Remark on Algorithm 1012: Computing projections with large data sets

TYLER H. CHANG, Argonne National Laboratory, USA
LAYNE T. WATSON, Virginia Polytechnic Institute and State University, USA
SVEN LEYFFER, Argonne National Laboratory, USA
THOMAS C. H. LUX, Meta, USA
HUSSAIN ALMOHRI, Kuwait University, Kuwait

In ACM TOMS Algorithm 1012, the DELAUNAYSPARSE software is given for performing Delaunay interpolation in medium to high dimensions. When extrapolating outside the convex hull of the training set, DELAUNAYSPARSE calls the nonnegative least squares solver DWNNLS to compute projections onto the convex hull. However, DWNNLS and many other available sum of squares optimization solvers were not intended for usage with many variable problems, which result from the large training sets that are typical in machine learning applications. Thus, a new PROJECT subroutine is given, based on the highly customizable quadratic program solver BQPD. This solution is shown to be as robust as DELAUNAYSPARSE for projection onto both synthetic and real-world data sets, where other available solvers frequently fail. Although it is intended as an update for DELAUNAYSPARSE, due to the difficulty and prevalence of the problem, this solution is likely to be of external interest as well.

CCS Concepts: • **Mathematics of computing** → **Mathematical software**; **Interpolation**; *Quadratic programming*; • **Theory of computation** → *Computational geometry*.

Additional Key Words and Phrases: Delaunay interpolation, projection, quadratic programming, data skew

ACM Reference Format:
Tyler H. Chang, Layne T. Watson, Sven Leyffer, Thomas C. H. Lux, and Hussain Almohri. 2023. Remark on Algorithm 1012: Computing projections with large data sets. *ACM Trans. Math. Softw.* 0, 0, Article 0 (2023), 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

ACM TOMS Algorithm 1012 provides the Fortran software DELAUNAYSPARSE for computing the Delaunay interpolant, an approximation to a d-dimensional function $f : \mathbb{R}^d \to \mathbb{R}^o$, in medium to high dimensions d [1].

Let $\mathcal{P} = \{p_1, \ldots, p_n\} \subset \mathbb{R}^d$ be a given set of "training data" with known response values $f(p_i)$ for all $p_i \in \mathcal{P}$. Let q be an interpolation point in the convex hull of $\mathcal{P}$ with unknown response

Authors' addresses: Tyler H. Chang, tchang@anl.gov, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 S Cass Ave, Lemont, IL, USA, 60516; Layne T. Watson, ltw@cs.vt.edu, Depts. of Computer Science, Mathematics, and Aerospace and Ocean Engineering, Virginia Polytechnic Institute and State University, Torgersen Hall, 620 Drillfield Dr, Blacksburg, VA, USA, 24061; Sven Leyffer, leyffer@anl.gov, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 S Cass Ave, Lemont, IL, USA, 60516; Thomas C. H. Lux, Meta, Menlo Park, CA, USA; Hussain Almohri, Dept. of Computer Science, Kuwait University, Kuwait.

0098-3500/2023/0-ART0 $15.00
https://doi.org/XXXXXXX.XXXXXXX

values. Denote the Delaunay triangulation of $\mathcal{P}$ by $DT(\mathcal{P})$, and let $s_1, \ldots, s_{d+1} \in \mathcal{P}$ be the vertices of a d-simplex in $DT(\mathcal{P})$ containing q. Then there exist interpolation weights w given by solving

$$\begin{bmatrix} s_1 & \ldots & s_{d+1} \\ 1 & \ldots & 1 \end{bmatrix} w = \begin{bmatrix} q \\ 1 \end{bmatrix},$$

and the Delaunay interpolant to approximate $f(q)$ is

$$\hat{f}_{DT}(q) = \sum_{i=1}^{d+1} f(s_i) w_i. \tag{1}$$

DELAUNAYSPARSE calculates the value of $\hat{f}_{DT}(q_i)$ for a finite set of interpolation points $\mathcal{Q} = \{q_1, \ldots, q_m\}$. This is done by computing only the m simplices in $DT(\mathcal{P})$ containing points in $\mathcal{Q}$, without computing the entire $DT(\mathcal{P})$, which grows exponentially large in d.

Note that so long as each $q \in \mathcal{Q}$ is inside the convex hull of $\mathcal{P}$, denoted $CH(\mathcal{P})$, then at least one simplex S in $DT(\mathcal{P})$ will always contain q. However, it is often the case that one wishes to make predictions for an *extrapolation point* z that is outside $CH(\mathcal{P})$. DELAUNAYSPARSE handles this situation by projecting z onto $CH(\mathcal{P})$ then interpolating at the projection $\hat{z}$.

Let W be the $d \times n$ matrix

$$W = [p_1 \quad p_2 \quad \ldots \quad p_n].$$

Then DELAUNAYSPARSE calculates the projection by $\hat{z} = Wx^\star$, where $x^\star$ solves the nonnegative least squares problem (NNLS) with a single equality constraint

$$\min_{x \in \mathbb{R}^n} \|Wx - z\|_2^2 \quad \text{subject to} \quad x \geq 0 \quad \text{and} \quad \sum_{i=1}^{n} x_i = 1. \tag{2}$$

Let e denote the n-vector of ones. To solve Equation (2), DELAUNAYSPARSE calls the double-precision SLATEC solver DWNNLS [8], which solves the equality-constrained NNLS

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} \|Wx - z\|_2^2 \\ &\text{subject to} \\ &e^\top x = 1, \qquad x \geq 0 \end{aligned}$$

as a *weighted* nonnegative least squares problem (WNNLS) by choosing a small penalty parameter μ and solving the NNLS

$$\min_x \left\| \begin{bmatrix} \mu W \\ e^\top \end{bmatrix} x - \begin{bmatrix} \mu z \\ 1 \end{bmatrix} \right\|_2^2 \quad \text{subject to} \quad x \geq 0. \tag{3}$$

The error when using the solution to (3) as the solution to (2) is proportional to the penalty parameter μ. In DWNNLS, the value of μ is chosen such that

$$\mu^2 = \frac{10^{-4}\eta}{\left\| \begin{bmatrix} W \\ e^\top \end{bmatrix} \right\|_\infty}$$

where $\eta \approx 2.22 \times 10^{-16}$ is the unit roundoff and $\|\cdot\|_\infty$ is the matrix infinity norm, given by the maximum row-sum. It is argued that this quantity is sufficient to guarantee accuracy proportional to the unit roundoff [8]. Because DELAUNAYSPARSE normalizes $\mathcal{P}$ to the unit sphere on input, W is componentwise less than or equal to 1, and $\| \begin{smallmatrix} W \\ e^\top \end{smallmatrix} \|_\infty = \|e^\top\|_\infty = n$.

The solution in (3) is effective when projecting onto the convex hull of a small number of points and when the matrix W is well-conditioned. However, at a fundamental level, DWNNLS and many similar solvers were not designed to handle problems with large numbers of variables, as is the

case when n grows large. In the particular case of DWNNLS, n being even moderately large drives μ to numerical zero, at which point μW becomes close to or exactly row-rank deficient. Alternatively, if certain dimensions of the training data set exhibit far more variability than others as is often the case in real-world data science problems, μW can become numerically singular for even smaller values of n after $\mathcal{P}$ is normalized to the unit hypersphere.

In Section 2, the generic quadratic program (QP) solver BQPD of R. Fletcher [6] is introduced. BQPD is robust against the issue discussed above, despite solving a broader class of problems than DWNNLS. As a result, an update to DELAUNAYSPARSE is needed, replacing all references to DWNNLS with BQPD, and producing a new public subroutine PROJECT for computing projections onto the convex hull via reference to BQPD. In Section 3, the original issue and effectiveness of the proposed solutions are demonstrated on both synthetic examples and a real data set. The difficulty of this problem is greater than originally thought, as observed through the failure of other well-known open source QP solvers on the failure cases of DWNNLS. Therefore, the new subroutine PROJECT may be of external interest as well.

2 USING BQPD TO COMPUTE PROJECTIONS ONTO THE CONVEX HULL

To pose the equality-constrained NNLS in (2) as a QP in general form, the sum-of-squares objective is expanded, divided by two, and the constant term is dropped, yielding

$$\min_{x\in\mathbb{R}^n} \frac{1}{2}x^\top W^\top W x - z^\top W x \quad \text{subject to} \quad 0 \le x \le e, \quad e^\top x = 1. \tag{4}$$

Interpreted as a generic QP, (4) is a hard problem for many QP solvers because the Hessian matrix $W^\top W$ is dense and the number of variables n can be prohibitively large for real-world data science applications (recall that n comes from the number of training points in $\mathcal{P}$). However, (4) has a lot of structure that is exploited in state-of-the-art active-set solvers.

In this paper, the active-set solver BQPD [6, 7] is used. BQPD is a primal active-set solver that is well-suited to solving large-scale instances of (4). One notable advantage of BQPD is that it does not require the Hessian matrix $W^\top W$ to be provided explicitly. Instead, one only needs to provide a subroutine to form products of the Hessian with a vector, v. In the case of (4), this product can be done in $O(nd)$ time while only explicitly storing W, which is an input. BQPD solves a sequence of equality-constrained QPs, where the equality constraints are $e^\top x = 1$, and a subset of currently active bounds, $x_i = 0$ (in this case, one can ignore the upper bounds, $x_i \le 1$, which is implied). BQPD forms and updates factors of two matrices at every iteration: (1) It forms LU factors of the active constraint matrix, $A_{\mathcal{A}} := [e : I_{\mathcal{A}} : I_{\mathcal{I}}]$, where $I_{\mathcal{A}}$ are the columns of active bounds, $x_i = 0$, and $I_{\mathcal{I}}$ is an arbitrary subset of the remaining columns of I. Because all but one column of $A_{\mathcal{A}}$ are unit columns, the factors can be stored and computed in $O(n)$ operations (updates are $O(1)$, if they only involve switching columns between $I_{\mathcal{A}}$ and $I_{\mathcal{I}}$). (2) BQPD also forms factors of the reduced Hessian, $Z^\top W^\top W Z$, where Z is implicitly defined as

$$[e \quad I_{\mathcal{A}} \quad I_{\mathcal{I}}]^{-1} = \begin{bmatrix} Y^\top \\ Z^\top \end{bmatrix}.$$

Note that Z is again comprised of unit columns. The reduced Hessian is typically small, because the solution $x^\star$ to (4) is always extremely sparse since each facet of the convex hull can be defined by at most d vertices in $\mathcal{P}$ and generally $d \ll n$ (the upper bound, $x \le e$, is redundant). Thus, the reduced Hessian is an $d \times d$ matrix, which can be factored efficiently.

Overall, BQPD requires $O(n)$ plus $O(d^2)$ additional storage, where $d \ll n$, and uses $O(nd + d^3)$ flops per iteration. This workspace overhead is less than the $O(nd)$ required to store the input data $\mathcal{P}$, so it does not increase the space complexity beyond the size of the input. However, it is still

significantly greater than the workspace storage required by DELAUNAYSPARSE, which only requires $O(d^2)$ additional storage. Thus, it is worth being mindful of this additional space requirement. On the other hand, BQPD's iteration complexity is significantly less than the iteration complexity of DELAUNAYSPARSE, which is $O(nd^2)$ [2]. Therefore, this approach does not increase the time complexity of the original DELAUNAYSPARSE software.

2.1 Software Updates and the PROJECT subroutine

To implement the solution proposed in Section 2, a new public subroutine named PROJECT has has been added to DELAUNAYSPARSE. This subroutine uses a private subroutine GDOTX for evaluating $W^\top Wx$ without ever explicitly forming $W^\top W$. An additional interface block is provided for PROJECT since it may be of external interest.

One limitation of BQPD is that it uses Fortran common blocks to pass data between several internal subroutines. This method of communication is not threadsafe, so the PROJECT subroutine must be called from within an OpenMP lock to avoid race conditions. However, as previously stated BQPD adds significant additional storage overhead, but this projection via BQPD is generally not the most time consuming part of the overall Delaunay interpolation. Therefore, the additional space requirement for shared-memory parallelism of BQPD may outweigh the negligible benefit of parallelizing a faster part of the computation in many use-cases.

Additionally, recall from ACM TOMS Algorithm 1012 [1] that DELAUNAYSPARSE uses a working precision of ε provided by the user (defaults to the square-root of the unit-roundoff). Among other things, ε is used as a tolerance when checking whether a point lies inside a simplex. However, when a simplex is ill-conditioned (as often occurs with high-dimensional simplices along the boundary of the convex hull) a point that is less than ε outside of $CH(\mathcal{P})$ could be judged as much farther due to numerical errors. In order to avoid these numerical issues, a slightly lower tolerance of $\varepsilon^{1.5}$ is required for BQPD.

3 DEMONSTRATION OF SOLUTIONS

This section demonstrates the prevalence of extrapolation in the context of machine learning, the prevalence of the error described in Section 1 on synthetic and real-world data sets, and the robustness of the BQPD-based solution on these problems. The purpose of this paper is not to study the interpolation/extrapolation accuracy of Delaunay interpolation, which has already been analyzed in previous works [9]. Therefore, in this section there is no consideration for any machine learning task (response values/class labels are not used and no predictions are made). Instead, the focus is on correctly computing the projection onto $CH(\mathcal{P})$ for various synthetic and real-world training sets $\mathcal{P}$.

3.1 Synthetic Examples

To assess the robustness of the new PROJECT subroutine powered by BQPD, this section considers the success rate when solving (2) for numerous synthetic data sets and extrapolation problems that are carefully designed to be challenging. Each data set is comprised of randomly generated training points $\mathcal{P}$ and 1000 extrapolation points drawn randomly from a sphere that contains $\mathcal{P}$. In order to consider different data distributions and problem conditioning, the input set $\mathcal{P}$ is pseudo-randomly sampled from several common distributions and undergoes several different transformations.

First, a *data geometry* is selected. Options include *lattice*, which means points are randomly selected from the smallest d-dimensional axis aligned lattice with k values on each axis such that $k^d \geq n$; *ball*, which means that points are chosen from a Latin hypercube design that is projected into a ball in a way that preserves cell volume; and *box*, which means a standard Latin hypercube design is used to generate points. Next, in order to check how DELAUNAYSPARSE handles rescaling

data, a uniform *scale factor* is applied across all dimensions of P with three possible values. Next, a *data modifier* is applied to control the conditioning of the data. In particular, scalar multipliers with a particular distribution are applied to each of the axes, where *identity* multiplies all by 1, *exponential skew* multiplies each component sequentially with uniformly spaced (in input) values from a exponential curve that intersects $(0, 0)$ and (s, s), and *linear skew* multiplies each component sequentially by values linearly spaced between 0 (noninclusive) and s (inclusive). The *skew factor* s has 3 possible values. In total there are 405 synthetic experimental designs generated by permuting unique combinations of these variables. Table 1 provides the experimental variables and their descriptions, of which all unique combinations are tested with 10 trials. In total there are 4050 experiments to simulate different conditions under which the Delaunay extrapolation routines might operate. Both the old version of DELAUNAYSPARSE using DWNNLS and the updated version using BQPD are tested on all generated synthetic data sets.

Table 1. Variables and their values in the synthetic data experiments. The values n and d define the size and shape of $\mathcal{P}$, as defined in Section 1. Data geometry, data modifier, skew, and scale factor are defined above.

Variable	Values
dimensions (d)	2, 8, 16
number of points (n)	$d+1$, 2^9 (512), 2^{14} (16, 384)
data geometry	lattice, ball, box
scale factor	2^{-20} ($9.5e^{-7}$), 1, 2^{63} ($9.2e^{18}$)
data modifier	identity, exponential skew, linear skew
skew (s)	1, 2^{10} (1024), 2^{20} ($1.1e^6$)

In total the synthetic benchmark produces 276 instances of error code 71 from DELAUNAYSPARSE using DWNNLS, and 0 instances of error code 71 when using BQPD on the same problems. Aggregated across all experiments, only the variable n has a noticeable effect on the frequency of DELAUNAYSPARSE error code 71 (due to failed projections of extrapolation points onto the convex hull of data). For all other variable values the prevalence of error 71 is uniform. Specifically, for $n = d + 1$ there were no occurrences of error 71, for $n = 2^9$ there are 15 occurrences of error 71, and for $n = 2^{14}$ there 261 occurrences of error 71. This synthetic benchmark supports the hypothesis that numerical instabilities associated with large numbers of variables are the core problem with a DWNNLS approach to projecting extrapolation points. This synthetic benchmark also shows that the errors no longer occur with a BQPD approach for projecting extrapolation points. Next a comparison of approaches to extrapolation on a real data set is considered.

3.2 KDDCUP99 Data Set

The previous section has shown how the new BQPD-based PROJECT subroutine performs on synthetic, uniformly-distributed data sets. However, to understand the prevalence of the issue and effectiveness of the solution, a test case based on real-world data is needed. Real-world data sets are often massive, ill-conditioned, and rank with geometric degeneracy, which presents a challenge for many geometric programming techniques [5].

KDDCUP99 [12] is a network intrusion detection data set. The data set includes predictors that were collected from network communications with their corresponding classification labels. The machine learning task is to classify a given communication instance, represented by its predictors, as one of the network attack types or as normal. The data set includes 42 predictors summarized in Table 2, with 94,008 training points and 50,620 testing points in the raw data set.

Table 2. Number of predictors and their types for the KDDCUP99 data set.

Feature Type	Count
Basic features of individual TCP connections	9
Content features within a connection	13
Traffic features using a two-second time window	20

As previously stated, for the scope of this paper there is no consideration for the machine learning task. Therefore, the class labels are unused and pre-processing of the data is kept to a minimum. However, DELAUNAYSPARSE will produce an error if $\mathcal{P}$ contains any duplicate data points or if all points in $\mathcal{P}$ lie in a lower-dimensional linear manifold. For the raw KDDCUP99 data set, both of these issues occur.

Therefore, taking the recommendations in the original software publication [1], the following steps were taken to ensure that DELAUNAYSPARSE will not raise an error.

(1) First, a reduced-dimensional set $\hat{\mathcal{P}}$ with improved conditioning is computed via a combination of rescaling and principle component analysis (PCA). This is done by calculating shift and scale factors $s, t \in \mathbb{R}^{41}$, where s is the barycenter of $\mathcal{P}$ and t is the componentwise maximum magnitude of $p - s$ over all $p \in \mathcal{P}$. To reduce the dimension, the singular value decomposition is computed as $\bar{W} = U\Sigma V^\top$ for $\bar{W} = [(p_1 - s)/t \quad (p_2 - s)/t \quad \dots \quad (p_n - s)/t]$. Then, the reduced-dimensional set $\hat{W} = U^{\dagger\top}\bar{W}$ where $U^\dagger$ contains the first 26 columns of U, corresponding to the singular values in Σ whose magnitudes are greater than 10^{-4}. Finally, the reduced-dimensional training points in $\hat{\mathcal{P}}$ are given by the columns of $\hat{W}$.
(2) Next, to eliminate redundant points in $\hat{\mathcal{P}}$, all tuples of points in $\hat{\mathcal{P}}$ whose pairwise 2-norm distance is less than 10^{-4} are clustered and replaced by a single synthetic training point whose value is their arithmetic mean.

After the above reduction, the resulting set $\hat{\mathcal{P}}$ that is used in this section contains 26 embedded features each in the normalized range $[-1, 1]$, 77,802 training points, and 50,620 testing points.

In order to asses the prevalence of the issue addressed in this paper and the performance of the new PROJECT subroutine on the cleaned KDDCUP99 data set, the following experiment was conceived. First, for various training sizes n_k, n_k of the total 77,802 indices in $\hat{\mathcal{P}}$ are sampled to produce a reduced training set of size n_k. Next, 20 testing points from the 50,620 in the total test set are sampled. Of the 20 testing points sampled, only those that are outside the convex hull of the training sample (extrapolation points) are kept. The above experiment is repeated for 10 different random seeds in order to produce several real-world data sets for each value of n_k.

For comparison, the original projection implementation using DWNNLS in DELAUNAYSPARSE version 1 is considered against the new version using BQPD. Additionally for the real-world data considered in Section 3.2, the projection is also computed using several open-source QP and second-order cone program (SOCP) solvers available in the popular convex programming modeling language CVXPY [3] (version 1.3.1). To do so, the problem (2) is modeled as a linearly constrained sum-of-squares minimization problem in CVXPY, then the OSQP (QP) [11], ECOS (SOCP) [4], and SCS (SOCP) [10] solvers are each used to solve the problem, requiring the same accuracy level as described in Section 2.1 ($\sim 1.8 \times 10^{-12}$ in 64-bit IEEE arithmetic) with an iteration limit of 10,000.

For each training size n_k and over all 10 random seeds, the total number of extrapolation points (out of 200 total sampled testing points), average condition number (over the 10 training samples), and probability of success for each solver is calculated. The results of this experiment are presented in Table 3.

			success rate				
n_k	extrap points	$\kappa(\hat{W})$	BQPD	DWNNLS	OSQP	ECOS	SCS
1,000	169	308.83	1.0	0.30	0.25	1.0	0.0
5,000	141	98.82	1.0	0.30	0.04	1.0	0.0
10,000	123	93.40	1.0	0.22	0.12	0.98	0.0
15,000	109	91.09	1.0	0.17	0.17	0.96	0.0
20,000	99	91.23	1.0	0.17	0.15	0.90	0.0

Table 3. Total number of extrapolation points, average condition number (κ), and solver success rates for various size subsamples of the cleaned KDDCUP99 data set.

From these results, several conclusions can be drawn. First, studying the meta data, it is clear that the number of extrapolation points is quite large. While it decreases with the size of the training set, at least 50% of the testing points sampled were extrapolation points for all training sizes used here. Second, the conditioning of the matrix $\hat{W}$ is somewhat poor, despite significant preprocessing to improve the problem conditioning, but improves as n_k grows. However, the performance of all methods except BQPD decreases as n_k increases, again suggesting that despite the poor conditioning, the value of n is the most important factor in the difficulty of (2).

Finally, looking at the success rates, only `BQPD` is able to compute all projections across all training sizes, showing the robustness of this solution. The majority of the solvers failed more than 50% of the time on even the smallest samples of the training set. Of the other solvers compared, only `ECOS` offers acceptable performance on the smaller problem sizes, but still fails to converge a significant percentage of the time on the larger problems. Given the robustness of the original Delaunay interpolation code and algorithm, only `BQPD` gives performance on these extrapolation tasks that is in keeping with the spirit of `DELAUNAYSPARSE`.

ACKNOWLEDGMENTS

This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357.

REFERENCES

[1] Tyler H. Chang, Layne T. Watson, Thomas C. H. Lux, Ali R. Butt, Kirk W. Cameron, and Yili Hong. 2020. Algorithm 1012: DELAUNAYSPARSE: Interpolation via a Sparse Subset of the Delaunay Triangulation in Medium to High Dimensions. *ACM Trans. Math. Softw.* 46, 4, Article 38 (2020), 20 pages. https://doi.org/10.1145/3422818

[2] Tyler H. Chang, Layne T. Watson, Thomas C. H. Lux, Bo Li, Li Xu, Ali R. Butt, Kirk W. Cameron, and Yili Hong. 2018. A polynomial time algorithm for multivariate interpolation in arbitrary dimension via the Delaunay triangulation. In *Proc. 2018 ACM Southeast Conference (ACMSE '18)* (Richmond, KY, USA). ACM, Article 12, 8 pages. https://doi.org/10.1145/3190645.3190680

[3] Steven Diamond and Stephen Boyd. 2016. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research* 17, 83 (2016), 1–5. http://jmlr.org/papers/v17/15-408.html

[4] Alexander Domahidi, Eric Chu, and Stephen Boyd. 2013. ECOS: An SOCP solver for embedded systems. In *European Control Conference (ECC)* (Zürich, Switzerland). IEEE, 3071–3076. https://doi.org/10.23919/ECC.2013.6669541

[5] Herbert Edelsbrunner and Ernst Peter Mücke. 1990. Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms. *ACM Transactions on Graphics (TOG)* 9, 1 (1990), 66–104. https://doi.org/10.1145/77635.77639

[6] Roger Fletcher. 1993. Resolving degeneracy in quadratic programming. *Annals of Operations Research* 46 (1993), 307–334. https://doi.org/10.1007/BF02023102

[7] Roger Fletcher. 2000. Stable reduced Hessian updates for indefinite quadratic programming. *Mathematical programming* 87 (2000), 251–264.

[8] Richard J. Hanson and Karen H. Haskell. 1982. Algorithm 587: Two Algorithms for the Linearly Constrained Least Squares Problem. *ACM Trans. Math. Softw.* 8, 3 (1982), 323–333. https://doi.org/10.1145/356004.356010

[9] Thomas C. H. Lux, Layne T. Watson, Tyler H. Chang, Jon Bernard, Bo Li, Li Xu, Godmar Back, Ali R. Butt, Kirk W. Cameron, and Yili Hong. 2021. Interpolation of sparse high-dimensional data. *Numerical Algorithms* 88, 1 (2021), 281–313. https://doi.org/10.1007/s11075-020-01040-2

[10] Brendan O'Donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. 2016. Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding. *Journal of Optimization Theory and Applications* 169, 3 (June 2016), 1042–1068. https://doi.org/10.1007/s10957-016-0892-3

[11] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. 2020. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation* 12, 4 (2020), 637–672. https://doi.org/10.1007/s12532-020-00179-2

[12] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani. 2009. A detailed analysis of the KDD CUP 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*. IEEE, Ottawa, ON, Canada, 1–6. https://doi.org/10.1109/CISDA.2009.5356528