.8) *has a unique solution $\hat{p}_k$ such that, if we set $p_k = P_k \hat{p}_k$, then*

$$p_k = -P_k \left(P_k^\top B_k P_k\right)^{-1} \left(P_k^\top \nabla f_k\right). \tag{2.19}$$

From this it follows that

$$\|p_k\|_2 \geq \frac{\cos_{\min}}{M_1} \|\nabla f_k\|_2 . \tag{2.20}$$

Furthermore, the Armijo condition (2.1) *is then satisfied for all $\alpha \in (0, \alpha_{\text{low}})$, where*

$$\alpha_{\text{low}} = \frac{2\,(1-\beta)\,M_2}{L}. \tag{2.21}$$

Proof. Since Assumption 2.6 holds, $\left(P_k^\top B_k P_k\right) \succ 0$, so that (2.8) is a strictly convex problem and has a unique solution $\hat{p}_k$.

Given that P_k also satisfies (2.18) and we set $p_k = P_k \hat{p}_k$, the remaining results are derived in [13, sec. A.5] and [13, sec. A.4] respectively. ■

Finally, we use the Armijo sufficient decrease condition (2.1) for Assumption 2.4.

Lemma 2.5 (Sufficient decrease in Newton-like search)**.** *Suppose the conditions of* Lemma 2.4 *hold, and p_k is computed as per* (2.19)*, with a corresponding trial iterate update $s_k = \alpha_k p_k$ for some $\alpha_k > 0$. Then* Assumption 2.4*—with true iterations given by* Definition 2.2*—holds with*

$$h(\epsilon, \alpha_k) = \frac{\beta M_2 \cos^2_{\min}}{M_1^2} \alpha_k \epsilon^2. \tag{2.22}$$

Proof. Given Assumption 2.6 and with P_k satisfying (2.18), it is proved in [13, sec. A.4] that $\left|\nabla f_k^\top p_k\right| \geq M_2 \|p_k\|_2^2$. The stated result follows from plugging this bound, the bound (2.20), and $\|\nabla f_k\|_2 > \epsilon$ (before convergence) into (2.1). ■

We have thus shown that this setup, too, satisfies Assumptions 2.3 to 2.5.

2.4.1.3 Newton-like and SD-like Truthfulness Definitions Compared

It is worth observing now that Definition 2.1 (for SD-like searches) requires a **stronger** set of conditions than Definition 2.2 (for Newton-like searches), since we use this fact at multiple points in later discussions.

Lemma 2.6. *In* Algorithm 2.2*, if an iteration fits* Definition 2.1 *for some* $\epsilon_S \in (0,1)$ *and* $P_{\max} > 0$*, then it also fits* Definition 2.2 *with* $\cos_{\min} = \epsilon_S / \left(\sqrt{m_p} P_{\max}\right)$*. However,* no *analogous converse holds.*

Proof. ($\implies$) For arbitrary ∇f_k, suppose $P_k \in \mathbb{R}^{n \times m_p}$ is such that the requirements of Definition 2.1 are satisfied. Then (2.10) holds.

Suppose also, for contradiction, that the requirements of Definition 2.2 do **not** hold, that is,

$$\left|\nabla f_k^\top P_k^j\right|^2 < \cos_{\min}^2 \left\|P_k^j\right\|_2^2 \|\nabla f_k\|_2^2 \,, \ \forall j \in \{1, \ldots, m_p\} \,. \tag{2.23}$$

Summing this over j gives

$$\left(\left\|P_k^\top \nabla f_k\right\|_2^2 = \right) \sum_{j=1}^{m_p} \left|\nabla f_k^\top P_k^j\right|^2 < \cos_{\min}^2 \|\nabla f_k\|_2^2 \sum_{j=1}^{m_p} \left\|P_k^j\right\|_2^2 . \tag{2.24}$$

Together with the bound (2.10), this implies

$$\cos_{\min}^2 \|\nabla f_k\|_2^2 \sum_{j=1}^{m_p} \left\|P_k^j\right\|_2^2 > \epsilon_S^2 \|\nabla f_k\|_2^2 \implies \cos_{\min}^2 \|P_k\|_F^2 > \epsilon_S^2 . \tag{2.25}$$

It is a well-known fact that $\|P_k\|_F^2 \leq m_p \|P_k\|_2^2$ and, since P_k satisfies Definition 2.1, this implies $\|P_k\|_F^2 \leq m_p P_{\max}^2$.

This last bound and (2.25) finally imply that $m_p \cos_{\min}^2 P_{\max}^2 > \epsilon_S^2$, which in turn is a contradiction provided that $\cos_{\min} \leq \epsilon_S / \left(\sqrt{m_p} P_{\max}\right)$.

($\not\Longleftarrow$) We use a simple counterexample. Suppose, for all $k \in \mathbb{N}$, that the first column of P_k is ∇f_k. Such matrices clearly satisfy Definition 2.2 even if their other columns grow unboundedly with k, which would clearly disqualify them from satisfying Definition 2.1. ■

2.4.1.4 A Practical Unified Definition of True Iterations

In the previous discussions we gave an attempt at describing the minimal known requirements to be placed on `ConstructSubspace` in Algorithm 2.2 (or, equivalently,

on the matrices P_k) in order for our first- and second-order methods—as described thus far—to satisfy the key Assumptions 2.2 to 2.5.

Now, we propose a simpler definition of a true iteration, unifying the two that we have stated previously (Definitions 2.1 and 2.2).

Definition 2.3 (True iteration—Unified)**.** *In* Algorithm 2.2*, let* $\cos_{\min} \in (0,1)$*,* $P_{\max} > 0$*, and* $\epsilon_P > 0$ *be iteration-independent constants. Iteration* k *is true if and only if, simultaneously,* (2.18) *holds for some* $\cos_{\min} \in (0,1)$*,* $\|P_k\|_2 \leq P_{\max}$*, and* $\left\|P_k^j\right\|_2 \geq \epsilon_P \ \forall j \in \{1,\ldots,m_p\}$.[5]

This "unified" definition is the **strictest** out of all three stated. The reason we propose it is how practical it is to adapt from the simple and **least strict** Definition 2.2, while also providing a single condition suitable for both SD- and Newton-like searches (which Definition 2.2 fails to do).

Lemma 2.7. *If an iteration fits* Definition 2.3 *for some* $\cos_{\min} \in (0,1)$*,* $P_{\max} > 0$*, and* $\epsilon_P > 0$*, then it also fits* Definition 2.1 *with the same* $P_{\max}$ *and* $\epsilon_S = \cos_{\min} \epsilon_P$*. However,* no *analogous converse holds.*

Proof. ($\Longrightarrow$) We assume without loss of generality that Definition 2.2 is ensured by the first column of P_k.[6]

Now we turn to Definition 2.1. Note that the $P_{\max}$ bound is easily seen to be inherited from Definition 2.3. Consider, then, the other condition of Definition 2.1:

$$\left\|P_k^\top \nabla f_k\right\|_2^2 = \sum_{j=1}^{m_p} \left|\nabla f_k^\top P_k^j\right|^2 = \left|\nabla f_k^\top P_k^1\right|^2 + \sum_{j=2}^{m_p} \left|\nabla f_k^\top P_k^j\right|^2 \tag{2.26}$$

$$\overset{\text{Def. 2.2}}{\geq} \cos_{\min}^2 \left\|P_k^1\right\|_2^2 \|\nabla f_k\|_2^2 + \sum_{j=2}^{m_p} \left|\nabla f_k^\top P_k^j\right|^2 \tag{2.27}$$

$$\geq \cos_{\min}^2 \left\|P_k^1\right\|_2^2 \|\nabla f_k\|_2^2 \geq \cos_{\min}^2 \epsilon_P^2 \|\nabla f_k\|_2^2 . \tag{2.28}$$

We note that the final inequality follows from the assumed lower bound on the Euclidian norm of all columns of P_k. This proves the first part of the lemma.

($\not\Longleftarrow$) We use a simple counterexample. It is clear from the first part of the proof above that P_k may satisfy Definition 2.1 while having one of its columns be a zero vector, which violates one of the conditions of Definition 2.3. ■

[5] Recall that we use P_k^j to denote the jth column of P_k, interpreted as a vector.

[6] Of course, by no means would this necessarily be known to us during the algorithm—hence the need for the lower bound on *all* the columns' Euclidian norms.

Using Lemmas 2.6 and 2.7 we can chart the relative strictness of all "true iteration" definitions stated here, which we do in Figure 2.1.

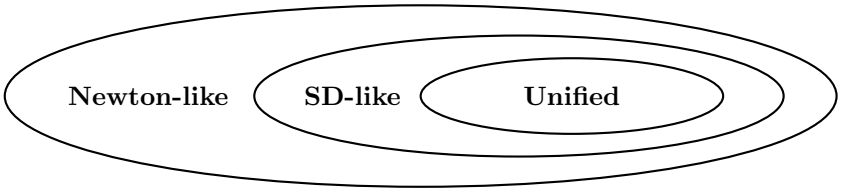

Figure 2.1: Subset relations of sets of P_k matrices (in Algorithm 2.2) satisfying the three definitions of true iterations proposed.

As we mentioned, we conclude this discussion by considering how a P_k matrix compliant with Definition 2.2 can—under the additional assumption that none of its columns are zero—easily be converted into another compliant with Definition 2.3. We make use of this observation in `L-HS`, which we describe in Chapter 3.

Lemma 2.8. *Suppose $\tilde{P}_k \in \mathbb{R}^{n \times m_p}$ satisfies the conditions of* Definition 2.2 *for some $\cos_{\min} \in (0, 1)$. Suppose additionally that none of its columns are zero vectors. Let P_k denote the matrix obtained from $\tilde{P}_k$ by normalising each of its columns to have unit Euclidian norm. Then P_k satisfies the conditions to* Definition 2.3 *with the same constant $\cos_{\min}$, $P_{\max} = \sqrt{m_p}$, and $\epsilon_P = 1$.*

Proof. Clearly the directions of the columns of P_k are the same as those of $\tilde{P}_k$, so the $\cos_{\min}$-related bound is inherited thusly.

The normalisation process trivially implies the ϵ_P-related bound (equality, in fact). It also implies that $\|P_k\|_F^2 = m_p$, which in turn implies that $\|P_k\|_2 \leq \sqrt{m_p}$. ■

2.4.2 Conditional Independence of Iteration Truthfulness

We have given definitions of iteration truthfulness suitable to ensure, given first- and second-order local models, the requirements of Assumptions 2.3 and 2.4. The particularisation for hybrid P_k construction methods is relegated to Chapter 3; however, at this point it is sensible to address the way in which Assumption 2.2 is adapted from its original statement in [18], since this is required by the main convergence and complexity result we aim to "inherit".

In [18], the original independence condition is only directly used in the proof of Lemma A.2. Close inspection of this proof reveals that, from the latest step of (A.6) to (A.7), we can replace the conditioning statements "x_{N-1}" and "$x_{N-1} = \bar{x}_{N-1}$" by "$x_\psi, \ldots, x_{N-1}$" and "$x_\psi = \bar{x}_\psi, \ldots, x_{N-1} = \bar{x}_{N-1}$" respectively, where

$\psi = \max\{0, (N-1) - p + 1\}$ for some $p \in \mathbb{N}_+$ (to be interpreted as in our Assumption 2.2). Lemma A.2 and the subsequent proof of the global convergence result are thus unaffected.

2.4.3 Global Convergence and Complexity Theorem

At last, we are able to piece together the requirements we have listed related to Assumptions 2.2 to 2.5 in Algorithm 2.2. To recap:

1. We require some notion of a "true iteration", which occurs with probability at least $1-\delta_S$ (Assumption 2.2). We consider how to practically achieve this when describing `L-HS` in Chapter 3.
2. True iterations before convergence must be guaranteed to also be successful given $\alpha_k \in (0, \alpha_{\text{low}})$, where α_{low} is iteration-independent (Assumption 2.3). We showed this in Section 2.4.1 given two classes of search directions and multiple definitions of a true iteration.
3. True successful iterations before convergence must satisfy a sufficient decrease condition which is increasing in α_k and in ϵ (Assumption 2.4). Algorithm 2.2 achieves this using the Armijo condition (2.1), and in our first- as well as second-order methods we achieve $h(\epsilon, \alpha_k) \propto \alpha_k \epsilon^2$ (see (2.17) and (2.22)).
4. Objective values must be monotonically decreasing (Assumption 2.5). This is guaranteed in Algorithm 2.2 by only updating the iterate when the Armijo condition (2.1) is satisfied.

With this, we finally state our general global convergence/complexity result.

Theorem 2.4.1 (Adapted from [18, Theorem 2.1])**.** *Let* Algorithm 2.2 *run with a target first-order optimality accuracy* $\epsilon > 0$ *and with* $\#_{\text{try}} = 1$*. Let the search direction* p_k *be obtained as described in* Section 2.4.1.1 *(first-order) or in* Section 2.4.1.2 *(second-order), with a notion of true iteration given by* Definition 2.1 *or* Definition 2.2 *respectively. Then* Assumptions 2.3 *to* 2.5 *hold with* α_{low} *given by* (2.14) *or* (2.21) *and with* h *given by* (2.17) *or* (2.22) *respectively.*

Suppose, also, that Assumption 2.2 *holds with* $\delta_S \in (0, 1)$ *such that*

$$\delta_S < \frac{c}{(c+1)^2}, \tag{2.29}$$

where $c \in \mathbb{N}_+$ *is chosen when initialising* Algorithm 2.2*.*

Now define $f^* = \min_{x \in \mathbb{R}^n} f(x)$*, as well as*

$$\xi_\alpha = \left\lceil \log_\tau \left(\min \left\{ \frac{\alpha_{\text{low}}}{\alpha_0}, \frac{1}{\nu} \right\} \right) \right\rceil \tag{2.30}$$

and

$$g\left(\delta_S, \delta_1\right) = \left[\left(1 - \delta_S\right)\left(1 - \delta_1\right) - 1 + \frac{c}{\left(c+1\right)^2} \right]^{-1}. \tag{2.31}$$

Let Algorithm 2.2 *run for* N *iterations. Then, for any* $\delta_1 \in (0, 1)$ *such that*

$$g\left(\delta_S, \delta_1\right) > 0, \tag{2.32}$$

if N *satisfies*

$$N \geq g\left(\delta_S, \delta_1\right) \left[\frac{f(x_0) - f^*}{h\left(\epsilon, \alpha_0 \tau^{(c+\xi_\alpha)}\right)} + \frac{\xi_\alpha}{1+c} \right], \tag{2.33}$$

then

$$\Pr\left\{N \geq N_\epsilon\right\} \geq 1 - \exp\left(-\frac{\delta_1^2}{2} \left(1 - \delta_S\right) N \right). \tag{2.34}$$

Proof. The first part of this result—concerning Assumptions 2.3 to 2.5—follows from the core work done in this chapter. The rest is a consequence of [18, Theorem 2.1] and the fact that Algorithm 2.2 with $\#_{\text{try}} = 1$ is a particular case of Algorithm 2.1. ■

Remark 2.1. *In both first- and second-order variants of* Algorithm 2.2*, we derived a sufficient decrease function* $h(\epsilon, \alpha_k) \propto \alpha_k \epsilon^2$ *(see* (2.17) *and* (2.22) *respectively). Putting this into* (2.33)*, we see that this probabilistic worst-case global complexity bound matches the* $O\left(\epsilon^{-2}\right)$ *worst-case bounds of deterministic, full-dimensional methods for nonconvex optimisation using first-order derivative information to achieve* ϵ*-first-order optimality* [22, sec. 2].

In this chapter we have shown that our proposed Algorithm 2.2—using $\#_{\text{try}} = 1$ and, if in a second-order variant, under Assumption 2.6—is a particular case of Algorithm 2.1, and that Theorem 2.4.1 is therefore applicable to it. To enable concrete practical implementations all that remains is to specify `ConstructSubspace`—that is, how P_k is computed at each iteration—and `ComputeDirection`—that is, how B_k is computed in second-order variants. Crucially, this entails ensuring that under suitable definitions of true iterations, Assumption 2.2 holds. These are the central concerns of Chapter 3.

Chapter 3

L-HS: Limited-Memory Hybrid Subspace Method

In this chapter we describe L-HS, a **l**imited-memory **h**ybrid **s**ubspace line search method. While our theoretical discussions in Chapter 2 could be applied to deterministic as well as fully random subspace choices, here we flesh out the "hybrid" subspace idea at the heart of this work. As we have explained, this involves using techniques which are, in some sense, between the deterministic subspace methods of [3, 10–13] and the random subspace methods of [14–19], by making use of partial/projected problem information in constructing subspaces.

From Algorithm 2.1, this particularisation essentially entails describing the functions `ConstructSubspace` (ensuring that true iterations occur with high probability) and `ComputeDirection` (ensuring, in second-order local models, that B_k satisfies Assumption 2.6). Building up to that, we keep in mind computational considerations in terms of derivative evaluations as well as numerical linear algebra costs.

The chapter is organised as follows. In Section 3.1 we motivate and describe our design of the functions `ConstructSubspace` and `ComputeDirection` within L-HS. Then, in Section 3.2, we show how L-HS fits into the theory presented in Chapter 2. We conclude in Section 3.3 by thoroughly describing the evaluation and numerical linear algebra costs of L-HS.

Note that from now on we drop the overbars used in Chapter 2 to distinguish random variables from their realisations, as the distinction becomes unimportant. Moreover, in an attempt to make descriptions easier to follow, we use indices in a liberal manner dependent on context, often referring only to successful or successful-adjacent iterations. We strive to make this clear as we go along.

3.1 Describing the L-HS Algorithm

3.1.1 Constructing the Subspace

In pursuing true iterations with high probability (Definition 2.3), the use of the current approximate gradient—which we define shortly—plays the leading role. However, taking inspiration from subspace ideas seen in existing deterministic as well as random subspace algorithms (e.g. in [13, 18]), we also consider the use of a fixed-size history of past gradients and iterate update directions, along with randomised directions.[1]

We use $\tilde{P}_k \in \mathbb{R}^{n \times m_p}$ to denote a "raw" subspace matrix obtained from assembling directions of interest into its columns. In L-HS, the subspace basis matrix itself, P_k, is invariably obtained from $\tilde{P}_k$ by either column-wise normalisation or orthogonalisation.

3.1.1.1 Including Projected Gradients in the Subspace

As we have said, one of the main draws of L-HS compared to deterministic subspace methods such as L-CommDir [13] is that at no point is the full-dimensional gradient required. Rather, given a tall *sketching matrix* $S_k \in \mathbb{R}^{n \times m_s}$ drawn from a suitable random ensemble, the algorithm requires only what we call an *approximate gradient*

$$g_k = S_k \hat{g}_k \in \mathbb{R}^n, \tag{3.1}$$

where

$$\hat{g}_k = S_k^\top \nabla f_k \in \mathbb{R}^{m_s}, \ m_s < n, \tag{3.2}$$

is the *projected gradient*.

For the sake of example, consider the case where S_k is a (full-rank) sampling matrix, that is, where each column is a distinct canonical basis vector. Then S_k is orthonormal and g_k above is the orthogonal projection of the gradient to a coordinate-aligned lower-dimensional subspace. This idea is used in popular block-coordinate descent methods [6–9]. Notably, $\hat{g}_k \in \mathbb{R}^{m_s}$ is then a vector with entries equal to a subset of the entries of ∇f_k.

In fact, it is always the case that $\hat{g}_k$ is a vector whose entries consist of m_s (scaled) directional derivatives of f at x_k. In our discussions, this motivates us to account for problem evaluation costs in terms of access to a directional derivative oracle. And

[1] Storing a **fixed-size** history of past approximate gradient and iterate step vectors is what earns L-HS its "limited-memory" qualifier. This is common practice in optimisation, as seen in the algorithm names L-CommDir [13], L-BFGS, and L-SR1 [4, sec. 9], to name but a few.

while we regard this as a black box—the focus being on the fact that only reduced-dimension derivative information is used—in practice this could correspond to the use of automatic differentiation [20] or, approximately, of finite differences [4, sec. 7.1].

3.1.1.2 Including Iterate Steps in the Subspace

Other natural directions to consider in constructing lower-dimensional subspaces are those of (successful) past iterate updates. For instance, a simple restatement of [3, Lemma 2.3], concerning common quasi-Newton methods, is that the kth iterate update lies in the subspace $\langle s_0, \ldots, s_{k-1}, \nabla f_k \rangle$.

Moreover, simple examples taken from [10, 23], as well as nonlinear conjugate gradient methods, have iterate updates satisfying $s_k \in \langle \nabla f_k, s_{k-1} \rangle$. These update terms are sometimes known as "momentum" directions—a name which comes about by analogy with the concept from classical mechanics, since each iterate update has some weight in the following update [23].

In the case of Algorithm 2.2 there is generally no simple relation between gradient and update vectors, so it may be sensible to include multiple past update vectors along multiple past approximate gradients. This is also seen, for instance, in `L-CommDir` [13] (although there the full-dimensional gradient vector is used).

3.1.1.3 Including Random Directions in the Subspace

Motivated by the encouraging results that past research has had in using randomly generated subspaces to reduce subproblem dimension (e.g. [8, 14–19]), we propose the inclusion of random directions in constructing each subspace. In `L-HS` we achieve this by drawing Gaussian vectors, where each entry is independently sampled from an identical zero-mean Gaussian distribution. Such vectors have uniformly distributed directions in $\mathbb{R}^n$—that is, Gaussian vectors are isotropic.[2] Intuitively, they can help to "shake up" the subspace information in an unbiased manner.

As we have already alluded to, in `L-HS` we obtain P_k from $\tilde{P}_k$ either by normalising each of its columns or by orthogonalising it. It therefore follows that the scale of the columns in $\tilde{P}_k$ is irrelevant, which is why the variance of the Gaussian distribution used here is irrelevant. It makes sense, however, to briefly comment on the rank

[2]This can be seen by the invariance of the Gaussian matrix distribution under orthogonal transformations—that is, if G is a Gaussian matrix (a matrix of identically distributed Gaussian vectors) and A and B are orthogonal matrices independent of G and with appropriate dimensions, then AG and BG are also Gaussian matrices [24, sec. 14.1.1].

of a matrix obtained from another by appending Gaussian columns, since this is of relevance to Assumption 2.6.

Lemma 3.1. *Suppose $A \in \mathbb{R}^{n\times m_1}$, where $n > m_1$, has full (column) rank. Suppose also that $G \in \mathbb{R}^{n\times m_2}$, with $n > m_2$ and $m_1 + m_2 \leq n$, is a Gaussian matrix (a matrix with identically distributed Gaussian vectors). Then the matrix formed from these two, $[A, B]$, has full rank with probability* 1.

Proof. Denote the columns of G with $\{g_1, \ldots, g_{m_2}\}$. Denote with E_j the event that the matrix $[A, g_1, \ldots, g_j]$ has full rank, with $j \in \{1, \ldots, m_2\}$. The superscript c denotes an event's complement.

We use an inductive argument. From the rank of A and [25, Proposition 7.1], we have that the matrix $[A, g_1]$ has full rank with probability 1. Now take the inductive hypothesis to be that $\Pr\{E_j\} = 1$ for some $j \in \{1, \ldots, m_2 - 1\}$.

We can then, for $j \in \{1, \ldots, m_2 - 1\}$, write

$$\Pr\{E_{j+1}\} = \Pr\{E_{j+1} \mid E_j\} \Pr\{E_j\} + \Pr\left\{E_{j+1} \mid E_j^c\right\} \Pr\left\{E_j^c\right\} = 1, \tag{3.3}$$

where the conclusion follows from the inductive hypothesis and from [25, Proposition 7.1]. This concludes the proof. ■

3.1.1.4 Raw Subspace Matrix $\tilde{P}_k$ Construction Schemes

We are now in a position to propose two $\tilde{P}_k$ construction schemes. The first option is to use $p > 0$ past approximate gradients, so that

$$\tilde{P}_k = \left[\begin{array}{c|c|c|c|c} g_k & \cdots & g_{k-p+1} & G_k \end{array}\right] \in \mathbb{R}^{n\times(p+r)}, \tag{3.4}$$

where $G_k \in \mathbb{R}^{n\times r}$ is a Gaussian matrix ($r = 0$ is acceptable). The second option is to *also* use p past iterate updates, so that

$$\tilde{P}_k = \left[\begin{array}{c|c|c|c|c|c|c} g_k & s_{k-1} & \cdots & g_{k-p+1} & s_{k-p} & G_k \end{array}\right] \in \mathbb{R}^{n\times(2p+r)}. \tag{3.5}$$

We then obtain P_k by either normalising each column of P_k or by generating an orthonormal basis for the subspace range$\left\{\tilde{P}_k\right\}$, which we now discuss more carefully.

3.1.1.5 Constructing the Subspace Basis Matrix P_k From $\tilde{P}_k$

In Chapter 2 we discussed a unified definition of a true iteration—Definition 2.3—suitable for search directions arising out of linear as well as convex quadratic local

models. We also demonstrated, in Lemma 2.8, how this could be achieved for P_k in practice by **normalising each column** of a matrix $\tilde{P}_k$ satisfying the conditions of Definition 2.2.

Further to this, here we propose another practical method geared toward Definition 2.3. After generating a raw subspace matrix $\tilde{P}_k$—using approximate gradients, previous updates, and/or random directions—we obtain P_k from the Q factor of a QR factorisation of $\tilde{P}_k$. Thus P_k is **orthonormal**, that is, $P_k^\top P_k = I_{m_p}$.

Clearly this method satisfies Definition 2.3 under the same conditions as does the column-wise normalisation method. However, for one, numerical experience shows this to enable improved performance (Chapter 4). Moreover, this is in some sense a more natural way to express the subspace, making some notions regarding the reduced-dimensional subproblem (2.8) closely analogous to full-space counterparts.

For one, with this method the SD-like search direction in (2.9) is simply the orthogonal projection of the gradient to the subspace spanned by P_k. Furthermore, the direction arising out of a convex quadratic local model (given by (2.19)) with $B_k = I_n$ is $-P_k P_k^\top \nabla f_k$—the same as when using a linear model. This "match" is also seen in full-space line search methods when a local quadratic model's B_k is the identity matrix.

As an additional benefit, note that this method guarantees that P_k has full rank, as required of second-order `L-HS` variants by Assumption 2.6.

3.1.2 Specifying `ConstructSubspace` and `ComputeDirection` Functions

In fitting `L-HS` under Algorithm 2.2, it remains for us to fully specify the functions `ConstructSubspace` and `ComputeDirection`.

Informed by the discussions in Section 3.1.1, we describe `ConstructSubspace` straight away in Algorithm 3.1. We only gloss over the first few calls to it in a given solver run—while not enough problem information (past approximate gradients and update steps) has been accumulated, Gaussian directions are used to fill in as many columns as is required to make up m_p directions in P_k.

Before describing `ComputeDirection`, we make a small detour to discuss a computationally efficient way to ensure that Assumption 2.6 holds in second-order `L-HS` variants using Hessian information.

Algorithm 3.1 `ConstructSubspace` in Algorithm 2.2 for `L-HS`.

Input: Current "raw" and processed subspace matrices, $\tilde{P}_k$ and P_k respectively. Also possibly the "last" iterate update s_{k-1}.
Initialisation: (At the start of Algorithm 2.2) Choose $p \in \mathbb{N}_+$ (number of problem-based directions) and $r \in \mathbb{N}$ (number of Gaussian directions) such that $n \geq m_p := p + r$. Choose whether to process $\tilde{P}_k$ using orthogonalisation or just column-wise normalisation.

Replace r right-most columns of current $\tilde{P}_k$ with r new randomised columns
Generate sketching matrix $S_k \in \mathbb{R}^{n \times m_s}$
Compute projected gradient $\hat{g}_k \leftarrow S_k^\top \nabla f_k$
Compute approximate gradient $g_k \leftarrow S_k \hat{g}_k$
Replace left-most column of $\tilde{P}_k$ with g_k
if following successful iteration **then** ▷ Line 8 of Algorithm 2.2.
    if using iterate update directions **then** ▷ (3.5).
        Replace second column from the left in $\tilde{P}_k$ with s_{k-1}
    end if
end if
Process $\tilde{P}_k$, by orthogonalisation or by column-wise normalisation, to rewrite P_k
return new subspace matrix P_k

3.1.2.1 Ensuring Bounds on the Spectrum of the Curvature Matrix B_k

Recall Assumption 2.6, concerning the matrix B_k, and which we used in our analysis of second-order methods. Thanks to the following lemma, we can ensure those conditions by using Hessian information along with a regularisation triggered by the smallest eigenvalue of $P_k^\top \nabla^2 f_k P_k \in \mathbb{R}^{m_p \times m_p}$. In this way we can avoid computing the smallest eigenvalue of the (much) larger Hessian matrix itself.

Lemma 3.2. *In* Algorithm 2.2*, suppose that f satisfies* Assumption 2.1 *and that $P_k \in \mathbb{R}^{n \times m_p}$ has full rank and columns with unit Euclidian norm for all $k \in \mathbb{N}$. If $\hat{B}_k$ is computed as per* Algorithm 3.2 *(second-order variant), this is equivalent to* Assumption 2.6 *holding with $M_2 = \lambda_{\text{reg}}/m_p$ and $M_1 = \lambda_{\text{reg}} + L m_p$ (where λ_{reg} is defined in* Algorithm 3.2*).*

Proof. When triggered, the regularisation procedure of Algorithm 3.2 is equivalent to regularising the Hessian matrix itself, since $\left(P_k^\top \nabla^2 f_k P_k\right) + (\lambda_{\text{reg}} - \lambda_{\min}) I_{m_p} = P_k^\top \left(\nabla^2 f_k + (\lambda_{\text{reg}} - \lambda_{\min}) I_n\right) P_k$ [13, p. 13]. With this in mind, let $B_k \in \mathbb{S}^n$ denote the matrix obtained from $\nabla^2 f_k$ after having regularised it (or not) with $\lambda_{\text{reg}} - \lambda_{\min}$, as described in Algorithm 3.2.

Thus, for all $\hat{p}_k \in \mathbb{R}^{m_p}$, we have that $(P_k \hat{p}_k)^\top B_k (P_k \hat{p}_k) = \hat{p}_k^\top \left(P_k^\top B_k P_k\right) \hat{p}_k \geq \lambda_{\text{reg}} \|\hat{p}_k\|_2^2$. On the other hand $\|P_k\|_2^2 \leq \|P_k\|_F^2 = m_p$, so $\|P_k \hat{p}_k\|_2^2 \leq m_p \|\hat{p}_k\|_2^2 \implies$

$\|\hat{p}_k\|_2^2 \geq \|P_k\hat{p}_k\|_2^2 / m_p$. This bound and the one in the previous sentence give the M_2-related bound in Assumption 2.6.

Now, since ∇f is L-Lipschitz-continuous, $LI_n \succeq \nabla^2 f(x) \succeq -LI_n$ for all $x \in \mathbb{R}^n$. Therefore $\lambda_{\min} = \min_{\|y\|_2=1} (P_k y)^\top \nabla^2 f_k (P_k y) \geq -Lm_p$; the last inequality follows from $\|P_k y\|_2^2 \leq \|P_k\|_2^2 \leq m_p$ for an arbitrary unit vector $y \in \mathbb{R}^{m_p}$ and with $\|P_k\|_F^2 = m_p$. This then implies $\nabla^2 f_k + (\lambda_{\text{reg}} - \lambda_{\min}) I_n \preceq (\lambda_{\text{reg}} + Lm_p) I_n$, as required. ■

Algorithm 3.2 `ComputeDirection` in Algorithm 2.2 for `L-HS`.

Input: Current subspace matrix $P_k \in \mathbb{R}^{n \times m_p}$.
Initialisation: (At the start of Algorithm 2.2) Choose between first- and second-order local models. If using second-order models, choose a constant $\lambda_{\text{reg}} > 0$.

if using first-order local models **then**
  $\hat{g}_k \leftarrow P_k^\top \nabla f_k$ ▷ Projected gradient.
  $p_k \leftarrow -P_k \hat{g}_k$
else ▷ Using second-order local models.
  Compute[3] "projected" Hessian $P_k^\top \left(\nabla^2 f_k P_k\right) \in \mathbb{R}^{m_p \times m_p}$
  Compute $\lambda_{\min}$, the smallest eigenvalue of $P_k^\top \nabla^2 f_k P_k$
  if $\lambda_{\min} < \lambda_{\text{reg}}$ **then**
    $\hat{B}_k \leftarrow \left(P_k^\top \nabla^2 f_k P_k\right) + (\lambda_{\text{reg}} - \lambda_{\min}) I_{m_p}$ ▷ Regularise.
  else
    $\hat{B}_k \leftarrow P_k^\top \nabla^2 f_k P_k$
  end if
  Solve $\hat{B}_k \hat{p}_k = -P_k^\top \nabla f_k$ ▷ Symmetric positive-definite $m_p \times m_p$ system.
  $p_k \leftarrow P_k \hat{p}_k$
end if
return search direction p_k

3.1.3 Summary

Finally, we can assemble `L-HS` easily using Algorithms 2.2, 3.1, and 3.2, which we do in Algorithm 3.3.

To shorten terminology, we use `L-HS-SD` ("`SD`" for "steepest descent") and `L-HS-N` ("`N`" for Newton) to refer to the first- and second-order `L-HS` classes described.

[3] The parentheses in this expression serve only to allude to the computation of $\nabla^2 f_k P_k$ via m_p Hessian actions, similarly to what we do computing $\hat{g}_k = S_k^\top \nabla f_k$, only then followed by $g_k = S_k \hat{g}_k$.

Algorithm 3.3 `L-HS`: **L**imited-memory **H**ybrid **S**ubspace method for (1.1).

Initialisation: Initialise as in Algorithm 2.2. Moreover, let the functions `ConstructSubspace` and `ComputeDirection` be defined by Algorithm 3.1 and Algorithm 3.2 respectively (each also has its own initialisation procedure).

1: Run Algorithm 2.2

3.2 L-HS in the Theory Framework of Chapter 2

Having fully described first- and second-order `L-HS` classes, `L-HS-SD` and `L-HS-N` respectively, we can now demonstrate how they fit into the theory that we discussed in Chapter 2. We begin by describing the choice of two suitable gradient sketching matrix ensembles.

3.2.1 Choice of Gradient Sketching Matrix Ensemble

A crucial aspect of `L-HS` is to make use of (randomly) projected gradient information to construct subspaces that enable good performance. Our "unified" Definition 2.3 hints at a familiar way in which we may view this issue: it is desirable to have the approximate gradient g_k uniformly bounded away from orthogonal to ∇f_k with high probability.

We denote sketching matrices by $S_k \in \mathbb{R}^{n \times m_s}$. Given $g_k = S_k \left(S_k^\top \nabla f_k\right)$, we know from Lemma 2.1 that having g_k bounded away from orthogonal to ∇f_k is ensured if S_k satisfies the conditions that Definition 2.1 imposes on P_k.

Fortunately, previous work has been done demonstrating random matrix ensembles that can satisfy these requirements with high probability—see [16, secs. 2 and 4.4.2]. Lemma 3.3 is useful in demonstrating this, and in its statement we recall these two key conditions.

Lemma 3.3 (Adapted from [18, Lemma 3.2])**.** *Let an arbitrary $\nabla f_k \in \mathbb{R}^n$ be fixed. Suppose a random matrix $S_k \in \mathbb{R}^{n \times m_s}$ is drawn from an ensemble such that it satisfies*

$$\left\|S_k^\top \nabla f_k\right\|_2 \geq \epsilon_S \left\|\nabla f_k\right\|_2 \tag{3.6}$$

for some $\epsilon_S \in (0, 1)$ with probability at least $1 - \delta_S^{(1)}$, and satisfies

$$\left\|S_k\right\|_2 \leq S_{\max} \tag{3.7}$$

for some $S_{\max} > 0$ with probability at least $1 - \delta_S^{(2)}$. If $\delta_S^{(1)} + \delta_S^{(2)} < 1$, then S_k satisfies

both conditions simultaneously with probability at least $1 - \left(\delta_S^{(1)} + \delta_S^{(2)}\right)$.

3.2.1.1 Scaled Gaussian Matrices

It is shown in [16, sec. 4.4.2] that a suitable option for our purposes is to draw S_k with entries from an appropriately scaled Gaussian distribution.

Definition 3.1 (Scaled Gaussian matrix [18, Def. 3.2])**.** *$S_k \in \mathbb{R}^{n \times m_s}$ is a scaled Gaussian matrix if its entries are independently drawn from $\mathcal{N}(0, 1/m_s)$.*

The following lemmas, also from [16, sec. 4.4.2], concretise the conditions to ensure that g_k is bounded away from orthogonal to the actual gradient.

Lemma 3.4 ([16, Lemma 4.4.4])**.** *Let $S_k \in \mathbb{R}^{n \times m_s}$ be a scaled Gaussian matrix, and fix an arbitrary $\nabla f_k \in \mathbb{R}^n$. Then* (3.6) *is satisfied with any $\epsilon_S \in (0, 1)$ with probability at least $1 - \delta_S^{(1)}$, where $\delta_S^{(1)} = \exp\left(-(1 - \epsilon_S^2)^2 m_s/4\right)$.*

Lemma 3.5 ([16, Lemma 4.4.6])**.** *Let $S_k \in \mathbb{R}^{n \times m_s}$ be a scaled Gaussian matrix. Then it satisfies* (3.7) *with probability at least $1 - \delta_S^{(2)}$ and with*

$$S_{\max} = 1 + \sqrt{\frac{n}{m_s}} + \sqrt{\frac{2 \log\left(1/\delta_S^{(2)}\right)}{m_s}}. \tag{3.8}$$

Remark 3.1. *Note that we can choose $\delta_S^{(2)}$ as small as we would like provided that $S_{\max}$ is allowed to be large enough.*[4] *On the other hand, the smallness of $\delta_S^{(1)}$ is limited by the sketch dimension m_s; even in the limit as we let $\epsilon_S \to 0$, we have $\delta_S^{(1)} \to \exp(-m_s/4)$. Recalling* Lemma 3.3 *and the fact that* Theorem 2.4.1 *requires a (conditional) probability of iteration truthfulness of at least* $3/4$*—note* (2.29)*—this implies that, for the theory to be applicable, there is the minimum requirement that* $\exp(-m_s/4) < 1/4 \implies m_s \geq 6$.

3.2.1.2 Haar-Distributed Orthonormal Matrices

The Haar measure is the uniform measure on orthonormal matrices [26, sec. 4.6]. Taking into account our previous discussions, in Section 3.1.1.5, regarding orthogonalisation of subspace basis matrices, we may also wish to use matrices from this ensemble to project gradient information.

[4] Of course, shrinking $\delta_S^{(2)}$ has implications on the constants of proportionality in our worst-case global complexity bounds; see (3.9) and (3.11) in Corollaries 3.1 and 3.2 respectively.

In order to generate Haar-distributed orthonormal matrices, [27, sec. 5] proposes a simple method based on computing a "constrained" QR factorisation of a Gaussian matrix. We can use this and previous results to show that this ensemble inherits its suitability for our purposes from that of scaled Gaussian matrices.

Lemma 3.6. *(In the following, tildes are used to distinguish constants playing similar roles for a matrix S_k (no tilde) and $\tilde{S}_k$ (with a tilde).)*

Let $S_k \in \mathbb{R}^{n\times m_s}$ be a Haar-distributed orthonormal matrix. Fix constants $\tilde{\epsilon}_S \in (0,1)$ and $\tilde{S}_{\max} > 0$ such that a scaled Gaussian matrix $\tilde{S}_k$ with the same dimensions satisfies simultaneously both conditions of Lemma 3.3 *with probability at least $\mu \in (0,1)$. Then, for arbitrary $\nabla f_k \in \mathbb{R}^n$, S_k satisfies the conditions of* Lemma 3.3 *with $S_{\max} = 1$ and $\epsilon_S = \tilde{\epsilon}_S / \left(\sqrt{m_s}\tilde{S}_{\max}\right)$ with probability at least μ.*

Proof. The proof is most easily followed through the subset relations of Figure 2.1, replacing P_k with S_k (and so on in a natural manner) in the lemmas and equations that we refer to.

Our assumptions imply that $\tilde{S}_k$ has a probability of at least μ of satisfying the conditions of Definition 2.1. Such a success, in turn, implies the satisfaction of (2.18) with $\tilde{\cos}_{\min} = \tilde{\epsilon}_S / \left(\sqrt{m_s}\tilde{S}_{\max}\right)$ (Lemma 2.6). Since S_k is an orthonormal basis matrix for $\tilde{S}_k$, it is guaranteed in turn to fall under Definition 2.3 with $\cos_{\min} = \tilde{\epsilon}_S / \left(\sqrt{m_s}\tilde{S}_{\max}\right)$ and $S_{\max} = 1$, which implies that it also falls under Definition 2.1 with $\epsilon_S = \tilde{\epsilon}_S / \left(\sqrt{m_s}\tilde{S}_{\max}\right)$ (Lemma 2.7) and $S_{\max} = 1$. ■

3.2.2 `L-HS` Corollaries of Theorem 2.4.1

We now show how our methods fit into the theory of Chapter 2 by presenting Corollaries 3.1 and 3.2—consequences of the principal convergence/complexity result, Theorem 2.4.1, applied to `L-HS` using scaled Gaussian sketching matrices.

Corollary 3.1 (Convergence/complexity of `L-HS-SD` variants)**.** *Let a variant of* `L-HS-SD` *be run with $\#_{\text{try}} = 1$, $p \geq 1$, and with sketching matrices $S_k \in \mathbb{R}^{n\times m_s}$ drawn randomly as scaled Gaussians. If P_k is obtained from $\tilde{P}_k$ by orthogonalisation, let $P_{\max} = 1$; else—if column-wise normalisation is used—let $P_{\max} = \sqrt{m_p}$ and assume that P_k has no zero columns for all $k \in \mathbb{N}$. Now let $\delta_S^{(2)}, \delta_1$, and ϵ_S be constants in $(0,1)$ such that, by defining $\delta_S = \exp\left(-\left(1-\epsilon_S^2\right)^2 m_s/4\right) + \delta_S^{(2)}$, we have that* (2.29) *and* (2.32) *hold.*

Then Theorem 2.4.1 *is applicable with*

$$h(\epsilon, \alpha_k) = \frac{\beta \epsilon_S^3}{P_{\max}} \left[1 + \sqrt{\frac{n}{m_s}} + \sqrt{2 {m_s}^{-1} \log\left(1/\delta_S^{(2)}\right)} \right]^{-3} \alpha_k \epsilon^2. \tag{3.9}$$

Therefore, if this `L-HS-SD` *variant runs for N iterations where N satisfies* (2.33)*, the convergence probability bound* (2.34) *holds.*

Proof. Since the aforementioned algorithm is a (first-order) variant of `L-HS` with $\#_{\text{try}} = 1$, it is a particular case of Algorithm 2.2 for which Theorem 2.4.1 is applicable.

Now, Assumption 2.2 is ensured by the use of Definition 2.3 ("unified") of true iterations for constants $\cos_{\min} \in (0, 1)$, $P_{\max}$ as stated, and $\epsilon_P = 1$.[5]

For the purpose of having a direction in P_k whose cosine with ∇f_k is larger (in absolute) than $\cos_{\min}$ with high probability, we "rely" solely on the current approximate gradient g_k. Then, by Lemma 2.1, a true iteration occurs when the scaled Gaussian matrix S_k satisfies the conditions (3.6) and (3.7) simultaneously, for fixed constants $\epsilon_S \in (0, 1)$ and $S_{\max} > 0$, with the relation $\cos_{\min} = \epsilon_S / S_{\max}$ (arising from Lemma 2.1).

Now consider δ_S; as stated in Assumption 2.2, the (conditional) probability of a true iteration is bounded below by $1 - \delta_S$, and our definition of δ_S above comes from applying Lemmas 3.3 to 3.5 with the aforementioned constants ϵ_S and $S_{\max}$.

It remains only to determine $h(\epsilon, \alpha_k)$. We must first adapt (2.17) to our present use of the "unified" definition of a true iteration. By applying Lemma 2.7, one can easily show that it becomes

$$h(\epsilon, \alpha_k) = \frac{\beta \epsilon_P^3 \cos_{\min}^3}{P_{\max}} \alpha_k \epsilon^2. \tag{3.10}$$

Now we put $\epsilon_P = 1$ and $\cos_{\min} = \epsilon_S / S_{\max}$ into (3.10). Further, we substitute $S_{\max}$ as per (3.8). We thus arrive at $h(\epsilon, \alpha_k)$ as given in (3.9) above. ■

Corollary 3.2 (Convergence/complexity of second-order `L-HS` variants)**.** *Let a second-order variant of* `L-HS` *be run with $\#_{\text{try}} = 1$ and $p \geq 1$, such that* Assumption 2.6 *holds with constants $M_1 \geq M_2 > 0$, and with sketching matrices $S_k \in \mathbb{R}^{n \times m_s}$ drawn*

[5] Recall that ϵ_P is, in Definition 2.3, a lower bound on the Euclidian norm of all columns of P_k. In `L-HS`, whether using column-wise normalisation (provided none of the columns in $\tilde{P}_k$ are zero) or orthogonalisation, this clearly holds with $\epsilon_P = 1$.

randomly as scaled Gaussians.[6] *Let* $\delta_S^{(2)}, \delta_1$, *and* ϵ_S *be constants in* $(0,1)$ *such that, by defining* $\delta_S = \exp\left(-\left(1-\epsilon_S^2\right)^2 m_s/4\right) + \delta_S^{(2)}$, *we have that* (2.29) *and* (2.32) *hold.*

Then Theorem 2.4.1 *is applicable with*

$$h(\epsilon, \alpha_k) = \frac{\beta M_2 \epsilon_S^2}{M_1^2}\left[1 + \sqrt{\frac{n}{m_s}} + \sqrt{2{m_s}^{-1}\log\left(1/\delta_S^{(2)}\right)}\right]^{-2}\alpha_k\epsilon^2. \tag{3.11}$$

Therefore, if this `L-HS` *variant runs for* N *iterations where* N *satisfies* (2.33)*, the convergence probability bound* (2.34) *holds.*

Proof. The proof is very closely analogous to that of Corollary 3.1, so we proceed to describe only the differing aspects between them. The aforementioned algorithm is a second-order variant of `L-HS` satisfying the conditions of Corollary 3.1 and, additionally, Assumption 2.6. Therefore Theorem 2.4.1 is applicable.

Assumption 2.2 can this time be ensured by the use of Definition 2.2 (Newton-like) of true iterations, for a constant $\cos_{\min} \in (0,1)$. We determine $h(\epsilon, \alpha_k)$ by putting $\cos_{\min} = \epsilon_S / S_{\max}$ into (2.22). Further, we substitute $S_{\max}$ as per (3.8). This completes the proof. ■

Remark 3.2. *From* (2.33)*,* (3.9)*, and* (3.11)*, note that* N *grows asymptotically (as* $n \to \infty$*) as* $O\left(\left(n/m_s\right)^{3/2}\epsilon^{-2}\right)$ *and as* $O\left(\left(n/m_s\right)\epsilon^{-2}\right)$ *for convergence in first- and second-order* `L-HS` *variants respectively. In this sense, the latter matches results such as* [18, Theorem 4.1] *and* [18, Theorem 4.4]*, also obtained using scaled Gaussian sketching matrices.*

By applying Lemma 3.6, we could similarly obtain complexity bounds with the use of Haar-distributed orthonormal sketching matrices. It suffices to follow the same steps as above, but with substitutions put in place for ϵ_S and $S_{\max}$ relative to the corresponding scaled Gaussian constants (denoted by $\tilde{\epsilon}_S$ and $\tilde{S}_{\max}$ in Lemma 3.6).

Moreover, one could analogously—albeit with more substantive modifications—analyse similar random subspace methods, that is, where the subspace matrix P_k is purely random ($p = 0$ in (3.4)). We omit this for brevity and because this analysis would closely resemble that of existing literature on random subspace methods.[7]

[6] N.B.: Here we do **not** refer to `L-HS-N` because this result applies broadly to `L-HS` variants using local quadratic models satisfying the relevant assumptions; `L-HS-N` is a further particularisation of this, which this result does not require.

[7] Namely we could define, for both first- and second-order variants, true iterations by Definition 2.1. Then the high-probability results we have stated for scaled Gaussian and Haar-distributed orthonormal matrices could be easily applied, and the analysis on the whole would very closely resemble the proofs of [18, Theorem 4.1] and [18, Theorem 4.4].

3.2.3 Summary

In this section we have demonstrated that, provided $\#_{\text{try}} = 1$, `L-HS` "inherits" the theory presented in Chapter 2. We proposed the use of two sketching matrix ensembles (scaled Gaussian and Haar-distributed orthonormal matrices), and presented global convergence/complexity corollaries of Theorem 2.4.1 in the context of `L-HS`.

3.3 `L-HS` Costs per Iteration

As far as `L-HS` is concerned, one obvious facet of analysis remains to be carried out: the costs per iteration, whether in numerical linear algebra operations or in problem (derivative) information. We address these in detail in Sections 3.3.1 and 3.3.2.

3.3.1 Numerical Linear Algebra Costs

The numerical linear algebra (NLA) costs associated with `L-HS` depend on the choice of algorithm attributes. These are: first- versus second-order local models; use of $\tilde{P}_k$ orthogonalisation versus column-wise normalisation; use of scaled Gaussian versus Haar-distributed orthonormal sketching matrices. We discuss these issues in sequence.

3.3.1.1 First- Versus Second-Order Variants

The choice between linear and (strictly) convex quadratic local models affects how the search direction p_k is computed.

Using **linear models**, this is done simply by $p_k = -P_k \left(P_k^\top \nabla f_k\right)$, which costs $O\left(m_p n\right)$ flops.

Using **quadratic models** comes about with additional expense. We must ensure Assumption 2.6, which, owing to Lemma 3.2, can be done in `L-HS` by regularising just the small matrix $P_k^\top B_k P_k$. This entails the computation of its eigenvalues, which can be done in $O\left(m_p^3\right)$ flops [24, sec. 10].

We must then form the linear system $\left(P_k^\top B_k P_k\right) \hat{p}_k = -P_k^\top \nabla f_k$, which costs $O\left(m_p n^2 + m_p^2 n\right)$ flops. Under Assumption 2.6,[8] this is a determined $m_p \times m_p$ system, which can be solved with classical methods in $O\left(m_p^3\right)$ flops. Computing $p_k = P_k \hat{p}_k$ is done at a smaller cost of $O\left(m_p n\right)$ flops.

[8] In the author's practical experience numerical rank deficiency of P_k was never observed (although admittedly most of this experience made use of orthonormal P_k). We also recall our discussion in Section 3.1.1.3 regarding the effect of the inclusion of Gaussian columns in P_k on its rank.

3.3.1.2 Orthogonalised Versus Column-Normalised Subspace Matrix P_k

In the case where P_k is obtained from $\tilde{P}_k$ by **column-wise normalisation**, the choice of subspace directions matters due to the recycled effort in having already normalised some of the columns that remain after a subspace update. The introduction of a new approximate gradient (and possibly a new update vector) entails $O(n)$ flops in normalisation. The introduction of r new randomised columns requires $O(rn)$ flops. Accounting for cases including $r = 0$, we abuse notation slightly to represent these costs as $O(n(r+1))$ flops.

In the case where P_k is **orthonormal**, the choice of directions defining the subspace matters due to the possible numerical efficiency in updating the QR factorisation of matrices derived from deleting/appending columns from/to a matrix whose factorisation we already have [28, sec. 6.5.2] (as opposed to recomputing it anew).

Recall (3.4) and (3.5); each time $\tilde{P}_k$ is updated following a successful iteration: the Gaussian columns (submatrix G_k) are deleted, along with the most recent stored approximate gradient g_k and, if applicable, iterate update s_{k-1}; a new set of r Gaussian columns is drawn and appended to the right; a new approximate gradient and, if applicable, iterate update are appended to the left. When $\tilde{P}_k$ is updated following an unsuccessful iteration due to the backtracking limit $\#_{\text{try}}$ being hit, the only difference is that nothing is done about stored iterate updates, but this has little effect here.

If we already have the QR factorisation of $\tilde{P}_k$, we also have the factorisation of the matrix obtained from it by deleting however many right-most rows (this follows from the R factor being upper-triangular).

What remains then is to update the factorisation due to the (re-)introduction of either $r+1$ or $r+2$ columns.[9] Let $a \in \{r+1, r+2\}$ denote the number of columns to be appended at such an update. Using [28, sec. 6.5.2] we deduce that this update can be carried out in $\sum_{j=m_p-a+1}^{m_p} jn = an(m_p - (a-1)/2)$ flops.

As one considers problems of increasing ambient dimension n, this is especially advantageous (compared to recomputing the factorisation in full each time) when $a = O(1)$; for instance, if $r = 0$ or if the number of Gaussian directions in $\tilde{P}_k$ is set to a constant independent of n. On the other hand, if $r = O(m_p)$ this cost is asymptotically the same as that of the naive approach.

Note that we can implicitly generate a Haar-orthonormal matrix to apply it to vectors without having to incur the costs of a QR factorisation as usual. Unfortunately,

[9] If using (3.4), we append $r+1$ columns here, regardless of whether the P_k update follows a successful or an unsuccessful iteration. If using (3.5), we append $r+1$ after unsuccessful iterations and $r+2$ after successful ones. In any case, this distinction is negligible here.

this "trick" fails to materialise in L-HS when generating Haar-orthonormal **sketching** matrices, due to our interpretation of $\hat{g}_k = S_k^\top \nabla f_k$ as m_s directional derivatives. Since it can, in any case, be used in random subspace methods, we describe this trick in Appendix A.1.

3.3.1.3 Scaled Gaussian Versus Haar-Distributed Orthonormal Gradient Sketching Matrix S_k

The generation of Gaussian entries when using scaled Gaussian sketching matrices entails a small, fixed arithmetic cost per entry, so such a sketching matrix takes $O(nm_s)$ flops to generate.

In generating a Haar-distributed orthonormal matrix explicitly, we resort to the QR factorisation of a Gaussian matrix [27, sec. 5]. This has the costs of an ordinary QR factorisation of $S_k \in \mathbb{R}^{n \times m_s}$, which is $O(nm_s^2)$ flops.

3.3.2 Derivative Evaluation Costs

We now move to discuss one of the main motivations of L-HS: the use of incomplete objective gradient information. As we have mentioned, our elementary unit of measurement of derivative information cost is the directional derivative, that is, an inner product $v^\top \nabla f_k$ for arbitrary $v, x_k \in \mathbb{R}^n$.

Note that in Appendix A.2 we discuss these costs in methods where the subspace matrix P_k is entirely randomised, which is useful here because we use such methods in numerical tests in Chapter 4.

3.3.2.1 Typical L-HS Variants

We first consider "typical" L-HS variants, that is, where neither m_p nor m_s are equal to the ambient dimension n.

Derivative costs are incurred in L-HS exactly whenever the subspace matrix P_k is updated, but an important distinction to make is whether this follows a successful iteration or an unsuccessful iteration.

Now, consider the case where a **successful** iteration has just occurred and the iterate has just been updated. The matrix $P_k \in \mathbb{R}^{n \times m_p}$ is to be updated, which requires computing a new approximate gradient $g_k = S_k \hat{g}_k$, where $S_k \in \mathbb{R}^{n \times m_s}$ is a sketching matrix. This entails m_s directional derivatives to compute $\hat{g}_k = S_k^\top \nabla f_k$.

Having updated P_k—with a new approximate gradient but also possibly a new iterate update vector and new randomised columns—we must then compute $P_k^\top \nabla f_k$.

We note that here the directional derivative along the approximate gradient g_k is "recycled", since $\nabla f_k{}^\top g_k = \|\hat{g}_k\|_2^2$. This therefore entails $m_p - 1$ derivatives. This is all that is needed following successful iterations in **first-order**/`L-HS-SD` variants.

In `L-HS-N` variants, the use of objective curvature information entails significant additional derivative costs. We choose to measure them by an approach based on finite differences of ∇f [4, Eq. (7.20)], taking a gradient evaluation therein to be equivalent to n directional derivative evaluations. Following a successful iteration, this results in $\nabla^2 f_k P_k$ costing $(m_p + 1)\, n$ directional derivatives.

Now consider that we are updating P_k following an **unsuccessful** iteration (because the backtracking limit $\#_{\text{try}}$ has been reached). Again, m_s derivatives are required in evaluating a new approximate gradient g_k. However, when computing $P_k^\top \nabla f_k$ we now recycle all derivatives except for the ones along the r newly drawn random directions; this is because each of the past stored approximate gradients and iterate updates is unchanged. So $P_k^\top \nabla f_k$ now entails just r derivatives, which is again sufficient in **first-order**/`L-HS-SD` variants.

In `L-HS-N` variants, $\nabla^2 f_k P_k$ now requires recomputing the actions associated with newly drawn random directions, along with the action along the direction of the newly determined approximate gradient direction. This entails $(1 + r)\, n$ directional derivative evaluations.[10]

3.3.2.2 Edge Cases

We now consider atypical edge cases of `L-HS`, where either m_p or m_s is equal to n. Note that in these cases, a step change comes about in practice: simply, when $m_p = n$ we set $P_k = I_n$, and when $m_s = n$ we set $S_k = I_n$ (for all $k \in \mathbb{N}$).

Briefly, in **first-order** variants, both cases have the same consequence—turning `L-HS-SD` effectively into a deterministic full-space method, whereby P_k updates following successful iterations incur n directional derivatives and those following unsuccessful updates incur 0 derivatives.

In **second-order** variants, the two cases are different. When $m_s = n$, the first-order information costs at successful iterations are simply of n directional derivatives, and the costs associated with Hessian actions go from $(m_p + 1)\, n$ to $m_p n$. At unsuccessful updates, the first-order derivative information cost is 0. The other case,

[10] N.B.: in our thinking of these costs as if in finite differences of ∇f, the case following a successful iteration would require the computing of the "central" value—∇f_k itself, whereas the case following an unsuccessful iteration would not (owing to this having been done following the latest successful iteration).

where $m_p = n$, supersedes this; again the method becomes a full-space one, so that successful iterations incur $(n+1)\,n$ directional derivatives, and unsuccessful updates incur 0 derivatives.

3.3.3 Summary

Following our detailed discussions of NLA and derivative costs of `L-HS`, we summarise it all in Tables 3.1 and 3.2 respectively. Note that, in Table 3.2, we leave aside the edge cases discussed with respect to derivative costs.

Note that we use `RS` to refer to **r**andom **s**ubspace analogues of `L-HS`. To this, we append suffixes `-SD` and `-N` playing the same role as for `L-HS`.

Table 3.1: Summary of NLA costs of `L-HS` variants. Recall the definition of a from Section 3.3.1.2.

Algorithm Step	**Variant Attribute**	**Cost (Flops)**
Generate sketching matrix $S_k \in \mathbb{R}^{n \times m_s}$	S_k scaled Gaussian	$O(nm_s)$
	S_k Haar-orthonormal	$O(nm_s^2)$
Compute subspace matrix $P_k \in \mathbb{R}^{n \times m_p}$	P_k **not** orthonormal	$O(n(r+1))$
	P_k orthonormal	$an(m_p - (a-1)/2)$
	(`RS` only) P_k orthonormal	$O(nm_p)$
Compute search direction $p_k \in \mathbb{R}^n$	First-order	$O(nm_p)$
	Second-order	$O(n^2 m_p)$

Table 3.2: Summary of (derivative) evaluation costs of considered algorithms. Costs are incurred immediately after the subspace basis matrix P_k is updated, and depend on whether this follows a successful or an unsuccessful iteration.

Algorithm	**Cost (Directional Derivatives)**			
	(`L-HS`/`RS`)`-SD`		(`L-HS`/`RS`)`-N`	
	Successful	**Unsuccessful**	**Successful**	**Unsuccessful**
`L-HS`	$m_p + m_s - 1$	$m_s + r$	$m_p + m_s + (m_p+1)\,n$	$m_s + r + (r+1)\,n$
`RS`	m_p		$m_p + (m_p+1)\,n$	$m_p + m_p n$

Chapter 4

Practical L-HS Implementation

So far we have provided a theoretical framework for subspace line search methods (Chapter 2) and described L-HS, a **l**imited-memory **h**ybrid **s**ubspace class of methods based on it (Chapter 3).

This class can yield a large variety of concrete algorithms depending on the parameter choices made by the user. In light of this—and having implemented L-HS in Python [29]—in this chapter we carry out a range of numerical experiments with two purposes.

The first of these, in Sections 4.2 and 4.4, is to find L-HS parameter choices leading to algorithms that perform well on relevant test problems—namely, a subset of the CUTEst problem suite [30] given in Table B.1. Note that throughout, we give priority to variants' performance based on derivative evaluation expenditure (see Table 3.2), leaving aside NLA cost considerations.

The second purpose, in Sections 4.3 and 4.5, is to compare the most promising L-HS variants to related existing methods on a larger set of CUTEst test problems, given in Table B.2.

We use abbreviations to refer to variants in plot labels. Importantly, we signal the construction of $\tilde{P}_k$ with nomenclature exemplified by L-HS-SD-5.0.15.[1] This refers to an L-HS-SD variant where $\tilde{P}_k$ has 5% of past approximate gradient directions, 0% of past iterate updates, and 15% of random directions. These percentages are with respect to each problem's ambient dimension, and the **ceiling** function is used for rounding where required.[2]

[1] Recall that $\tilde{P}_k$ is the "raw" subspace matrix, assembled from our chosen directions—see (3.4) and (3.5).

[2] Obvious alternatives to the ceiling function—such as rounding to the nearest integer or using the floor function—would work badly. Then we could see, for example, L-HS-SD-10.10.1 misleadingly have zero random subspace directions in problems with ambient dimension less than 50.

The exception is the form `xd`, whereby `x` represents a fixed **number** of directions, independent of ambient dimension—for example, in `L-HS-SD-1d.1d.0`. Other variant attributes are labelled explicitly as appropriate for each numerical experiment. These conventions allow us to convey the important information about each variant assessed concisely and within the plot labels.

4.1 Experimental Setup

Throughout this chapter we make use of **numerical illustrations**—applied to (B.1), a (nonconvex) extended Rosenbrock test function [31] with ambient dimension $n = 100$—as well as **data profiles** [32, sec. 5.1]. For all of these, we use **"equivalent gradient evaluations"**—directional derivative evaluations divided by the problem's ambient dimension—as the horizontal coordinate, and to define solution budgets.

We normalise objective values by defining

$$\bar{f}(x) = \frac{f(x) - f^*}{f(x_0) - f^*}, \tag{4.1}$$

where x_0 is the initial iterate (part of each problem's data) and f^* is either the problem's known global minimiser (in the case of the extended Rosenbrock function) or the lowest loss value achieved in the problem by any solver. Note that, with the purpose of finding low values for f^*, full-space regularised Newton line search methods were run with generous budgets on all relevant problems to begin with.

Given a problem Π, a solver Γ, and an accuracy level $\omega \in (0, 1)$, we say that Π is (Γ, ω, z)-**solved** if solver Γ finds an iterate x_k such that $\bar{f}(x_k) \leq \omega$ within z equivalent gradient evaluations. Then, given a set of test problems $\mathcal{Q}$, a data profile is a plot

$$d_{\Gamma,\omega}(z) = \frac{1}{|\mathcal{Q}|}\Big|\Pi \in \mathcal{Q} : \Pi \text{ is } (\Gamma, \omega, z)\text{-solved}\Big|. \tag{4.2}$$

Since `L-HS` is stochastic, we run each variant 10 times per test problem and consider each run to correspond to a distinct element of $\mathcal{Q}$ in all profiles.[3] Moreover, all data profiles use $\omega = 10^{-2}$.

Throughout the experiments that follow, whenever an `L-HS` variant attribute is left undiscussed, it is assigned a default value as per Table 4.1. These were chosen through a mixture of numerical experience, heuristics, and reference to resources such

[3]This data profile methodology for stochastic algorithms is taken from [18, sec. 5.1], while the use of 10 runs per variant is seen in [32, sec. 5.1].

as [4, 21].

In particular we point out the use of "$c = \infty$" (in practice this simply means that, following any successful iteration, the step size parameter is reset to $\alpha_{\max}$) and $\#_{\text{try}} = 200$, which fail to satisfy the conditions of our main convergence/complexity result, Theorem 2.4.1. Note that, while the theory is stated in terms of iterations, we are interested in problem derivative evaluations, for which these parameter choices are found to perform better than theory-compliant ones. For instance, setting $\#_{\text{try}} = 1$ is inefficient because it forces costly updates to P_k at every single iteration, even if unsuccessful!

Table 4.1: Default `L-HS` attribute values (refer to Algorithm 2.2).

`L-HS` Attribute	**Description**	**Default Value**
τ	Backtracking factor	0.5
β	Armijo condition (2.1) scaling	0.001
c	"Forwardtracking" factor is $\nu = 1/\tau^c$	∞
$\alpha_{\max}$ (`L-HS-SD`)	Maximum step size parameter	100
$\alpha_{\max}$ (`L-HS-N`)	Maximum step size parameter	1
p	Initial step size is $\alpha_0 = \alpha_{\max}\tau^p$	1
$\#_{\text{try}}$	Limit to unsuccessful iterations before updating P_k	200
λ_{reg} (`L-HS-N`)	Projected Hessian regularisation parameter	0.01

From all suitable CUTEst problems (twice-continuously differentiable and unconstrained) we take those with ambient dimension between 100 and 200, giving us the 73 problems shown in Table B.2. From these we randomly choose 20 name-ambient dimension pairs, to obtain those shown in Table B.1. We use the latter, smaller subset for data profiles in seeking good variants (Sections 4.2 and 4.4) and the former, larger one for benchmarks (Sections 4.3 and 4.5).

4.2 Seeking the Best First-Order `L-HS` Variants

As we discussed, in this section we aim to find good choices for `L-HS-SD` variant attributes mainly through assessments in data profiles of the 20 CUTEst problems in Table B.1.

4.2.1 Column-Wise Normalisation or Orthogonalisation of the Subspace Matrix

We begin our investigations by making the decision of whether it is best to obtain the subspace basis matrix P_k by orthogonalisation or just column-wise normalisation of $\tilde{P}_k$. The results in Figures 4.1 and 4.2 clearly suggest the consistent use of **orthogonalised** P_k, which we use for the remainder of our `L-HS-SD` numerics.

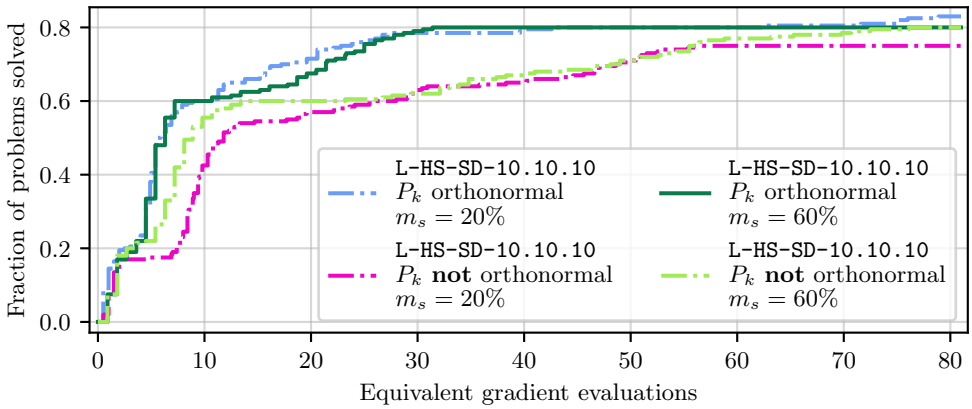

Figure 4.1: Data profile for 20 CUTEst problems, with 10 runs per solver variant. All variants here use Haar-orthonormal sketching matrices.

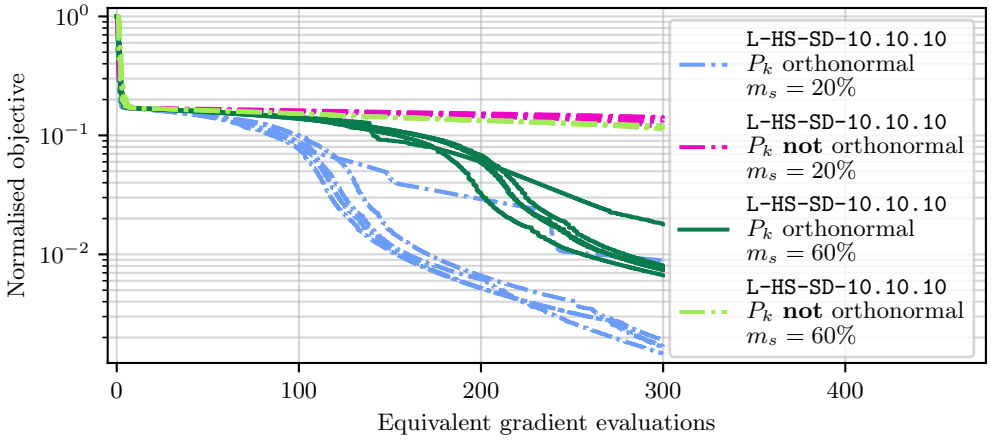

Figure 4.2: Five distinct runs per solver variant applied to (B.1) with ambient dimension $n = 100$. All variants here use Haar-orthonormal sketching matrices.

4.2.2 Scaled Gaussian or Haar-Distributed Orthonormal Sketching Matrices

We now turn to another binary decision that our previous discussions presented—whether we ought to take S_k, the gradient sketching matrix in (3.1), to be a scaled Gaussian matrix (Section 3.2.1.1) or a Haar-orthonormal matrix (Section 3.2.1.2). To this end we use the results in Figures 4.3 and 4.4.

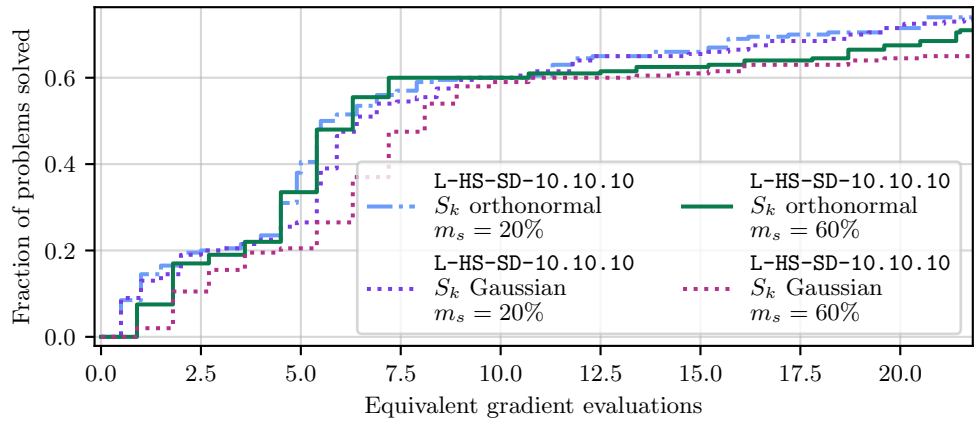

Figure 4.3: Data profile for 20 CUTEst problems, with 10 runs per solver variant.

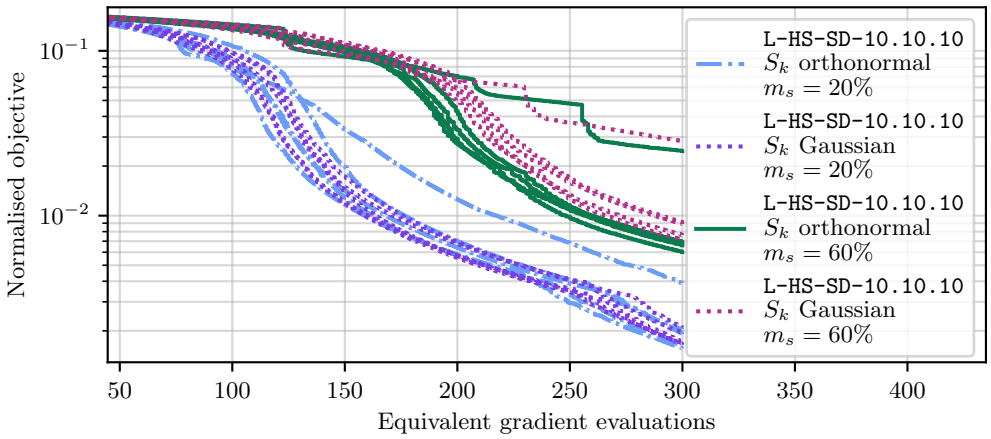

Figure 4.4: Five distinct runs per solver variant applied to (B.1) with ambient dimension $n = 100$.

It appears that the impact of this choice is larger when the sketch size is large, as we may have expected—as per the Marchenko-Pastur rule, the scaled Gaussian is likely to be approximately orthonormal (very well-conditioned) anyway if the sketch size is small [24, Theorem 14.1]. The results favour the use of **Haar-orthonormal**

sketching matrices, which we therefore adopt for the remainder of our L-HS-SD numerical tests.

4.2.3 Gradient Sketch Size

Now we would like to get a sense for a good value of m_s relative to each problem's ambient dimension. For this we use the results in Figures 4.5 to 4.7.[4] Also note that the solid red plots correspond to a full-space deterministic steepest descent (SD) method.[5]

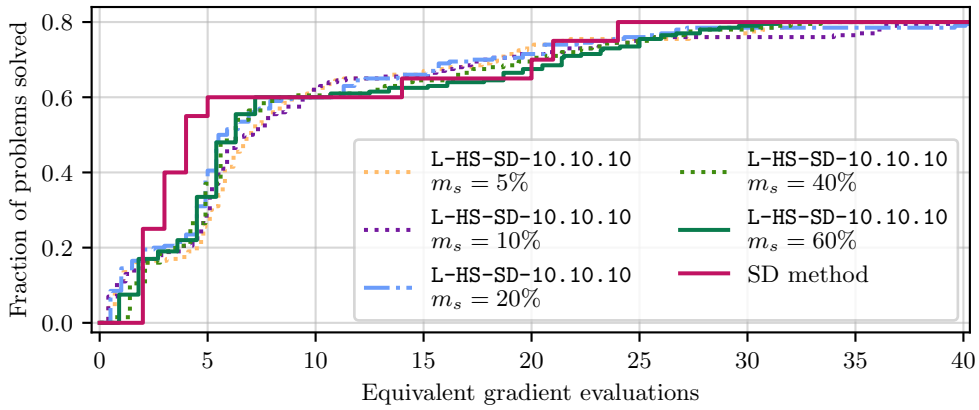

Figure 4.5: Data profile for 20 CUTEst problems, with 10 runs per solver variant.

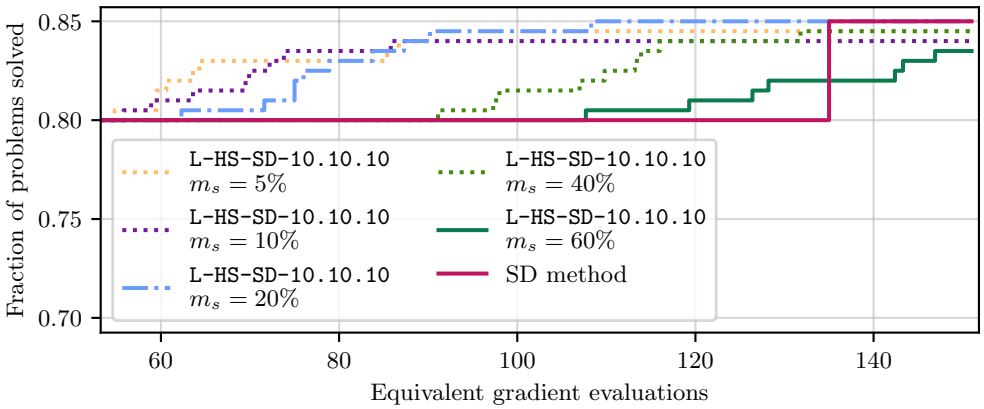

Figure 4.6: Same data profile as in Figure 4.5. Different view.

[4]Note that we "skip" the $m_s = 80\%$ case because such a specification would result in a solver with an expenditure of directional derivatives larger than n per subspace update (recall Table 3.2), at which point a full-space method can be used.

[5]This is the natural successor to the other variants considered here, effectively corresponding to the sketch size $m_s = 100\%$.

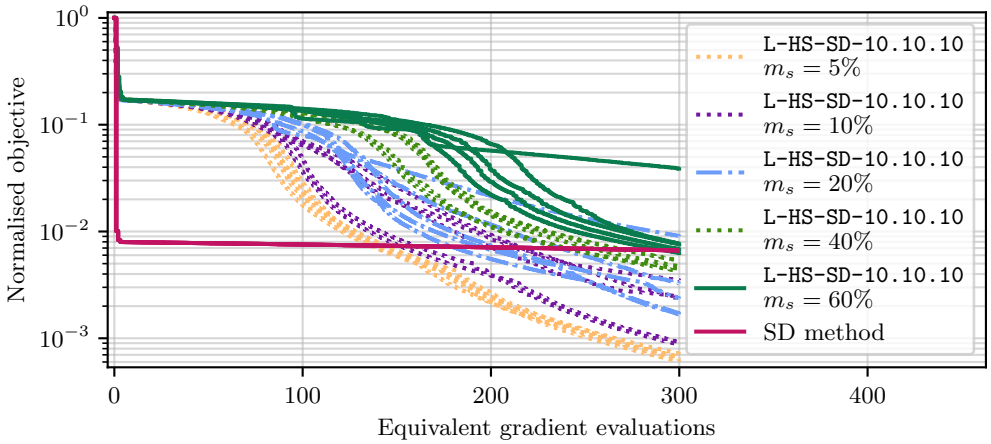

Figure 4.7: Five distinct runs per solver variant applied to (B.1) with ambient dimension $n = 100$.

At first, we may have—understandably—associated subspace methods with the principal benefit of making progress at small evaluation budgets—even under a single equivalent gradient evaluation—which is indeed a benefit demonstrated in Figure 4.5. It is striking, then, that `L-HS-SD` beats its full-space counterpart **at high budgets** in Figure 4.6; this hints at the potential for its usefulness against harder problems requiring higher evaluation budgets to solve.

In Figures C.1 and C.2 we provide similar data profile views to Figures 4.5 and 4.6 but using `L-HS-SD-20.0.10` instead; the results are broadly of a similar nature.

The top contenders in sketch size are $m_s \in \{5\%, 10\%, 20\%\}$. Taking into account performance at low as well as high budgets—using `L-HS-SD-10.10.10` as well as `L-HS-SD-20.0.10`—we choose to proceed with `L-HS-SD` using $m_s = 20\%$.

4.2.4 Including Momentum Directions in the Subspace Construction

What remains to assess now is the make-up of the subspace construction; that is, the three integers `xx`, `yy`, and `zz` in our `L-HS-SD-xx.yy.zz` notation.

We begin by trying to ascertain whether it is worthwhile to include iterate update directions.[6] A wealth of relevant results is shown in Figures 4.8 to 4.11 and C.4 to C.18.

In many of the data profiles presented, variants have very much comparable performance. Insofar as the data profiles presented differ—and keeping in mind that

[6]Recall that this is what distinguishes the schemes (3.4) and (3.5).

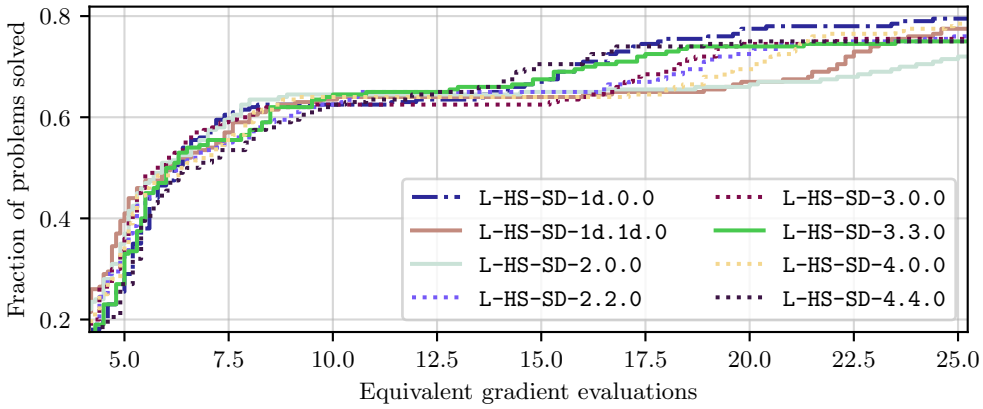

Figure 4.8: Data profile for 20 CUTEst problems, with 10 runs per solver variant.

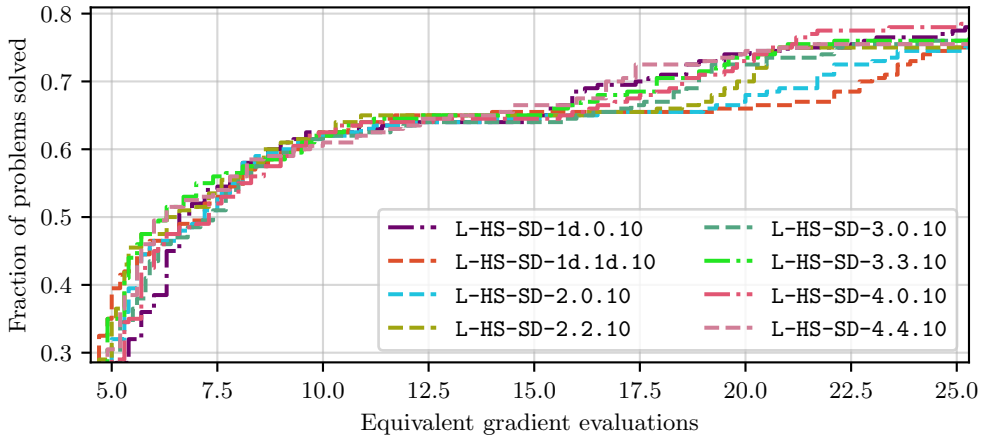

Figure 4.9: Data profile for 20 CUTEst problems, with 10 runs per solver variant.

these are random variables—the takeaways seem to be as follows. The methods `L-HS-SD-1d.0.zz` and `L-HS-SD-1d.1d.zz` are exceptional in that the former performs better than the latter across Figures 4.8, 4.9, and C.4 to C.10. For other variants in these same figures—where the subspace dimension m_p is small—the picture is muddier, although it tilts slightly in favour of including momentum directions.

In variants with larger subspace dimension (see Figures 4.10, 4.11, and C.11 to C.14, where m_p ranges from 5 to 30) we see no such exceptions.[7] For the most part, the effect of including momentum directions is either seemingly negligible or—in

[7]Note that some care should be taken with variants of high subspace dimension at low evaluation budgets. For example, `L-HS-SD-15.15.0` with $m_s = 20\%$ expends about half of an equivalent gradient evaluation per successful iteration. Thus, in a problem with ambient dimension 100, only by around 8 equivalent gradient evaluations has the solver actually accumulated $15\% = 15$ past approximate gradients. Up to that point, the "gaps" in the subspace construction are filled in with **random** directions—as we had explained in Section 3.1.2.

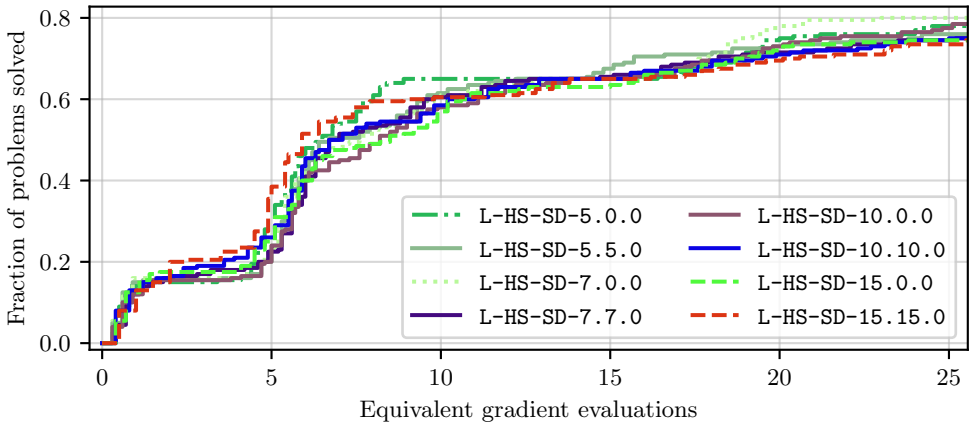

Figure 4.10: Data profile for 20 CUTEst problems, with 10 runs per solver variant.

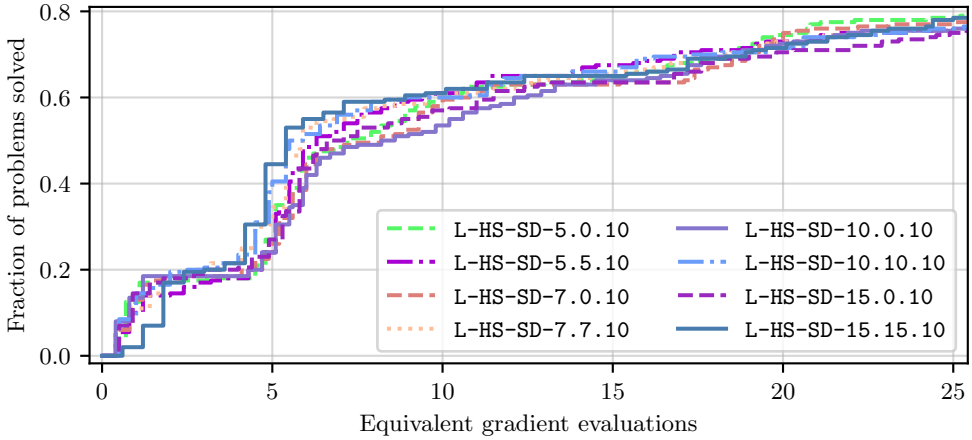

Figure 4.11: Data profile for 20 CUTEst problems, with 10 runs per solver variant.

`L-HS-SD-7.0.zz`, `L-HS-SD-10.0.zz`, and `L-HS-SD-15.0.zz`—slightly positive (especially in Figure 4.11, at low evaluation budgets and where random subspace directions are included). However, there seems to be a negative impact in including momentum directions at high evaluation budgets and where random subspace directions are **not** included—see Figure C.11, variants `L-HS-SD-10.yy.0` and `L-HS-SD-15.yy.0`.

For a different perspective, in Figures C.15 to C.18 our comparisons use fixed, moderately large subspace dimensions, and the inclusion of momentum directions there has a positive effect.

Based on these findings, we choose to **include momentum directions** in all subsequent `L-HS-SD` variants considered **except** for `L-HS-SD-1d.0.zz`.

4.2.5 Including Random Directions in the Subspace Construction

We now attempt to isolate the effects of including varying quantities of random directions in the subspace construction. Relevant results are presented in Figures 4.12 to 4.14 and C.19 to C.27.

We can say that, at higher budgets, `L-HS-SD-5.5.zz` and `L-HS-SD-10.10.zz` suffer from the inclusion of a large amount of random directions and seem to benefit the most from the use of `zz` $= 10$ (Figures C.21 and C.23). At lower budgets the inclusion of more random directions appears to be more beneficial (Figures 4.13 and 4.14). The case of `L-HS-SD-1d.0.zz` is, once more, a bit of an exception; using `zz` $= 2$ seems to lead to some of the best results in low as well as high budgets (Figures 4.12 and C.19).

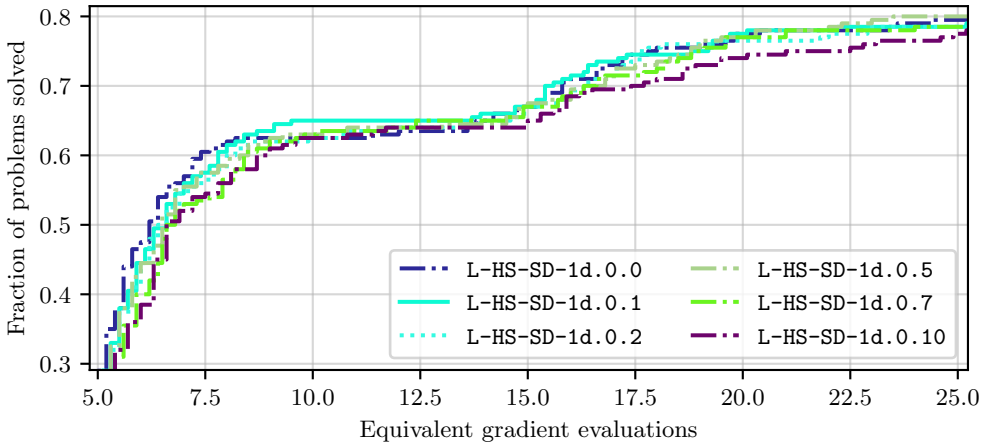

Figure 4.12: Data profile for 20 CUTEst problems, with 10 runs per solver variant.

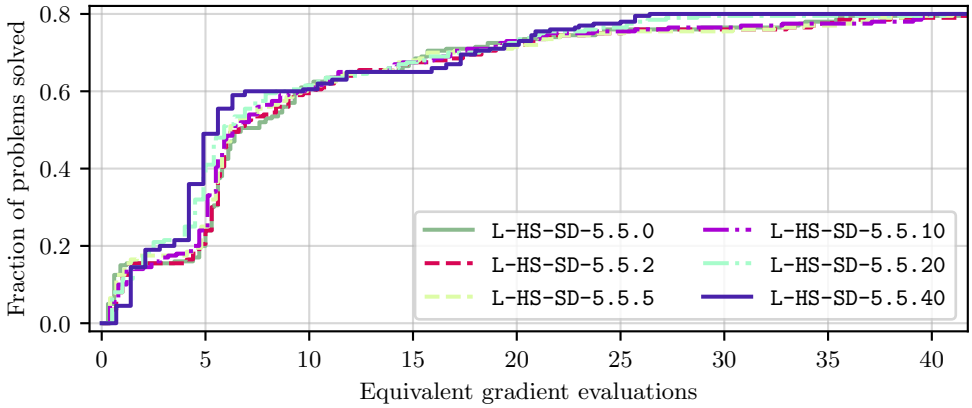

Figure 4.13: Data profile for 20 CUTEst problems, with 10 runs per solver variant.

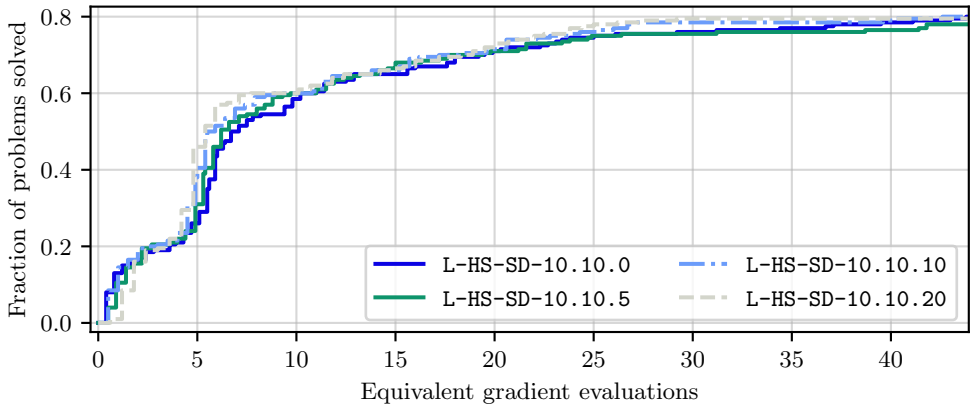

Figure 4.14: Data profile for 20 CUTEst problems, with 10 runs per solver variant.

4.3 First-Order Method Benchmarks

We now seek to assess `L-HS-SD` variants on the set of 73 CUTEst problems listed in Table B.2, which we do with three views of the same data profile in Figures 4.15 to 4.17.

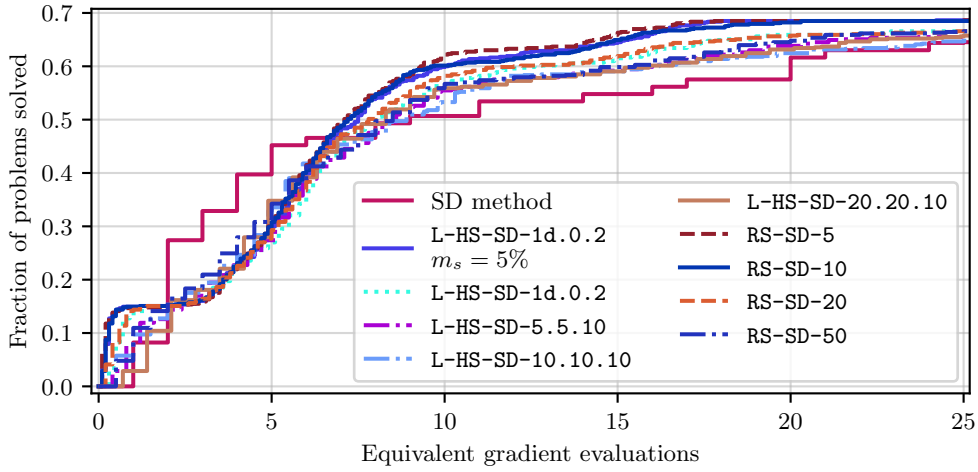

Figure 4.15: First-order methods benchmark on 73 CUTEst problems, with 10 runs per solver variant. All `L-HS-SD` variants use $m_s = 20\%$ apart from the one labelled with $m_s = 5\%$.

Aside from some of the most promising `L-HS-SD` variants we have found in this chapter, we also assess methods where the subspace matrix P_k is fully random. We denote these by `RS-SD-zz`, where `zz` denotes the subspace dimension as a percentage of the ambient dimension. In all `RS-SD` methods considered here, we take P_k as a Haar-orthonormal random matrix.

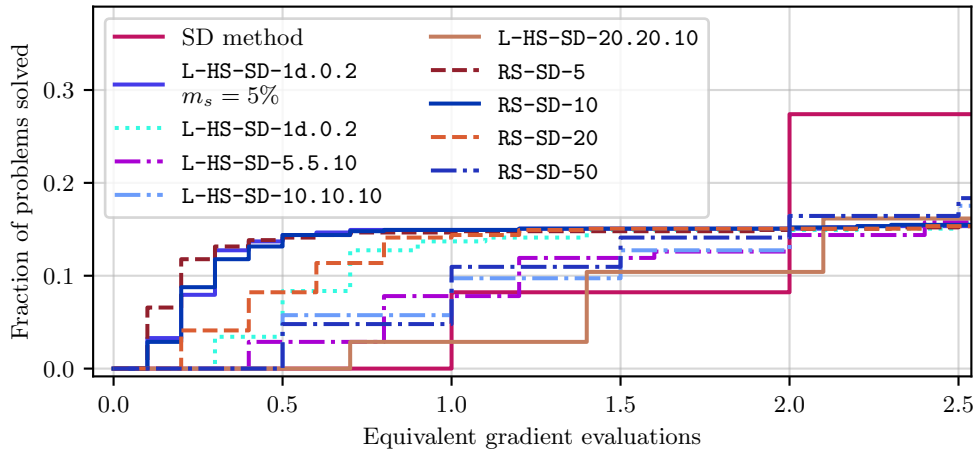

Figure 4.16: Same benchmark as in Figure 4.15. Different view.

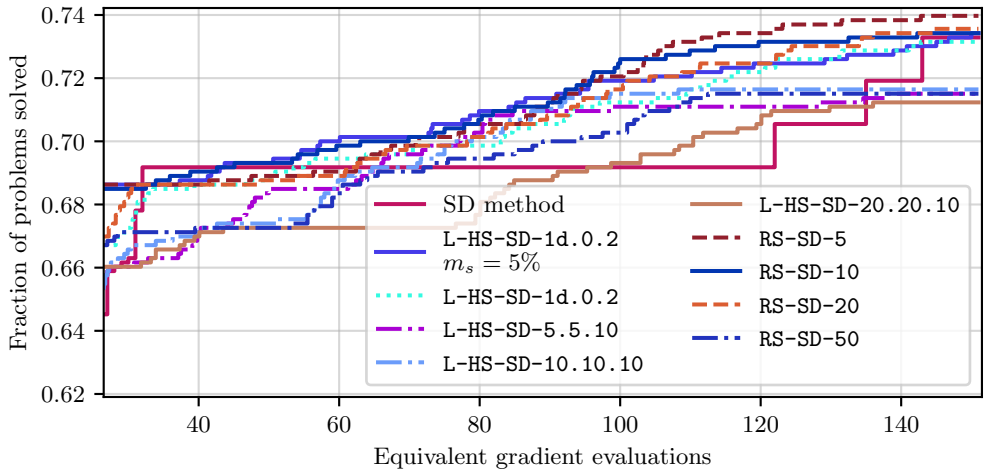

Figure 4.17: Same benchmark as in Figure 4.15. Different view.

We also include `L-HS-SD-1d.0.2` using $m_s = 5\%$, mainly due to the strong performance achieved with a very small subspace dimension by `RS-SD-5`. Our having missed this well-performing `L-HS-SD` variant in our previous experiments highlights the heuristic nature of our variant parameter search in Section 4.2.

Overall, it seems that subspace methods with smaller subspace dimensions perform better, which is undoubtedly linked to our budgeting by directional derivative evaluations—variants with very low subspace dimension can make progress with very little costs per iteration on average. The `RS-SD` and `L-HS-SD` methods have roughly similar performance, which beats the classical steepest descent (SD) method on our metrics at very low budgets—where the SD method expends too much problem information to achieve sizeable objective decrease—as well as high budgets—where the

SD method is liable to get "stuck" in difficult problems, as we had seen in Figure 4.7.

4.4 Seeking the Best Second-Order L-HS Variants

Owing to the high costs of running many sizeable numerical tests with L-HS-N—as we did for L-HS-SD—we present just a few of the most important numerical studies in a shortened form in this section.

We address principally the following issues, in the same order as we did for L-HS-SD in Section 4.2: whether to use orthonormal or simply column-normalised subspace basis matrices P_k; whether to use scaled Gaussian or Haar-distributed orthonormal sketching matrices S_k; what trade-offs occur in performance as we vary the sketch size m_s.

4.4.1 Column-Wise Normalisation or Orthogonalisation of the Subspace Matrix

Regarding the issue of column-wise normalisation versus orthogonalisation of the subspace basis matrix, we refer to Figures 4.18 and C.28. In contrast with our findings for L-HS-SD, whether P_k is orthonormal or not seems to make little difference to the observed performance. If nothing else, in the interest of simplicity we choose to stick with the **orthonormal** option.

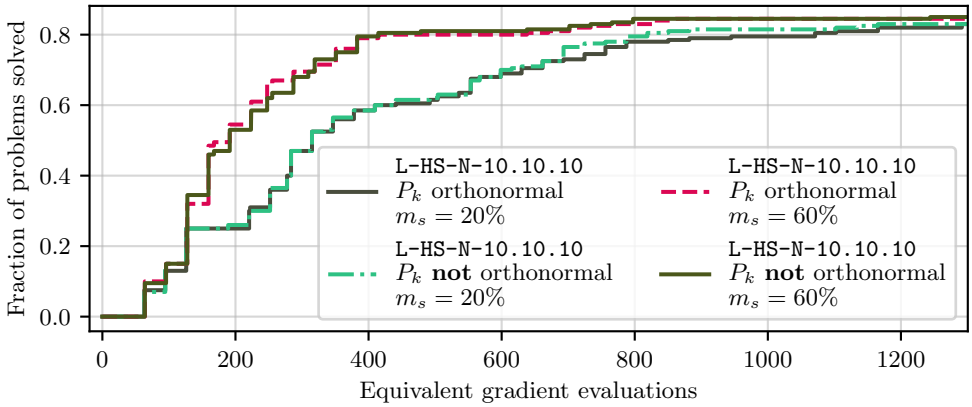

Figure 4.18: Data profile for 20 CUTEst problems, with 10 runs per solver variant. All variants here use Haar-orthonormal sketching matrices. To clarify the similar colours of two of the plots: the curves are grouped by sketch size.

4.4.2 Scaled Gaussian or Haar-Distributed Orthonormal Sketching Matrices

We now address the matter of the sketching matrix ensemble using the results in Figures 4.19 and C.29. Here the suggested choice is clearer, and once more in favour of using **Haar-orthonormal** sketching matrices, which we therefore use for the remainder of this chapter.

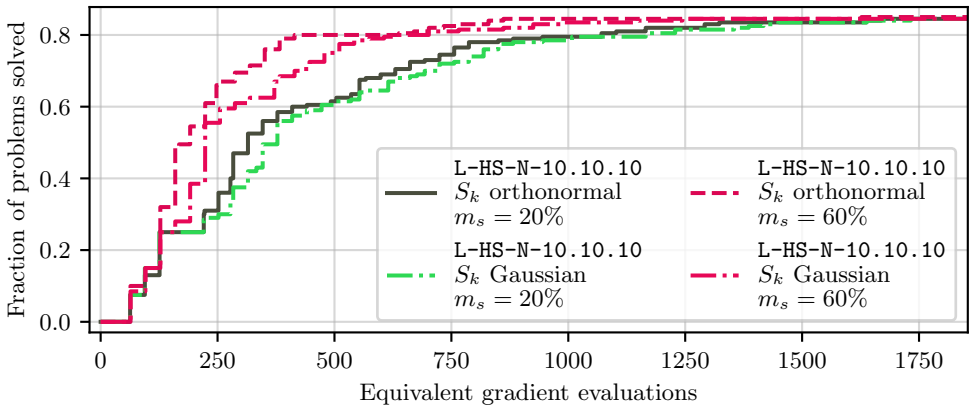

Figure 4.19: Data profile for 20 CUTEst problems, with 10 runs per solver variant.

4.4.3 Gradient Sketch Size

We finally address the issue of the sketch size m_s using the results in Figures 4.20 and C.30.

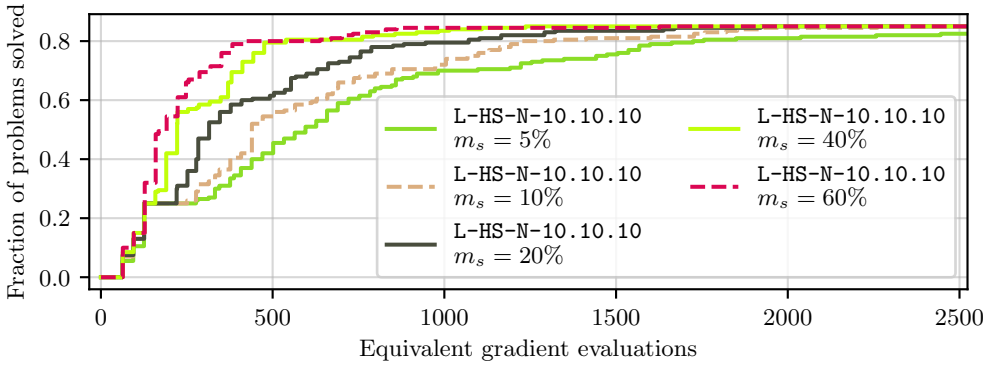

Figure 4.20: Data profile for 20 CUTEst problems, with 10 runs per solver variant.

Clearly it pays off, by our metrics, to have large m_s in `L-HS-N`—recalling Table 3.2, we may deduce that this is because the derivative evaluation costs in this algorithm

originate predominantly from the second-order information used in the form of Hessian actions, whereas the impact of m_s is modest by comparison.

We therefore take the question of which sketch size to use in `L-HS-N` to come down—to a larger extent than in `L-HS-SD`—to problem specifics determining the precise computational and memory costs of computing first- versus second-order objective derivative information. Recall that we proposed the "equivalent gradient evaluations" metric in large part to unify our measurements of problem information used by each algorithm considered, with little regard for problem/algorithm structure—partial separability or Hessian sparsity, for example—which could be exploited in the use of techniques such as automatic differentiation [20, 33]. In practice, taking these matters into account could tilt the scales in favour of using more first-order information (increasing m_s) or using more second-order information (increasing m_p).

In light of these observations, we retain varying sketch size percentages in the benchmark exercise of Section 4.5.

4.5 Second-Order Method Benchmarks

Much as in Section 4.3, we now assess a mixture of second-order methods on the full set of 73 CUTEst problems in Table B.2. The results are shown in Figure 4.21. In particular we note that the variants of `L-HS-N` with $m_s = 100\%$ and with zero random columns in the subspace construction are deterministic and fall within the `L-CommDir` method of [13].

As we had seen and discussed in Section 4.4.3, one of the main determinants of performance seems to be the sketch size m_s, with higher values all the way up to $m_s = 100\%$ invariably leading to better performance. Unlike in our `L-HS-SD` benchmarks, none of the subspace methods can be reasonably said to beat the full-space counterpart, which is a higher price to pay in performance for the lack of a need to use full-dimensional derivative information.

These results from `L-HS-N` lead us to a short discussion of quasi-Newton variants of `L-HS`.

4.6 A Note on Quasi-Newton `L-HS` Variants

Quasi-Newton methods exploit first-order derivative information gathered during an algorithm run to construct a matrix B_k approximating the true Hessian of the ob-

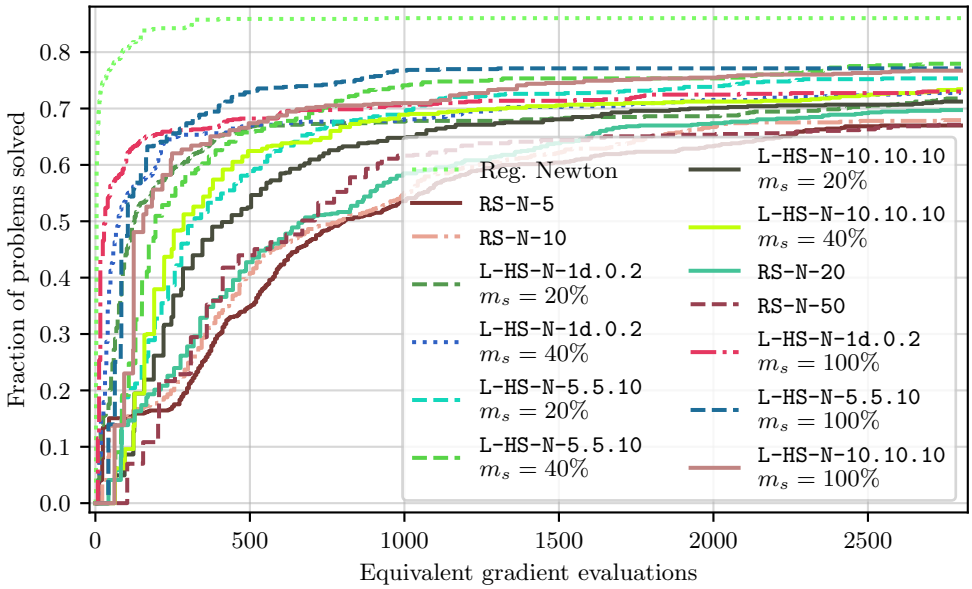

Figure 4.21: Benchmark on 73 CUTEst problems, with 10 runs per solver variant.

jective [4, 34]. The most well-known such methods—such as `SR1` and `BFGS`—achieve this through incremental low-rank updates to B_k which solve a variational problem (often minimising some norm of the update) and ensure that the *secant condition*

$$B_{k+1}s_k = \nabla f_{k+1} - \nabla f_k =: y_k \tag{4.3}$$

is satisfied. Each set $\{s_k, y_k\}$ is known as a *secant pair*.

These methods are appealing in our setting especially because they suggest a way to improve on the performance of `L-HS-SD`[8] while keeping derivative evaluation costs just as low. Unfortunately, our preliminary results suggest that our naive adaptation—which we call `L-HS-QN`—fails to unlock this potential.

We use the `L-SR1` method [4, sec. 9.2], and substitute the B_k matrix obtained thusly for the true Hessian in our description of `L-HS-N` in Chapter 3. The `SR1` method updates the running B_k matrix by **s**ymmetric **r**ank-**one** matrices. Its **l**imited-memory version `L-SR1` accumulates only a limited set of recent secant pairs—in our illustrations we maintain 10, and use the identity matrix as the running "initial" Hessian estimate. Our naive adaptation consists of replacing, in all of this, ∇f_k by g_k as computed in `L-HS`—hence the lack of a need for any additional function derivative

[8]Under certain conditions, classical quasi-Newton methods have superlinear local convergence, an improvement on the linear convergence of steepest-descent methods [4, Theorems 3.4 and 3.6].

evaluations compared to `L-HS-SD`.[9]

Illustrations on the Rosenbrock problem (B.1) are shown in Figure 4.22. Clearly there is a step change in performance when $m_s = 100\%$—that is, when $g_k = \nabla f_k$, so that the B_k updates are obtained from the `L-SR1` procedure exactly.[10]

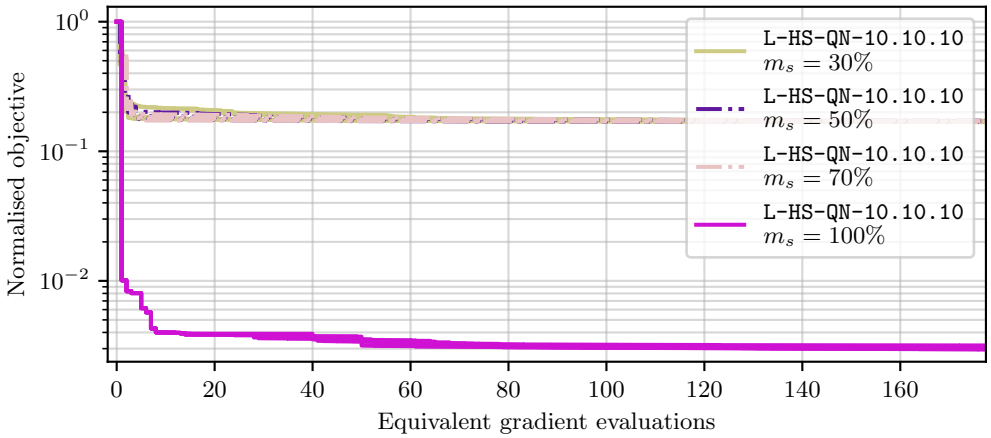

Figure 4.22: Five distinct runs per solver variant applied to (B.1) with ambient dimension $n = 100$. All variants use Haar-orthonormal sketching matrices and orthonormal subspace basis matrices.

We conjecture that it is possible to improve on this by adapting typical quasi-Newton updates to the `L-HS` setting more carefully than we have done. For instance, we can pre-multiply (4.3) by P_{k+1} and substitute $s_k = P_k \hat{s}_k$ to obtain

$$\left(P_{k+1}^\top B_{k+1} P_k\right) \hat{s}_k = P_{k+1}^\top \left(\nabla f_{k+1} - \nabla f_k\right), \tag{4.4}$$

which is something like a reduced-dimension secant condition. Note how $P_{k+1}^\top B_{k+1} P_k$ is rather similar to $P_{k+1}^\top B_{k+1} P_{k+1}$ (a "reduced" approximation to the Hessian similar to what we described for general second-order `L-HS` methods) especially if, as is the case in many `L-HS` variants, P_{k+1} and P_k have columns in common.

This note highlights the interest of thinking `L-HS-QN` through more carefully, but we leave that work to future research endeavours.

[9]For brevity, we are avoiding detailed description of a few further implementation details—for example, the most recent secant pair in `L-HS-QN` requires overwriting when P_k is updated following an unsuccessful iteration; moreover, we use the safeguard proposed in [4, sec. 8.2] to skip ill-defined `SR1` updates.

[10]Note that, as far as derivative evaluation costs are concerned, at this point we may as well have use a usual full-space quasi-Newton method—our objective here is investigative.

Chapter 5

Conclusion

5.1 Summary

In this dissertation we were concerned with nonconvex unconstrained optimisation over decision variable spaces with large ambient dimension. This problem arises in applications such as training large machine learning models [1] or data assimilation in weather forecasting [2], to name a few.

Within this setting, the allure of subspace methods is clear: since computing full-dimensional derivative information may be extremely computationally intensive, their virtue lies in enabling good progress towards a solution while using only reduced-dimension information—for instance, subsets of partial derivatives in block-coordinate methods.

However, existing subspace methods occupy distant ends of a spectrum. They either require full-dimensional gradient information to construct subspaces—as in the case of most deterministic methods [3, 10–13]—or fail to exploit any problem information at all in constructing subspaces—as in the case of random methods [7, 8, 14–19] or deterministic cyclic block-coordinate methods [6].

In this dissertation we aimed to fill this gap by proposing `L-HS`, a class of **l**imited-memory **h**ybrid **s**ubspace methods. The term "hybrid" alludes to the fact that these methods sit in between the two existing classes, by using (randomly) projected derivative information in constructing subspaces.

In Chapter 2, by adapting the theoretical framework of [18], we stated conditions required of a generic hybrid subspace line search algorithm to secure probabilistic global worst-case convergence and complexity bounds.

Then, in Chapter 3, we particularised this for `L-HS` subclasses using first- and

second-order local models—L-HS-SD and L-HS-N respectively—and showed that they inherited those theoretical guarantees. Throughout the development of these subclasses, we kept in mind the associated numerical linear algebra and problem evaluation costs, aiming to keep them low and, at the end, providing thorough but summarised accounts.

Having implemented these algorithms in Python [29], this enabled us to carry out extensive numerical tests on problems from the CUTEst suite [30] in Chapter 4. Using these, we could assess the most worthwhile L-HS parameter choices for performance against our evaluation cost metrics; show regimes in which L-HS-SD beats its full-space counterpart; and demonstrate the trade-offs in the use of first- as well as second-order information by L-HS-N.

5.2 Subsequent Research

Our numerical results, along with the intuition resulting from our theoretically-minded proofs and discussions, suggest pressing further research questions.

As we discussed in Section 4.6, the development of quasi-Newton Hessian approximation techniques using projected gradient information offers a seemingly feasible as well as worthwhile investment of future research efforts. This could provide improved performance without having to expend any more derivative information per iteration than L-HS-SD, by exploiting projected information a step further than we did here.

Also motivated in part by large problems in machine learning, it would be interesting to merge hybrid subspace ideas with sampling in the data space—that is, when the objective is a sum of functions over a large data set. Indeed, related work has already been done for random subspaces [35].

A smaller idea worth pursuing would be to combine the directional derivatives computed in $S_k^\top \nabla f_k$ with those computed in $P_k^\top \nabla f_k$. While here we limit ourselves to storing the former, we could combine the directional derivatives of the latter to make for higher-dimensional past projected gradient information without any additional evaluation costs.

Yet another idea would be to carry out multiple (successful) inner iterations before renewing the hybrid subspace—using our terminology, this is of interest mainly when $m_p \ll m_s$.

In all of these cases, using more computing power to enable extensive tests on very large problems—and considering algorithm parallelism—would be valuable, since smaller problems are likely not to reflect structure seen in higher-dimensional spaces.

Appendix A

L-HS Additional Discussions

A.1 Computing Haar-Orthonormal Matrix-Vector Products

Here we consider a trick which allows us to implicitly generate Haar-distributed orthonormal matrices when all we need is their application to vectors. This does not include situations where we need to interpret these products as directional derivatives, which disqualifies this trick from being used in generating Haar-orthonormal gradient sketching matrices in L-HS. But it does allow for cheaper generation of random Haar-orthonormal subspace matrices in RS (random subspace) methods.

Consider the process of computing a QR factorisation. In deterministic scenarios (and as is done for computing Haar-orthonormal matrices in the method of [27]), the computation of each reflector relies on having multiplied the working matrix by the previous reflector, so as to reveal the current "target" vector. These multiplications are costly, but can be avoided here: the key insight is that the columns of the original matrix—Gaussian vectors—are independently drawn from an orthogonally invariant distribution. That is, the distribution of each target vector in the working matrix is unaffected by the application (or not) of each reflector, so we may simply generate each Gaussian column of the working matrix as we go along, avoiding the costly matrix multiplications incurred by the method of [27]. This is proposed, for square matrices, in [36, Theorem 3.3].

For a matrix in $\mathbb{R}^{n\times m_s}$, the procedure of [36] implies only the computation of m_s Householder vectors (e.g. by [28, Alg. 5.1.1]), which can be done at a cost of $\sum_{k=0}^{m_s-1} 3\,(n-k) \sim 3nm_s - 3m_s^2/2$ flops. The subsequent left-application of $O\,(m_s)$ Householder reflectors to a vector in $\mathbb{R}^n$ could then be ordinarily computed in $O\,(nm_s)$

flops, still falling within quadratic complexity in the dimensions of S_k.

This trick **can** be used in `RS` methods—we account for this possibility in the cost summary of Table 3.1. However, in the context of computing g_k, `L-HS`'s motivation in terms of derivative costs does **not** allow for these savings in reflector applications. Owing to our interpretation of $\hat{g}_k = S_k^\top \nabla f_k$ as a set of m_s directional derivatives, we are instead forced to form S_k explicitly, computing g_k by

$$g_k = H_1 \dots H_{m_s} \left[I_{n \times m_s} \left(\overbrace{(I_{m_s \times n} H_{m_s} \dots H_1)}^{\in \mathbb{R}^{m_s \times n}} \nabla f_k \right) \right]. \tag{A.1}$$

Forming S_k incurs $O(nm_s^2 + m_s^3)$ flops [28, sec. 5.1.6], while the left-application of m_s reflectors incurs a smaller cost of $O(nm_s)$ flops [28, sec. 5.1.4].

A.2 Derivative Costs When P_k is Fully Randomised

Consider now the case where P_k is fully randomised. No use is made of approximate gradients, so the only expense incurred when P_k is updated in **first-order** variants—following either a successful iteration or an unsuccessful iteration—is to compute $P_k^\top \nabla f_k$, which plainly costs m_p directional derivatives.

In **second-order** variants, updates to P_k following successful and unsuccessful iterations are only distinguished—in our finite difference framing—by the need (or lack thereof) to compute the "central" gradient. This leads to additional costs of $(m_p + 1)\, n$ and $m_p n$ following successful and unsuccessful updates respectively.

Appendix B

Problems Used in Numerical Studies and Benchmarks

Multiple distinct definitions of an "extended Rosenbrock test function" can be found in the literature. We choose to use

$$f(x) = \sum_{i=1}^{n/2} 100\big(x_{2i} - x_{2i-1}^2\big)^2 + (1 - x_{2i-1})^2 \tag{B.1}$$

for some even ambient dimension n. This is proposed, for example, in [31].

Table B.1: CUTEst problems [30] and corresponding ambient dimensions used in small data profiles for numerical studies.

CUTEst Name	**Dimension**	**CUTEst Name**	**Dimension**
ARGLINA	200	LUKSAN22LS	100
ARGLINC	100	MANCINO	100
ARWHEAD	100	NCB20B	180
BOXPOWER	100	OSCIPATH	100
CURLY10	100	SBRYBND	100
DQDRTIC	100	SCHMVETT	100
ENGVAL1	100	SCURLY10	100
FLETCBV3	100	SSBRYBND	100
LIARWHD	100	TRIDIA	100
LUKSAN15LS	100	VARDIM	100

Table B.2: CUTEst problems [30] and corresponding ambient dimensions used in benchmark exercises.

CUTEst Name	**Dimension**	**CUTEst Name**	**Dimension**
ARGLINA	100, 200	LUKSAN16LS	100
ARGLINB	100, 200	LUKSAN17LS	100
ARGLINC	100, 200	LUKSAN21LS	100
ARGTRIGLS	100, 200	LUKSAN22LS	100
ARWHEAD	100	MANCINO	100
BDQRTIC	100	MOREBV	100
BOXPOWER	100	NCB20B	100, 180
BOX	100	NONCVXU2	100
BROWNAL	100, 200	NONCVXUN	100
BROYDN3DLS	100	NONDIA	100
BROYDNBDLS	100	NONDQUAR	100
BRYBND	100	OSCIGRAD	100
COSINE	100	OSCIPATH	100
CURLY10	100	PENALTY1	100
CURLY20	100	POWELLSG	100
CURLY30	100	POWER	100
DIXON3DQ	100	QUARTC	100
DQDRTIC	100	SBRYBND	100
DQRTIC	100	SCHMVETT	100
ENGVAL1	100	SCOSINE	100
EXTROSNB	100	SCURLY10	100
FLETBV3M	100	SCURLY20	100
FLETCBV2	100	SCURLY30	100
FLETCBV3	100	SENSORS	100
FLETCHBV	100	SINQUAD	100
FLETCHCR	100	SPARSINE	100
GENHUMPS	100	SPARSQUR	100
GENROSE	100	SSBRYBND	100
INDEFM	100	SSCOSINE	100
INDEF	100	TOINTGSS	100
LIARWHD	100	TQUARTIC	100
LUKSAN11LS	100	TRIDIA	100
LUKSAN15LS	100	VARDIM	100, 200

Appendix C

Further Numerical Study Figures

C.1 Appendix to Section 4.2.3

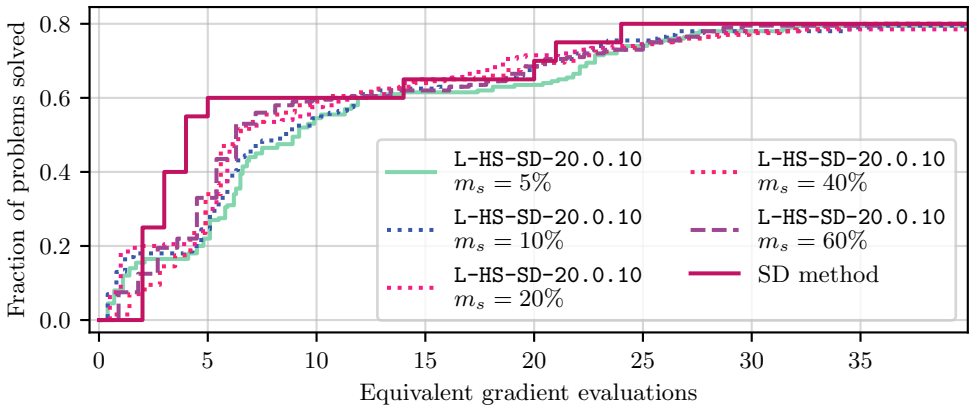

Figure C.1: Data profile for 20 CUTEst problems, with 10 runs per solver variant.

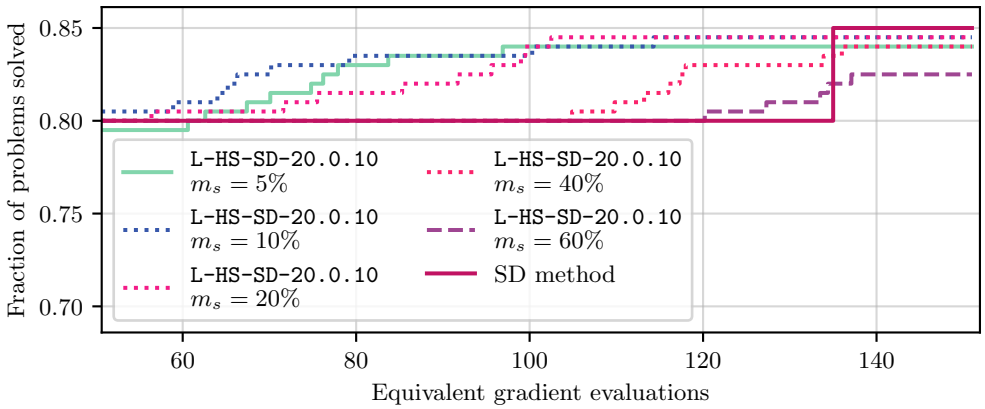

Figure C.2: Data profile for 20 CUTEst problems, with 10 runs per solver variant. Different view.

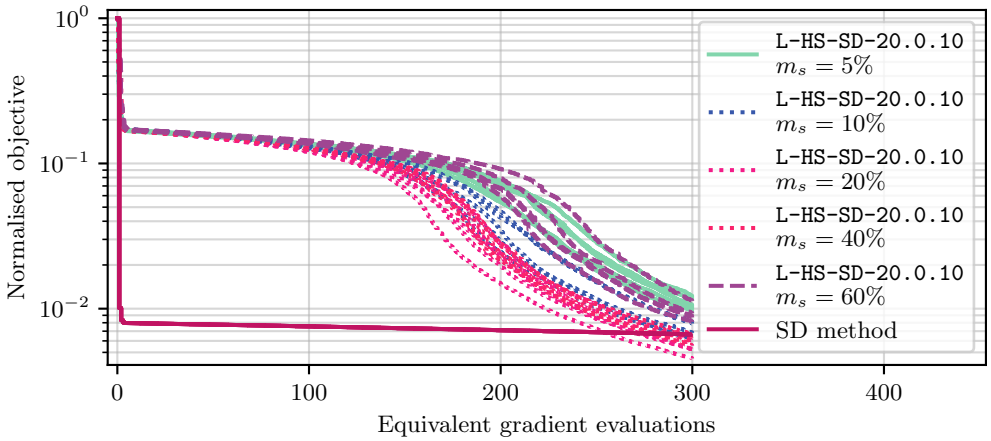

Figure C.3: Five distinct runs per solver variant applied to (B.1) with ambient dimension $n = 100$.

C.2 Appendix to Section 4.2.4

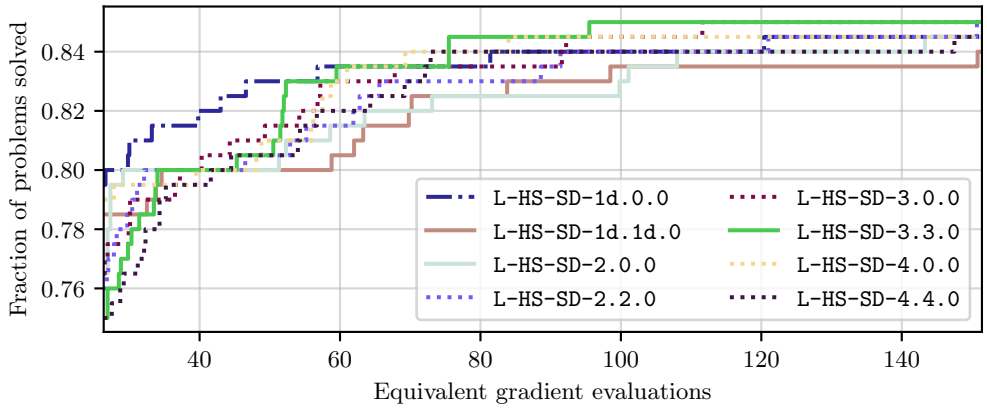

Figure C.4: Data profile for 20 CUTEst problems, with 10 runs per solver variant. Different view.

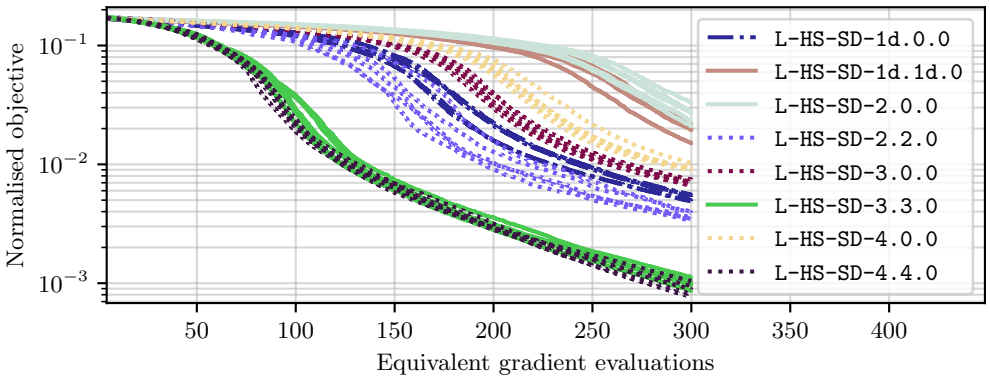

Figure C.5: Five distinct runs per solver variant applied to (B.1) with ambient dimension $n = 100$.

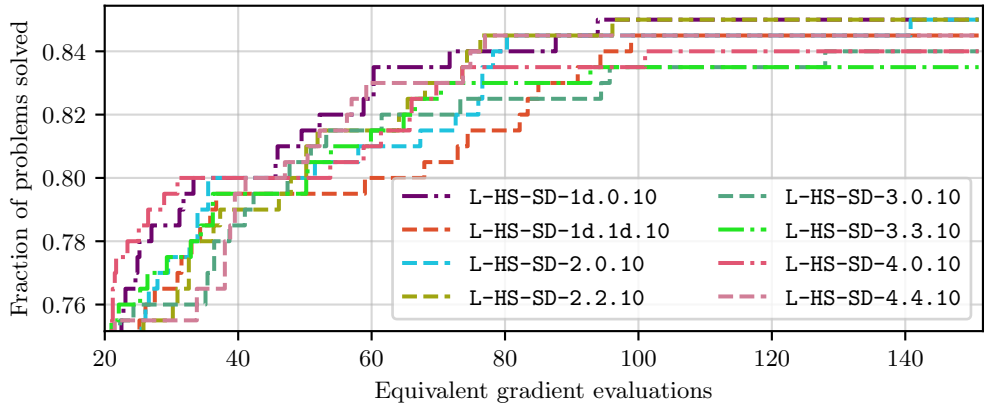

Figure C.6: Data profile for 20 CUTEst problems, with 10 runs per solver variant. Different view.

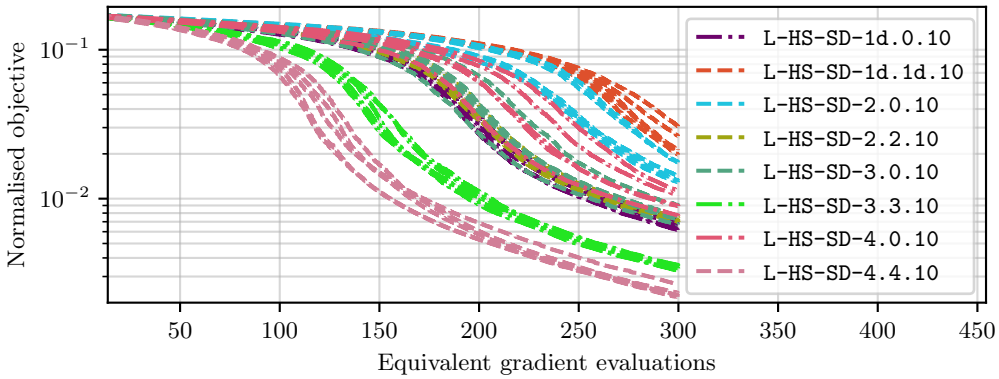

Figure C.7: Five distinct runs per solver variant applied to (B.1) with ambient dimension $n = 100$.

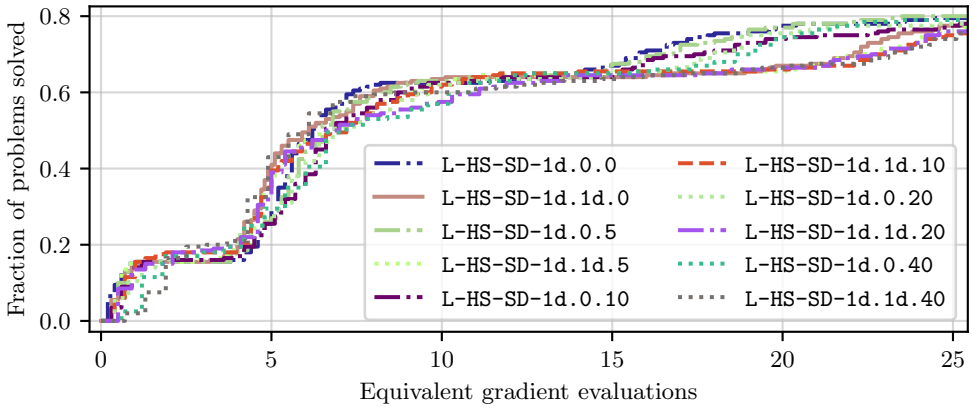

Figure C.8: Data profile for 20 CUTEst problems, with 10 runs per solver variant. In the "gulf" around 22 equivalent gradient evaluations, all the best-performing solvers are those with 0 past update directions.

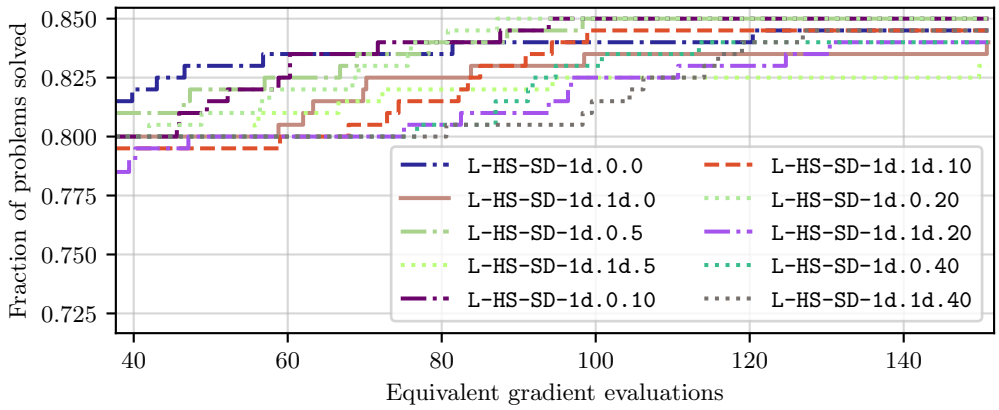

Figure C.9: Data profile for 20 CUTEst problems, with 10 runs per solver variant. Different view.

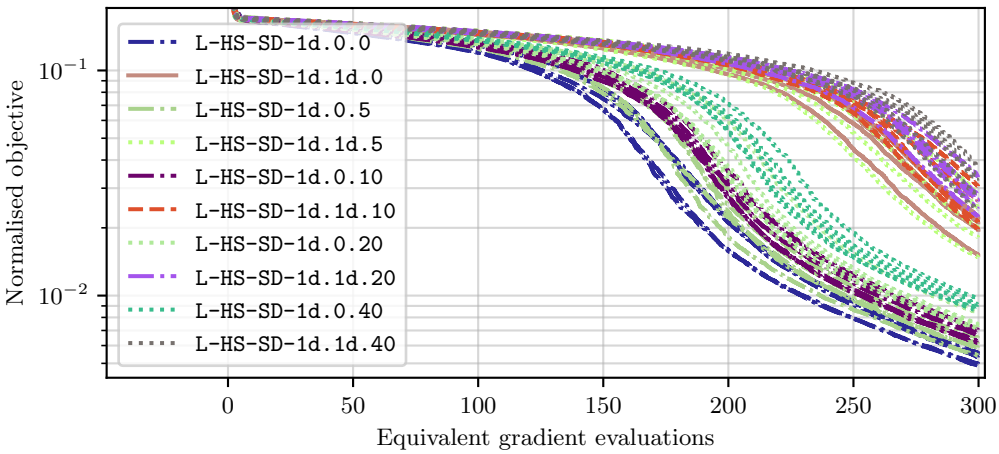

Figure C.10: Five distinct runs per solver variant applied to (B.1) with ambient dimension $n = 100$. As in Figures C.8 and C.9, the best-performing variants are those that do **not** use a past update direction in the subspace construction.

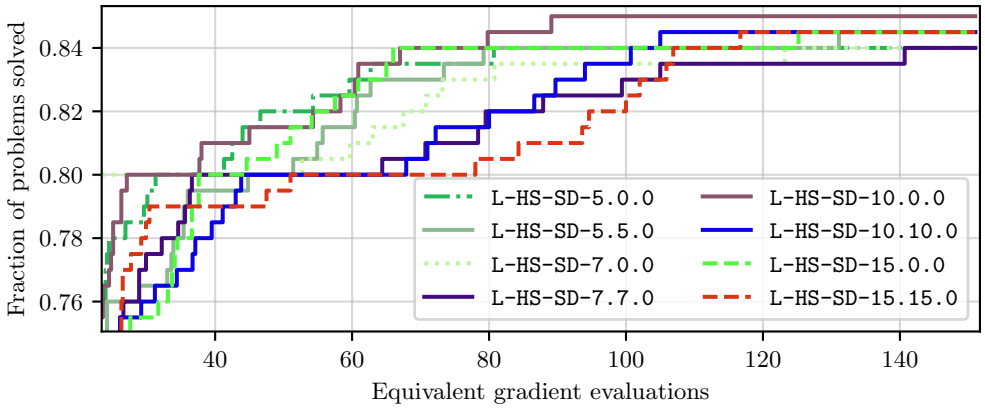

Figure C.11: Data profile for 20 CUTEst problems, with 10 runs per solver variant. Different view.

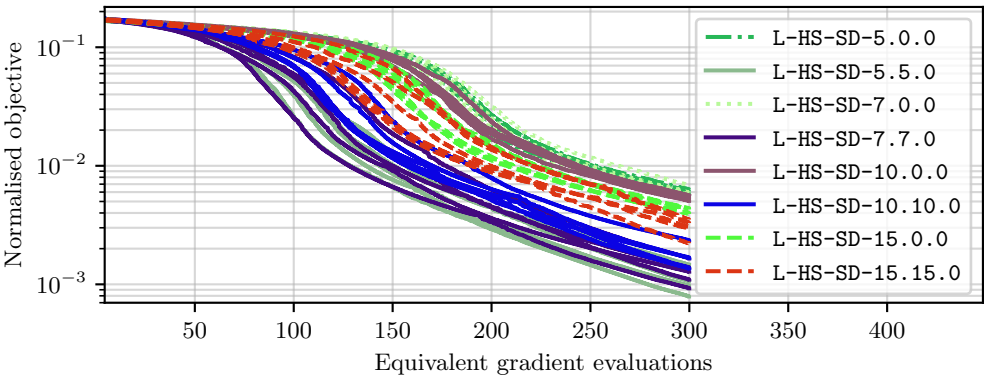

Figure C.12: Five distinct runs per solver variant applied to (B.1) with ambient dimension $n = 100$.

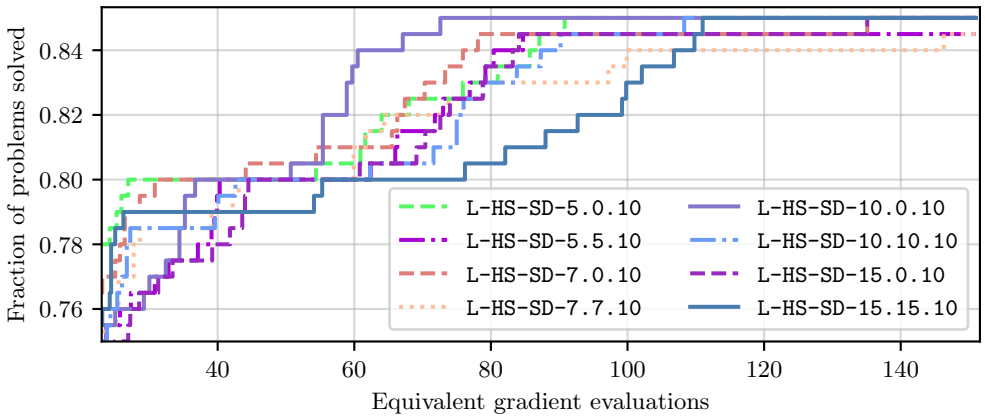

Figure C.13: Data profile for 20 CUTEst problems, with 10 runs per solver variant.

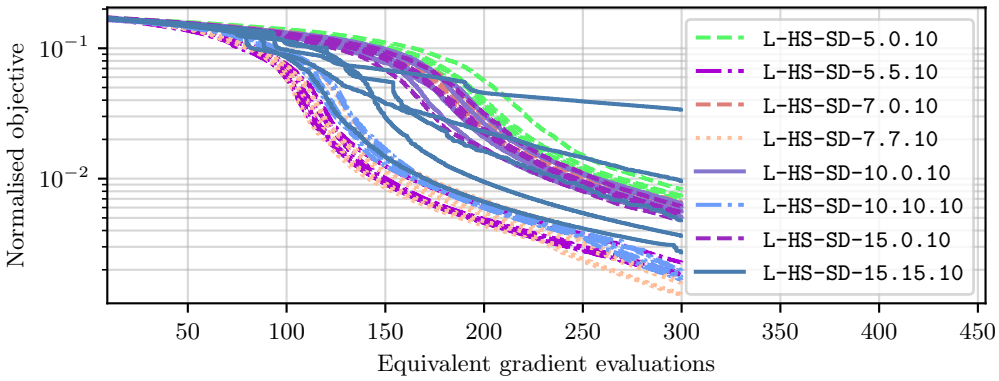

Figure C.14: Five distinct runs per solver variant applied to (B.1) with ambient dimension $n = 100$.

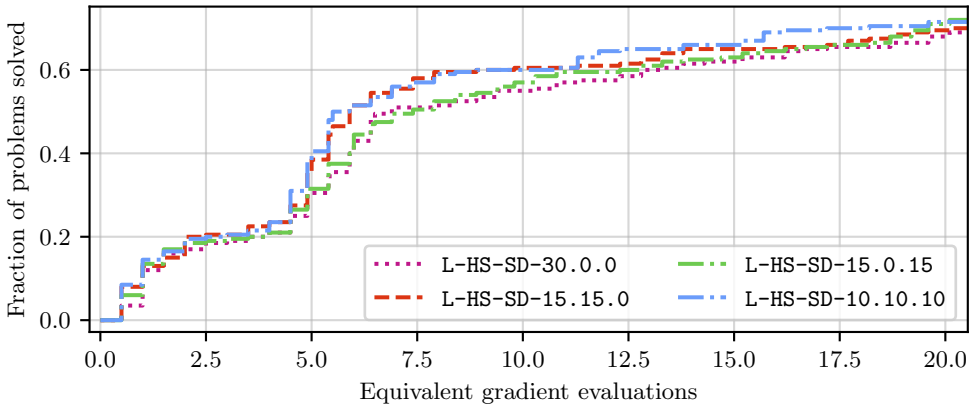

Figure C.15: Data profile for 20 CUTEst problems, with 10 runs per solver variant.

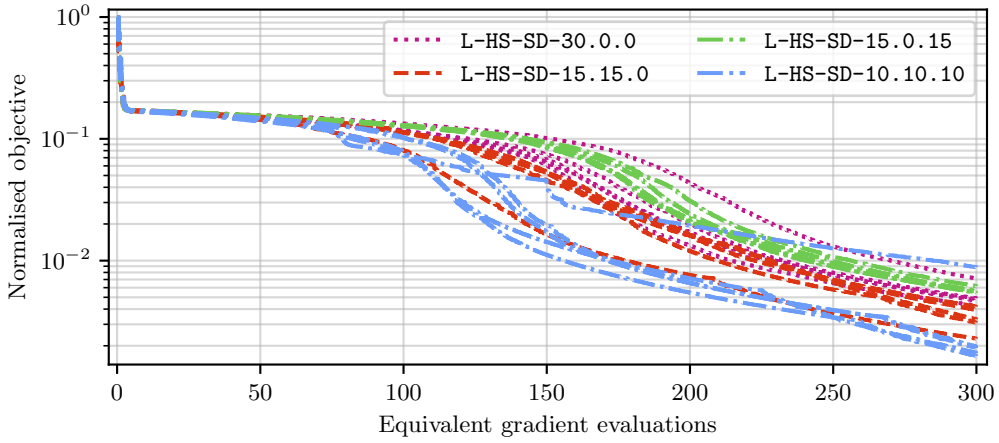

Figure C.16: Five distinct runs per solver variant applied to (B.1) with ambient dimension $n = 100$.

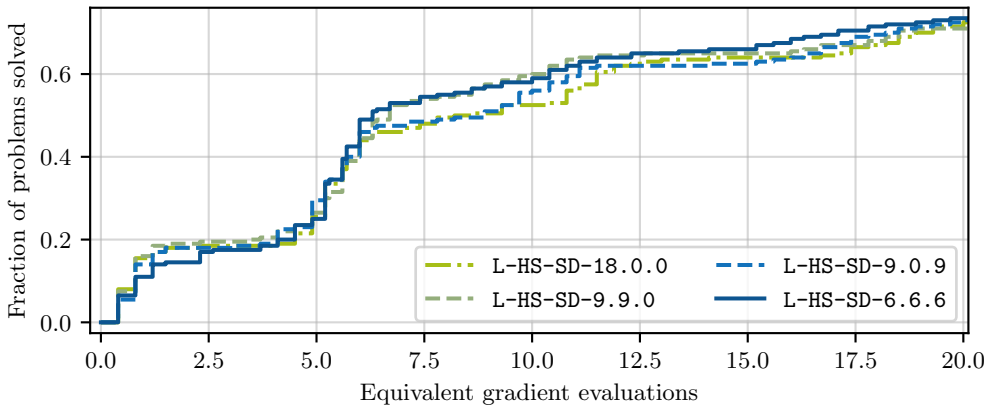

Figure C.17: Data profile for 20 CUTEst problems, with 10 runs per solver variant.

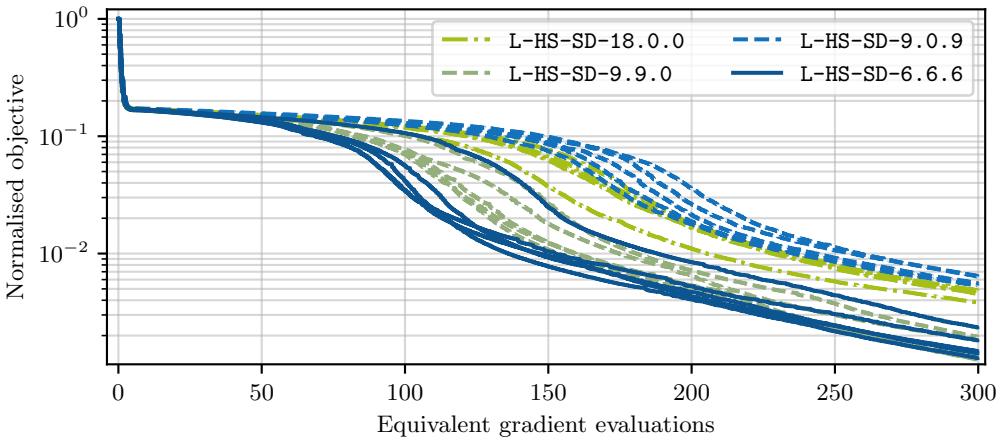

Figure C.18: Five distinct runs per solver variant applied to (B.1) with ambient dimension $n = 100$.

C.3 Appendix to Section 4.2.5

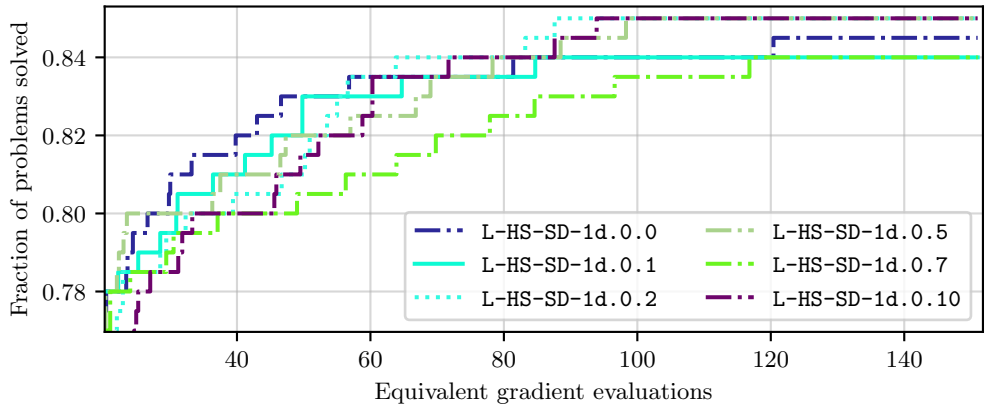

Figure C.19: Data profile for 20 CUTEst problems, with 10 runs per solver variant. Different view.

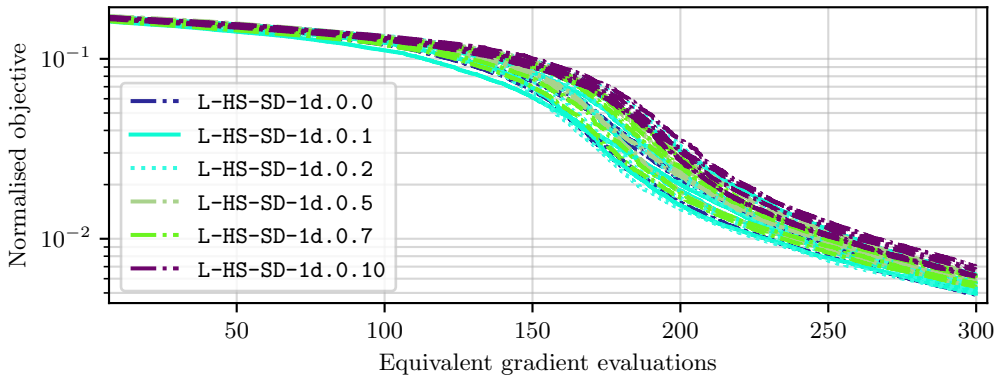

Figure C.20: Five distinct runs per solver variant applied to (B.1) with ambient dimension $n = 100$.

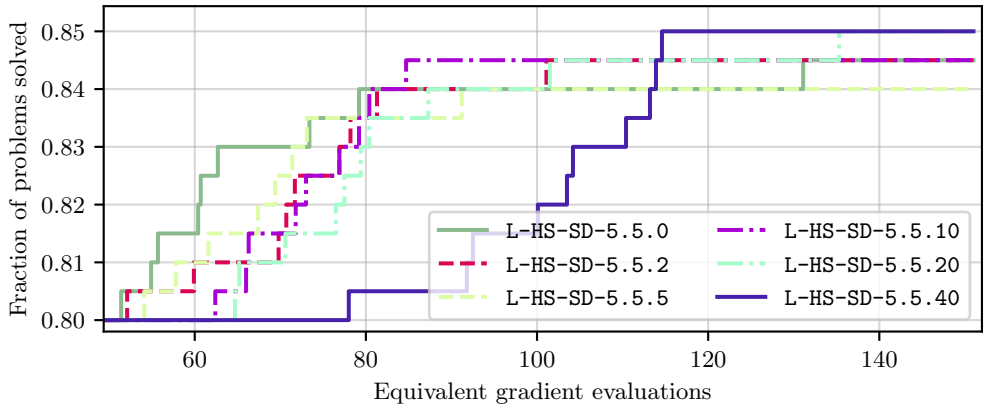

Figure C.21: Data profile for 20 CUTEst problems, with 10 runs per solver variant. Different view.

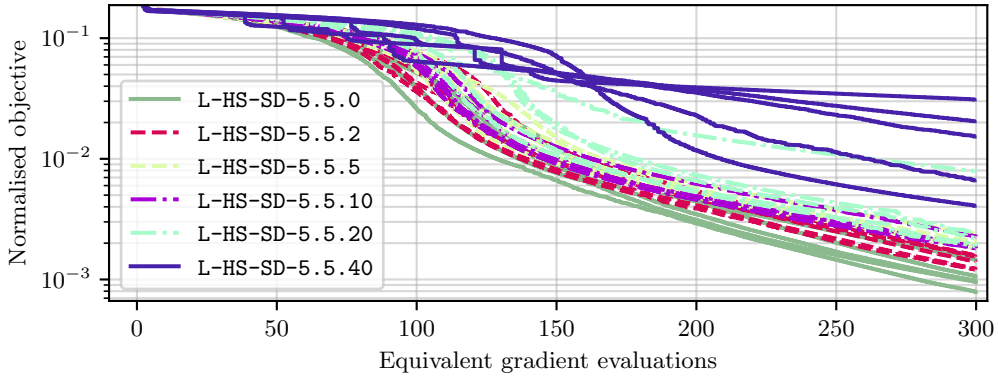

Figure C.22: Five distinct runs per solver variant applied to (B.1) with ambient dimension $n = 100$.

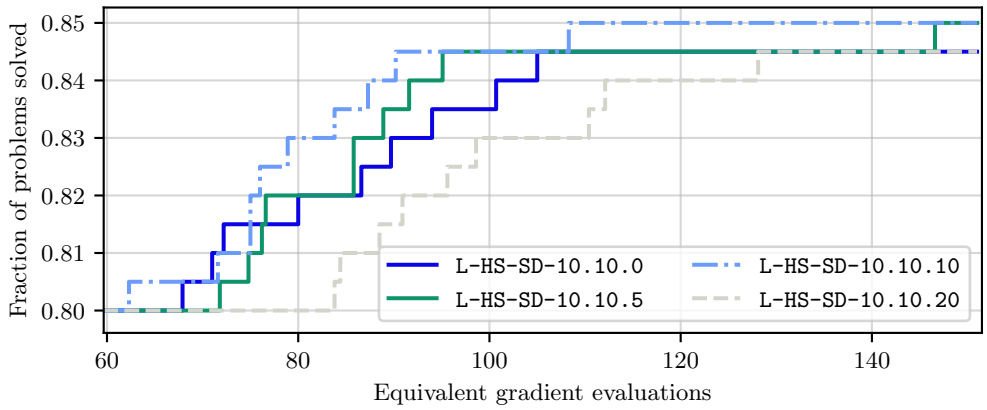

Figure C.23: Data profile for 20 CUTEst problems, with 10 runs per solver variant. Different view.

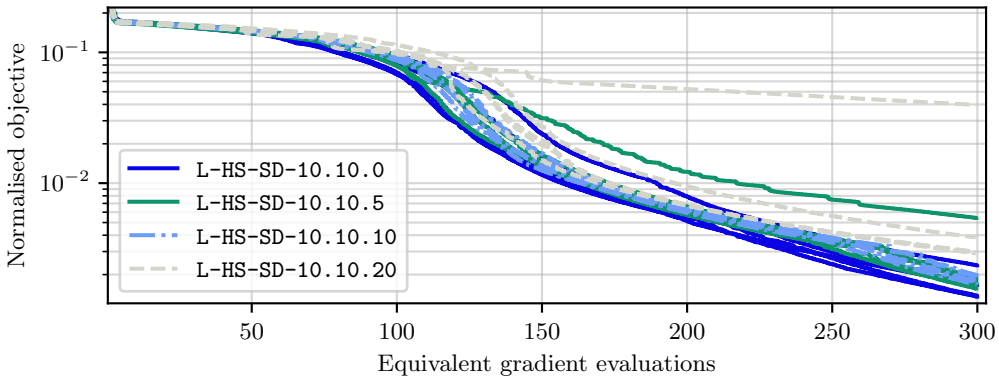

Figure C.24: Five distinct runs per solver variant applied to (B.1) with ambient dimension $n = 100$.

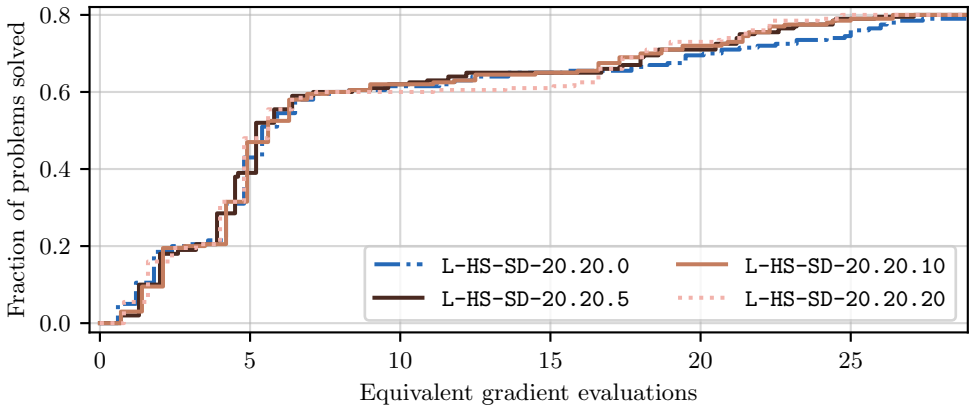

Figure C.25: Data profile for 20 CUTEst problems, with 10 runs per solver variant.

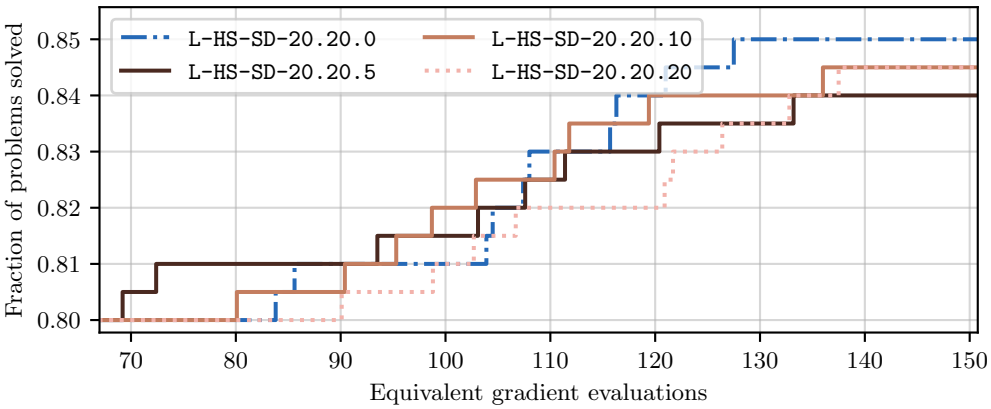

Figure C.26: Data profile for 20 CUTEst problems, with 10 runs per solver variant. Different view.

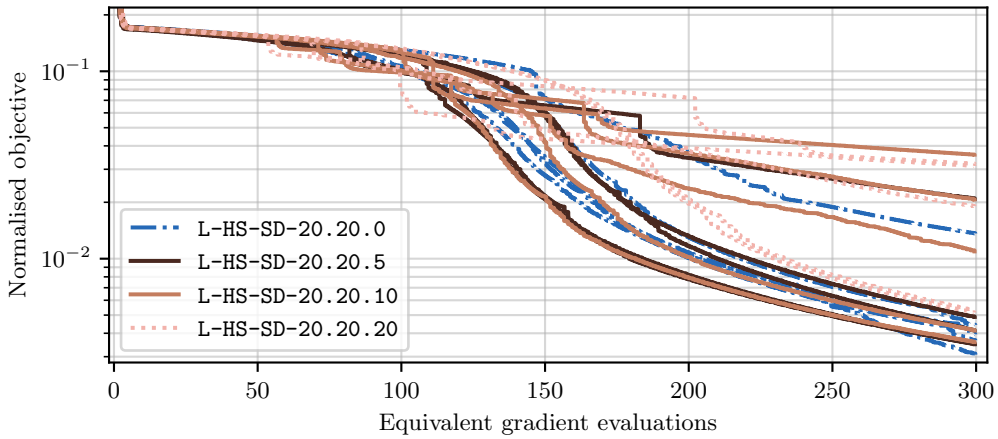

Figure C.27: Five distinct runs per solver variant applied to (B.1) with ambient dimension $n = 100$.

C.4 Appendix to Section 4.4.1

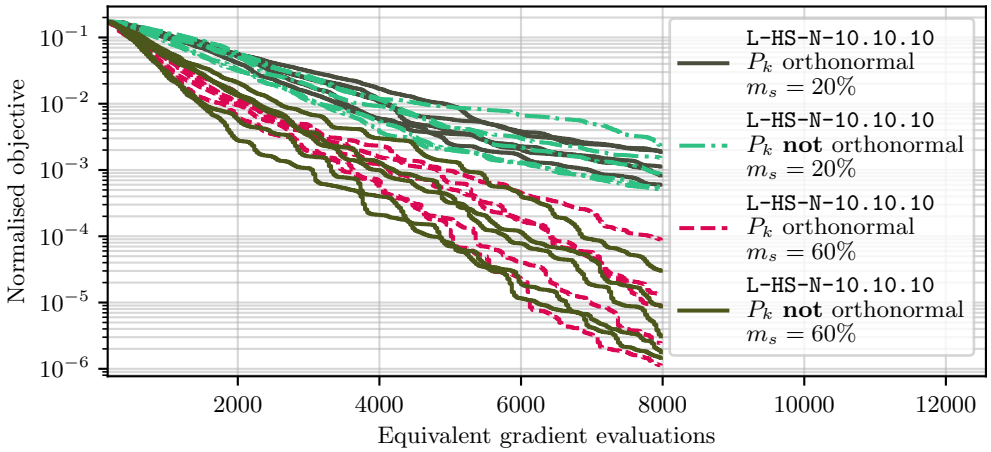

Figure C.28: Five distinct runs per solver variant applied to (B.1) with ambient dimension $n = 100$.

C.5 Appendix to Section 4.4.2

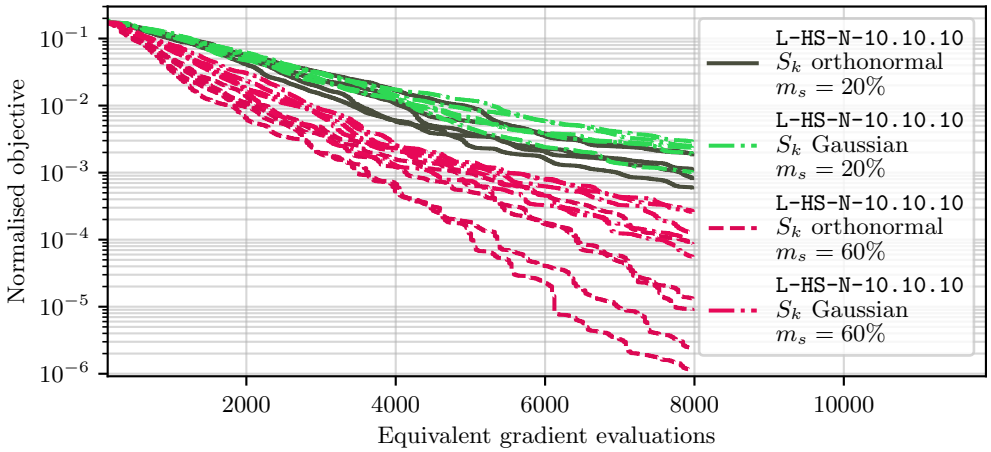

Figure C.29: Five distinct runs per solver variant applied to (B.1) with ambient dimension $n = 100$.

C.6 Appendix to Section 4.4.3

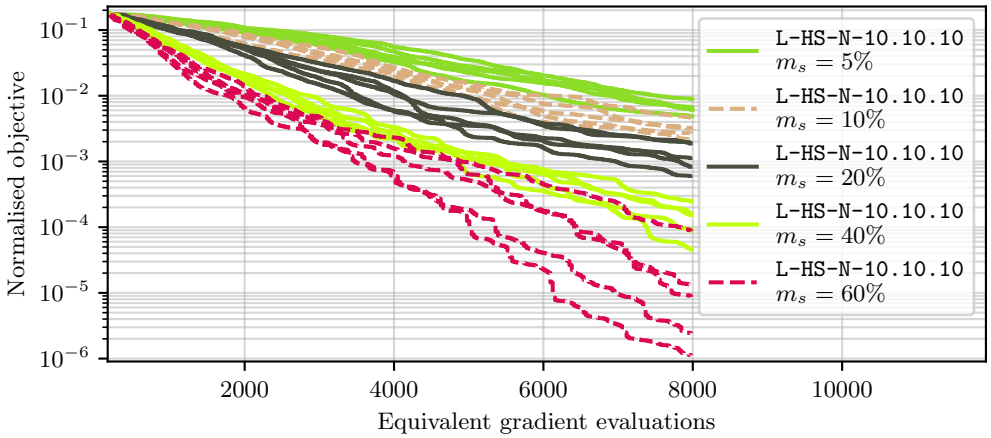

Figure C.30: Five distinct runs per solver variant applied to (B.1) with ambient dimension $n = 100$.

Bibliography

[1] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization Methods for Large-Scale Machine Learning", *SIAM Review*, vol. 60, no. 2, pp. 223–311, Jan. 2018, ISSN: 0036-1445, 1095-7200. DOI: 10.1137/16M1080173. [Online]. Available: https://epubs.siam.org/doi/10.1137/16M1080173.

[2] A. Carrassi, M. Bocquet, L. Bertino, and G. Evensen, "Data assimilation in the geosciences: An overview of methods, issues, and perspectives", *WIREs Climate Change*, vol. 9, no. 5, e535, Sep. 2018, ISSN: 1757-7780. DOI: 10.1002/wcc.535. [Online]. Available: https://doi.org/10.1002/wcc.535.

[3] Z.-H. Wang and Y.-X. Yuan, "A subspace implementation of quasi-Newton trust region methods for unconstrained optimization", *Numerische Mathematik*, vol. 104, no. 2, pp. 241–269, Aug. 2006, ISSN: 0945-3245. DOI: 10.1007/s00211-006-0021-6. [Online]. Available: https://doi.org/10.1007/s00211-006-0021-6.

[4] J. Nocedal and S. J. Wright, *Numerical Optimization* (Springer series in operations research). New York: Springer, 1999, ISBN: 978-0-387-98793-4.

[5] J. Liesen and Z. Strakos, *Krylov Subspace Methods: Principles and Analysis*. Oxford University Press, Oct. 2012, ISBN: 978-0-19-965541-0. DOI: 10.1093/acprof:oso/9780199655410.001.0001. [Online]. Available: https://doi.org/10.1093/acprof:oso/9780199655410.001.0001.

[6] A. Saha and A. Tewari, *On the Finite Time Convergence of Cyclic Coordinate Descent Methods*, May 2010. [Online]. Available: http://arxiv.org/abs/1005.2146.

[7] Y. Nesterov, "Efficiency of Coordinate Descent Methods on Huge-Scale Optimization Problems", *SIAM Journal on Optimization*, vol. 22, no. 2, pp. 341–362, Jan. 2012, ISSN: 1052-6234. DOI: 10.1137/100802001. [Online]. Available: https://doi.org/10.1137/100802001.

[8] P. Richtárik and M. Takáč, "Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function", *Mathematical Programming*, vol. 144, no. 1, pp. 1–38, Apr. 2014, ISSN: 1436-4646. DOI: 10.1007/s10107-012-0614-z. [Online]. Available: https://doi.org/10.1007/s10107-012-0614-z.

[9] S. J. Wright, "Coordinate descent algorithms", *Mathematical Programming*, vol. 151, no. 1, pp. 3–34, Jun. 2015, ISSN: 1436-4646. DOI: 10.1007/s10107-015-0892-3. [Online]. Available: https://doi.org/10.1007/s10107-015-0892-3.

[10] Y.-X. Yuan, "Subspace Techniques for Nonlinear Optimization", in *Some Topics in Industrial and Applied Mathematics*, ser. Series in Contemporary Applied Mathematics Volume 8, vol. Volume 8, World Scientific Publishing, Aug. 2007, pp. 206–218, ISBN: 978-981-270-934-9. DOI: 10.1142/9789812709356_0012. [Online]. Available: https://doi.org/10.1142/9789812709356_0012.

[11] Y.-X. Yuan, "Subspace methods for large scale nonlinear equations and nonlinear least squares", *Optimization and Engineering*, vol. 10, no. 2, pp. 207–218, Jun. 2009, ISSN: 1573-2924. DOI: 10.1007/s11081-008-9064-0. [Online]. Available: https://doi.org/10.1007/s11081-008-9064-0.

[12] P.-W. Wang, C.-p. Lee, and C.-J. Lin, "The Common-directions Method for Regularized Empirical Risk Minimization", *Journal of Machine Learning Research*, vol. 20, no. 58, pp. 1–49, 2019. [Online]. Available: http://jmlr.org/papers/v20/16-309.html.

[13] C.-p. Lee, P.-W. Wang, and C.-J. Lin, "Limited-memory common-directions method for large-scale optimization: Convergence, parallelization, and distributed optimization", *Mathematical Programming Computation*, vol. 14, no. 3, pp. 543–591, Sep. 2022, ISSN: 1867-2957. DOI: 10.1007/s12532-022-00219-z. [Online]. Available: https://doi.org/10.1007/s12532-022-00219-z.

[14] C. Cartis and K. Scheinberg, "Global convergence rate analysis of unconstrained optimization methods based on probabilistic models", *Mathematical Programming*, vol. 169, no. 2, pp. 337–375, Jun. 2018, ISSN: 1436-4646. DOI: 10.1007/s10107-017-1137-4. [Online]. Available: https://doi.org/10.1007/s10107-017-1137-4.

[15] R. Gower, D. Kovalev, F. Lieder, and P. Richtarik, "RSN: Randomized Subspace Newton", in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d. Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/bc6dc48b743dc5d013b1abaebd2faed2-Paper.pdf.

[16] Z. Shao, "On random embeddings and their application to optimisation", DPhil Thesis, University of Oxford, 2021.

[17] C. Cartis, J. Fowkes, and Z. Shao, *A Randomised Subspace Gauss-Newton Method for Nonlinear Least-Squares*, Nov. 2022. [Online]. Available: http://arxiv.org/abs/2211.05727.

[18] C. Cartis, J. Fowkes, and Z. Shao, *Randomised subspace methods for non-convex optimization, with applications to nonlinear least-squares*, 2022. DOI: 10.48550/ARXIV.2211.09873. [Online]. Available: https://arxiv.org/abs/2211.09873.

[19] T. Fuji, P.-L. Poirion, and A. Takeda, *Randomized subspace regularized Newton method for unconstrained non-convex optimization*, 2022. DOI: 10.48550/ARXIV.2209.04170. [Online]. Available: https://arxiv.org/abs/2209.04170.

[20] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, 2nd ed. Philadelphia: SIAM, 2008, ISBN: 978-0-89871-659-7.

[21] C. Cartis and Y. Nakatsukasa, *C6.2 Continuous Optimisation Lecture Notes*, Oct. 2023. [Online]. Available: https://courses.maths.ox.ac.uk/course/view.php?id=5058.

[22] C. Cartis, N. I. M. Gould, and P. L. Toint, *Evaluation Complexity of Algorithms for Nonconvex Optimization: Theory, Computation, and Perspectives*. Philadelphia: Society for Industrial and Applied Mathematics, 2022, ISBN: 978-1-61197-699-1. [Online]. Available: https://epubs.siam.org/doi/10.1137/1.9781611976991.

[23] N. Qian, "On the momentum term in gradient descent learning algorithms", *Neural Networks*, vol. 12, no. 1, pp. 145–151, Jan. 1999, ISSN: 0893-6080. DOI: 10.1016/S0893-6080(98)00116-6. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608098001166.

[24] Y. Nakatsukasa, *C6.1 Numerical Linear Algebra Lecture Notes*, Oct. 2023. [Online]. Available: https://courses.maths.ox.ac.uk/pluginfile.php/94849/mod_resource/content/30/NLA_lecture_notes.pdf.

[25] M. L. Eaton, *Multivariate Statistics: A Vector Space Approach* (Lecture notes-monograph series / Institute of Mathematical Statistics 53). Beachwood, Ohio: Inst. of Mathematical Statistics, 2007, ISBN: 978-0-940600-69-0.

[26] A. Edelman and N. R. Rao, "Random matrix theory", *Acta Numerica*, vol. 14, pp. 233–297, 2005, ISSN: 0962-4929. DOI: 10.1017/S0962492904000236. [Online]. Available: https://www.cambridge.org/core/product/B291B4E6728E10537C2406CE4C341923.

[27] F. Mezzadri, *How to generate random matrices from the classical compact groups*, Feb. 2007. [Online]. Available: http://arxiv.org/abs/math-ph/0609050.

[28] G. H. Golub and C. F. Van Loan, *Matrix Computations* (Johns Hopkins studies in the mathematical sciences), 4th ed. Baltimore: The Johns Hopkins University Press, 2013, ISBN: 978-1-4214-0794-4.

[29] G. B. Pereira, *L-HS*, May 2024. [Online]. Available: https://github.com/berkpereira/L-HS.

[30] N. I. M. Gould, D. Orban, and P. L. Toint, "CUTEst: A Constrained and Unconstrained Testing Environment with safe threads for mathematical optimization", *Computational Optimization and Applications*, vol. 60, no. 3, pp. 545–557, Apr. 2015, ISSN: 1573-2894. DOI: 10.1007/s10589-014-9687-3. [Online]. Available: https://doi.org/10.1007/s10589-014-9687-3.

[31] L. C. W. Dixon and D. J. Mills, "Effect of rounding errors on the variable metric method", *Journal of Optimization Theory and Applications*, vol. 80, no. 1, pp. 175–179, Jan. 1994, ISSN: 1573-2878. DOI: 10.1007/BF02196600. [Online]. Available: https://doi.org/10.1007/BF02196600.

[32] C. Cartis and L. Roberts, "Scalable subspace methods for derivative-free nonlinear least-squares optimization", *Mathematical Programming*, vol. 199, no. 1, pp. 461–524, May 2023, ISSN: 1436-4646. DOI: 10.1007/s10107-022-01836-1. [Online]. Available: https://doi.org/10.1007/s10107-022-01836-1.

[33] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, *Automatic differentiation in machine learning: A survey*, 2015. DOI: 10.48550/ARXIV.1502.05767. [Online]. Available: https://arxiv.org/abs/1502.05767.

[34] J. E. Dennis and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations* (Classics in Applied Mathematics 16). Philadelphia: Society for Industrial and Applied Mathematics, 1996, ISBN: 978-0-89871-364-0.

[35] R. M. Gower, P. Richtárik, and F. Bach, "Stochastic quasi-gradient methods: Variance reduction via Jacobian sketching", *Mathematical Programming*, vol. 188, no. 1, pp. 135–192, Jul. 2021, ISSN: 1436-4646. DOI: 10.1007/s10107-020-01506-0. [Online]. Available: https://doi.org/10.1007/s10107-020-01506-0.

[36] G. W. Stewart, "The Efficient Generation of Random Orthogonal Matrices with an Application to Condition Estimators", *SIAM Journal on Numerical Analysis*, vol. 17, no. 3, pp. 403–409, Jun. 1980, ISSN: 0036-1429. DOI: 10.1137/0717034. [Online]. Available: https://doi.org/10.1137/0717034.