HONES: A Fast and Tuning-free Homotopy Method For Online Newton Step

Yuting Ye
Division of Biostatistics
Univ. of California, Berkeley
yeyt@berkeley.edu

Cheng Ju
Division of Biostatistics
Univ. of California, Berkeley
cju@berkeley.edu

Lihua Lei
Department of Statistics
Univ. of California, Berkeley
lihua.lei@berkeley.edu

Abstract

In this article, we develop and analyze a homotopy continuation method, referred to as HONES, for solving the sequential generalized projections in Online Newton Step [Hazan et al., 2006b], as well as the generalized problem known as sequential standard quadratic programming. HONES is fast, tuning-free, error-free (up to machine error) and adaptive to the solution sparsity. This is confirmed by both careful theoretical analysis and extensive experiments on both synthetic and real data.

1 Introduction

Online convex optimization (OCO) is an appealing framework that unifies online and sequential optimization problems in various areas. In OCO, a player sequentially makes decisions by choosing a point in a convex set and a concave payoff function is revealed after each decision. The player aims to “maximize” her cumulative payoff, or formally minimize the regret, which measures the gap between the average payoff of her decision strategy and that of the best fixed-action strategy from hindsight. One of the high-profile motivation is the universal portfolio management problem [Cover, 1991], where an investor seeks an online strategy to allocate her wealth on a set of financial instruments without making any assumption on the market behaviors. The payoff can be quantified by logarithmic wealth growth ratio, formulated as \[ \sum_{t=1}^{T} \log(x_t^T \gamma_t) \] where \( x_t(j) \) \( (j = 1, \ldots, n) \) is the share of the \( j \)-th stock in the portfolio and \( \gamma_t(j) \) is the ratio of the closing price of stock \( j \) on time \( t \) to that on time \( t - 1 \). In view of the prohibition of short sales in most markets, the decision space is thus the \( n \)-dimensional simplex \( \Delta_n = \{ x \in \mathbb{R}^n : \sum_{i=1}^{n} x_i = 1, x_i \geq 0 \} \). The regret of a given strategy that outputs \( \{x_t\}_{t=1}^{T} \) is then

\[
\sup_{x \in \Delta_n} \sum_{t=1}^{T} \log(x_t^T \gamma_t) - \sum_{t=1}^{T} \log(x_t^T \gamma_t). \tag{1}
\]

A rich class of algorithms has been developed since [Cover, 1991], which proposed an algorithm with regret \( O(\sqrt{n \log T}) \) but with exponential computation cost per period. [Helmbold et al., 1998] developed an algorithm that reduces the computation cost to \( O(n) \) but incurs a sub-optimal regret \( O(\sqrt{T \log n}) \) in terms of horizon dependence. Later [Kalai and Vempala, 2002] gave a polynomial-time algorithm that achieves the \( O(\sqrt{n \log T}) \) regret, though the order of polynomials is still high. In 2003, the pioneering work by [Zinkevich, 2003] proposed the influential Online Gradient Method.
which achieves $O(\sqrt{T})$ regret for general OCO problems. The next milestone, among others, is achieved by Hazan et al. [2006b], which proposed Online Newton Step that achieves $O(\log T)$ regret under mild conditions, satisfied in universal portfolio management problems, with a practical computation cost per period. Hazan et al. [2006b] also shows that Online Gradient Method is able to achieve $O(\log T)$ regret but requires the loss functions to be strongly convex and, more stringently, the player knowing the strong-convexity modulus apriori. We refer the readers to Shalev-Shwartz [2011] for the history and to Elad Hazan’s thesis [Hazan et al., 2006a] for detailed description of Online Newton Step.

Despite the promising theoretical guarantee of Online Newton Step, the computation efficiency remains a considerable concern for practitioners as the algorithm involves solving a sequence of generalized projections. Specifically, at time $t$ one needs to solve

$$
\min \frac{1}{2} x^T A^{(t)} x - r^{(t)^T} x, \quad s.t. \quad x \in \Delta_n
$$

(2)

where $A^{(t)}$ (resp. $r^{(t)}$) is a sequence of matrices (resp. vectors) such that

$$
A^{(t+1)} = A^{(t)} + g^{(t)} g^{(t)^T},
$$

(3)

for some time-varying vectors $g^{(t)}$. In the special case $A = I$, (2) can be solved quite efficiently in $O(n)$ time [Duchi et al., 2008] due to the explicit form of the solution. Unfortunately, in Online Newton Step, $A^{(t)}$ is never a scaled identity matrix and such benefits disappear for general matrices. Hazan et al. [2006b] suggests using iterative algorithms such as interior-point method [Wright, 1997]. However, it is known that interior-point method has $O(n^3)$ computation cost per iteration, which could be prohibitive for large problems or high-frequency online problems. For this reason, the sub-problem (2) becomes the bottleneck of Online Newton Step, which motivates our work.

Interestingly, (2) is also the generic problem in other areas such as Markowitz’s portfolio management [Markowitz, 1952] and resource allocation [Ibaraki and Katoh, 1988]. This is referred to as standard quadratic optimization dating back to 1950s; See Bomze [1998].

Without the rank-one update structure (3), we should not expect significant improvement over interior-point method as (2) leads to multiple unrelated quadratic programming problems. Nevertheless, (3) is a “huge bonus” that connects the consecutive problems: In fact $A^{(t)}$ is perturbed in only one direction at each step and hence the optimal solutions in consecutive steps should be close. A widely used strategy to exploit the minor change is warm-start, i.e. initializing the iterate as the optimal solution in the last step. Spectral projected gradient (SPG) method [Birgin et al., 2000] is a typical algorithm falling into this category, which combines projected gradient method with smart line search. However, a warm-start is not always allowed. For example, the interior-point method [Wright, 1997] requires the initializer to be an interior point of the constraint set, but as shown in various settings and applications, including our experiments in section 5, the solution in each step often lies on the boundary of the simplex. Another potential algorithm is Exponentiated Gradient Descent [Kivinen and Warmuth, 1997] or Mirror Descent [Beck and Teboulle, 2003]. However, it also requires the initializer to be an interior point in that any zero entry will stay zero. Furthermore, it is lack of an efficient stopping rule, which might not be essential for solving a single problem but is quite important for solving thousands of problems.

On the other hand, it has been proved that the minimizer of a standard quadratic programming problem tends to be sparse under fairly general structural assumptions [Chen et al., 2013; Chen and Peng, 2015]. We also observed the sparsity in both synthetic and real datasets; see section 3 for details. However, none of existing algorithm takes the solution sparsity into account.\footnote{We hope the readers not be confused by the word “sparsity”, which usually appears as an “assumption” on underlying parameters in literature. Here it is a “phenomenon”, observed in both theory and practice, that the solution of the optimization problem tends to be sparse.}

In summary, to the best of our knowledge, existing methods are neither tailored for the sequential problem with structure (3) nor designed to adapt to the solution sparsity. To exploit the structure (3), we resort to the homotopy method, which is proposed decades ago and widely used in optimizing highly non-convex problems such as polynomial systems [Chow et al., 1979; Li, 1983]. The basic idea is to construct a bivariate function $H(x, u)$ on $\mathbb{R}^n \times [0, 1]$ with $H(x, 0) = g(x)$ and $H(x, 1) = f(x)$. In order to optimize $f(x)$ one can start from the optimizer of $g(x)$ and move towards $f(x)$ by
We first consider the problem for a given $w$. Given sufficient smoothness of $H$ along with the non-singularity of the Hessian matrix of $H$ w.r.t $x$, one can obtain a smooth trajectory, or a solution path, penetrating the optimizers of $H(x, w)$ for all $w \in [0, 1]$ with the optimizer of $f(x)$ being the ending point. Homotopy methods for quadratic programming problems have been studied and applied for decades [Frank and Wolfe 1956, Bank et al. 1982, Ritter 1981, Murty and Yu 1988, Best 1996, Efron et al. 2004]. However, all these methods are designed for a specific problem. Recently, the homotopy methods have been applied to sequential problems. For example, Garrigues and Ghaoui (2009) proposed a homotopy method to solve the online LASSO regression problem where the objectives are updated in a similar fashion as (3).

For our problem (2), we define the homotopy function in a zigzag fashion which moves $(A^{(t-1)}, r^{(t-1)})$ to $(A^{(t)}, r^{(t-1)})$ and to $(A^{(t)}, r^{(t)})$ then. See Section 2.2.1 for the explicit construction. We show that the solution path can be calculated efficiently and exactly using the Karush-Kuhn-Tucker (KKT) conditions. By carefully analyzing the evolution of solutions, we propose an algorithm, referred to as Homotopy Online NEwton Step (HONES), which is fast, tuning-free, error-free (up to machine error) and adaptive to solution sparsity.

We compute the number of atomic operations exactly, up to an additive constant, in Theorem 3. As almost all other homotopy continuation methods, the theoretical complexity of our algorithm is in general incomparable to other iterative algorithms like SPG and interior-point method because the former is proportional to the number of turning points on the trajectory (see Section 2.4) while the latter is proportional to the number of iterations to achieve an accurate solution (see Section 2.3). For this reason, we compare the algorithms by the running time on both synthetic and real datasets. To conclude, HONES has a superior performance to SPG and interior-point method and the gain of computational efficiency of HONES is more significant when the solutions are sparser.

The rest of the paper is organized as follows: HONES algorithm is detailed in Section 2, followed by the theory and complexity analysis. The practical implementation is more delicate than the general idea and hence stated in the Supplementary Material. In Section 3, we apply HONES to NYSE and NASDAQ data using Online Newton Step for universal portfolio management. We also conduct experiments on synthetic data and for Markowitz’s portfolio management on real data. Section 4 concludes the article.

2 Proposed Algorithm

A generic framework to solve problem (2) with matrix flow (3) is summarized in Algorithm 1 where ALGO1 and ALGO2 could be arbitrary sub-routines producing the solution of (2) in step 0 and the following steps.

Algorithm 1 Framework to solve (2)

Inputs: Initial matrix $A^{(0)}$, vectors $\{g^{(t)}, r^{(t)}, t = 1, 2, \ldots\}$.

Procedure:

1: Initialize: $x^{(0)} \leftarrow \text{ALGO1}(A^{(0)}, r^{(0)})$;
2: for $t = 1, 2, \ldots$ do
3: $x^{(t)} \leftarrow \text{ALGO2}(A^{(t-1)}, g^{(t)}, r^{(t)}; x^{(t-1)})$;
4: end for

Output: $\{x^{(t)} : t = 0, 1, \ldots\}$.

In this article, we will focus on the online part, namely ALGO2. The complexity of ALGO1 will be increasingly less important as $t$ increases. ALGO1 can be simply chosen as any state-of-the-art algorithm such as the interior-point method. Note that in Online Newton Step [Hazan et al. 2006b], $A^{(0)} = \epsilon I$ is a scaled identity matrix and hence $x^{(0)} = \frac{1}{\epsilon} 1$.

2.1 KKT Condition Within A Step

We first consider the problem for a given $t$. The aim is to minimize $\frac{1}{2} x^T A x - r^T x$ over $\Delta_n$, where $A$ and $r$ are abbreviation of $A^{(t)}$ and $r^{(t)}$. By strong duality, it is equivalent to minimize the Lagrangian...
form
\[ L(x; \mu_0, \mu) = \frac{1}{2} x^T A x - r^T x + \mu_0 (1 - 1^T x) - \mu^T x \]  
where \( \mu_0 \) and \( \mu \) are Lagrangian multipliers with constraint \( \mu_i \geq 0 \) for \( i = 1, \ldots, n \). Denote \( S_x \) by the support of vector \( x \). To be concise, the subscript \( x \) is suppressed in the following context. KKT condition together with Slater’s condition implies that \( (x, \mu_0, \mu) \) is the solution of (4) if and only if
\[ Ax - \mu_0 \mathbf{1} - \mu - r = 0; \]  
\[ 1^T x = 1; \]  
\[ \mu_i x_i = 0, \mu_i \geq 0, x_i \geq 0, \forall i = 1, \ldots, n. \]  
(5)
(6)
(7)

Here (7) is dubbed complementary slackness condition. The definition of \( S = \text{supp}(x) \) entails that \( x_{S^c} = 0 \), and (7) further implies that \( \mu_S = 0 \). Then the condition (5) can be reformulated as
\[
\begin{pmatrix}
A_{SS} & A_{SS^c} \\
A_{S^c S} & A_{S^c S^c}
\end{pmatrix}
\begin{pmatrix}
x_S \\
0
\end{pmatrix}
= \mu_0 \begin{pmatrix} 1_S \\ 1_{S^c} \end{pmatrix} + \begin{pmatrix} 0 \\ \mu_{S^c} \end{pmatrix} + \begin{pmatrix} r_S \\ r_{S^c} \end{pmatrix}
\]  
By separating \( S \) and \( S^c \), we have the following equations for \( x_S \) and \( \mu_{S^c} \).
\[ x_S = \mu_0 A_S^{-1} S 1_S + A_S^{-1} S \mu_{S^c} r_S; \]  
\[ \mu_{S^c} = A_{S^c S} x_S - \mu_0 1_{S^c} r_{S^c} \]
\[ = -\mu_0 (1_{S^c} - A_{S^c S} A_S^{-1} 1_S) - (r_{S^c} - A_{S^c S} A_S^{-1} r_S). \]  
(8)
(9)

The other parameter \( \mu_0 \) can be solved from (6) and (8). In fact,
\[ 1 = 1^T x = 1^T_S x_S = \mu_0 1^T_S A_S^{-1} S 1_S + 1^T_S A_S^{-1} S r_S \]
which implies that
\[ \mu_0 = \frac{1 - 1^T_S A_S^{-1} S r_S}{1^T_S A_S^{-1} S 1_S}. \]  
(10)

In summary, the quadruple \( (S, x_S, \mu_{S^c}, \mu_0) \) which solves (8)-(10) produces the unique solution of (4). Moreover, given the correct support \( S \), we can uniquely solve the other three parameters. Thus, determining \( S \) is the key part in this problem.

2.2 HONES Algorithm

2.2.1 Construction of Homotopy Continuation

Based on the above argument, the problem is reduced to updating support \( S \) with \( A \) replaced with \( A + gg^T \), and \( r \) replaced by \( r + \ell \), where \( g, \ell \) are shorthand notations of \( g^{(l)} \) and \( r^{(l)} - r^{(l-1)} \). Heuristically, \( S \) will not be significantly disturbed when \( g, \ell \) are small perturbations. However, in real problems, there is usually no such constraints on \( g \). Instead, we can consider a homotopy from \((A, r)\) to \((A + gg^T, r + \ell)\). The most natural one is \((A + \lambda gg^T, r + \ell)\) with \( \lambda \in [0, 1] \). In other words, if we denote \( x(\lambda, \lambda) \) be the solution of (4) with \((A, r)\) replaced by \((A + \lambda gg^T, r + \ell)\), then \( x(0, 0) \) is the solution in the last step and \( x(1, 1) \) is the solution after the update. The idea of homotopy continuation method is to calculate \( x(\lambda, \lambda) \) over a path linking \((0, 0)\) to \((1, 1)\). Theoretically, any path suffices and the goal is to find a path which leads to a simple computation. In this article we will consider the Manhattan path from \((0, 0)\) to \((1, 1)\), namely the union of three segments: \( \{(z, 0) : z \in [0, 1]\} \) and \( \{(1, z) : z \in [0, 1]\} \). In other words we first minimize
\[ H^{(1)}(\lambda) = \frac{1}{2} x^T (A + \lambda gg^T) x - r^T x \]
for each \( \lambda \in [0, 1] \) and then minimize
\[ H^{(2)}(\lambda) = \frac{1}{2} x^T (A + gg^T) x - (r + \ell)^T x \]
for each \( \lambda \in [0, 1] \).
Although the problem is augmented, the update is efficient since the support $S$ is shown to be a piecewise constant set on the path and explicit formulas, namely (8) - (10), can be used to compute $(x_S, \mu_S, \mu_0)$ directly when $S$ is fixed. In fact, the triple $(x_S, \mu_S, \mu_0)$ is a simple function of $(\lambda, \lambda)$ as shown in the following theorem.

**Theorem 1**

1. For a given $\lambda$, there exists vectors $u_1, u_2 \in \mathbb{R}^{n+1}$ and scalars $D_1, D_2 \in \mathbb{R}$, which only depend on $S$, such that

\[
\begin{pmatrix}
x_S(\lambda) \\
-\mu_S(\lambda) \\
\mu_0(\lambda)
\end{pmatrix} = \frac{u_1 - u_2 \lambda}{D_1 - D_2 \lambda}.
\]

(11)

2. For given $\lambda$, there exists vectors $u_1, u_2 \in \mathbb{R}^{n+1}$, which only depend on $S$, such that

\[
\begin{pmatrix}
x_S(\lambda) \\
-\mu_S(\lambda) \\
\mu_0(\lambda)
\end{pmatrix} = u_1 - u_2 \lambda.
\]

(12)

**Proof**

1. The proof is quite involved and we relegate it into Theorem [2] in Appendix [B]. The theorem also gives the exact formula of $u_1, u_2, D_1, D_2$.

2. By (10), we have

\[
\mu_0(\lambda) = \frac{1 - 1^T A_S^{-1} (r_S + \lambda \ell_S)}{1^T A_S^{-1} 1_S} = \mu_0(0) - \frac{1^T A_S^{-1} \ell_S}{1^T A_S^{-1} 1_S} \lambda.
\]

Then it follows from (8) that

\[
x_S(\lambda) = \mu_0(0) A_S^{-1} 1_S + A_S^{-1} (r_S + \lambda \ell_S)
\]

\[
= x_S(0) - \left( \frac{1^T A_S^{-1} \ell_S}{1^T A_S^{-1} 1_S} A_S^{-1} 1_S - A_S^{-1} \ell_S \right) \lambda.
\]

Similarly, by (9), we obtain that

\[
-\mu_S(\lambda) = -\mu_S(0) - \left( \frac{1^T A_S^{-1} \ell_S}{1^T A_S^{-1} 1_S} (1_S - A_S A_S^{-1} 1_S) \right) \lambda.
\]

2.2.2 Update of Support

Once $S = S(\lambda)$ is obtained for all $\lambda$, the solution path can be efficiently solved by Theorem [1]. Heuristically, $S$ is piecewise constant and the task is reduced to find the next $\lambda$ that $S(\lambda)$ changes. We consider the update of $S$ in optimizing $H^{(1)}(\lambda)$. The update of $S$ in optimizing $H^{(2)}(\lambda)$ can be obtained in the same way.

For a given $\lambda_0 \in [0, 1]$, if $x_S(\lambda_0) > 0$ and $\mu_S(\lambda_0) > 0$, then (11) implies that there exists $\eta > 0$, such that for any $\lambda \in (\lambda_0 - \eta, \lambda_0 + \eta)$, both $x_S(\lambda)$ and $\mu_S(\lambda)$ remains positive by setting $S(\lambda) = S(\lambda_0)$. Since (8) - (10) are sufficient and necessary, we conclude that $S(\lambda) = S(\lambda_0)$. This argument remains valid until an entry of either $x_S$ or $\mu_S$ hits zero. Denote $j$ by the index of this entry. In the former case, $j$ leaves $S$ and $S$ is updated to $S \setminus \{j\}$. In the latter case, $j$ enters into $S$ and $S$ is updated to $S \cup \{j\}$. The other three parameters are then updated correspondingly by Theorem [1]. Theorem [2] formalizes the above claim. The proof is omitted since it is a direct consequence of sufficiency and necessity of KKT conditions (8) - (10).
Theorem 2 For any given $\lambda_0 \geq 0$, let $\lambda_{\text{new}}$ be the next smallest $\lambda$ such that one entry of either $x_S$ or $\mu_S$ hits 0, i.e.

$$
\lambda_{\text{new}} = \min_+ \left\{ \frac{u_{1i}}{u_{2i}} : i = 1, 2, \ldots, n \right\},
$$

where $u_1$ and $u_2$ are defined in \[1\] and $\min_+$ evaluates the minimum positive number in the set and defined to be $\infty$ if all elements are non-positive. Then $S(\lambda) \equiv S(\lambda_0)$ for $\lambda \in [\lambda_0, \lambda_{\text{new}})$. Further, let

$$
I_1 = \{ i \in S : u_{1i} = u_{2i}\lambda_{\text{new}} \},
$$

$$
I_2 = \{ i \in S^c : u_{1i} = u_{2i}\lambda_{\text{new}} \},
$$

then $S(\lambda_{\text{new}})$ is updated by

$$
S(\lambda_{\text{new}}) = (S(\lambda_0) \setminus I_1) \cup I_2.
$$

Remark 1 According to our experience, $I_1 \cup I_2$ at most contains one element. In other words, $S$ is updated by one element at each iteration.

In summary, the algorithm starts from $\lambda = 0$ and searches for the next smallest $\lambda$ such that one entry of $x_S$ or $\mu_S$ hits zero, then updates $\lambda$ as well as the quadruple $(S, x_S, \mu_S, \mu_0)$. The procedure is repeated until $\lambda$ crosses 1. In other words, there exist a sequence $0 = \lambda_0 < \lambda_1 < \ldots < \lambda_k = 1$, which we call turning points, such that $x(\lambda)$ has the same support between any two consecutive turning points and the value can be calculated by Theorem 1. A counterpart of Theorem 2 can be established for $\lambda$. The whole task reduces to finding all turning points and we call this procedure HONES algorithm. The complexity of HONES algorithm is determined by both the number of turning points and the value can be calculated by Theorem 1. A counterpart of Theorem 2 can be written in the same way as Algorithm 2 by changing $\lambda$ into $\tilde{\lambda}$.

To be more clear, we state the main steps in Algorithm 2 for optimizing $H^{(1)}(\lambda)$ holding $\lambda = 0$. As a convention, the minimum of an empty set is set to be infinity (line 3). The algorithm for optimizing $H^{(2)}(\lambda)$ holding $\lambda = 1$ can be written in the same way as Algorithm 2 by changing $\lambda$ into $\tilde{\lambda}$.

Algorithm 2 Main steps of HONES algorithm in optimizing $H^{(1)}(\lambda)$

Inputs: parameters $A, y, r, g$; initial optimum $x$ (corresponding to $A$)

Procedure:

1: Initialize $\lambda \leftarrow 0$, $S \leftarrow \text{supp}(x)$;
2: while $\lambda < 1$ do
3: $\lambda = \min \{ \lambda_1 > \lambda : v_i(\lambda_1) = 0 \}$ for some $i$;
4: $I_1 \leftarrow \{ i \in S : v_i(\lambda) = 0 \}$;
5: $I_2 \leftarrow \{ i \in S^c : v_i(\lambda) = 0 \}$;
6: if $\lambda \leq 1$ then
7: $S \leftarrow (S \setminus I_1) \cup I_2$;
8: else
9: $\lambda \leftarrow 1$;
10: end if
11: $(x_S, \mu_S, \mu_0) \leftarrow (x_S(\lambda), \mu_S(\lambda), \mu_0(\lambda))$ via \[8\]-\[10\];
12: end while

Output: $(S, x_S, \mu_S, \mu_0)$.

2.3 Implementation and Complexity Analysis

Algorithm 2 presents the main idea without the implementation details. Although we can implement Algorithm 2 by directly computing quantities, e.g. $u_1, u_2$, in every step to find the next turning point as in line 3 and also directly computing the iterates via \[8\]-\[10\] as in line 11, it is fairly inefficient since many quantities appear in several computation steps and we can store them to save the computation. A careful derivation in Appendices B and C shows that the computation complexity is indeed low. For example, although $u_1$ and $u_2$ involves $A_S\widetilde{S}$, there is no need to calculate the matrix inverse directly.
Theorem 3 summarizes the complexity for optimizing $H^{(1)}(\lambda), H^{(2)}(\lambda)$ separately. As a convention, we assume the scalar-scalar multiplication takes a unit time and ignore the addition for simplicity when computing the complexity. Since the real implementation is involved, we state it as well as the proof of theorem 3 in Appendices B and C for two cases separately.

**Theorem 3** In step $t$, denote by $k_A, k_r$ the number of turning points in optimizing $H^{(1)}(\lambda)$ and $H^{(2)}(\lambda)$. Further let $s$ be the maximum support size over the path of $(\lambda, \lambda)$ and $s_*$ by the size of union of all supports from step 1 to step $t$. Let $C_{jt}$ be the computation cost of HONES algorithm in optimizing $H^{(j)}$, then

1. $C_{1t} = ns_* + ns(3k_A + 1) + n(12k_A + 2) + O(k_A)$;
2. $C_{2t} = ns(2k_r + 1) + n(6k_r + 1) + O(k_r)$.

It is clear that the algorithm adapts to the sparsity when optimizing both $H^{(1)}(\lambda)$ and $H^{(2)}(\lambda)$. For each step, the complexity is $O(ns(k_A + k_r))$, upper bounded by $O(n^2(k_A + k_r))$ for the dense case, which is the same as other algorithms due to the inevitable multiplication of $A$ by $x$. In some special regimes, the solution is guaranteed to be sparse with high probability, e.g., the data matrix is randomly generated form a certain distribution such as uniform and exponential distributions [Chen et al., 2013, Chen and Peng, 2015]. We also observe this in our experiments; See Section 3 for details.

2.4 Number of Turning Points

Let $S_t$ be the support of the optimum and $k_t$ be the number of total turning points, then we can derive a generic bound that

\[ k_t \geq |S_t \setminus S_{t-1}| + |S_{t-1} \setminus S_t| \]  

provided that only one element is added to or removed from the support at each update; see Remark 4. This is because it requires at least $|S_{t-1} \setminus S_t|$ steps to pop out the elements in $S_{t-1} \setminus S_t$ and $|S_t \setminus S_{t-1}|$ steps to push in the elements in $S_t \setminus S_{t-1}$ to translate $S_{t-1}$ into $S_t$.

On the other hand, suppose that no other coordinates than those in $S_t \cup S_{t-1}$ enter into the support in the path, then

\[ k_t = |S_t \setminus S_{t-1}| + |S_{t-1} \setminus S_t|. \]  

Heuristically, the equation (14) should hold since if a coordinate, not in $S_t \cup S_{t-1}$, entered into the support on the path, it must be popped out before the end, which, however, should be rare to happen. For both synthetic data and real data in section 3, we observe that there are at least 95% of steps with $k_t$ satisfying (14) and over 99% of steps with $k_t \leq |S_t \setminus S_{t-1}| + |S_{t-1} \setminus S_t| + 6$, i.e. with at most 3 outside coordinates entered into the path. Thus, (14) is a highly reliable result for $k_t$.

As a direct consequence of (14), HONES algorithm is efficient when the support changes slowly in which case $k_t$ is small. In addition, if the solution is sparse, then a rough bound suggests that $k_t \leq |S_t| + |S_{t-1}|$ is small. These phenomena are observed in various situations (see section 3) and (14) explains the good performance of HONES algorithm.

In general, the worst-case bound for the number of turning points can be exponential as [Mairal and Yu, 2012, Gärtner et al., 2009] pointed out for Lasso and SVM respectively. But the number of turning points is usually not large in practice. The same issue appears in Simplex method for linear programming. Although it is known that the worst case complexity is $2^n$, it usually converges in $O(n)$ operations; See Bertsimas and Tsitsiklis [1997].

3 Experiments

In this section, we compare the performance of HONES with SPG and the interior-point method on both real and synthetic data. We implement HONES in MATLAB and implement SPG and interior-point methods by using existing code. To make a fair comparison, SPG uses the solution to step $t$ as the warm start for the solution to step $t + 1$. To evaluate the performance, we display the
cumulative running time as a measure of efficiency. All experiments are conducted on a machine with 3 GHz Intel Core i7 processor, OS X Yosemite system and Matlab 2015a.

3.1 Universal Portfolio Management

[Hazan and Arora, 2006] proposed a version of Online Newton Step (Figure 3.7) that is equivalent to (2) with

\[ A^{(0)} = I, \quad g^{(t)} = \frac{\gamma_t}{x_t^T \gamma_t}, \quad r^t = \frac{1}{4} \sum_{\tau=1}^{t} \frac{\gamma_t}{x_{\tau}^T \gamma_{\tau}}. \]

We apply our algorithm on two datasets from NYSE and NASDAQ\(^6\) with daily stock price data from Jan. 3, 2005 to May. 13, 2016. The NYSE dataset contains 1544 stocks and NASDAQ dataset contains 1101 stocks. This differs from classical studies where at most hundreds of stocks, such as S&P500, are incorporated. Still, we should emphasize that for some financial institutions like hedge funds, the number of base assets is huge and the computation efficiency becomes important when the trading frequency is high. Here we consider a large number of stocks to show the potential of HONES algorithm in optimizing a large basket of assets.

The cumulative running time, measured in seconds, is reported in Figure 1. The interior-point method is quite inefficient as the running time for 10 steps (0.04 epochs) exceeds the total running time of HONES and SPG. Thus the proposal by Hazan et al. [2006b] is not desirable. In addition, HONES is much more efficient than both SPG, especially in the more volatile case (NASDAQ). In fact, HONES achieves a \(2 \times\) speedup on NYSE data and a \(6 \times\) speedup on NASDAQ data! By excluding the first 4 epochs, HONES even achieves a \(12.5 \times\) speedup on NASDAQ data.

To explain the different behavior on two datasets, we report the average solution sparsity in Table 1. Recall that HONES is accurate in every step and we confirm this by checking the KKT condition for each solution. Surprisingly, the solutions are generally sparse for both datasets, as suggested by existing theory [Chen et al., 2013; Chen and Peng, 2015]. It is not surprising that the solutions are sparser on NASDAQ due to the high volatility. As indicated by our theory, the efficiency gain should be more significant in this case.

Finally, we examine our conjecture in Section 2.4 on the number of turning points. As explained there, a benchmark for \(k_t\) is \(|S_t \setminus S_{t-1}| + |S_{t-1} \setminus S_t|\). We refer to \(e_t = (k_t - |S_t \setminus S_{t-1}| - |S_{t-1} \setminus S_t|)/2\)

\(^5\)Here we exclude the running time incurred by updating \(A^{(t-1)}\) to \(A^{(t)}\) since this is unrelated to the optimization and is unavoidable for whatever algorithms.

\(^6\)Data available at https://github.com/Elric2718/HOP.
Table 1: Characteristics of HONES for universal portfolio management on NYSE and NASDAQ data. (Top) solution Sparsity on NYSE and NASDAQ datasets, including the average, standard error, maximum and minimum of the support size; (Bottom) distribution of \( e_t \), the number of excess turning points, on NYSE and NASDAQ datasets.

| Dataset | sparsity |          |          |
|---------|----------|----------|----------|
|         | mean     | std.     | max      | min      |
| NYSE    | 16.0     | 8.6      | 98       | 4        |
| NASDAQ  | 6.46     | 3.7      | 64       | 2        |

| Dataset | proportion of zeros | quantiles |
|---------|---------------------|-----------|
| NYSE    | 65.8%               | 99%       |
| NASDAQ  | 85.4%               | 99.9%     |

as the number of *excess turning points*; see Section 2.4 for details. For each synthetic dataset, we report the proportion of zero \( e_t \) in Table 1. It is clear that most steps (65.8% for NYSE data and 85.4% for NASDAQ data) are predicted by our conjecture and almost all steps (over 99% for both data) are not far away from our conjecture. This indicates that \( k_t \) is an accurate proxy for the number of turning points.

### 3.2 Synthetic Data

![Graph showing cumulative running time of HONES and SPG on synthetic datasets](image)

Figure 2: Cumulative running time of HONES and SPG on synthetic datasets. Each row corresponds to a dimension \( n \) and each column corresponds to a factor \( c \).

As discussed in Section 1, the problem (2) is indeed more general than universal portfolio management. To examine our algorithm comprehensively, we consider (2) under other setups. First we consider the problem with the following structure on synthetic data:

\[
\min_{x \in \Delta_n} \frac{1}{2} (x - y)^T A^{(t)} (x - y). \tag{15}
\]

Without the superscript \( t \), this problem is called *standard quadratic programming problem* and has attracted the attention in various fields, e.g., [Bomze 1998], [Scozzari and Tardella 2008], [Bomze et al.].
Table 2: Solution Sparsity and overall computation gain of HONES over SPG on synthetic datasets. The first two columns correspond to the dimension and the factor $c$; the third column gives the mean of support size with its standard deviation (in the parentheses); the fourth column gives the maximum support size along the path; the last two columns show the ratio of overall running time between SPG and HONES.

| scenarios | sparsity | speed ratio |
|-----------|----------|-------------|
| $n$       | $c$      | mean (s.e.) | max |
| 100       | 0.01     | 81.3 (3.5)  | 88  | 0.97 |
| 1000      | 0.01     | 158.3 (30.6)| 190 | 1.13 |
| 3000      | 0.01     | 146.4 (34.9)| 225 | 1.68 |
| 100       | 0.1      | 19.5 (2.3)  | 26  | 1.34 |
| 1000      | 0.1      | 22.0 (5.3)  | 32  | 5.26 |
| 3000      | 0.1      | 18.3 (4.2)  | 27  | 4.51 |

[2008]. It is of particular interest to study the case where $A(t)$ is a random matrix generated from some distribution. For instance, Chen et al. [2013] consider a Wigner matrix $A$ with $\{A_{ij} : 1 \leq i \leq j \leq n\}$ being i.i.d. random variables and $A_{ij} = A_{ji}$. In this article, we consider another important class of matrices in random matrix theory — covariance matrix of rectangular matrices with i.i.d. entries, i.e. $A(t) = \sum_{s=1}^{t} g(s) (g(s))^T$, where $g(s) \in \mathbb{R}^n$ has i.i.d. Gaussian entries; see Bai and Silverstein [2010] for more discussion. In this case the matrix flow $\{A(t) : t = 1, 2, \ldots\}$ satisfies (3). To avoid singularity, we set $A(0) = \epsilon I$ where $\epsilon = 10^{-4}$ is a small positive number. Then it is easy to see that $A(t)$ is non-singular for all $t$ with probability 1 so that the solution $x(t)$ is unique. The vector $y$ governs the sparsity of the solution. To see this, consider the isotropic case where $A = I$, the solution of (13) is the projection of $y$ onto the simplex. If $y$ is a zero vector, the optimum is a dense vector with all entries $\frac{1}{n}$. In contrast, if $y$ has large entries, the simplex constraint will pull the optimum towards that direction and forces the other entries to be zero, in which case the solution is sparse. The same phenomenon is observed in anisotropic case as will be shown below.

Our goal is to explore the scalability, in terms of the dimension, and the adaptivity to solution sparsity of the algorithms. For the aspect of the dimension, we consider three dimensions: $\{100, 1000, 3000\}$; for the aspect of the sparsity, we set $y = cy_0$ with $y_0$ generated from $N(0, I_{n \times n})$ and $c \in \{0.01, 0.1\}$. For each case, we set the total number of steps as 5000 and treat every 250 steps as an epoch (20 epochs in total). Similar to the previous case, we report the cumulative running time in Figure 2 and other information in Table 2 and Table 3. Here we exclude the interior-point method since it is too slow.

Table 3: Distribution of $e_t$, the number of excess turning points, on synthetic datasets. The first two columns correspond to the dimension and the factor $c$; the third column gives the proportion of zero $e_t$; the last two columns give the 99% and 99.9% quantiles of $e_t$.

| scenarios | proportion of zeros | quantiles |
|-----------|---------------------|-----------|
| $n$       | $c$                 | 99% | 99.9% |
| 100       | 0.01                | 99.8% | 0 | 1   |
| 1000      | 0.01                | 98.9% | 1 | 4   |
| 3000      | 0.01                | 98.7% | 1 | 6   |
| 100       | 0.1                 | 99.9% | 0 | 0   |
| 1000      | 0.1                 | 99.8% | 0 | 1   |
| 3000      | 0.1                 | 99.8% | 0 | 1   |

First we notice that HONES is more scalable in high dimension. Moreover, as expected, a larger $c$ gives sparser solutions along the path and HONES significantly outperforms SPG when the solution is sparse ($c = 0.1$) especially for large-scale problems ($n = 1000, 3000$), in which HONES is over 4.5 times faster than SPG. When the solution is not sparse ($c = 0.01$), HONES is similar to SPG in small-scale problem ($n = 100$) and increasingly more efficient when the size of the problem grows.
Finally the results in Table 3 show that our conjecture in Section 2.4 is extremely accurate in this setting.

### 3.3 Markowitz Portfolio Selection

In this Subsection, we consider the application of HONES algorithm on sequential Markowitz portfolio selection problem. In this problem, the vector of stock prices is assumed to be a random vector with mean $\mu$ and covariance matrix $\Sigma$ and the investor observes a realization from this distribution. The goal is to minimize the risk, measured by the variance $x^T \Sigma x$ of a given portfolio while maintaining a reasonably high average return $x^T \mu$. The problem is usually formulated as follows:

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} x^T \Sigma x - \lambda x^T \mu.$$  

For simplicity we assume $\lambda = 0$. In practice, $\mu$ and $\Sigma$ are unknown and one has to replace them by estimators. The most natural estimators $\hat{\Sigma}(t)$ and $\hat{\mu}(t)$ are the sample covariance matrix and the sample mean, i.e.

$$\hat{\mu}(t) = \frac{1}{t} \sum_{s=1}^{t} w(s), \quad \hat{\Sigma}(t) = \frac{1}{t} \sum_{s=1}^{t} (w(s) - \hat{\mu}(s))(w(s) - \hat{\mu}(s))^T,$$

where $w(t)$ is the vector of daily gains, measured by the entrywise log return $\log(\gamma_t)$, of all assets of interest. Via some algebra, it can be shown that the problem is equivalent to (2) with

$$A(t) = t\hat{\Sigma}(t), \quad r(t) = t\hat{\mu}(t), \quad g(t) = \sqrt{\frac{t-1}{t}} \left( w(t) - \hat{\mu}(t-1) \right).$$

We should emphasize that the solution in this way is optimal from hindsight, which is different from the notions in online learning regret minimization. Nonetheless, it is an interesting and important problem in the context of back testing and risk management since the result can reveal the hidden structure of the assets; see Brodie et al. [2009], Fan et al. [2008, 2012] for more details.

![Figure 3: Cumulative running time of HONES and SPG on NYSE and NASDAQ dataset for Markowitz portfolio management. Each epoch has 252 measurements.](image)

Similar to the Section 3.1, we report the cumulative running time of HONES and SPG in Figure 3 and report other information in Table 4. Again, HONES is more efficient than SPG. We also try the interior-point method on each dataset. It is 67 times slower than HONES on NYSE dataset and 31 times slower than HONES on NASDAQ dataset. Finally, the conjecture on the number of turning points is also validated by the results in the bottom panel of Table 4.
Table 4: (Top) solution Sparsity on NYSE and NASDAQ datasets for Markowitz portfolio management, including the average, standard error, maximum and minimum of the support size; (Bottom) distribution of $e_t$, the number of excess turning points, on NYSE and NASDAQ datasets.

| Dataset | mean | s.e. | max | min |
|---------|------|------|-----|-----|
| NYSE   | 49.8 | 22.7 | 108 | 30  |
| NASDAQ | 148.9| 17.8 | 192 | 127 |

| Dataset | proportion of zeros | 99% | 99.9% |
|---------|---------------------|-----|-------|
| NYSE   | 97.9%               | 1   | 10    |
| NASDAQ | 96.2%               | 3   | 22    |

4 Conclusion

In this article, we propose an efficient algorithm HONES to solve the sequential generalized projection problem (2) with rank-one update (3), appeared as the building block and the bottleneck of Online Newton Step. HONES is a homotopy continuation method that interpolates the consecutive objectives. By a careful derivation, we calculate the exact number of atomic operations, up to an additive constant (Theorem 3) and show that HONES has a good performance when the support of the solution changes slowly with time or is sparse as in many applications. We also provide a heuristic conjecture on the number of turning points which plays an important role in the computation complexity. The efficiency of HONES algorithm is confirmed by extensive experiments on both synthetic and real data. The experimental results also strongly support our heuristic conjecture on the number of turning points and shed light on the theoretical efficiency of HONES.

References

Zhidong Bai and Jack W Silverstein. *Spectral analysis of large dimensional random matrices*, volume 20. Springer, 2010.

B Bank, J Guddat, D Klatte, B Kummer, and K Tammer. *Non-linear parametric optimization*. Akademie-Verlag, Berlin, 1982.

Amir Beck and Marc Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175, 2003.

Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

Michael J Best. An algorithm for the solution of the parametric quadratic programming problem. In *Applied mathematics and parallel computing*, pages 57–76. Springer, 1996.

Ernesto G Birgin, José Mario Martínez, and Marcos Raydan. Nonmonotone spectral projected gradient methods on convex sets. *SIAM Journal on Optimization*, 10(4):1196–1211, 2000.

Immanuel M Bomze. On standard quadratic optimization problems. *Journal of Global Optimization*, 13(4):369–387, 1998.

Immanuel M Bomze, Marco Locatelli, and Fabio Tardella. New and old bounds for standard quadratic optimization: dominance, equivalence and incomparability. *Mathematical Programming*, 115(1):31–64, 2008.

Joshua Brodie, Ingrid Daubechies, Christine De Mol, Domenico Giannone, and Ignace Loris. Sparse and stable markowitz portfolios. *Proceedings of the National Academy of Sciences*, 106(30):12267–12272, 2009.

Xin Chen and Jiming Peng. New analysis on sparse solutions to random standard quadratic optimization problems and extensions. *Mathematics of Operations Research*, 40(3):725–738, 2015.
Xin Chen, Jiming Peng, and Shuzhong Zhang. Sparse solutions to random standard quadratic optimization problems. *Mathematical Programming*, 141(1-2):273–293, 2013.

Shui-Nee Chow, John Mallet-Paret, and James A Yorke. A homotopy method for locating all zeros of a system of polynomials. In *Functional differential equations and approximation of fixed points*, pages 77–88. Springer, 1979.

Thomas M Cover. Universal portfolios. *Mathematical finance*, 1(1):1–29, 1991.

John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the $l_1$-ball for learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pages 272–279. ACM, 2008.

Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. Least angle regression. *The Annals of statistics*, 32(2):407–499, 2004.

Jianqing Fan, Jingjin Zhang, and Ke Yu. Asset allocation and risk assessment with gross exposure constraints for vast portfolios. *Available at SSRN 1307423*, 2008.

Jianqing Fan, Jingjin Zhang, and Ke Yu. Vast portfolio selection with gross-exposure constraints. *Journal of the American Statistical Association*, 107(498):592–606, 2012.

Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956.

Pierre Garrigues and Laurent E Ghaoui. An homotopy algorithm for the lasso with online observations. In *Advances in neural information processing systems*, pages 489–496, 2009.

Bernd Gärtner, Martin Jaggi, and Clément Maria. An exponential lower bound on the complexity of regularization paths. *arXiv preprint arXiv:0903.4817*, 2009.

Elad Hazan and Sanjeev Arora. Efficient algorithms for online convex optimization and their applications. Princeton University, 2006.

Elad Hazan, Adam Kalai, Satyen Kale, and Amit Agarwal. Logarithmic regret algorithms for online convex optimization. In *International Conference on Computational Learning Theory*, pages 499–513. Springer, 2006a.

Elad Hazan, Adam Kalai, Satyen Kale, and Amit Agarwal. Logarithmic regret algorithms for online convex optimization. In *International Conference on Computational Learning Theory*, pages 499–513. Springer, 2006b.

David P Helmbold, Robert E Schapire, Yoram Singer, and Manfred K Warmuth. On-line portfolio selection using multiplicative updates. *Mathematical Finance*, 8(4):325–347, 1998.

Toshihide Ibaraki and Naoki Katoh. *Resource allocation problems: algorithmic approaches*. MIT press, 1988.

Adam Kalai and Santosh Vempala. Efficient algorithms for universal portfolios. *Journal of Machine Learning Research*, 3(Nov):423–440, 2002.

Jyrki Kivinen and Manfred K Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1–63, 1997.

Tien-Yien Li. On chow, mallet-paret and yorke homotopy for solving systems of polynomials. *Bull. Inst. Math. Acad. Sinica*, 11(3):433–437, 1983.

Julien Mairal and Bin Yu. Complexity analysis of the lasso regularization path. *arXiv preprint arXiv:1205.0079*, 2012.

Harry Markowitz. Portfolio selection. *The journal of finance*, 7(1):77–91, 1952.

Katta G Murty and Feng-Tien Yu. *Linear complementarity, linear and nonlinear programming*. Citeseer, 1988.
Klaus Ritter. On parametric linear and quadratic programming problems. Technical report, DTIC Document, 1981.

Andrea Scozzari and Fabio Tardella. A clique algorithm for standard quadratic programming. *Discrete Applied Mathematics*, 156(13):2439–2448, 2008.

Shai Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 4(2):107–194, 2011.

Stephen J Wright. *Primal-Dual Interior-Point Methods*, volume 54. SIAM, 1997.

Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 928–936, 2003.

A Roadmap of Appendices

The general idea of HONES algorithm has been presented in section 2. However, to implement it efficiently, we need much more effort to explore the structure of the solution path and find common quantities which are used by multiple sub-routines. To make the derivation well-organized, we start from considering the case where only one of $A$ and $r$ is time-varying while the other parameter is fixed. The case where $A$ is time-varying and the case where $r$ is time-varying are considered separately in Appendix B and C. Then in Appendix D, we combine two components and state the implementation for the general case.

In each following appendix, we will first define a list of case-specific intermediate variables, which are the key ingredients to improve efficiency. Then we describe the whole procedure followed by details of each sub-routine. Finally, we give a detailed complexity analysis at the end of each appendix.

B Implementation of HONES Algorithm With Time-Varying $A$ and Fixed $r$

B.1 Intermediate Variables

Although (8)-(10) completely define the solution, they involves messy terms. To simplify the notations, we define three lists of intermediate variables. We should emphasize that these variables also play important roles in the algorithm design since they capture the quantities repeatedly appeared and unnecessary computation can be avoided by storing their values in memory.

The intermediate variables are defined as follows. First, let $M$ be a $n \times n$ matrix such that

$$
M_{SS} = A_{SS}^{-1}, \quad M_{S^c S} = -A_{S^c S} A_{SS}^{-1}, \quad M_{. S^c} = 0.
$$

(16)

For large-scale problem where $n$ is prohibitively large, we can only store a $n \times |S|$ matrix by removing the zero entries of $M$. This saves storage cost significantly. Then we define two vectors $\eta, \tilde{\eta} \in \mathbb{R}^n$ such that

$$
\eta_S = M_{SS} g_S, \quad \eta_{S^c} = g_{S^c} + M_{S^c S} g_S, \quad \tilde{\eta}_S = M_{SS} 1_S, \quad \tilde{\eta}_{S^c} = 1_{S^c} + M_{S^c S} 1_S \in \mathbb{R}^n.
$$

(17)

Last we define four scalars.

$$
D = 1_S^T A_{SS}^{-1} 1_S, \quad D_g = 1_S^T A_{SS}^{-1} g_S, \quad D_{gg} = g_S^T A_{SS}^{-1} g_S \quad D_{gr} = -\eta_{S^c}^T r_S.
$$

(18)

Note that all variables are functions of $\lambda$ if $A$ is replaced by $A + \lambda g g^T$ and we denote them by $\cdot(\lambda)$. For example,

$$
D(\lambda) = 1_S^T (A_{SS} + \lambda g g^T)^{-1} 1_S,
$$

and others can be defined in a similar fashion. The following lemma formulates these functions.

**Lemma 1** Let $\alpha(\lambda) = \frac{\lambda}{1 + \lambda D_{gg}}$. Before any entry of $(x_S, \mu_{S^c})$ hitting 0, it holds that

- $M_{. S}(\lambda) = M_{. S} - \alpha(\lambda) \eta_{S^c}^T$;
- $\eta(\lambda) = \frac{\eta}{1 + \lambda D_{gg}}$;

**Proof**
Thus, this implies that

\[ D = D - \alpha(\lambda) D^2 \eta; \]

\[ (D_g(\lambda), D_{gg}(\lambda), D_{gr}(\lambda)) = \frac{1}{1 + \lambda D_{gg}} (D_g, D_{gg}, D_{gr}). \]

**Proof** By Sherman-Morrison-Woodbury formula,

\[ M_{SS}(\lambda) = (A_{SS} + \lambda g_S g_S^T)^{-1} = A_{SS}^{-1} - \frac{A_{SS}^{-1} g_S g_S^T A_{SS}^{-1}}{1 + \lambda g_S^T A_{SS}^{-1} g_S} = M_{SS} - \alpha(\lambda) \eta \eta_S^T. \]

This implies that

\[
M_{S^cS}(\lambda) = -(A_{S^cS} + \lambda g_{S^c} g_{S^c}^T)(A_{SS}^{-1} - \alpha(\lambda) \eta \eta_S^T)
\]

\[
= M_{S^cS} - \lambda g_{S^c} g_{S^c}^T A_{SS}^{-1} + \lambda \alpha(\lambda) g_{S^c} g_{S^c}^T \eta \eta_S + \alpha(\lambda) A_{S^cS} \eta \eta_S^T
\]

\[
= M_{S^cS} - \lambda g_{S^c} \eta_{S^c}^T + \alpha(\lambda) D_{gg} g_{S^c} \eta_{S^c}^T + \alpha(\lambda) A_{S^cS} \eta \eta_S
\]

\[ = M_{S^cS} - \alpha(\lambda) g_{S^c} \eta_{S^c}^T + \alpha(\lambda) A_{S^cS} \eta \eta_S^T \quad [\text{Use } D_{gg} = g_S^T \eta_S] \]

\[ = M_{S^cS} - \alpha(\lambda) g_{S^c} \eta_{S^c}^T + \alpha(\lambda) A_{S^cS} \eta \eta_S^T \quad [\text{Use } \lambda - \alpha(\lambda) D_{gg} = \alpha(\lambda)] \]

Putting pieces together, we obtain that

\[ M_{S^cS}(\lambda) = M_{S^cS} - \alpha(\lambda) \eta \eta_S^T. \]

Based on \( M_{S^cS}(\lambda) \), it is straightforward to derive other variables. For \( \eta(\lambda) \),

\[
\eta_{S^c}(\lambda) = M_{S^cS}(\lambda) g_{S^c} = \eta_{S^c} - \alpha(\lambda) \eta \eta_S^T g_{S^c}
\]

\[
= (1 - \alpha(\lambda) D_{gg}) \eta_{S^c} = \eta_{S^c} \frac{1}{1 + \lambda D_{gg}}.
\]

Thus,

\[ \eta(\lambda) = \frac{\eta}{1 + \lambda D_{gg}}. \]

Similarly,

\[
\hat{\eta}_S(\lambda) = M_{SS}(\lambda) 1_S = \hat{\eta}_S - \alpha(\lambda) \eta \eta_S^T 1_S
\]

\[
= \eta_S - \alpha(\lambda) D_g \eta_S;
\]

\[
\hat{\eta}_{S^c}(\lambda) = 1_{S^c} + M_{S^cS}(\lambda) 1_S = 1_{S^c} - \alpha(\lambda) \eta \eta_S^T 1_S
\]

\[
= \eta_{S^c} - \alpha(\lambda) D_{gg} \eta_{S^c},
\]

and hence

\[ \hat{\eta}(\lambda) = \hat{\eta}_S - \alpha(\lambda) D_g \eta. \]

The last four scalars are even easier to handle. In fact, \( D(\lambda) \) can be derived directly by

\[ D(\lambda) = 1_S^T (M_{SS} - \alpha(\lambda) \eta \eta_S^T) 1_S = D - \alpha(\lambda) D_g^2 \]

By reformulating the other three variables, the last statement can be proved,

\[
(D_g(\lambda), D_{gg}(\lambda), D_{gr}(\lambda)) = (1_S^T \eta_S(\lambda), g_S^T \eta_S(\lambda), -r_S^T \eta_S(\lambda))
\]

\[
= \frac{1}{1 + \lambda D_{gg}} (1_S^T \eta_S, g_S^T \eta_S, -r_S^T \eta_S)
\]

\[
= \frac{1}{1 + \lambda D_{gg}} (D_g, D_{gg}, D_{gr}).
\]
B.2 Implementation

Lemma 1 implies that given the function values of the intermediate variables at \( \lambda = 0 \), the function values at a neighborhood of 0 can be calculated directly. Within time \( t \), all intermediate variables will be update correspondingly when the support changes. It has been shown in Theorem 2 that updating \( M \) requires \( n|S| \) operations while updating other variables only requires \( n \) operations. When the problem transfer from time \( t \) to time \( t + 1 \), the variables \((\eta, D_g, D_{gg}, D_{gr})\) needs to be recalculated since it depends on a new \( g^{(t+1)} \). In contrast, \((M, \tilde{\eta}, D)\) can be updated in the same way as in time \( t \).

In summary, \((M, \tilde{\eta}, D)\) is shared by for all times while \((\eta, D_g, D_{gg}, D_{gr})\) is only used in a single time. For compact notations, we define \( \text{Par}_1 \) and \( \text{Par}_2 \) as

\[
\text{Par}_1 = \{M, \tilde{\eta}, D\}, \quad \text{Par}_2 = \{\eta, D_g, D_{gg}, D_{gr}\}. \tag{19}
\]

In addition, we denote \( v \) by the concatenation of \( x_S \) and \( -\mu_{S^c} \), i.e.

\[
v_S = x_S, \quad v_{S^c} = -\mu_{S^c}, \tag{20}
\]

as a \( n \times 1 \) vector. It will be shown in the next subsection that \( v(\lambda) \) can be expressed in a concise way.

Algorithm 3 describes the full implementation of HONES algorithm, which solves the online problem (2) with \( \eta \) fixed. The sub-routines involved will be discussed separately in following subsections. Roughly speaking, after initialization, we enter into the outer-loop and try to solve (3) at time \( t \) using the information from time \( t - 1 \). Starting from \( \lambda = 0 \), we search for the next \( \lambda \) that pushes one entry of \( v \) to zero. FIND_LAMBDA fulfills this goal and also reports the corresponding entry \( j \). Since \((v, \mu_0, \text{Par}_1, \text{Par}_2)\) are all functions of \( \lambda \), we update them by UPDATE_BY_LAMBDA, in which \( \lambda^{\text{inc}} \) denotes the increment to reach the next turning point from the current one. Unlike \((v, \mu_0, \text{Par}_1, \text{Par}_2)\) has discontinuity at each turning point \( \lambda \) due to the change of support \( S \). They are updated by UPDATE_SHRINK_SUPPORT and UPDATE_EXPAND_SUPPORT depending on whether \( S \) is shrunk or expanded. The procedure is repeated until \( \lambda \) cross over 1 and an inner-loop finishes. At the end, \( \text{Par}_2 \) is recomputed for new \( g^{(t+1)} \), which is achieved by DIRECT_UPDATE.

B.3 FIND_LAMBDA

With the help of intermediate variables, we can express \((x_S, \mu_{S^c}, \mu_0)\) in a compact way.

**Theorem 4** Before any entry of \((x_S, \mu_{S^c})\) hitting 0, it holds that

\[
\mu_0(\lambda) = \mu_0 + \frac{\alpha(\lambda)}{D - \alpha(\lambda)D_g^2} \cdot D_g(D_g\mu_0 - D_{gr}). \tag{21}
\]

and

\[
v(\lambda) = v + \frac{\alpha(\lambda)}{D - \alpha(\lambda)D_g^2} \cdot (D_g\mu_0 - D_{gr}) \cdot (D_g\tilde{\eta} - D\eta). \tag{22}
\]

**Proof** First we prove (21). By definition,

\[
\frac{1}{1_S} A_{SS}(\lambda)^{-1} 1_S = \frac{1}{1_S} M_{SS}(\lambda) r_S.
\]

By Lemma 1

\[- M_{SS}(\lambda) r_S = -(M_{SS} - \alpha(\lambda)\eta_S\eta_S^T) r_S = -M_{SS} r_S - \alpha(\lambda) D_{gr} \eta_S.
\]

Thus the numerator of \( \mu_0(\lambda) \) can be written as

\[1 - 1_S y_S + 1_S (M_{S^c} y_{S^c} - M_{SS} r_S) - \alpha(\lambda) D_{gr} 1_S \eta_S = D\mu_0 - \alpha(\lambda) D_{gr} D_g.
\]

The denominator of \( \mu_0(\lambda) \), by Lemma 1 is formulated as

\[D(\lambda) = D - \alpha(\lambda)D_g^2.
\]
Algorithm 3 HONES Algorithm for time-varying $A$ and fixed $r$

**Inputs:** Initial matrix $A^{(0)}$, vectors $r$, matrix-update-vectors $\{g^{(t)}, t = 1, 2, \ldots \}$.

**Initialization:**

$x \leftarrow$ as the optimum corresponding to $A^{(0)}$;

$S \leftarrow \text{supp}(x)$;

Calculate $(x, \mu, \mu_0)$ via \[3, 10\]

$v_S \leftarrow x_S, v_{S^c} \leftarrow -\mu_{S^c}$;

Calculate intermediate variables $(\text{Par}_1, \text{Par}_2)$ via \[16, 18\] with $g = g^{(1)}$.

**Procedure:**

1: for $t = 1, 2, \ldots$ do
2: \quad $\lambda \leftarrow 0$;
3: \quad while $\lambda < 1$ do
4: \quad \quad $(\lambda^{\text{inc}}, j, S^{\text{new}}) \leftarrow \text{FIND\_LAMBDA}(S, v, \mu_0; \text{Par}_1, \text{Par}_2)$;
5: \quad \quad $\lambda^{\text{inc}} \leftarrow \min\{\lambda^{\text{inc}}, 1 - \lambda\}$;
6: \quad \quad $\lambda \leftarrow \lambda + \lambda^{\text{inc}}$
7: \quad \quad $(v, \mu_0; \text{Par}_1, \text{Par}_2) \leftarrow \text{UPDATE\_BY\_LAMBDA}(\lambda^{\text{inc}}, v, \mu_0; \text{Par}_1, \text{Par}_2)$;
8: \quad \quad if $S^{\text{new}} = S \cup \{j\}$ then
9: \quad \quad \quad $(\text{Par}_1, \text{Par}_2) \leftarrow \text{UPDATE\_EXPAND\_SUPPORT}(\lambda, S, j; r, g^{(t)}, \text{Par}_1, \text{Par}_2)$;
10: \quad \quad else if $S^{\text{new}} = S \setminus \{j\}$ then
11: \quad \quad \quad $(\text{Par}_1, \text{Par}_2) \leftarrow \text{UPDATE\_SHRINK\_SUPPORT}(S, j; r, g^{(t)}, \text{Par}_1, \text{Par}_2)$;
12: \quad end if
13: \quad $S \leftarrow S^{\text{new}}$
14: \quad end while
15: $\text{Par}_2 \leftarrow \text{DIRECT\_UPDATE}(S, r, g^{(t+1)}; \text{Par}_1, \text{Par}_2)$;
16: $A \leftarrow A + g^{(t)}(g^{(t)})^T$;
17: $x_S^{(t)} \leftarrow x_S$, $x_S^{(t)} \leftarrow 0$
18: end for

**Output:** $x^{(1)}, x^{(2)}, \ldots$.

Putting the pieces together results in

$$\mu_0(\lambda) = \frac{D\mu_0 - \alpha(\lambda)DgD\eta}{D - \alpha(\lambda)Dg^2} = \mu_0 + \frac{\alpha(\lambda)}{D - \alpha(\lambda)Dg^2} \cdot Dg(Dg\mu_0 - Dgr).$$

Plug $\mu_0(\lambda)$ into 3, we obtain that

$$x_S(\lambda) = \mu_0(\lambda)\eta_S(\lambda) + A_{SS}(\lambda)^{-1}r_S$$

$$= A_{SS}^{-1}r_S + \alpha(\lambda)Dgr\eta_S + (\mu_0(\lambda) - \mu_0)\eta_S(\lambda) + \mu_0\eta_S(\lambda)$$

$$= x_S + \alpha(\lambda)Dgr\eta_S + (\mu_0(\lambda) - \mu_0)\eta_S(\lambda) + \mu_0\eta_S(\lambda)$$

$$= x_S + \alpha(\lambda)Dgr\eta_S + \frac{\alpha(\lambda)}{D - \alpha(\lambda)Dg^2} \cdot Dg(Dg\mu_0 - Dgr)\eta_S(\lambda) - \mu_0\alpha(\lambda)Dg\eta_S$$

[Use Lemma 1]

$$= x_S + \frac{\alpha(\lambda)}{D - \alpha(\lambda)Dg^2} \cdot Dg(Dg\mu_0 - Dgr)\eta_S(\lambda) - \alpha(\lambda)(Dg\mu_0 - Dgr)\eta_S$$

$$= x_S + \frac{\alpha(\lambda)}{D - \alpha(\lambda)Dg^2} \cdot (Dg\mu_0 - Dgr) \cdot (Dg\eta_S(\lambda) - (D - \alpha(\lambda)Dg^2)\eta_S)$$

$$= x_S + \frac{\alpha(\lambda)}{D - \alpha(\lambda)Dg^2} \cdot (Dg\mu_0 - Dgr) \cdot (Dg\eta_S - \alpha(\lambda)Dg^2\eta_S - (D - \alpha(\lambda)Dg^2)\eta_S)$$

$$= x_S + \frac{\alpha(\lambda)}{D - \alpha(\lambda)Dg^2} \cdot (Dg\mu_0 - Dgr) \cdot (Dg\eta_S - D\eta_S).$$
Similarly, it follows from (9) that
\[ -\mu S^c(\lambda) = \mu_0(\lambda)\tilde{\eta}S^c(\lambda) + r_{S^c} + M_{S^cS}(\lambda)r_S \]
\[ = -\mu S^c - \mu_0\tilde{\eta}S^c + (M_{S^cS}(\lambda) - M_{S^cS})r_S + \mu_0(\lambda)\tilde{\eta}S^c(\lambda) \]
\[ = -\mu S^c - \mu_0\tilde{\eta}S^c + \alpha(\lambda)D_g\eta S^c + \mu_0(\lambda)\tilde{\eta}S^c(\lambda) \]
\[ = -\mu S^c + \alpha(\lambda)D_g\eta S^c + (\mu_0(\lambda) - \mu_0)\tilde{\eta}S^c(\lambda) + \mu_0(\tilde{\eta}S^c(\lambda) - \tilde{\eta}S^c) \]
\[ = -\mu S^c + \alpha(\lambda)D_g\eta S^c + \frac{\alpha(\lambda)}{D - \alpha(\lambda)D_g^2} \cdot D_g(D_g\mu_0 - D_g\tilde{\eta}S^c(\lambda) - \alpha(\lambda)(D_g\mu_0 - D_g)\eta S^c) \]
\[ = -\mu S^c + \frac{\alpha(\lambda)}{D - \alpha(\lambda)D_g^2} \cdot (D_g\mu_0 - D_g)(D_g\tilde{\eta}S^c - \alpha(\lambda)D_g^2\eta S^c - (D - \alpha(\lambda)D_g^2)\eta S^c) \]
\[ = -\mu S^c + \frac{\alpha(\lambda)}{D - \alpha(\lambda)D_g^2} \cdot (D_g\mu_0 - D_g)(D_g\tilde{\eta}S^c - D\eta S^c) \]

In sum,
\[ \begin{pmatrix} x_S(\lambda) \\ -\mu S^c(\lambda) \end{pmatrix} = \begin{pmatrix} x_S \\ -\mu S^c \end{pmatrix} + \frac{\alpha(\lambda)}{D - \alpha(\lambda)D_g^2} \cdot (D_g\mu_0 - D_g) \cdot (D_g\tilde{\eta} - D\eta). \]

\[ \square \]

Theorem 4 indicates that searching for next \( \lambda \) is equivalent to solve \( n \) linear equations. In fact, (22) can be abbreviated as
\[ v(\lambda) = v + \frac{\alpha(\lambda)}{D - \alpha(\lambda)D_g^2} u = \frac{Dv - (D_g^2v - u)\alpha(\lambda)}{D - \alpha(\lambda)D_g^2}, \]
for \( u = (D_g\mu_0 - D_g\tilde{\eta} - D\eta) \). Let
\[ \alpha = \min_+ \left\{ \frac{Dv_i}{D_g^2v_i - u_i} : i = 1, 2, \ldots, n \right\} \]
where \( \min_+ (\Omega) \) denotes the minimum of all positive numbers contained in set \( \Omega \). Then the target \( \lambda \) is the solution of \( \alpha(\lambda) = \alpha \), i.e.
\[ \lambda = \frac{\alpha}{1 - \alpha D_{gg}}. \]

We should emphasize that the right-handed side might be negative if \( \alpha D_{gg} \geq 1 \) in which case \( v \) never hits 0. Thus, we should set \( \lambda \) to be infinity. The implementation of FIND_LAMBDA is stated in Algorithm 4.

**B.4 Variables Update**

**B.4.1 UPDATE_BY_LAMBDA**

Once the next \( \lambda \) has been calculated, all variables can be updated via Lemma 1 and Theorem 2.

**B.4.2 UPDATE_EXPAND_SUPPORT**

Suppose \( S \) is updated to \( S \cup \{j\} \) for some \( j \in S^c \). Denote \( \tilde{S} \) by \( S \cup \{j\} \) and we add a subscript + to each variable to denote the value after update. The key tool is the following formula showing the relation between matrix inverses after adding one row and one column.

**Proposition 5** Let \( \tilde{A}_{jj} = A_{jj} - A_{jS}A_{SS}^{-1}A_{Sj} \),
\[ A_{SS}^{-1} = \begin{pmatrix} A_{SS}^{-1} & 0 \\ 0 & 0 \end{pmatrix} + \frac{1}{\tilde{A}_{jj}} \cdot \begin{pmatrix} -A_{SS}^{-1}A_{Sj} \\ 0 \end{pmatrix} \begin{pmatrix} -A_{SS}^{-1}A_{Sj} & 1 \end{pmatrix}. \]
Algorithm 4 FIND_LAMBDA

Input: Support $S$, iterate $v = \begin{pmatrix} x_S \\ -\mu_S \end{pmatrix}$, $\mu_0$, intermediate variables $\text{Par}_1, \text{Par}_2$.

Procedure:
1: $v \leftarrow (D_g \mu_0 - D_{gr})(D_g \tilde{\eta} - D\eta)$;
2: $\alpha \leftarrow \min \left\{ \frac{D_{gr} \eta}{D_{gr} \mu} : i = 1, 2, \ldots, n \right\}$;
3: $j \leftarrow \arg\min \left\{ \frac{D_{gr} \eta}{D_{gr} \mu} : i = 1, 2, \ldots, n \right\}$;
4: if $\alpha D_{gr} < 1$ then
5: $\lambda^{inc} \leftarrow \frac{\alpha}{1 - \alpha D_{gr}}$;
6: else
7: $\lambda^{inc} \leftarrow \infty$;
8: end if
9: if $j \in S$ then
10: $S^{new} = S \setminus \{j\}$;
11: else
12: $S^{new} = S \cup \{j\}$.
13: end if
Output: $\lambda^{inc}, j, S^{new}$.

Algorithm 5 UPDATE_BY_LAMBDA

Input: Increment $\lambda^{inc}$, iterate $v = \begin{pmatrix} x_S \\ -\mu_S \end{pmatrix}$, $\mu_0$; intermediate variables $\text{Par}_1, \text{Par}_2$.

Procedure:
1: $\alpha_0 \leftarrow 1 + \lambda^{inc} D_{gg}$;
2: $\alpha \leftarrow \lambda^{inc} \cdot \alpha_0$;
3: $\tilde{\alpha} \leftarrow \frac{\alpha}{D - \alpha D_{2g}}$
4: $v \leftarrow v + \tilde{\alpha} \cdot (D_g \mu_0 - D_{gr})(D_g \tilde{\eta} - D\eta)$;
5: $\mu_0 \leftarrow \mu_0 + \tilde{\alpha} \cdot D_g (D_g \mu_0 - D_{gr})$;
6: $D \leftarrow D - \tilde{\alpha} D_{2g}$;
7: $(g, g, g) \leftarrow \alpha_0 (D_g, D_{gg}, D_{gr})$.
8: $M \leftarrow M_s - \alpha \eta S^T$
9: $\tilde{\eta} \leftarrow \tilde{\eta} - \alpha D_{gg} \eta$
10: $\eta \leftarrow \alpha_0 \eta$
Output: $v, \mu_0, \text{Par}_1, \text{Par}_2$.

Similar to section 4.1, the key is to update $M$ and other variables are easy to update based on $M$. Denote a class of operator $\{R_j : j \in S^c\}$ for matrix $W \in \mathbb{R}^{n \times p}$, $R_j(W)$ sets the $j$-th row and $j$-th column of $W$ to be zero and for vector $z \in \mathbb{R}^{n \times 1}$, $R_j(z)$ sets the $j$-th coordinate of $z$ to be zero. One property of $R_j$ to be used is that For any matrix-vector pair $(W, z)$,

$$R_j(W)z = R_j(Wz) - z_j R_j(W_j) \tag{23}$$

where $W_j$ is $j$-th column of $W$.

**Theorem 6** Let $\gamma$ and $\tilde{\gamma}$ be two $n \times 1$ vectors with

$$\gamma_{S} = \tilde{\gamma}_S = (M_j S, 1)^T, \quad \gamma_{S^c} = -A_S e_j - A_{S^c} M_{jS}^T, \quad \tilde{\gamma}_{S^c} = 0.$$

Then

$$M^+ = R_j(M) + \frac{1}{A_{jj}} \cdot \gamma_{S}^T.$$

**Proof** By definition,

$$M_{SS} = A_{SS}^{-1}, \quad M_{SS}^+ = -A_{S} S A_{SS}^{-1}.$$
By Proposition 5, 
\[ M^{+}_{SS} = \begin{pmatrix} M_{SS} & 0 \\ 0 & 0 \end{pmatrix} + \frac{1}{A_{jj}} \begin{pmatrix} M_{jS}^T \\ 1 \end{pmatrix} (M_{jS})^{-1} = \begin{pmatrix} M_{SS} & 0 \\ 0 & 0 \end{pmatrix} + \frac{1}{A_{jj}} \gamma_{S} \gamma_{S}^T, \]

and 
\[ M^{+}_{\tilde{S}c} = -(A_{\tilde{S}c} - A_{Sj}) \left\{ \begin{pmatrix} M_{SS} & 0 \\ 0 & 0 \end{pmatrix} + \frac{1}{A_{jj}} \begin{pmatrix} M_{jS}^T \\ 1 \end{pmatrix} (M_{jS})^{-1} \right\} = \begin{pmatrix} M_{SS} & 0 \\ 0 & 0 \end{pmatrix} + \frac{1}{A_{jj}} \gamma_{S} \gamma_{S}^T. \]

Note that \( M_{\cdot, \tilde{S}} \) is always a zero matrix by definition, the above results imply that 
\[ M^{+} = R_{j}(M) + \frac{1}{A_{jj}} \cdot \gamma \gamma^T. \]

The update of other parameters can be derived as a consequence of Theorem 6. Theorem 7 summarizes the result.

**Theorem 7** Let \( b_{j,S} = -r_{S}^{T} \gamma_{S} \), then 

- \( \eta^{+} = R_{j}(\eta) + \frac{\eta_j}{A_{jj}} \cdot \gamma \);
- \( \tilde{\eta}^{+} = R_{j}(\tilde{\eta}) + \frac{\eta_j}{A_{jj}} \cdot \gamma \);
- \( D^{+} = D + \frac{g^2}{A_{jj}} \);
- \( D_{g}^{+} = D_{g} + \frac{\eta \cdot \tilde{\eta}}{A_{jj}} \);
- \( D_{gg}^{+} = D_{gg} + \frac{g^2}{A_{jj}} \);
- \( D_{gr}^{+} = D_{gr} + \frac{\eta \cdot b_{j,S}}{A_{jj}} \).

**Proof** Since \( M_{j} = 0 \), \( \{23\} \) implies that for any \( z \in \mathbb{R}^{n \times 1} \) 
\[ R_{j}(Mz) = R_{j}(M)z. \]

By definition, 
\[ \eta = \begin{pmatrix} 0 \\ g_{S} \end{pmatrix} + Mg, \quad \tilde{\eta} = \begin{pmatrix} 0 \\ 1_{S} \end{pmatrix} + M1. \]

Also notice that \( \tilde{\gamma}_{S}^{T}g_{S} = g_{j} + M_{jS}^{T}g_{S} = \eta_j \) and \( \gamma_{S}^{T}1_{S} = 1 + M_{jS}1_{S} = \tilde{\eta_j} \), thus, 
\[ \eta^{+} = \begin{pmatrix} 0 \\ g_{S} \end{pmatrix} + M^{+}g = \begin{pmatrix} 0 \\ g_{S} \end{pmatrix} + R_{j}(M)g + \frac{\tilde{\gamma}_{S}^{T}g_{S}}{A_{jj}} \gamma \]
\[ = \begin{pmatrix} 0 \\ g_{S} \end{pmatrix} + R_{j}(M)g + \frac{\tilde{\gamma}_{S}^{T}g_{S}}{A_{jj}} \gamma \]
\[ = \begin{pmatrix} 0 \\ g_{S} \end{pmatrix} + R_{j}(\eta) - R_{j} \left( \begin{pmatrix} 0 \\ g_{S} \end{pmatrix} \right) + \frac{\eta_j}{A_{jj}} \gamma \]
\[ = \begin{pmatrix} 0 \\ g_{S} \end{pmatrix} + R_{j}(\eta) - \begin{pmatrix} 0 \\ g_{S} \end{pmatrix} + \frac{\eta_j}{A_{jj}} \gamma \]
\[ = R_{j}(\eta) + \frac{\eta_j}{A_{jj}} \gamma. \]
The update of \( \tilde{\eta} \) can be obtained by replacing \( g \) by 1 in the above derivation. The four scalars \( D, D_g, D_{gg}, D_{gr} \) can be updated as follows.

\[
\begin{align*}
D^+ &= 1^T_S \tilde{\eta}^+_S = 1^T_S \left( R_j(\tilde{\eta})_S + \frac{\tilde{\eta}}{A_{jj}} \gamma_S \right) = D + \frac{\tilde{\eta}^2}{A_{jj}}; \\
D_g^+ &= 1^T_S \tilde{\eta}^+_S = 1^T_S \left( R_j(\eta)_S + \frac{\eta}{A_{jj}} \gamma_S \right) = D_g + \frac{\eta \tilde{\eta}}{A_{jj}}; \\
D_{gg}^+ &= g_S^T \tilde{\eta}^+_S = g_S^T \left( R_j(\eta)_S + \frac{\eta}{A_{jj}} \gamma_S \right) = D_{gg} + \frac{\eta^2}{A_{jj}}; \\
D_{gr}^+ &= -r_S^T \eta^+_S = -r_S^T \left( R_j(\eta)_S + \frac{\eta}{A_{jj}} \gamma_S \right) = D_{gr} - \frac{\eta \gamma}{A_{jj}} r_S^T \gamma_S \\
&= D_{gr} + \frac{\eta b_j \gamma}{A_{jj}}.
\end{align*}
\]

The implementation of UPDATE_EXPAND_SUPPORT is summarized in Algorithm 6. Note that both \( A_{jj} \) and \( \gamma_S \) depend on \( \lambda \) and it is easy to see that

\[
\tilde{A}_{jj}(\lambda) = A_{jj} + \lambda g^2_j + M_j S (A_{Sj} + \lambda g_j g_S) = A_{jj} + M_j S A_{Sj} + \lambda g_j \eta_j, \\
\gamma_{S^c}(\lambda) = -(A_{S^c} + \lambda g_S g_j) - (A_{S^c} + \lambda g_S g_S^T) M_j S = -A_{S^c} - A_{S^c} M_j S - \lambda \eta_j g_{S^c}.
\]

**Algorithm 6** UPDATE_EXPAND_SUPPORT

**Inputs:** Current \( \lambda \), original support \( S \); new index \( j \), matrix \( A \), vectors \( y, r, g \), intermediate variables \( \text{Par}_1, \text{Par}_2 \).

**Procedure:**
1. \( \tilde{A}_{jj} \leftarrow A_{jj} + M_j S A_{Sj} + \lambda g_j \eta_j; \)
2. \( \gamma_S \leftarrow (M_j S, 1)^T; \gamma_{S^c} \leftarrow -A_{S^c} = -A_{S^c} M_j S - \lambda \eta_j g_{S^c}; \)
3. \( \tilde{\gamma}_S \leftarrow (M_j S, 1)^T; \tilde{\gamma}_{S^c} \leftarrow 0; \)
4. \( b \leftarrow -r_S^T \tilde{\gamma}_S; \)
5. \( D \leftarrow D + \frac{\tilde{\eta}^2}{A_{jj}}; \)
6. \( D_g \leftarrow D_g + \frac{\eta \tilde{\eta}}{A_{jj}}; \)
7. \( D_{gg} \leftarrow D_{gg} + \frac{\eta^2}{A_{jj}}; \)
8. \( D_{gr} \leftarrow D_{gr} + \frac{\eta b_j \tilde{\gamma}}{A_{jj}}; \)
9. \( M_{.S} \leftarrow R_j(M_{.S}) + \frac{1}{A_{jj}} \tilde{\gamma}^T_S; \)
10. \( \tilde{\eta} \leftarrow R_j(\eta) + \frac{\eta}{A_{jj}} \gamma; \)
11. \( \tilde{\eta} \leftarrow R_j(\eta) + \frac{\eta}{A_{jj}} \gamma; \)

**Output:** \( \text{Par}_1, \text{Par}_2 \).

**B.4.3 UPDATE_SHRINK_SUPPORT**

Suppose \( S \) is updated to \( S \setminus \{j\} \) for some \( j \in S^c \). Denote \( \tilde{S} \) by \( S \setminus \{j\} \) and we add a superscript \(-\) to each variable to denote the value after update. Similar to last subsection, we start from deriving \( M^- \) and apply the result to calculate other variables.

**Theorem 8** Let \( \beta \) and \( \tilde{\beta} \) be two \( n \times 1 \) vectors with

\[
\beta^-_S = \tilde{\beta}^-_S = M^-_{Sj}; \quad \beta^-_{S^c} = \begin{pmatrix} -1 \\ M^-_{S^c} \end{pmatrix}, \quad \tilde{\beta}^-_{S^c} = 0.
\]
Then
\[ M^- = R_j(M) - \frac{1}{M_{jj}} \cdot \hat{\beta}^T. \]

**Proof** By definition,
\[
\begin{pmatrix}
M_{S\tilde{S}} & M_{\tilde{S}j} \\
M_{\tilde{S}j} & M_{jj}
\end{pmatrix}
= A^{-1}_{S\tilde{S}} = \begin{pmatrix}
A_{S\tilde{S}} & A_{\tilde{S}j} \\
A_{\tilde{S}j} & A_{jj}
\end{pmatrix}^{-1}.
\]

Then Proposition 5 implies that
\[
\begin{pmatrix}
M_{S\tilde{S}} & M_{\tilde{S}j} \\
M_{\tilde{S}j} & M_{jj}
\end{pmatrix}
= \begin{pmatrix}
A^{-1}_{S\tilde{S}} & 0 \\
0 & 0
\end{pmatrix}
+ \frac{1}{A_{jj} - A_{j\tilde{S}} A^{\text{-1}}_{S\tilde{S}} A_{\tilde{S}j}}
\begin{pmatrix}
-A^{-1}_{S\tilde{S}} A_{\tilde{S}j} \\
A_{j\tilde{S}} A_{S\tilde{S}}^{-1} A_{\tilde{S}j}
\end{pmatrix}
\begin{pmatrix}
-1 \\
1
\end{pmatrix}.
\]

This entails that
\[ A_{S\tilde{S}}^{-1} = M_{S\tilde{S}} - \frac{M_{\tilde{S}j} M_{jj}}{M_{jj}}, \quad A_{S\tilde{S}}^{-1} = \frac{M_{\tilde{S}j}}{M_{jj}}. \] (24)

On the other hand,
\[
\begin{pmatrix}
M_{S\tilde{S}} & M_{\tilde{S}j} \\
M_{\tilde{S}j} & M_{jj}
\end{pmatrix}
= -A_{S\tilde{S}} A_{S\tilde{S}}^{-1} = -(A_{S\tilde{S}} A_{S\tilde{S}}) \begin{pmatrix}
M_{S\tilde{S}} & M_{\tilde{S}j} \\
M_{\tilde{S}j} & M_{jj}
\end{pmatrix}
= -(A_{S\tilde{S}} M_{\tilde{S}j} + A_{S\tilde{S}} M_{\tilde{S}j}) = A_{S\tilde{S}} M_{\tilde{S}j} + A_{S\tilde{S}} M_{\tilde{S}j}. \] (25)

If follows from (24), (25) and (26) that
\[
-A_{S\tilde{S}} A_{S\tilde{S}}^{-1} = -A_{S\tilde{S}} \left( M_{S\tilde{S}} - \frac{M_{\tilde{S}j} M_{jj}}{M_{jj}} \right)
= M_{S\tilde{S}} + A_{S\tilde{S}} M_{\tilde{S}j} + \frac{A_{S\tilde{S}} M_{\tilde{S}j} M_{jj}}{M_{jj}}
= M_{S\tilde{S}} + (A_{S\tilde{S}} M_{\tilde{S}j} + A_{S\tilde{S}} M_{\tilde{S}j}) \frac{M_{\tilde{S}j}}{M_{jj}}
= M_{S\tilde{S}} - \frac{M_{S\tilde{S}} M_{\tilde{S}j}}{M_{jj}}. \] (27)

Putting (25) and (26) together, we obtain that
\[
M_{S\tilde{S}} = \begin{pmatrix}
A_{S\tilde{S}}^{-1} \\
-A_{S\tilde{S}} A_{S\tilde{S}}^{-1}
\end{pmatrix}
= \begin{pmatrix}
A_{S\tilde{S}}^{-1} \\
-A_{S\tilde{S}} A_{S\tilde{S}}^{-1}
\end{pmatrix}
= R_j(M)_{.,S} - \frac{1}{M_{jj}} \cdot \hat{\beta}^T.
\]

Since \(M_{,S}\) is a zero matrix,
\[ M^- = R_j(M) - \frac{1}{M_{jj}} \cdot \hat{\beta}^T. \]

**Theorem 9** Let \( \beta_{j,S} = -r_j^T \beta - r_j M_{jj} \), then
\[
\begin{array}{l}
\eta^- = R_j(\eta) - \frac{\eta}{M_{jj}} \beta; \\
\tilde{\eta}^- = R_j(\tilde{\eta}) - \frac{\tilde{\eta}}{M_{jj}} \beta; \\
D^- = D - \frac{\delta^2}{M_{jj}}; \\
D_g^- = D_g - \frac{\delta^2}{M_{jj}}; \\
D_{gg}^- = D_{gg} - \frac{\delta^2}{M_{jj}};
\end{array}
\]

\]

22
Algorithm 7 UPDATE_SHRINK_SUPPORT

Inputs: Original support \( S \), new index \( j \), matrix \( A \), vector \( y, r, g \), intermediate variables \( \text{Par}_1, \text{Par}_2 \).

Procedure:

1: \( \beta_S \leftarrow M_j^T \beta_S, \beta_{Sc} \leftarrow \left( \frac{-1}{M_{S,j}} \right), \beta_{Sc} \leftarrow M_j^T \beta_{Sc}, \beta_{Sc} \leftarrow -1 \);
2: \( \tilde{b} \leftarrow -r_j^T \beta_S - r_j M_{jj} \);
3: \( D \leftarrow D - \frac{\eta_j^2}{M_{jj}} \);
4: \( D_g \leftarrow D_g - \frac{\eta_j g_j J}{M_{jj}} \);
5: \( D_{gg} \leftarrow D_{gg} - \frac{\eta_j^2}{M_{jj}} \);
6: \( D_{gr} \leftarrow D_{gr} - \frac{\eta_j b_j J}{M_{jj}} \);
7: \( M \cdot \tilde{S} \leftarrow R_j (M \cdot \tilde{S}) - \frac{1}{M_{jj}} \beta_S^T \beta_S, M \cdot j \leftarrow 0 \);
8: \( \eta \leftarrow R_j (\eta) - \frac{\eta_j^2}{M_{jj}} \beta \);
9: \( \tilde{\eta} \leftarrow R_j (\tilde{\eta}) - \frac{\eta_j}{M_{jj}} \beta \).

Output: \( \text{Par}_1, \text{Par}_2 \).

\[ D_{gr} = D_{gr} - \frac{\eta_j b_j J}{M_{jj}}. \]

Proof By [23].

\[ R_j (M) g = R_j (M g) - g_j R_j (M \cdot j) \]

Let \( e_j \) is the \( j \)-th basis vector with \( j \)-th entry equal to 1 and all other entries equal to 0. Then

\[ \eta^- = \begin{pmatrix} 0 \\ g_{Sc} \end{pmatrix} + M^{-1} g = \begin{pmatrix} 0 \\ g_{Sc} \end{pmatrix} + R_j (M) g - \frac{\tilde{\beta}^T g}{M_{jj}} \beta \]

\[ = \begin{pmatrix} 0 \\ g_{Sc} \end{pmatrix} + R_j (M g) - g_j R_j (M \cdot j) - \frac{\tilde{\beta}^T g}{M_{jj}} \beta \]

\[ = \begin{pmatrix} 0 \\ g_{Sc} \end{pmatrix} + R_j (M g) - g_j e_j - g_j \beta - \frac{\tilde{\beta}^T g}{M_{jj}} \beta \]

\[ = \begin{pmatrix} 0 \\ g_{Sc} \end{pmatrix} + R_j (M g) - g_j e_j - \frac{M_j g_{Sc}}{M_{jj}} \beta \]

\[ = \begin{pmatrix} 0 \\ g_{Sc} \end{pmatrix} + R_j (\eta) - \frac{M_j g_{Sc}}{M_{jj}} \beta \]

\[ = \begin{pmatrix} 0 \\ g_{Sc} \end{pmatrix} + R_j (\eta) - g_j e_j - \frac{\eta_j}{M_{jj}} \beta \]

\[ = R_j (\eta) - \frac{\eta_j}{M_{jj}} \beta. \]
Although agnostic to \( S \) words, we update \( A \) where \( S \) is sparse and the number of turning points is small. Fortunately, UPDATE_EXPAND_SUPPORT is only required the sub-routine which extracts information from \( S \).

The implementation of UPDATE_SHRINK_SUPPORT is summarized in Algorithm 8.

### B.4.4 DIRECT_UPDATE

At the beginning of each time \( t \), we need to recompute \( \text{Par}_2 = \{ \eta, D_g, D_{gg}, D_{gr} \} \). The implementation is summarized in Algorithm 8.

**Algorithm 8 DIRECT_UPDATE**

**Inputs:** Support \( S \), vector \( y, r, g \), intermediate variables \( \text{Par}_1, \text{Par}_2 \).

**Procedure:**
1. \( \eta_S \leftarrow M_{ssg_S}, \eta_S^* \leftarrow gs^* + M_{s^*sg_S} \);
2. \( D_g \leftarrow 1^T_S \eta_S \);
3. \( D_{gg} \leftarrow \eta_S^T gs \);
4. \( D_{gr} \leftarrow -\eta_S r_S \).

**Output:** \( \text{Par}_2 \).

### B.5 Update of \( A \)

As will be shown in next subsection, the complexities of all above sub-routines are at most \( O(ns) \) where \( s = |S| \). However, the complexity of line 16 is \( O(n^2) \) which might dominate when the solution is sparse and the number of turning points is small. Fortunately, UPDATE_EXPAND_SUPPORT is the only sub-routine which extracts information from \( A \). In fact, in line 1 and line 2,

\[
\begin{pmatrix}
\tilde{A}_{jj} \\
\gamma_{S^*}
\end{pmatrix} = \begin{pmatrix}
A_{jj} + M_{js} S j \\
- A_{S^*j} - A_{S^*S} M_{jS}^T
\end{pmatrix} + \lambda \eta_j \begin{pmatrix}
g_j \\
g_{S^*}
\end{pmatrix}.
\]

This only requires the \( j \)-th column of \( A \). Let \( S_a \) be the union of all supports appeared in Algorithm 8. Suppose we know \( S_a \) apriori, we can only update the columns of \( A \) with indices in \( S_a \). In other words, we update \( A_{.,S_a} \) by \( A_{.,S_a} + \lambda g g_{S_a}^T \) at the beginning of each step and hence the complexity is reduced to \( O(n|S_a|) \).

Although agnostic to \( S_a \), we can do this in reality, we can initialize it by \( \text{supp}(x_k) \) for some positive \( k \), e.g., \( k = 1 \), and keep track by adding index into \( S_a \) once the index is not included in \( S_a \). Once a new index \( j \)
Algorithm 9 HONES Algorithm for time-varying $A$ and fixed $r$ with sparse update of $A$

**Inputs:** Initial matrix $A^{(0)}$, vectors $r$, matrix-update-vectors $\{g^{(t)}, t = 1, 2, \ldots \}$.

**Initialization:**

$x \leftarrow$ as the optimum corresponding to $A^{(0)}$;
$S \leftarrow \text{supp}(x)$, $S_0 \leftarrow S$;
Calculate $(x, \mu, \mu_0)$ via (5-10);
$v_S \leftarrow x_S$, $v_{S^C} \leftarrow -\mu_S$;
Calculate intermediate variables $(\text{Par}_1, \text{Par}_2)$ via (16-18) based on $g^{(1)}$.

**Procedure:**

1: for $t = 1, 2, \ldots$ do
2: $\lambda \leftarrow 0$;
3: while $\lambda < 1$ do
4: $(\lambda_{\text{inc}}, j, S_{\text{new}}) \leftarrow \text{FIND}\_\text{LAMBDA}(S, v; \text{Par}_1, \text{Par}_2)$;
5: $\lambda_{\text{inc}} \leftarrow \min\{\lambda_{\text{inc}}, 1 - \lambda\}$;
6: $\lambda \leftarrow \lambda + \lambda_{\text{inc}}$;
7: $(v, \mu_0; \text{Par}_1, \text{Par}_2) \leftarrow \text{UPDATE}\_\text{BY}\_\text{LAMBDA}(\lambda_{\text{inc}}; v, \mu_0; \text{Par}_1, \text{Par}_2)$;
8: if $S_{\text{new}} = S \cup \{j\}$ then
9: $(\text{Par}_1, \text{Par}_2) \leftarrow \text{UPDATE}\_\text{EXPAND}\_\text{SUPPORT}(\lambda, S, j; A, r, g^{(t)}; \text{Par}_1, \text{Par}_2)$;
10: if $j \notin S$, then
11: $G \leftarrow (g^{(1)}, \ldots, g^{(t-1)})$;
12: $A_{\cdot j} \leftarrow A_{\cdot j} + GG_T$;
13: $S_s = S_s \cup \{j\}$;
14: end if
15: else if $S_{\text{new}} = S \setminus \{j\}$ then
16: $(\text{Par}_1, \text{Par}_2) \leftarrow \text{UPDATE}\_\text{SHRINK}\_\text{SUPPORT}(S, j; r, g^{(t)}; \text{Par}_1, \text{Par}_2)$;
17: end if
18: $S \leftarrow S_{\text{new}}$;
19: end while
20: $\text{Par}_2 \leftarrow \text{DIRECT}\_\text{UPDATE}(S, r, g^{(t+1)}; \text{Par}_1, \text{Par}_2)$;
21: $A_{\cdot S_s} \leftarrow A_{\cdot S_s} + g^{(t)}(g^{(t)}_S)^T$;
22: $x_S^{(t)} \leftarrow x_S$, $x_{S^C}^{(t)} \leftarrow 0$.
23: end for

**Output:** $x^{(1)}, x^{(2)}, \ldots$.

### B.6 Complexity Analysis

In this subsection, we analyze the complexity of the algorithm. We distinguish four types of computation, namely matrix-vector product, outer-product of two vectors, inner-product of two vectors and vector-scalar product. Denote by $W \in \mathbb{R}^{m \times p}$, $(z, \bar{z}) \in \mathbb{R}^{p \times p}$ and $a \in \mathbb{R}$ the generic matrix, vector and scalar respectively. As a convention, the complexity is defined as the number of scalar-multiplications. The addition is omitted here for simplicity. Note that the complexities of $Wz, z^T \bar{z}$ and $az$ are $mp$, $pq$, $p$ and $p$, respectively. The results for a single step are summarized in Table 5, where $s_* = |S_*|$ be the size of $S_*$ at the final round. We should emphasize that our complexity analysis is exact.

For given $t$, denote $k_A^+$ by the number of turning points which add element to $S$ and $k_A^-$ by the number of turning points which delete element from $S$. Let $k_A = k_A^+ + k_A^-$ be the total number of turning points and $s$ be the maximum size of $S$ in the iteration. Then the complexity of HONES for a single time $t$ is at most

$$ns(3k_A^+ + 2k_A^-) + n(12k_A^+ + 10k_A^-) + O(k_A),$$

Therefore, the complexity at time $t$ is at most

$$C_t \leq ns_* + ns(3k_A^+ + 2k_A^- + 1) + n(12k_A^+ + 10k_A^- + 2) + O(k_A) \leq ns_* + ns(3k_A + 1) + n(12k_A + 2) + O(k_A).$$
Table 5: Computation complexity of each sub-routine in Algorithm 3

| Sub-routine                  | (Wz)-type | (z\tilde{z})^T-type | (z^Tz)-type | (az)-type |
|-----------------------------|-----------|----------------------|-------------|----------|
| FIND_LAMBDA                 | 0         | 0                    | n           | 2n       |
| UPDATE_BY_LAMBDA            | 0         | ns                   | 0           | 4n       |
| UPDATE_EXPAND_SUPPORT       | s(n - s - 1) | n(s + 1)         | n           | 2n       |
| UPDATE_SHRINK_SUPPORT       | 0         | n(s - 1)             | n           | 2n       |
| DIRECT_UPDATE               | ns        | 0                    | n + s       | 0        |
| Update of A                | 0         | ns_s                 | 0           | 0        |

C Implementation of HONES Algorithm With Time-Varying \(r\) and Fixed \(A\)

C.1 Intermediate Variables

Similar to Appendix B, we define \(\text{Par}_1 = \{M, \tilde{\eta}, D\}\) where the parameters are defined in (16)-(18). Moreover, we define a vector \(\xi\) such that

\[
\xi_S = -A_{S}^{-1}\ell_S, \quad \xi_{S^c} = -\ell_{S^c} + A_{S^c}A_S^{-1}\ell_S,
\]

and a scalar \(D_\ell\) as

\[
D_\ell = 1_S^T\xi_S.
\]

We write \(\text{Par}_3\) for \(\{\xi, D_\ell\}\) for convenience.

C.2 Implementation

Algorithm 10 describes the full implementation in this case and the sub-routines will be discussed separately in following subsections.

Algorithm 10 HONES Algorithm for constant \(A\), \(y\) and time-varying \(r\)

**Inputs:** Initial matrix \(A\), vector \(r^{(0)}\), vector-update-vector \(\{\ell^{(t)} = r^{(t)} - r^{(t-1)} : t = 1, 2, \ldots\}\).

**Initialization:**

1. \(x \leftarrow \) as the optimum corresponding to \(r^{(0)}\).
2. \(S \leftarrow \text{supp}(x)\);
3. \(v_S \leftarrow x_S, v_{S^c} \leftarrow -\mu_{S^c}\);
4. Calculate intermediate variables \(\text{Par}_1\), \(\text{Par}_3\) via (16)-(18) based on \(\ell^{(1)}\).

**Procedure:**

1. for \(t = 1, 2, \ldots\) do
2. \(\lambda \leftarrow 0\);
3. while \(\lambda < 1\) do
4. \((\lambda_{\text{inc}}, j, S^{\text{new}}) \leftarrow \text{FIND UTILDE LAMBDA}(S, v; \text{Par}_1, \text{Par}_3)\);
5. \(\lambda_{\text{inc}} \leftarrow \min\{\lambda_{\text{inc}}, 1 - \lambda\}\);
6. \((v, \mu_0, \text{Par}_1, \text{Par}_3) \leftarrow \text{UPDATE UTILDE LAMBDA}(\lambda_{\text{inc}}; v, \mu_0, \text{Par}_1, \text{Par}_3)\);
7. if \(S^{\text{new}} = S \cup \{j\}\) then
8. \((\text{Par}_1, \text{Par}_3) \leftarrow \text{UPDATE UTILDE EXPAND SUPPORT}(S, j, A, \ell; \text{Par}_1, \text{Par}_3)\);
9. else if \(S^{\text{new}} = S \setminus \{j\}\) then
10. \((\text{Par}_1, \text{Par}_3) \leftarrow \text{UPDATE UTILDE SHRINK SUPPORT}(S, j, \ell; \text{Par}_1, \text{Par}_3)\);
11. end if
12. \(S \leftarrow S^{\text{new}}\);
13. \(\lambda \leftarrow \lambda + \lambda_{\text{inc}}\).
14. end while
15. \(\text{Par}_3 \leftarrow \text{DIRECT UTILDE UPDATE}(S, \text{Par}_1, h^{(t+1)})\);
16. \(x_S^{(t)} \leftarrow x_S, \quad x_{S^c}^{(t)} \leftarrow 0\).
17. end for

**Output:** \(x^{(1)}, x^{(2)}, \ldots\)
C.3 FIND_UTILDE_LAMBDA

Define $v$ as in (20). Then Theorem 1 implies that

$$v(\lambda) = v(0) - \left( \xi - \frac{D}{D^T} \tilde{\eta} \right) \lambda, \quad \mu(\lambda) = \mu_0 + \frac{D}{D^T} \lambda.$$  

Thus, searching for $\lambda$ is equivalent to solve simple linear equations. Algorithm 11

**Algorithm 11 FIND_UTILDE_LAMBDA**

**Input:** Support $S$, iterate $v = \left( \begin{array}{c} x_S \\ -\mu_{S^c} \end{array} \right)$, intermediate variables $\text{Par}_1, \text{Par}_3$.

**Procedure:**

1. $\lambda^{\text{inc}} \leftarrow \min_+ \left\{ \frac{v_i}{\xi_i - \frac{D}{D^T} \tilde{\eta}_i} : i = 1, 2, \ldots, n \right\}$;
2. $j \leftarrow \arg\min_+ \left\{ \frac{v_i}{\xi_i - \frac{D}{D^T} \tilde{\eta}_i} : i = 1, 2, \ldots, n \right\}$;
3. if $j \in S$ then
   4. $S^{\text{new}} = S \setminus \{j\}$;
   5. else
   6. $S^{\text{new}} = S \cup \{j\}$.
7. **end if**

**Output:** $\lambda^{\text{inc}}, j, S^{\text{new}}$.

C.4 Variables Update

C.4.1 UPDATE_BY_UTILDE_LAMBDA

Note that all intermediate variables are not affected by $\lambda$, we only need to update $v$ and $\mu_0$ accordingly.

**Algorithm 12 UPDATE_BY_UTILDE_LAMBDA**

**Input:** Increment $\lambda^{\text{inc}}$, iterate $v = \left( \begin{array}{c} x_S \\ -\mu_{S^c} \end{array} \right)$, $\mu_0$; intermediate variables $\text{Par}_1, \text{Par}_3$.

**Procedure:**

1. $v \leftarrow v - \left( \xi - \frac{D}{D^T} \tilde{\eta} \right) \lambda^{\text{inc}}$;
2. $\mu_0 \leftarrow \mu_0 + \frac{D}{D^T} \lambda^{\text{inc}}$.

**Output:** $v, \mu_0$.

C.4.2 UPDATE_UTILDE_EXPAND_SUPPORT

Since $M$ is exactly the same as in Appendix B, we can directly apply Theorem 6 to obtain an update of $M$ and the updates of other parameters as a consequence.

**Theorem 10** Let $\gamma$ and $\tilde{\gamma}$ be defined in Theorem 6, i.e.

$$\gamma_S = \tilde{\gamma}_S = (M_j S^T, 1)^T, \quad \gamma_{S^c} = -A_{S^c j} - A_{S^c S} M_j^T, \quad \tilde{\gamma}_{S^c} = 0,$$

then

- $M^+ = R_j(M) + \frac{1}{\lambda_{jj}} \gamma_j^T$;
- $\tilde{\eta}^+ = R_j(\tilde{\eta}) + \frac{\tilde{\eta}_j}{\lambda_{jj}} \gamma$;
- $D^+ = D + \frac{\gamma_j^2}{\lambda_{jj}}$;
\begin{itemize}
  \item \( \xi^+ = R_j(\xi) + \frac{\xi_j}{A_{jj}} \gamma; \)
  \item \( D^+ = D_\ell + \frac{\xi_j \bar{\eta}}{A_{jj}} \)
\end{itemize}

**Proof** The update of \( M, \bar{\eta} \) and \( D \) has been proved in Theorem 6. For any subset \( S \), let \( I_S \) denote the matrix with \( j \)-th diagonal element equal to 1 for any \( j \in S \) and all other elements equal to 0. Then \( \xi \) and \( \xi^+ \) can be rewritten as

\[
\xi = -(M + I_S) \ell, \quad \xi^+ = -(M^+ + I_{\bar{S}}) \ell.
\]

Note that \( I_{S^c} - I_{\bar{S}} = e_j e_j^T \) where \( e_j \) is the \( j \)-th basis vector, then we have

\[
\xi^+ - \xi = (M - M^+ + e_j e_j^T) \ell = \left( M - R_j(M) - \frac{1}{A_{jj}} \gamma \bar{\gamma}^T + e_j e_j^T \right) \ell = -\frac{\bar{\gamma}^T \ell}{A_{jj}} \gamma + (\ell_j - M_j S^c) e_j.
\]

\[
\implies \xi^+ = \xi - \frac{\bar{\gamma}^T \ell}{A_{jj}} \gamma + (\ell_j - M_j S^c) e_j = R_j(\xi) - \frac{\bar{\gamma}^T \ell}{A_{jj}} \gamma.
\]

Note that \( \bar{\gamma}^T \ell = \ell_j + M_j S \ell_S = -\xi_j \), we obtain that

\[
\xi^+ = R_j(\xi) + \frac{\xi_j}{A_{jj}} \gamma.
\]

For \( D^+ \), we have

\[
D^+ = 1_S^T \xi^+_S = D_\ell + \frac{\xi_j}{A_{jj}} \cdot 1_S^T \gamma_S = D_\ell + \frac{\xi_j \bar{\eta}}{A_{jj}}.
\]

The implementation of \textit{UPDATE_TILDE_EXPAND_SUPPORT} is summarized in Algorithm 13.

**Algorithm 13 UPDATE_TILDE_EXPAND_SUPPORT**

**Inputs:** Original support \( S \), new index \( j \), matrix \( A \), vector \( \ell \), intermediate variables \( \text{Par}_1, \text{Par}_3 \).

**Procedure:**

1. \( A_{jj} \leftarrow A_{jj} + M_j S A_{Sj}; \)
2. \( \gamma_S \leftarrow (M_j S, 1)^T, \gamma_{S^c} \leftarrow -A_{S^c j} - A_{S^c S} M_j^T; \)
3. \( \bar{\gamma}_S \leftarrow (M_j S, 1)^T, \bar{\gamma}_{S^c} \leftarrow 0; \)
4. \( D \leftarrow D + \frac{\bar{\gamma}_j}{A_{jj}}; \)
5. \( D_\ell \leftarrow D_\ell + \frac{\xi_j \bar{\eta}}{A_{jj}}; \)
6. \( \xi \leftarrow R_j(\xi) + \frac{\bar{\gamma}_j}{A_{jj}} \gamma; \)
7. \( \bar{\eta} \leftarrow R_j(\bar{\eta}) + \frac{\bar{\gamma}_j}{A_{jj}} \gamma; \)
8. \( M, \bar{\gamma} \leftarrow R_j(M, \bar{\gamma}) + \frac{1}{A_{jj}} \bar{\gamma}^T \bar{\gamma}_S. \)

**Output:** \( \text{Par}_1, \text{Par}_3. \)

**C.4.3 UPDATE_TILDE_SHRINK_SUPPORT**

Since \( M \) is exactly the same as in Appendix B, we can directly apply Theorem 8 to obtain an update of \( M \) and the updates of other parameters as a consequence.

**Theorem 11** Let \( \beta \) and \( \tilde{\beta} \) be defined in Theorem 8, i.e.

\[
\beta_S = \tilde{\beta}_S = M_{Sj}, \quad \beta_{S^c} = \left( \begin{array}{c} -1 \\ M_{S^c j} \end{array} \right), \quad \tilde{\beta}_{S^c} = 0,
\]

then
• $M^- = R_j(M) - \frac{1}{M_{jj}} \cdot \beta^T$;
• $\bar{\eta}^- = R_j(\bar{\eta}) - \frac{\bar{\eta}j}{M_{jj}} \beta$;
• $D^- = D - \frac{\bar{\eta}j}{M_{jj}}$;
• $\xi^- = R_j(\xi) - \frac{\xi j}{M_{jj}} \beta$;
• $D^- = D_h - \frac{\xi j}{M_{jj}}$.

Proof The update of $M$, $\bar{\eta}$ and $D$ has been proved in Theorem 8 and Theorem 9. For any subset $S$, let $I_S$ denote the matrix with $j$-th diagonal element equal to 1 for any $j \in S$ and all other elements equal to 0. Then $\xi$ and $\xi^-$ can be rewritten as

$$\xi = -(M + I_{Sc}) \ell, \quad \xi^- = -(M^- + I_{Sc}) \ell.$$

Note that $I_{Sc} - I_{Sc} = e_j e_j^T$ where $e_j$ is the $j$-th basis vector, then we have

$$\xi^- - \xi = (M - M^- - e_j e_j^T) \ell = \left( \frac{1}{M_{jj}} \beta^T + M - R_j(M) - e_j e_j^T \right) \ell = \frac{\beta^T \ell}{M_{jj}} \beta + (M - R_j(M) - e_j e_j^T) \ell = \frac{\beta^T \ell}{M_{jj}} \beta + \bar{\xi}. $$

By definition of $\bar{\xi}$

$$\bar{\xi}_S = \ell_j M_{Sj}, \quad \bar{\xi}_j = M_{jS} \ell_S + \ell_j M_{jj} - \ell_j = -\xi_j - \ell_j, \quad \bar{\xi}_{Sc} = \ell_j M_{Scj},$$

and thus,

$$\bar{\xi} = \ell_j \beta - \xi_j e_j.$$ 

This implies that

$$\xi^- = \xi - \xi_j e_j + \frac{\beta^T \ell_S}{M_{jj}} \beta + \ell_j M_{jj} = R_j(\xi) - \frac{\xi_j}{M_{jj}} \beta.$$ 

For $D^-$, we have

$$D^- = 1^T_S \bar{\xi}^- = D - \xi_j - \frac{\xi_j}{M_{jj}} \cdot 1^T_S \beta_S = D - \frac{\xi_j (1^T_S \beta_S + M_{jj})}{M_{jj}} = D - \frac{\xi_j \bar{\eta}_j}{M_{jj}},$$

The implementation of UPDATE_TILDE_SHRINK_SUPPORT is summarized in Algorithm 14.

Algorithm 14 UPDATE_TILDE_SHRINK_SUPPORT

Inputs: Original support $S$, new index $j$, vector $\ell$, intermediate variables Par1, Par3.

Procedure:
1: $\beta_S \leftarrow M^T_{jS}, \beta_{Sc} \leftarrow \left( -\frac{1}{M_{Sj}} \right), \bar{\beta}_S \leftarrow M^T_{jS}, \bar{\beta}_{Sc} \leftarrow 0$;
2: $D \leftarrow D - \frac{\bar{\eta}j}{M_{jj}}$;
3: $D_\ell \leftarrow D_\ell - \frac{\xi_j}{M_{jj}}$;
4: $\xi \leftarrow R_j(\xi) - \frac{\xi_j}{M_{jj}} \beta$;
5: $\bar{\eta} \leftarrow R_j(\bar{\eta}) - \frac{\bar{\eta}j}{M_{jj}} \beta$;
6: $M_{.,S} \leftarrow R_j(M_{.,S}) - \frac{1}{M_{jj}} \beta^T \bar{\beta}_S, M_{.,j} \leftarrow 0$.

Output: Par1, Par3.
Algorithm 15 DIRECT_UTILDE_UPDATE

Inputs: Support $S$, vector-update-vector $\ell$, intermediate variables $M$.

Procedure:
1. $\xi_S \leftarrow -M_S S \ell_S$, $\xi_S^c \leftarrow -\ell_S^c - M_S^c S \ell_S$;
2. $D_\ell \leftarrow T_S^T \xi_S$.

Output: $\xi, D_\ell$.

C.4.4 DIRECT_UTILDE_UPDATE

At the beginning of each time $t$, we need to recompute $\xi$ and $D_\ell$. The implementation is summarized in Algorithm 15.

C.5 Complexity Analysis

Similar to Appendix B, we can analyze the computation complexity. The analysis here is much simpler than the last case since the implementation is quite straightforward. Table 6 summarizes the results.

| Sub-routine                                | $(Wz)$-type | $(z^Tz)$-type | $(z^Tz)$-type | $(az)$-type |
|--------------------------------------------|-------------|---------------|---------------|-------------|
| FIND_UTILDE_LAMBDA                         | 0           | 0             | 2$n$          | 0           |
| UPDATE_BY_UTILDE_LAMBDA                   | 0           | 0             | 0             | $n$         |
| UPDATE_UTILDE_EXPAND_SUPPORT              | $(n - s - 1)s$ | $n(s + 1)$    | $s$           | $2n$        |
| UPDATE_UTILDE_SHRINK_SUPPORT              | 0           | $n(s - 1)$    | 0             | $2n$        |
| DIRECT_UTILDE_UPDATE                      | $ns$        | 0             | $s$           | 0           |

Let $k^+_r$ and $k^-_r$ be the number of turning points that $S$ is expanded and shrunk respectively and $k_r = k^+_r + k^-_r$ be the total number of tuning points, then the complexity is

$$C_{2t} \leq ns + n + 3nk_r + n(2s + 3)k^+_r + n(s + 1)k^-_r + O(k_r) = ns(2k_r + 1) + n(6k_r + 1) + O(k_r).$$

D Implementation of HONES Algorithm With Time-Varying $A, r$

D.1 Intermediate Variables

Based on the results in Appendix B and Appendix C, we can concatenate Algorithm 3 and Algorithm 10. Thus we define Par$_1, Par_2, Par_3$ as Par$_1 = \{M, \hat{\eta}, D\},$ Par$_2 = \{\eta, D_g, D_{gg}, D_{gr}\},$ Par$_3 = \{\xi, D_\ell\}$ where all parameters are defined in previous appendices.

D.2 Implementation

Note that only two sub-routines involves the matrix $A$, namely UPDATE_EXPAND_SUPPORT and UPDATE_UTILDE_EXPAND_SUPPORT, and moreover they only involve the $j$-th column of $A$. Thus, we can use the sparse update of $A$ as in Algorithm 9 for acceleration. Algorithm 16 below describes the implementation.

D.3 Complexity Analysis

The complexity of Algorithm 16 is just the sum of that of Algorithm 3 and Algorithm 10, i.e.

$$C_t = C_{1t} + C_{2t} = ns_s + ns(3k_A + 2k_r + 2) + n(12k_A + 6k_r + 3) + O(k_A + k_r).$$
Algorithm 16 HONES Algorithm for time-varying $A$, $r$ with sparse update of $A$

Inputs: Initial parameters $A^{(0)}$, vectors \{r^{(t)} : t = 1, 2, \ldots\}, matrix-update-vectors \{g^{(t)} : t = 1, 2, \ldots\}.

Initialization:

\begin{itemize}
\item $x \leftarrow \text{as the optimum corresponding to } A^{(0)}, r^{(0)}.$
\item $S \leftarrow \text{supp}(x), S_* \leftarrow S$;
\item \text{Calculate } (x, \mu, \mu_0) \text{ via (8)-(10)};
\item $v \leftarrow (x_S, -\mu_S)$;
\item \text{Calculate intermediate variables } (\text{Par}_1, \text{Par}_2) \text{ via (16)-(18) based on } r^{(0)}, g^{(1)};
\end{itemize}

Procedure:

\begin{enumerate}
\item for $t = 1, 2, \ldots$ do
\item $\lambda \leftarrow 0$;
\item while $\lambda < 1$ do
\item $(\lambda_{\text{inc}}, j, S_{\text{new}}) \leftarrow \text{FIND\_LAMBDA}(S, v; \text{Par}_1, \text{Par}_2);$\item $\lambda_{\text{inc}} \leftarrow \min\{\lambda_{\text{inc}}, 1 - \lambda\}$;
\item $\lambda \leftarrow \lambda + \lambda_{\text{inc}}$;
\item $(v, \mu_0; \text{Par}_1, \text{Par}_2) \leftarrow \text{UPDATE\_BY\_LAMBDA}(\lambda_{\text{inc}}; v, \mu_0, \text{Par}_1, \text{Par}_2)$;
\item if $S_{\text{new}} = S \cup \{j\}$ then
\item $(\text{Par}_1, \text{Par}_2) \leftarrow \text{UPDATE\_EXPAND\_SUPPORT}(\lambda, S, j; A, r^{(t-1)}, g^{(t)}, \text{Par}_1, \text{Par}_2)$
\item if $j \notin S$, then
\item $G \leftarrow (g^{(1)}, \ldots, g^{(t-1)})$;
\item $A_{.,j} \leftarrow A_{.,j} + GG_{j,.;}$
\item $S_* = S_* \cup \{j\}$;
\item end if
\item end if
\item end if
\item $(\text{Par}_1, \text{Par}_2) \leftarrow \text{UPDATE\_SHRINK\_SUPPORT}(S, j; r^{(t-1)}, g^{(t)}, \text{Par}_1, \text{Par}_2)$
\item end if
\item $S \leftarrow S_{\text{new}}$;
\item end while
\item $(\lambda_{\text{inc}}, j, S_{\text{new}}) \leftarrow \text{FIND\_UTILDE\_UPDATE}(v, \text{Par}_1, \text{Par}_3)$;
\item $\lambda_{\text{inc}} \leftarrow \min\{\lambda_{\text{inc}}, 1 - \lambda\}$;
\item $(v, \mu_0) \leftarrow \text{UPDATE\_BY\_UTILDE\_LAMBDA}(\lambda_{\text{inc}}; v, \mu_0, \text{Par}_1, \text{Par}_3)$;
\item if $S_{\text{new}} = S \cup \{j\}$ then
\item $(\text{Par}_1, \text{Par}_3) \leftarrow \text{UPDATE\_UTILDE\_EXPAND\_SUPPORT}(S, j, A, \ell^{(t)}; \text{Par}_1, \text{Par}_3)$
\item if $j \notin S$, then
\item $G \leftarrow (g^{(1)}, \ell^{(t)}; \text{dot}_s, g^{(t-1)})$
\item $A_{.,j} \leftarrow A_{.,j} + GG_{j,.;}$
\item $S_* = S_* \cup \{j\}$;
\item end if
\item end if
\item $(\text{Par}_1, \text{Par}_3) \leftarrow \text{UPDATE\_UTILDE\_SHRINK\_SUPPORT}(S, j, \ell^{(t)}; \text{Par}_1, \text{Par}_3)$
\item end if
\item $S \leftarrow S_{\text{new}}$;
\item $\lambda \leftarrow \lambda + \lambda_{\text{inc}}$;
\item end while
\item $\text{Par}_2 \leftarrow \text{DIRECT\_UPDATE}(S, r^{(t)}; g^{(t+1)}; \text{Par}_1, \text{Par}_2)$;
\item $x_{S}^{(t)} \leftarrow x_S$, $x_{S_*}^{(t)} \leftarrow 0$.
\item end for
\end{enumerate}

Output: $x^{(1)}, x^{(2)}, \ldots$, 31