Adapting Zeroth Order Algorithms for Comparison-Based Optimization

Isha Slavin[†]

Project advisor: Daniel McKenzie[‡]

Abstract. Comparison-Based Optimization (CBO) is an optimization paradigm that assumes only very limited access to the objective function $f(x)$. Despite the growing relevance of CBO to real-world applications, this field has received little attention as compared to the adjacent field of Zeroth-Order Optimization (ZOO). In this work we propose a relatively simple method for converting ZOO algorithms to CBO algorithms, thus greatly enlarging the pool of known algorithms for CBO. Via PyCUTEst, we benchmarked these algorithms against a suite of unconstrained problems. We then used hyperparameter tuning to determine optimal values of the parameters of certain algorithms, and utilized visualization tools such as heat maps and line graphs for purposes of interpretation. All our code is available at https://github.com/ishaslavin/Comparison_Based_Optimization.

1. Introduction.

Zeroth-Order Optimization (ZOO) is a branch of mathematical optimization in which one tries to minimize the objective function:

$$f\colon \mathbb{R}^n \to \mathbb{R}.$$

In this paradigm the gradient $\nabla f(x)$ cannot be accessed, and only function evaluations $f(x)$ are available. Comparison-Based Optimization (CBO), which is the focus of this paper, further restricts what one can calculate. For this form of optimization, we assume the user has very limited access to $f(x)$. More explicitly, it is assumed that the only method to obtain information about the function is to use a *Comparison Oracle*, which—when given $x, y \in \mathbb{R}^n$—returns a single bit of information representing whether $f(x) < f(y)$ or $f(y) < f(x)$. More formally:

Definition 1.1 (Comparison Oracle). *A Comparison Oracle is a function* $\mathcal{C}_f(\bullet,\bullet)\colon \mathbb{R}^d \times \mathbb{R}^d \to \{-1,+1\}$ *defined as*

$$\mathcal{C}_f(x,y) = \operatorname{sign}\left(f(y) - f(x)\right).$$

In certain applications it is useful to consider an oracle which occasionally returns the incorrect answer. So, we also define a *noisy* comparison oracle:

Definition 1.2 (Noisy Comparison Oracle). *A Noisy Comparison Oracle with parameter* $p \in [0.5, 1]$ *is a function* $\mathcal{C}_f(\bullet,\bullet)\colon \mathbb{R}^d \times \mathbb{R}^d \to \{-1,+1\}$ *defined as*

$$\mathbb{P}\left[\mathcal{C}_f(x,y) = \operatorname{sign}\left(f(y) - f(x)\right)\right] = p.$$

There are many reasons to consider a comparison oracle and a variety of natural situations where CBO arises. Consider the case of applying reinforcement learning to real-world tasks,

[†]New York University (ivs225@nyu.edu)

[‡]Colorado School of Mines (dmckenzie@mines.edu)

in which a real-valued reward seems erroneous to define [5]. A dditionally, i n m any real-world domains numerical feedback signals are either unavailable or are created arbitrarily to support conventional reinforcement learning algorithms [9, 25]. In these cases, human comparison feedback can be used in comparison oracle form in lieu of a numerical reward function. For example, when attempting to optimize exoskeleton gait researchers determined that since exoskeleton-walking is non-intuitive, users can provide preference between multiple gaits much more reliably than numerically quantifying experience [24]. Another application for using CBO involves optimizing information retrieval systems through maximizing user utility [26]. In this scenario it is infeasible to assign a numerical utility value to a result served to a user, yet simple to obtain judgements of utility by asking a user to compare two results; this is a form of comparison oracle. It is important to note that in such situations, CBO is the only option. Gradients or even function values are simply not available.

Despite the wealth of potential applications, relatively few algorithms for CBO have been proposed [4, 3, 16]. On the other hand, there is a wealth of algorithms available for ZOO, see for example the recent survey [18]. We observe that some, but not all, ZOO algorithms can be adapted to CBO. The first c ontribution o f t his p aper is a s imple c riterion f or determining when this is the case, and a procedure for doing so (See Section 2).

There is also a lack of clear comparison between CBO algorithms in prior literature. In particular, it is not clear how such algorithms perform on large scale continuous optimization problems. This makes it difficult for pr actitioners wishing to use a CB O algorithm in practice to select the appropriate algorithm for their problem. As our second main contribution we provide a software suite for easy benchmarking of CBO algorithms, using the CuTEST set of test functions. We use this to compare five C BO a lgorithms—two n ative C BO m ethods and three that are converted from ZOO methods using the procedure mentioned above.

Finally, CBO algorithms often have many hyperparameters, and it is not always clear from theoretical grounds which hyperparameter settings are optimal. Using the software tools mentioned above, we search the hyperparameter space of two CBO algorithms and identify values which empirically work well on the CuTEST problem set. These could be used by practioners working on similar problems.

The paper is organized as follows. Our main contributions are available in Section 2, containing a novel utility we introduce. Pseudocode for our modifications of current algorithms are presented in Appendix A and experimental results are presented in Section 4. A discussion of those results can be found in Section 5 and Section 6.

2. From ZOO to CBO. The motivation for this work was the observation that many ZOO algorithms *do not use the function values directly*. Rather, such algorithms proceed by sampling a small number of points $z_1, \ldots, z_m$ near the current iterate x_k and then ranking the function evaluations $f(z_1), \ldots, f(z_m)$. The next iterate is then determined using this ranking. We note that such a ranking can be done using only a comparison oracle, and formalize this as Property 2.1.

Property 2.1 (The "Comparison Only" property). *Suppose $\mathcal{A}$ is a ZOO algorithm. We say $\mathcal{A}$ satisfies the "Comparison Only" property if it only uses function values within an* argmin *over a finite set,* e.g $\text{argmin}_{i=1,\dots,m} f(z_i)$ *or to sort a list $z_1, \dots, z_m$ according to $f(z_i)$* i.e. *to find a permutation π such that $f(z_{\pi(i)}) \leq f(z_{\pi(2)}) \leq \dots \leq f(z_{\pi(m)})$.*

If $\mathcal{A}$ satisfies Property 2.1, it can easily be converted to a CBO algorithm using the utility Algorithm 2.1 (an implementation of `Bubble Sort` [6, Section 2.3] using the comparison oracle) or Algorithm 2.2 (an implementation of the `Minimum` algorithm [6, Section 9.1] using the comparison oracle). Observe that the above modification only holds when the zeroth order algorithm finds an argmin over a finite set; otherwise, Property 2.1 will not hold. We illustrate this with the Stochastic Three Point (`STP`) method [1], see Algorithm 2.3. For an example of a ZOO algorithm which does not satisfy our condition, and so cannot be converted into a CBO algorithm, consider the `RSGF` algorithm of [11] (see also [21]). Here, at each iteration a gradient estimator is constructed from function evaluations and used in place of the gradient:

$$\hat{g}_k = \frac{f(x_k + \delta u_i) - f(x_k)}{\delta} \approx \nabla f(x_k) \tag{2.1}$$

$$x_{k+1} = x_k - \alpha \hat{g}_k \tag{2.2}$$

A second example could be any interpolation-based method, *e.g.* `NEWUOA` [22], or the recently introduced `HJ-Mad` [15]. That `CMA-ES` (and related algorithms) satisfy the Comparison Only property appears to be well-known in the evolutionary computing community, where it is frequently mentioned as a source of robustness [10].

Algorithm 2.1 Comparison-based Sort (`CompSort`)

```
**Initialization**
Take in Comparison Oracle: $\mathcal{C}_f$, lst $= [z_1, z_2, ..., z_m]$: list of input values
for $i = 1, \dots, m-1$ do
  for $j = 1, \dots, m-i-2$ do
    if $\mathcal{C}_f\,(\text{lst}[j+1], \text{lst}[j]) = +1$ then
      Swap lst[$j$] and lst[$j+1$]
    end if
  end for
end for
return Sorted list $[z_{\pi(1)}, z_{\pi(2)}, \dots, z_{\pi(m)}]$.
```

The two procedures introduced above are used to convert ZOO to CBO algorithms. Instead of using direct function evaluations to sort a list (Algorithm 2.1) or find t he argmin (Algorithm 2.2), it queries a Comparison Oracle. It then uses the one-bit comparisons outputted by the oracle to either return a list of input vectors, arranged by function values in ascending order (Algorithm 2.1) or output the input which yields the smallest function value (Algorithm 2.2). This is done without ever evaluating the objective function at an input directly.

Algorithm 2.2 Comparison-based Min (`CompMin`)

```
Initialization
Take in Comparison Oracle: $\mathcal{C}_f$, lst $= [z_1, z_2, ..., z_m]$: list of input values
$z_+ = z_1$
for $k = 2, \ldots, m$ do
    if $\mathcal{C}_f(z_+, z_k) = +1$ then
        $z_+ = z_+$
    else if $\mathcal{C}_f(z_+, z_k) = -1$ then
        $z_+ = z_k$
    else
        $z_+ = z_+$ or $z_+ = z_k$ with equal probability.
    end if
end for
return $z_+$
```

As proof of concept, we identified t hree Z OO a lgorithms s atisfying P roperty 2 .1 and transform them to CBO algorithms using Algorithm 2.2 or Algorithm 2.1. The algorithms we consider are the Stochastic Three Points Method (`STP`) [1], Covariance Matrix Adaptation Evolutionary Strategies (`CMA-ES`) [14], and Gradientless Descent (`GLD`) [12]. For all algorithms considered, we have the functionality to generate multiple types of distributions $\mathcal{D}$ including the Uniform distribution over $\{e_1, \ldots, e_n\}$, where e_i denotes the i-th canonical basis vector, Gaussian, Uniform over the unit sphere, and Rademacher (see Appendix A). Note that direct search methods such as the Nelder-Mead simplex algorithm [19], Powell's method [23], and various directional direct search algorithms [18] also satisfy Property 2.1 and can thus be converted to Comparison-Based algorithms.

Algorithm 2.3 shows how we were able to convert the STP optimization algorithm from Zeroth-Order to Comparison-Based. When using step $3a$ the algorithm uses function evaluations to determine the argmin over a set of three input vectors. When using a modification, step $3b$, Algorithm 2.2 is employed to find the argmin, thus side-stepping function evaluations and using a Comparison Oracle instead. Note that the four distributions mentioned above can be used to randomly generate random vectors s_k in the algorithm below. For CBO conversion of `CMA-ES` and `GLD` zeroth-order algorithms see Appendix A.

Algorithm 2.3 Stochastic Three Point (STP). For original algorithm use 3a. For comparison-based version, use 3b.

```
Initialization
Choose $x_0 \in \mathbb{R}^n$, stepsizes $\alpha_k > 0$, probability distribution $\mathcal{D}$ on $\mathbb{R}^n$
for $k = 0, 1, 2, ....$ do
    1. Generate a random vector $s_k \sim \mathcal{D}$
    2. Let $x_+ = x_k + \alpha_k s_k$ and $x_- = x_k - \alpha_k s_k$
    3a. $x_{k+1} = \text{argmin}\{f(x_-), f(x_+), f(x_k)\}$
    3b. $x_{k+1} = \texttt{CompMin}(x_-, x_+, x_k)$
end for
```

Converting a ZOO algorithm to a CBO one using our utilities changes the *query complexity* of the algorithm, defined as the number of function evaluations (resp. comparison oracle queries) required to find a suitable solution, in a predictable way:

Theorem 2.2. *Suppose $\mathcal{A}$ is a ZOO algorithm satisfying Property 2.1 and making m function evaluations per iteration. Then the associated CBO algorithm constructed using Algorithm 2.1 (resp. Algorithm 2.2) makes at most m^2 (resp $m-1$) oracle queries per iteration.*

Proof. This follows from standard complexity analysis of Bubble Sort and Minimization, see [6]. ∎

Remark 2.3. When m is large, using Algorithm 2.1 increases the query complexity of an algorithm significantly, as it requires m^2 queries per iteration. This could be improved by using a more sophisticated sorting algorithm, *e.g.* QuickSort. Nonetheless, care must be taken when adapting ZOO algorithms requiring many sort operations. In particular, we caution against naive comparison-based implementations of the Nelder-Mead simplex algorithm [20], as this may make as many as $\mathcal{O}(n^2)$ queries per iteration.

3. A benchmarking utility for CBO algorithms.

To the best of our knowledge, there does not exist a suite of test problems for benchmarking CBO algorithms. So, we create one using the well-known CuTEST package [13] and the PyCUTEst interface [8]. We do so by providing a simple wrapper which turns any test function f into a comparison oracle $\mathcal{C}_f$; see Algorithm 3.1. We also provide a wrapper allowing for noisy comparison oracles; see Algorithm 3.2. These functions are available at https://github.com/ishaslavin/Comparison_Based_Optimization.

4. Experimental results.

We empirically compare five CBO algorithms. Two are specifically designed for CBO problems (SCOBO [3] and SignOPT [4]) while three (GLD, CMA-ES, and STP) are adapted from ZOO algorithms using the method of Section 2. See Section 2 for further details, and Appendix A for pseudocode. We first benchmark these algorithms on three simple test problems: Sparse Quadratic, MaxK, and (non-sparse) Quadratic, studied in [3, 2].

Algorithm 3.1 Oracle Utility ($\mathcal{C}_f$)

```
Initialization
Take in function $f\colon \mathbb{R}^n \to \mathbb{R}$, $x$: first input value, $y$: second input value
if $f(x) < f(y)$ then
  return 1
else if $f(y) < f(x)$ then
  return -1
else
  return 0
end if
```

Algorithm 3.2 Noisy Oracle Utility

```
Initialization
Take in function $f\colon \mathbb{R}^n \to \mathbb{R}$, $p \in [0,1]$: noisy-ness of oracle, $x$: first input value, $y$: second input value
Generate $r \in [0,1]$ randomly
if $r < p$ then
  if $f(x) < f(y)$ then
    return 1
  else if $f(y) < f(x)$ then
    return -1
  else
    return 0
  end if
else if $r \geq p$ then
  if $f(x) < f(y)$ then
    return -1
  else if $f(y) < f(x)$ then
    return 1
  else
    return 0
  end if
end if
```

Definition 4.1 (SparseQuadratic). *For fixed parameters $n = 200$ and $k = 20$, define*

$$f : \mathbb{R}^n \to \mathbb{R} \tag{4.1}$$

$$f(x) = \sum_{i=1}^{k} x_i^2 \tag{4.2}$$

Definition 4.2 (MaxK). *Fix the parameters $n = 200$ and $k = 20$. For any $x \in \mathbb{R}^n$, let π*

denote a permutation such that $x_{\pi(1)} \geq x_{\pi(2)} \geq \cdots \geq$. *Define:*

$$f_{\mathrm{max-k}} \colon \mathbb{R}^n \to \mathbb{R} \tag{4.3}$$

$$f_{\mathrm{max-k}}(x) = \sum_{i=1}^{k} {x^2}_{\pi(i)} \tag{4.4}$$

By non-sparse quadratic we mean the function

Definition 4.3 (NonSparseQuadratic).

$$f : \mathbb{R}^{200} \to \mathbb{R} \tag{4.5}$$

$$f(x) = \sum_{i=1}^{200} x_i^2 \tag{4.6}$$

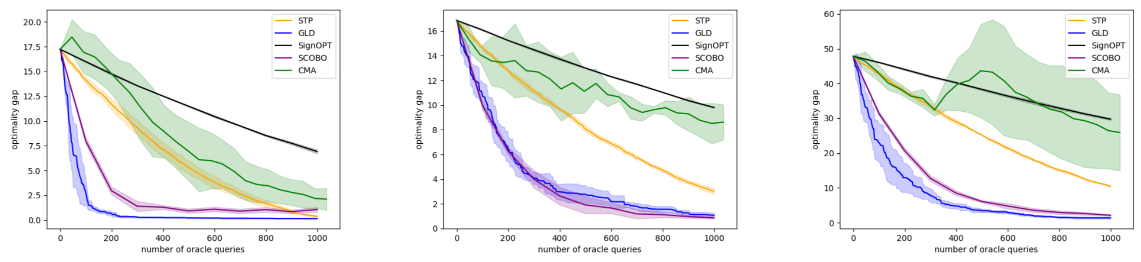

Figure 1. Left: *SparseQuadratic.* **Center:** *MaxK.* **Right:** *NonSparseQuadratic. Graphs display the mean optimality gap plotted against the cumulative number of comparison oracle queries for CBO algorithms against the three functions mentioned above.*

We test our five algorithms on these three functions. For each problem, we run each algorithm using the same initial point x_0 and repeat this five times. Figure 1 shows the mean optimality gap (*i.e.* $f(x_k) - f(x_\star)$) plotted against the cumulative number of comparison oracle queries. The shading indicates the min–max range. As expected [3] `SCOBO` optimizes the fastest on the function MaxK which exhibits gradient sparsity, *i.e.*

$$\|\nabla f(x)\|_0 := |\{i : \nabla_i f(x) \neq 0\}| \leq 20 \text{ for all } x \in \mathbb{R}^n. \tag{4.7}$$

For the function without gradient sparsity (the non-sparse quadratic) as well as SparseQuadratic, `GLD` (originally zeroth-order but modified h ere t o b e c omparison-based) performs best. `STP`, `SignOPT`, and `CMA` show similar patterns of minimizing the functions linearly and not fast.

4.1. PyCUTEst Results. The functions mentioned above are fairly simple. To benchmark against problems that generalize to an overall population of functions we utilized the PyCUTEst [8] Python wrapper to the Fortran package CUTEst [13], used to test optimization software. Available in this package are 117 unconstrained problems, each varying in input vector dimension. We benchmarked our CBO algorithms against 22 of these problems, ranging in input vector dimension from $\mathbb{R}^{10}$ to $\mathbb{R}^{100}$. The 22 PyCUTEst functions used and their dimensions are provided in Table 1. To turn these functions into CBO problems we used the utility described in Section 3. Representative results are shown in Figure 2. To gain a clearer perspective on how these CBO algorithms compare over the entire benchmark set, we use performance profiles [7]. As described in [17], performance profiles are constructed as follows:

Let $\mathcal{P}$ denote the set of benchmark problems and $\mathcal{S}$ denote the set of algorithms under consideration. For each $p \in \mathcal{P}$ and $s \in \mathcal{S}$ the *performance ratio* $r_{p,s}$ is defined by

$$r_{p,s} = \frac{t_{p,s}}{\min_{s' \in \mathcal{S}} t_{p,s'}},$$

where $t_{p,s}$ is the number of comparison oracle queries required for algorithm s to solve problem p (lower is better). So, $r_{p,s}$ represents the performance of s on p relative to the best algorithm in $\mathcal{S}$ for p. The *performance profile* of s, $\rho_s : [1, \infty) \to [0, 1]$ is

$$\rho_s(\tau) = \frac{|\{p \in \mathcal{P} : r_{p,s} \leq \tau\}|}{|\mathcal{P}|}.$$

In other words, $\rho_s(1)$ is the fraction of problems for which s solves the problem first, so a higher value of $\rho_s(1)$ is better. When τ is larger $\rho_s(\tau)$ represents the fraction of problems for which the performance of algorithm s is at most τ times worse than the performance of the best algorithm tested on this problem. Again, higher values of $\rho_s(\tau)$ are preferred and indicate that the algorithm s is *robust*. Our success condition for performance profiling is determined by the *relative* size of either the function values or the gradient. More specifically, we define two success criterion terms to profile against: one in which the final function evaluation $f(x_k)$ is 0.05 times the initial function evaluation $f(x_0)$, and one in which the euclidean norm of the gradient of the function evaluated at x_k is 0.05 times the 2-norm of the gradient of the function evaluated at the starting input x_0. Performance profiles for `SignOPT`, `GLD`, `CMA-ES`, `SCOBO`, and `STP`, tested on the 22 problems in Table 1, are shown in Figure 3.

Table 1
PyCUTEst problems used in benchmarking.

Problem	**CHNROSNB**	**CHNRSNBM**	**ERRINROS**	**ERRINRSM**
Dimension	50	50	50	50
Problem	**HILBERTB**	**QING**	**LUKSAN11LS**	**LUKSAN12LS**
Dimension	10	100	100	98
Problem	**LUKSAN13LS**	**LUKSAN14LS**	**LUKSAN15LS**	**LUKSAN16LS**
Dimension	98	98	100	100
Problem	**LUKSAN17LS**	**LUKSAN21LS**	**LUKSAN22LS**	**MANCINO**
Dimension	100	100	100	100
Problem	**STRTCHDV**	**SENSORS**	**VANDANMSLS**	**WATSON**
Dimension	10	100	22	12
Problem	**TRIGON1**	**TRIGON2**		
Dimension	10	10		

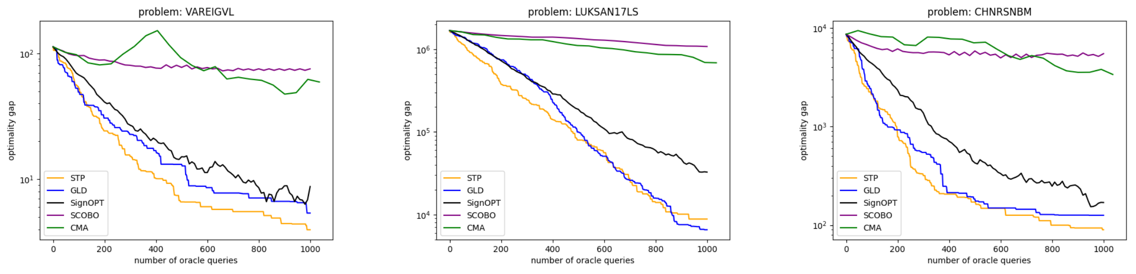

Figure 2. *Optimality gap* $(f(x_k) - f(x_\star))$ vs *number of iterations for three typical PyCUTEst functions.* **Left:** *VAREIGVL.* **Center:** *LUKSAN17LS.* **Right:** *CHNRSNBM.*

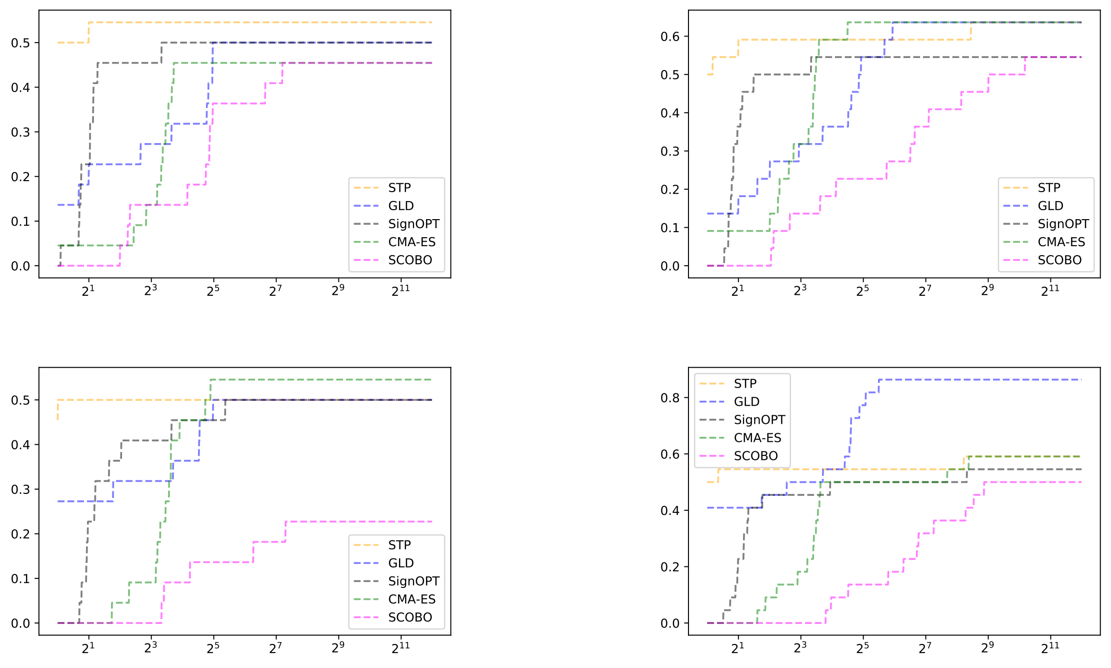

Figure 3. *Performance profiles f or* G LD, S ignOPT, S TP, C MA-ES, *a nd* S COBO. **T op L eft:** *A q uery budget of* 10^4 *and success criterion* $f(x_k) \leq 0.05 f(x_0)$. **Top Right:** *A query budget of* 10^5 *and success criterion* $f(x_k) \leq 0.05 f(x_0)$. **Bottom Left:** *A query budget of* 10^4 *and success criterion* $\|\nabla f(x_k)\|_2 \leq 0.05 \|\nabla f(x_0)\|_2$. **Bottom Right:** *A query budget of* 10^5 *and success criterion* $\|\nabla f(x_k)\|_2 \leq 0.05 \|\nabla f(x_0)\|_2$.

4.2. Noisy Oracle. In practice comparison oracles may occasionally be unreliable. To test the robustness of the CBO algorithms to noise, we ran experiments using the SparseQuadratic function (see Definition 4.1) with the noisy comparison oracle utility (see Definition 1. 2). The noisy oracle takes in a parameter p determining the probability that its output is accurate. We ran experiments using $p = 0.7$ and $p = 0.9$. The results are shown in Figure 4. Notice that `SCOBO` and `GLD` show similar trends, while `STP`, `SignOPT`, and `CMA` show similarities as well with a greater margin of error. Additionally, we can see that `CMA` and `GLD` show low robustness

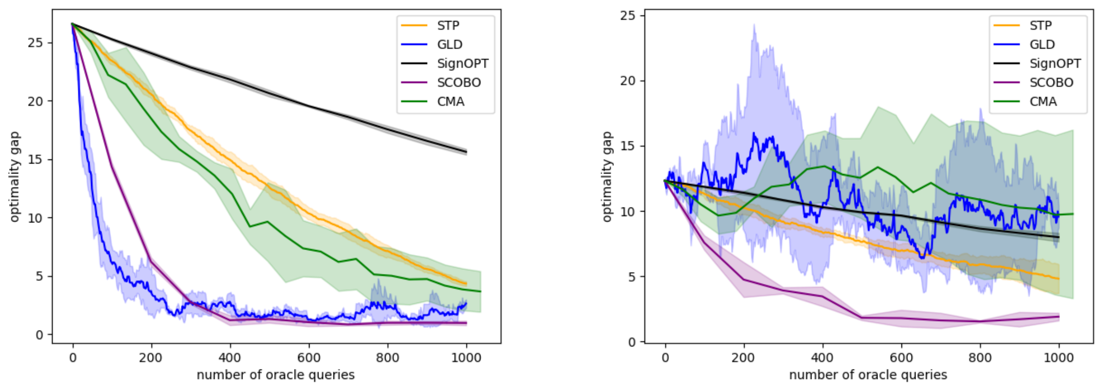

Figure 4. *Minimizing the SparseQuadratic function using a noisy comparison oracle.* **Left:** $p = 0.9$ **Right:** $p = 0.7$.

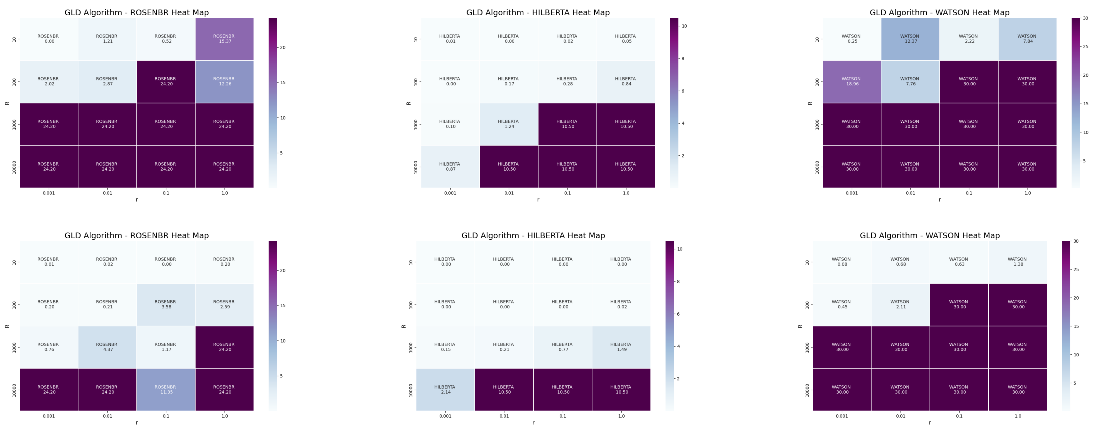

Figure 5. **Top Left:** *ROSENBR (100 queries).* **Top Center:** *HILBERTA (100 queries).* **Top Right:** *WATSON (100 queries).* **Bottom Left:** *ROSENBR (5,000 queries).* **Bottom Center:** *HILBERTA (5,000 queries).* **Bottom Right:** *WATSON (5,000 queries).*

to noise with greater margins of error and a clear struggle to minimize the function, whereas SCOBO maintains performance in the presence of noise well.

4.3. Hyperparameter Tuning.

To demonstrate how our PyCUTEst utility might be used in practice, we tune the hyperparameters for two algorithms, GLD and SCOBO, using three PyCUTEst functions: Rosenbrock (ROSENBR), Hilbert (HILBERTA), and Watson (WATSON). For more information on these functions, see [8].

GLD takes in two scalar parameters R and r, representing the upper and lower bounds of the search radii, respectively. Figure 5 shows results in the form of heat map visualizations of GLD's performance for 100 and 5000 oracle queries. Darker colors represent higher final function evaluations and lighter colors represent lower final function evaluations. Thus, pairings of

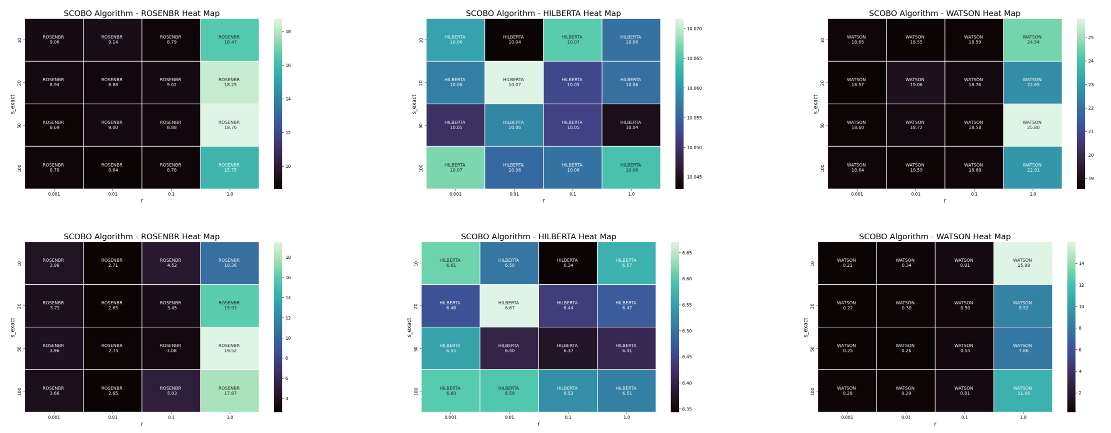

Figure 6. **Top Left:** *ROSENBR (1,000 queries).* **Top Center:** *HILBERTA (1,000 queries).* **Top Right:** *WATSON (1,000 queries).* **Bottom Left:** *ROSENBR (10,000 queries).* **Bottom Center:** *HILBERTA (10,000 queries).* **Bottom Right:** *WATSON (10,000 queries).*

r, R that are a lighter blue on the heat map represent more optimal parameter value pairings than those that are purple. We considered r values of 0.001, 0.01, 0.1, 1.0, displayed on the x-axis, and R values of 10000, 1000, 100, 10, displayed on the y-axis.

SCOBO takes in scalar parameters r, s, and m, see [3] for a discussion on the meaning of these parameters. Heat maps were generated by varying values of s and r, while keeping m fixed, and finding the last function evaluation for each of these pairings. The same was done for s and m, keeping r fixed. Results showing the heat maps for s and r variations are displayed in Figure 6, and heat maps for s and m variations are shown in Figure 7, containing graphs for both 1,000 and 10,000 oracle queries. Darker colors represent lower final function evaluations and greener colors represent higher final function evaluations (notice that this is a different color scale to Figure 5). Thus, pairings that are dark are better than those that are light. Values of r range on the x-axis (0.001, 0.01, 0.1, 1.0) and values of s range on the y-axis (100, 50, 20, 10).

5. Discussion.

There are many insights to gather from the results of our experiments. Firstly, certain CBO algorithms work better than others in various situations. For example, notice that SignOPT and CMA tend to consistently lack in ability to optimize the Sparse-Quadratic, MaxK, and NonSparseQuadratic functions whereas GLD and SCOBO optimize them fast and efficiently (F igure 1). Ho wever in No nSparseQuadratic, a fu nction wi thout sparse gradients (which SCOBO specializes in), GLD converges the function values faster. Therefore in this case it may be in a scientist's best interest to use GLD. Furthermore we see in Figure 2 that SCOBO is unable to generalize to problems of higher dimensions whereas GLD and STP optimize the functions the best, relatively. Thus in the situation of non-sparse gradients and high dimensional input spaces, it may be in the scientist's favor to choose GLD, STP, or SignOPT over SCOBO.

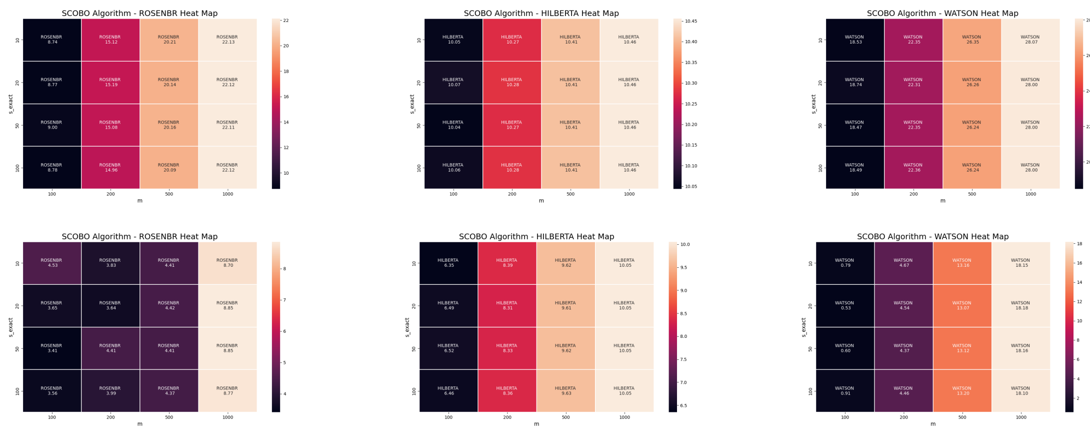

Figure 7. **Top Left:** *ROSENBR (1,000 queries).* **Top Center:** *HILBERTA (1,000 queries).* **Top Right:** *WATSON (1,000 queries).* **Bottom Left:** *ROSENBR (10,000 queries).* **Bottom Center:** *HILBERTA (10,000 queries).* **Bottom Right:** *WATSON (10,000 queries).*

Performance profiles provide information on the fraction of problems for which certain algorithms perform the best and on the robustness of each algorithm. Robustness refers to the ability of an algorithm to eventually solve hard problem instances. From the performance profiles shown in Figure 3, notice that each algorithm solves the same fraction of problems given the query budget. When success criterion is set to be $f(x_k) \leq 0.05f(x_0)$, we see approximately a 55% solve rate given a 10^4 budget and 65% given a 10^5 budget; when success criterion is defined as $\|\nabla f(x_k)\|_2 \leq 0.05\|\nabla f(x_0)\|_2$, we observe the same solve rates respective to query budgets apart from `GLD` which is able to solve 80% of problems given a 10^5 query budget. Thus, while the robustness of each algorithm is relatively equal, this value is still low. However, a low value of robustness is expected as CBO is much harder than ZOO. Notice that `STP` and `SignOPT` have the highest rate of initial increase, indicating they are able to solve easier problems the fastest. `SCOBO` clearly struggles the most in optimizing CUTEst functions. `GLD` has a slower rate of initial increase yet eventually levels off at a high rate, indicating that if left with a high-enough query budget it can successfully solve a substantial portion of problems.

Focusing on the `GLD` hyperparameter tuning experiments, we can determine that the best values of R and r tested were the smallest combination of each ($r = 0.001, R = 10$). Observe that the top-left of each graph becomes the lightest in the smallest number of queries for each of the three PyCUTEst problems we tuned against. This corner of the heat map corresponds to the smallest value of each search radii bound (Figure 5).

For `SCOBO`, we see interesting and perhaps less interpretable results. We first tuned hyperparameters s and m to find the best pairing. We found that s had no effect on the performance of the pairing, whereas smaller m led to better optimization. Recalling that m denotes the number of queries made per iteration (see [3]), this reveals an interesting tradeoff. Although higher m means more accurate gradient approximations, it is better to take a smaller m and hence less accurate gradient approximations, as this allows for more (albeit noisier) iterations given a fixed query budget. We next tuned s and r with $m = 100$ fixed. Altering values for these parameters seemed to make no difference on performance when tuned against the problem HILBERTA, yet for ROSENBR and WATSON any pairing which had an r value of 0.001, 0.01, or 0.1 minimized the function the best. When $r = 1$, any value of s yielded poor performance. For the ROSENBR problem, when $r = 0.01$ any value of s had a very strong minimization. For the WATSON problem, $r = 0.01$ and $r = 0.001$ were both ideal. This indicates the best optimization occurs with a small r value (Figure 6).

Finally, our experiments with a noisy oracle reveal some interesting insights into the robustness of the CBO algorithms considered. When the success rate is $p = 0.9$, we hardly see a difference compared to using a normal Comparison Oracle (Figure 4). However, when the success rate is $p = 0.7$ the benchmarked algorithms are noticeably more noisy. `GLD`'s optimization is considerably changed as it is sensitive to error. This can be explained by the pseudocode Algorithm 2.3 and Algorithm A.2, as this algorithm takes strides in the direction of the smaller input value. Thus if the wrong value is being outputted by the oracle 30% of the time, the algorithm will step in the wrong direction often and considerably diminish its minimization strength. While `SCOBO`, `SignOPT`, and `STP` also seem to get noisier, they are much less impacted by the error. These three algorithms are more *robust* to noise than `GLD` or `CMA`.

6. Conclusions.

We have provided a novel utility for converting ZOO algorithms to CBO algorithms, and showcased how we did so with three state-of-the-art algorithms. We described our benchmarking experiments across these three converted and two already-existing CBO algorithms and analyzed results extensively. Users now have access to a suite of CBO algorithms as well as guidance in their application to continuous, large-scale CBO problems.

There is future work to conduct in this area. One idea is to use human comparison rather than a comparison oracle, so that instead of modeling what a noisy oracle may look like we establish it with human error. This continuation would tie into Cognitive Science, as we would work with human participants. Additionally, we can conduct hyperparameter tuning for more CBO algorithms, as we only covered two (`GLD`, `SCOBO`).

Acknowledgments. Most of this work was conducted while the authors were affiliated with the Mathematics Department at UCLA. The authors are grateful for the stimulating environment provided by this department.

REFERENCES

[1] E. H. Bergou, E. Gorbunov, and P. Richtárik, *Stochastic three points method for unconstrained smooth minimization*, SIAM Journal on Optimization, 30 (2020), pp. 2726–2749.

[2] H. Cai, Y. Lou, D. McKenzie, and W. Yin, *A zeroth-order block coordinate descent algorithm for huge-scale black-box optimization*, in International Conference on Machine Learning, PMLR, 2021, pp. 1193–1203.

[3] H. Cai, D. Mckenzie, W. Yin, and Z. Zhang, *A one-bit, comparison-based gradient estimator*, Applied and Computational Harmonic Analysis, 60 (2022), pp. 242–266.

[4] M. Cheng, T. Le, P.-Y. Chen, H. Zhang, J. Yi, and C.-J. Hsieh, *Query-efficient hard-label black-box attack: An optimization-based approach*, in International Conference on Learning Representation (ICLR), 2019.

[5] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, *Deep reinforcement learning from human preferences*, Advances in neural information processing systems, 30 (2017).

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, MIT press, 2022.

[7] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Mathematical Programming, 91 (2002), pp. 201–213.

[8] J. Fowkes and L. Roberts, *PyCUTEst: Python interface to the CUTEst optimization test environment*, 2019.

[9] J. Fürnkranz, E. Hüllermeier, W. Cheng, and S.-H. Park, *Preference-based reinforcement learning: a formal framework and a policy iteration algorithm*, Machine learning, 89 (2012), pp. 123–156.

[10] S. Gelly, S. Ruette, and O. Teytaud, *Comparison-based algorithms are robust and randomized algorithms are anytime*, Evolutionary Computation, 15 (2007), pp. 411–434.

[11] S. Ghadimi and G. Lan, *Stochastic first-and zeroth-order methods for nonconvex stochastic programming*, SIAM Journal on Optimization, 23 (2013), pp. 2341–2368.

[12] D. Golovin, J. Karro, G. Kochanski, C. Lee, X. Song, and Q. Zhang, *Gradientless descent: High-dimensional zeroth-order optimization*, in International Conference on Learning Representations, 2019.

[13] N. I. Gould, D. Orban, and P. L. Toint, *CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization*, Computational optimization and applications, 60 (2015), pp. 545–557.

[14] N. Hansen, *The cma evolution strategy: A tutorial*, arXiv preprint arXiv:1604.00772, (2016).

[15] H. Heaton, S. W. Fung, and S. Osher, *Global solutions to nonconvex problems by evolution of hamilton-jacobi pdes*, arXiv preprint arXiv:2202.11014, (2022).

[16] M. O. Karabag, C. Neary, and U. Topcu, *Smooth convex optimization using sub-zeroth-order oracles*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, 2021, pp. 3815–3822.

[17] B. Kim, H. Cai, D. McKenzie, and W. Yin, *Curvature-aware derivative-free optimization*, arXiv preprint arXiv:2109.13391, (2021).

[18] J. Larson, M. Menickelly, and S. M. Wild, *Derivative-free optimization methods*, Acta Numerica, 28 (2019), pp. 287–404.

[19] J. A. Nelder and R. Mead, *A simplex method for function minimization*, Computer Journal, 7 (1965), pp. 308–313.

[20] J. A. Nelder and R. Mead, *A simplex method for function minimization*, The computer journal, 7 (1965), pp. 308–313.

[21] Y. Nesterov and V. Spokoiny, *Random gradient-free minimization of convex functions*, Foundations of Computational Mathematics, 17 (2017), pp. 527–566.

[22] M. J. Powell, *The newuoa software for unconstrained optimization without derivatives*, Large-scale nonlinear optimization, (2006), pp. 255–297.

[23] M. J. D. Powell, *An efficient method for finding the minimum of a function of several variables without calculating derivatives*, The Computer Journal, 7 (1964), p. 155, https://doi.org/10.1093/comjnl/7.2.155, +http://dx.doi.org/10.1093/comjnl/7.2.155.

[24] M. Tucker, E. Novoseller, C. Kann, Y. Sui, Y. Yue, J. W. Burdick, and A. D. Ames, *Preference-based learning for exoskeleton gait optimization*, in 2020 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2020, pp. 2351–2357.

[25] G. E. Wimmer, N. D. Daw, and D. Shohamy, *Generalization of value in reinforcement learning by humans*, European Journal of Neuroscience, 35 (2012), pp. 1092–1104.

[26] Y. Yue and T. Joachims, *Interactively optimizing information retrieval systems as a dueling bandits problem*, in Proceedings of the 26th Annual International Conference on Machine Learning, 2009, pp. 1201–1208.

Appendix A. Pseudocode.

This Appendix contains pseudocode for three of the five CBO algorithms (*STP, GLD CMA*). The other two algorithms, *SCOBO* and *SignOPT*, were originally CBO and not altered; thus, we did not feel the need to provide pseudocode for them as they can be found in their original respective papers (see references). The Appendix also contains pseudocode for three test problems (*Sparse Quadratic, MaxK, Non-Sparse Quadratic*), and four distributions (*Uniform distribution of canonical basis vectors, Gaussian, Uniform under Sphere, Rademacher*).

Algorithm A.1 Stochastic Three Point (`STP`). For the original algorithm use 3a. For the comparison-based version, use 3b.

```
Initialization
Choose $x_0 \in \mathbb{R}^n$, stepsizes $\alpha_k > 0$, probability distribution $\mathcal{D}$ on $\mathbb{R}^n$
for $k = 0, 1, 2, ....$ do
    1. Generate a random vector $s_k \sim \mathcal{D}$
    2. Let $x_+ = x_k + \alpha_k s_k$ and $x_- = x_k - \alpha_k s_k$
    3a. $x_{k+1} = \text{argmin}\{f(x_-), f(x_+), f(x_k)\}$
    3b. $x_{k+1} = \texttt{CompMin}(x_-, x_+, x_k)$
end for
```

Algorithm A.2 Gradientless Descent with Binary Search (`GLD`). For the original algorithm use a. For the comparison-based version, use b.

```
Initialization
Take in function $f\colon \mathbb{R}^n \to \mathbb{R}$, $\mathcal{T} \in \mathbb{Z}_+$: number of iterations, $x_0$: starting point, $\mathcal{D}$:
sampling distribution, $\mathcal{R}$: maximum search radius, $r$: minimum search radius
$\mathcal{K} = log(\mathcal{R}/r)$
for $t = 0, ..., \mathcal{T}$ do
    Ball Sampling Trial:
    for $k = 0, ..., \mathcal{K}$ do
        Set $r_k = 2^{-k}\mathcal{R}$
        Sample $v_k \sim r_k\mathcal{D}$
    end for
    a. Update: $x_{t+1} = \text{argmin}_k\{f(y) | y = x_t, y = x_t + v_k\}$
    b. Update: $x_{t+1} = \texttt{CompMin}(x_t, x_t + v_k)$
end for
return $x_t$
```

Algorithm A.3 Covariance Matrix Adaptation - Evolution Strategy (`CMA-ES`). For the original algorithm use a. For the comparison-based version, use b.

```
**Set parameters**
Set parameters $\lambda$, $w_{i=1\cdots\lambda}$, $c_\sigma$, $d_\sigma$, $c_c$, $c_1$, and $c_\mu$
**Initialization**
Set evolution paths $p_\sigma = 0$, $p_c = 0$, covariance matrix $\mathcal{C} = I$, and $g = 0$
**for** $k = 1, \ldots, \mathcal{K}$ **do**
    Sample new population of search points, for $k = 1, \cdots, \lambda$
    (A.1)    $z_k \sim \mathcal{N}(0, \mathcal{I})$
    (A.2)    $y_k = \mathcal{B}\mathcal{D}z_k \sim \mathcal{N}(0, \mathcal{C})$
    (A.3)    $x_k = m + \sigma y_k \sim \mathcal{N}(m, \sigma^2\mathcal{C})$
    a. Sort: Find permutation $\pi$ such that $f(x_{\pi(1)}) \leq f(x_{\pi(2)}) \leq \ldots \leq f(x_{\pi(\lambda)})$
    b. Sort: $[x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(\lambda)}] = \texttt{CompSort}(x_1, \ldots, x_n)$
    Recombination
    (A.4)    $\langle y \rangle_w = \sum_{i=1}^{\mu} w_i y_{\pi(i)}$
    (A.5)    $m \leftarrow m + c_m \sigma \langle y \rangle_w$
    Step - size control
    (A.6)    $p_\sigma \leftarrow (1 - c_\sigma)p_\sigma + \sqrt{c_\sigma(2 - c_\sigma)\mu_{eff}}\mathcal{C}^{-1/2}\langle y \rangle_w$
    (A.7)    $\sigma \leftarrow \sigma \times exp(\frac{c_\sigma}{d_\sigma}(\frac{||p_\sigma||}{\mathcal{E}||\mathcal{N}(0, \mathcal{I})||} - 1))$
    Covariance matrix adaptation
    (A.8)    $p_c \leftarrow (1 - c_c)p_c + h_\sigma\sqrt{c_c(2 - c_c)\mu_{eff}}\langle y \rangle_w$
    (A.9)    $w_i^\circ \leftarrow w_i \times (1 \text{ if } w_i \geq 0 \text{ else } n/||C^{-\frac{1}{2}}y_{i:\,\lambda}||^2)$
    (A.10)   $\mathcal{C} \leftarrow (1 + c_1\delta(h_\sigma) - c_1 - c_\mu \sum w_j)\ \mathcal{C} + c_1 p_c p_c^T + c_\mu \sum_{i=1}^{\lambda} w_i^\circ y_{i:\,\lambda} y_{i:\,\lambda}^T$
**end for**
```

Algorithm A.4 Random Sampling Directions.
Based on type of probability distribution. a: Original (uniform distribution of canonical basis vectors). b: Gaussian distribution. c: Uniform under Sphere. d: Rademacher distribution.

```
**Initialization**
Take in the following inputs: $x$ = number of direction vectors (*type=int*), $y$ = length of each direction vector (*type=int*), $z$ = the type of distribution (*original, gaussian, uniform under sphere, or rademacher*)
if a. $z$ = "Original (uniform distribution of canonical basis vectors)" then
    if $x = 1$ then
        randDirection $= w \in (0, y-1)$
        $s_k = [0, 0, ...., 0]$ where $len(s_k) = y$
        $s_k$[randDirection] $= 1$
    else if $x > 1$ then
        directionVectors = [ ]
        for $i$ from $0, \dots, x-1$ do
            randDirection $= w \in (0, y-1)$
            $s_k = [0, 0, \dots, 0]$ where $len(s_k) = y$
            $s_k$[randDirection] $= 1$
            directionVectors.append($s_k$)
        end for
        $s_k \in \mathbb{R}^{x*y} = [\ [v_1], [v_2], \dots, [v_x]\ ]$ for $v_i \in$ directionVectors, $i \in \{1 \dots x\}$, $v_i \in \mathbb{R}^y$
    end if
end if
if b. $z$ = "Gaussian Distribution" then
    if $x = 1$ then
        $s_k = [d_1, d_2, ..., d_y]$ where $\{d_i\} \in$ standard normal distribution; $i \in \{1, \dots, y\}$
    else if $x > 1$ then
        $s_k \in \mathbb{R}^{x*y} = [\ [v_1], [v_2], \dots, [v_x]\ ]$ for $v_i \in$ standard normal distribution $\subset \mathbb{R}^y$, $i \in \{1 \dots x\}$
    end if
end if
```

Continued on next page....

Algorithm A.5 Random Sampling Directions **Cont'd.**

```
....
if c. $z$ = "Uniform Under Sphere" then
  if $x = 1$ then
    $s_k = [d_1, d_2, ..., d_y]$ where $\{d_i\} \in$ standard normal distribution; $i \in \{1, \ldots, y\}$
    norm = Frobenius norm of $s_k$
    $s_k = s_k$ / norm
  else if $x > 1$ then
    directionVectors = [ ]
    for $i$ from $0, \ldots, x - 1$ do
      $s_k = [d_1, d_2, ..., d_y]$ where $\{d_i\} \in$ standard normal distribution; $i \in \{1, \ldots, y\}$
      norm = Frobenius norm of $s_k$
      $s_k = s_k$ / norm
      directionVectors.append($s_k$)
    end for
    $s_k \in \mathbb{R}^{x*y}$ = [ [$v_1$], [$v_2$], ..., [$v_x$] ] for $v_i \in$ directionVectors, $i \in \{1 \ldots x\}$, $v_i \in \mathbb{R}^y$
  end if
end if
if d. $z$ = "Rademacher Distribution" then
  if $x = 1$ then
    $s_k = 2 * \text{round}([d_1, d_2, ..., d_y]) + 1$ where $\{d_i\} \in$ uniform distribution over [0, 1); $i \in \{1, \ldots, y\}$
    $s_k = s_k / \sqrt{y}$
  else if $x > 1$ then
    directionVectors = [ ]
    for $i$ from $0, \ldots, x - 1$ do
      $s_k = 2 * \text{round}([d_1, d_2, ..., d_y]) + 1$ where $\{d_i\} \in$ uniform distribution over [0, 1); $i \in \{1, \ldots, y\}$
      $s_k = s_k / \sqrt{y}$
      directionVectors.append($s_k$)
    end for
    $s_k \in \mathbb{R}^{x*y}$ = [ [$v_1$], [$v_2$], ..., [$v_x$] ] for $v_i \in$ directionVectors, $i \in \{1 \ldots x\}$, $v_i \in \mathbb{R}^y$
  end if
end if
return $s_k$
```