Fast and Reliable Methods in Numerical Linear Algebra, Signal Processing, and Image Processing

by

James Folberth

B.S., Rose-Hulman Institute of Technology, 2013

M.S., University of Colorado, 2016

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Applied Mathematics

2018

This thesis entitled:
Fast and Reliable Methods in Numerical Linear Algebra, Signal Processing, and Image Processing
written by James Folberth
has been approved for the Department of Applied Mathematics

Prof. Stephen Becker

Prof. Ian Grooms

Date

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Folberth, James (Ph.D., Applied Mathematics)

Fast and Reliable Methods in Numerical Linear Algebra, Signal Processing, and Image Processing

Thesis directed by Prof. Stephen Becker

In this dissertation we consider numerical methods for a problem in each of numerical linear algebra, digital signal processing, and image processing for super-resolution fluorescence microscopy. We consider first a fast, randomized mixing operation applied to the unpivoted Householder QR factorization. The method is an adaptation of a slower randomized operation that is known to provide a rank-revealing factorization with high probability. We perform a number of experiments to highlight possible uses of our method and give evidence that our algorithm likely also provides a rank-revealing factorization with high probability.

In the next chapter we develop fast algorithms for computing the discrete, narrowband cross-ambiguity function (CAF) on a downsampled grid of delay values for the purpose of quickly detecting the location of peaks in the CAF surface. Due to the likelihood of missing a narrow peak on a downsampled grid of delay values, we propose methods to make our algorithms robust against missing peaks. To identify peak locations to high accuracy, we propose a two-step approach: first identify a coarse peak location using one of our delay-decimated CAF algorithms, then compute the CAF on a fine, but very small, grid around the peak to find its precise location. Runtime experiments with our C++ implementations show that our delay-decimated algorithms can give more than an order of magnitude improvement in overall runtime to detect peaks in the CAF surface when compared against standard CAF algorithms.

In the final chapter we study non-negative least-squares (NNLS) problems arising from a new technique in super-resolution fluorescence microscopy. The image formation task involves solving many tens of thousands of NNLS problems, each using the same matrix, but different right-hand sides. We take advantage of this special structure by adapting an optimal first-order method to efficiently solve many NNLS problems simultaneously. Our NNLS problems are extremely ill-conditioned, so we also experiment with using a block-diagonal preconditioner and the alternating direction method of multipliers (ADMM) to improve convergence speed. We also develop a safe feature elimination strategy for general NNLS problems. It eliminates features only when they are guaranteed to have weight zero at an optimal point. Our strategy is inspired by recent works in the literature for ℓ_1-regularized least-squares, but a notable exception is that we develop our method to use an inexact, but feasible, primal-dual point pair. This allows us to use feature elimination reliably on the extremely ill-conditioned NNLS problems from our microscopy application. For an example image reconstruction, we use our feature elimination strategy to certify that the reconstructed super-resolved image is unique.

Dedication

To my parents.

Acknowledgements

First and foremost, many thanks go to Stephen Becker. This dissertation would not be possible without his guidance, patience, and encouragement. In working with him I've been exposed to many exciting and practical ideas. I will always be grateful for the time I've spent working with him.

There are many others that I would like to thank for their help throughout the years: my undergraduate advisors Dr. David Goulet, Dr. David Finn, and Dr. Rick Ditteon; the fantastic folks at Northrop Grumman for their generous support of the CAF project; Dr. Carol Cogswell, Dr. Jiun-Yann Yu, Simeng Chen, and Jian Xing, for inviting Stephen and I to collaborate with them on their microscopy research; Dr. Chris Ketelsen, for numerous enlightening and fun conversations; Dr. Daniel Appelö, for helpful discussions, particularly on iterative refinement; my fellow graduate students and friends, for making my time in graduate school enjoyable.

Corinne, your partnership has helped me more than I know how to put into words. Our furry friend, Bagheera, deserves the "panther's share" of the credit for helping me type this document. I am so appreciative of his determined assistance with my keyboard.

Contents

Tables

Figures

Chapter 1

URV Factorization with Random Orthogonal System Mixing

The unpivoted and pivoted Householder QR factorizations are ubiquitous in numerical linear algebra. A difficulty with pivoted Householder QR is the communication bottleneck introduced by pivoting. In this chapter we propose using random orthogonal systems to quickly mix together the columns of a matrix before computing an unpivoted QR factorization. This method computes a URV factorization which forgoes expensive pivoted QR steps in exchange for mixing in advance, followed by a cheaper, unpivoted QR factorization. The mixing step typically reduces the variability of the column norms, and in certain experiments, allows us to compute an accurate factorization where a plain, unpivoted QR performs poorly. We experiment with linear least-squares, rank-revealing factorizations, and the QLP approximation, and conclude that our randomized URV factorization behaves comparably to a similar randomized rank-revealing URV factorization, but at a fraction of the computational cost. Our experiments provide evidence that our proposed factorization might be rank-revealing with high probability.

1.1 Introduction

The QR factorization of a matrix $A \in \mathbb{R}^{m\times n}$ is a widely used decomposition, with applications in least-squares solutions to linear systems of equations, eigenvalue and singular value problems, and identification of an orthonormal basis of the range of A. The form of the decomposition is $A = QR$, where Q is $m \times m$ and orthogonal and R is $m \times n$ and upper triangular. When A is dense and has no special structure, Householder reflections are often preferred to Gram-Schmidt (and its variants) and Givens rotations, due to their precise orthogonality and computational efficiency via the (compact) WY representation [42, 14, 80], which can utilize level-3 BLAS. Indeed, Householder QR with a compact WY representation is implemented in the LAPACK routine `_geqrf` [5] (we use the prefix `_` to denote one of `s`, `d`, etc.).

A common variant of the QR factorization is column pivoted QR, which computes the factorization $A\Pi = QR$, where Π is a permutation matrix. At the ith stage of the decomposition, the column of the submatrix $A(i{:}m, i{:}n)$ (in MATLAB notation) with the largest norm is permuted to the leading position of $A(i{:}m, i{:}n)$ and then a standard QR step is taken. The LAPACK routine `_geqp3` implements column pivoted Householder QR using level-3 BLAS [5]. However, it is typically much slower than the unpivoted `_geqrf`, as `_geqp3` still suffers from high communication costs [28] and cannot be cast entirely in level-3 operations [65]. We refer to Householder QR without pivoting as unpivoted QR (`QR`), and Householder QR with column pivoting as `QRCP`.

Improving on `QRCP`, recent works have used random projections to select blocks of pivots, emulating the behavior of `QRCP`, while more fully utilizing level-3 BLAS [64, 65, 34]. Indeed, this approach is quite efficient on modern computer architectures; for example, it is implemented in Elemental, a framework for distributed memory dense matrix computations [79]. Another approach uses so called "tournament pivoting" to select blocks of pivots and is shown to minimize communication up to polylogarithmic factors [28]. In each of these cases, a pivoted QR factorization is produced.

URV factorizations decompose A as $A = URV$, where U and V have orthonormal columns and R is upper triangular. One can think of URV factorizations as a relaxation of the SVD, where instead of a diagonal singular value matrix, we require only that R is upper-triangular. Similarly, `QRCP` can be thought of as a URV factorization where V is a permutation matrix, a special case of an orthogonal matrix. In Section 1.3 we discuss how URV factorizations can be used to solve linear least-squares problems in much the same manner as QR factorizations or the SVD.

For another example, let V be a random orthogonal matrix sampled from the Haar distribution on orthogonal matrices. The matrices U and R are computed with an **unpivoted** QR factorization of $\hat{A} = AV^T$, and the resulting URV factorization is a strong rank-revealing factorization with high probability (see Subsection 1.2.1) [25]; we call this randomized factorization `RURV_Haar`. This demonstrates that one can forego column pivoting at the cost of mixing together the columns of A and still have a safe factorization. However, taking V to be a random, dense orthogonal matrix is rather computationally expensive, as V is generated with an $n \times n$ unpivoted QR and must be applied with dense matrix multiplication.

We instead propose mixing with an alternating product of orthogonal Fourier-like matrices (e.g., discrete cosine, Hadamard, or Hartley transforms) and diagonal matrices with random ± 1 entries, forming a so-called random orthogonal system (ROS) [1, 95, 60, 68]. This provides mixing, but with a fast transform, as V is never formed explicitly and can be applied with the FFT, or FFT-style algorithms (see Subsection 1.2.2). We call this randomized URV factorization with ROS mixing `RURV_ROS`.

Numerical experiments with our implementation of `RURV_ROS` demonstrate that for large enough matrices (i.e., where communication is the bottleneck of `QRCP`), `RURV_ROS` runs only slightly slower than `_geqrf`, but significantly faster than `_geqp3`. Figure 1.1 shows the average performance of `dgeqrf`, `dgeqp3`, and `RURV_ROS`, computed as mn^2 divided by the runtime in seconds (and further divided by 10^9, to be analogous to GFLOP/second). Note that each algorithm performs a different number of floating point operations, but computing performance as mn^2 divided by runtime treats each algorithm fairly at each matrix size. Indeed it is valid to directly compare the performance of the algorithms with each other.

We used MATLAB's LAPACK [66] and the reference FFTW [37] with 1 and 8 threads on a desktop workstation with two Intel® Xeon® E5-2630 v3 CPUs running at 2.4 GHz. See Subsection 1.2.2 for more details on our implementation of `RURV_ROS`.

Around $n = 1000$, we begin to see a sharp increase in the runtime of `dgeqp3`, owing to the communication bottleneck of column pivoting. In this region, `dgeqp3` with 8 threads does not see an appreciable improvement over running just a single thread. In contrast, `dgeqrf` parallelizes much more nicely, as we can see an order of magnitude improvement in runtime when using 8 threads. When using RURV_ROS, we also see a noticeable improvement in runtime when using 8 threads versus 1 thread.

We also run timing and accuracy experiments on over- and underdetermined linear least-squares problems in Section 1.3. In Subsection 1.4.1 we sample the rank-revealing conditions of [45, 44] for a variety of QR and URV factorizations, which suggest that RURV_ROS behaves similarly to RURV_Haar. This provides evidence suggesting that RURV_ROS is rank-revealing with high probability. We also examine using RURV_Haar and RURV_ROS in a QLP approximation to the SVD in Subsection 1.4.3.

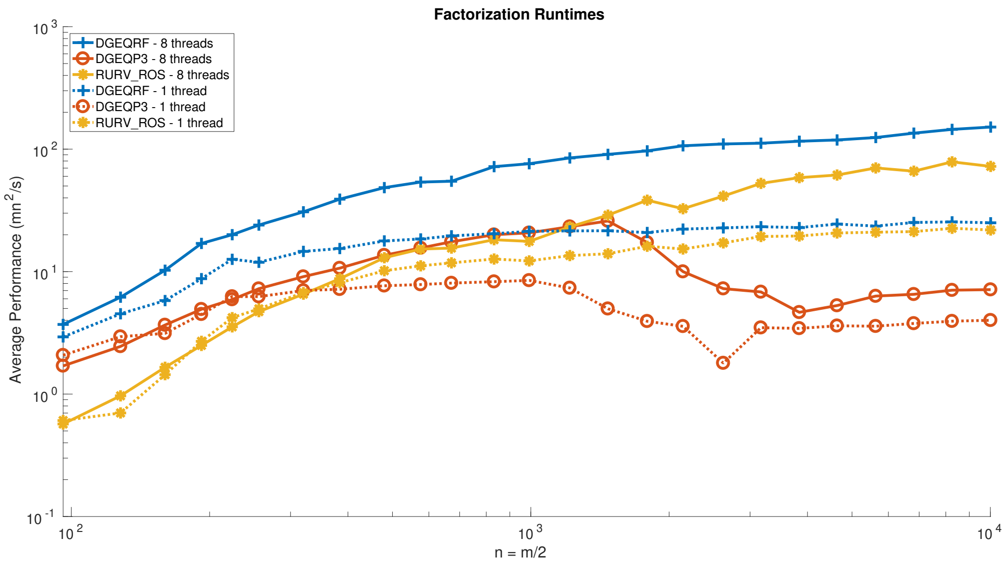


Figure 1.1: Average performance over five runs of dgeqrf, dgeqp3, and RURV_ROS on slightly tall-skinny matrices ($n = m/2$). Note that we do not include the time to generate the orthogonal factor Q (labeled U for RURV_ROS), as all routines would use dorgqr. For the run with 8 threads, the sharp decrease in performance beginning around size 2000×1000 matrices corresponds to the beginning of the regime where communication is the bottleneck of QRCP.

1.2 Randomized URV Factorization

1.2.1 Randomized URV Factorization via Haar Random Orthogonal Mixing

Demmel et al. proposed in [25] a randomized URV factorization (RURV), which we call RURV_Haar, to use as part of eigenvalue and singular value decompositions. Their RURV of an

$m \times n$ matrix A is based on sampling from the Haar distribution on the set of orthogonal (or unitary) matrices [69], using that sampled matrix to mix the columns of A, and then performing an **unpivoted** QR on the mixed A, resulting in the factorization $A = URV$.

Algorithm 1 `RURV_Haar` - Randomized URV with Haar mixing from [25]

```
Input: $A \in \mathbb{R}^{m \times n}$
Output: $U, R, V$
1: Generate a random $n \times n$ matrix $B$ whose entries are i.i.d. $N(0,1)$.
2: $[V, \hat{R}] = \texttt{qr}(B)$          ▷ $V$ is Haar distributed; $\hat{R}$ is unused
3: $\hat{A} = AV^T$
4: $[U, R] = \texttt{qr}(\hat{A})$
```

Haar orthogonal matrices are known to smooth the entries of the vectors on which they operate. By multiplying A on the right by a Haar orthogonal matrix V^T, we can mix together the columns of A, and reduce the variance of the column norms (see Figure 1.2). The intuition behind the mixing is that by reducing the variance of the column norms, we reduce the effect that column pivoting would have, and can get away with unpivoted QR. Indeed, in [25] it is shown that Algorithm 1 produces a rank-revealing factorization with high probability, and can be used for eigenvalue and SVD problems. It was further shown that Algorithm 1 produces a **strong** rank-revealing factorization in [9]. Criteria for a (strong) rank-revealing factorization of the form $A = URV$ are as follows (taken from [45, 44], but slightly weaker conditions were used in [9]):

(1) U and V are orthogonal and $R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix}$ is upper-triangular, with R_{11} $k \times k$ and R_{22} $(n-k) \times (n-k)$;

(2) For any $1 \le i \le k$ and $1 \le j \le \min(m,n) - k$,

$$1 \le \frac{\sigma_i(A)}{\sigma_i(R_{11})}, \frac{\sigma_j(R_{22})}{\sigma_{k+j}(A)} \le q(k,n), \tag{1.1}$$

where $q(k,n)$ is a low-degree polynomial in k and n.

(3) In addition, if

$$\|R_{11}^{-1} R_{12}\|_2 \tag{1.2}$$

is bounded by a low-degree polynomial in n, then the rank-revealing factorization is called **strong**.

These conditions state that the singular values of R_{11} and R_{12} are not too far away from the respective singular values of A. Thus, by performing a rank-revealing factorization instead of an expensive SVD, we can still gain insight into the singular values of A.

Both QR factorizations in Algorithm 1 are **unpivoted**, and thus can be considerably cheaper than the standard column-pivoted Householder QR, `QRCP`. However, a major drawback is the expense of generating and applying the random matrix V. To sample an $n \times n$ matrix V from the Haar distribution on orthogonal matrices, we take the Q factor from an unpivoted QR factorization of an $n \times n$ matrix B whose entries are i.i.d. $N(0,1)$ [69]. The dominant cost of this computation is the unpivoted QR factorization, which requires $\mathcal{O}(n^3)$ FLOPs. We then compute $\hat{A} = AV^T$, which requires $\mathcal{O}(mn^2)$ FLOPs, followed by the unpivoted QR factorization to find U and R, which costs $\mathcal{O}(mn^2)$ FLOPs. To reduce the cost of forming and applying V, we propose replacing V with a product of random orthogonal systems, which can each be applied implicitly and quickly, although providing slightly worse mixing.

1.2.2 Randomized URV Factorization via Fast Random Orthogonal Mixing

Consider a real $m \times n$ matrix A and a product of random orthogonal systems (ROS) of the form

$$V = \Pi \left[\prod_{i=1}^{N} F D_i \right], \tag{1.3}$$

where each D_i is a diagonal matrix of independent, uniformly random ± 1 and F is an orthogonal Fourier-like matrix with a fast transform. Just like in `RURV_Haar`, we mix together the columns of A as $\hat{A} = AV^T$. The matrix Π is a permutation matrix chosen so $\hat{A}\Pi^T$ sorts the columns of $\hat{A}$ in order of decreasing norm. Replacing the Haar matrix V in Algorithm 1 with the ROS based V in (1.3) yields the new algorithm we call `RURV_ROS`, shown in Algorithm 2.

Algorithm 2 `RURV_ROS` - Randomized URV with ROS mixing

```
Input: $A \in \mathbb{R}^{m \times n}$, number of mixing steps $N$, $\{D_i\}_{i=1}^{N}$ diagonal $\pm 1$ matrices
Output: $U, R$        ▷ The $V$ matrix is not output because it is never explicitly formed
1: $\hat{A} = A \prod_{i=N}^{1} (D_i F^T)$
2: $\hat{A} = \hat{A} \Pi^T$        ▷ Sort the columns of $\hat{A}$ so they are in order of decreasing $\ell_2$ norm.
3: $[U, R] = \texttt{qr}(\hat{A})$
4: $V = \Pi \prod_{i=1}^{N} F D_i$
```

Each product FD_i is referred to as a random orthogonal system (ROS) [1, 95, 60, 68]. Examples of real-to-real, orthogonal Fourier-like transforms are the discrete cosine transform (e.g., DCT-II and DCT-III), the discrete Hartley transform, and the discrete Hadamard transform. The Fourier-like matrix is never explicitly constructed, but rather is only used as an operator, for which we use a fast transform. This brings the FLOP count for computing AV^T from $\mathcal{O}(mn^2, n^3)$ to $\mathcal{O}(mn \log n)$. In our experiments, we use the DCT-II and DCT-III for F and F^T, as implemented in FFTW [37].

Figure 1.2 shows the effect of mixing with Haar matrices and ROS on the column norms of a random 250×250 matrix A, formed in MATLAB with `A = bsxfun(@times, randn(m,n)+ exp(10*rand(m,n)), exp(2*rand(1,n)))`, followed by `A = A/mean(sqrt(sum(A.*A)))`, so that the mean column norm is one. The variance of the column norms is clearly decreased by the mixing, and notably, Haar and ROS (with $N = 1$) affect the distribution of column norms in a similar manner. A theme of this chapter is that `RURV_ROS` behaves similarly to `RURV_Haar`, which likely stems from their similar effect on the distribution of column norms.

To mix together the columns of A, we compute $\hat{A} = A \prod_{i=N}^{1} (D_i F^T)$. The permutation/pre-sort matrix Π is chosen so the columns of $\hat{A}\Pi^T$ are sorted in decreasing order of column norm. The pre-sort is included to potentially enhance the accuracy and stability of `RURV_ROS`. Note that the cost of this one-time sort is much smaller than the cost of the repeated column pivots in `QRCP`.

A MATLAB implementation of `RURV_ROS` with F taken to be the DCT-II is shown in Listing 1.1. For in-core computations, it is sometimes more efficient to compute the mixing on left of A^T

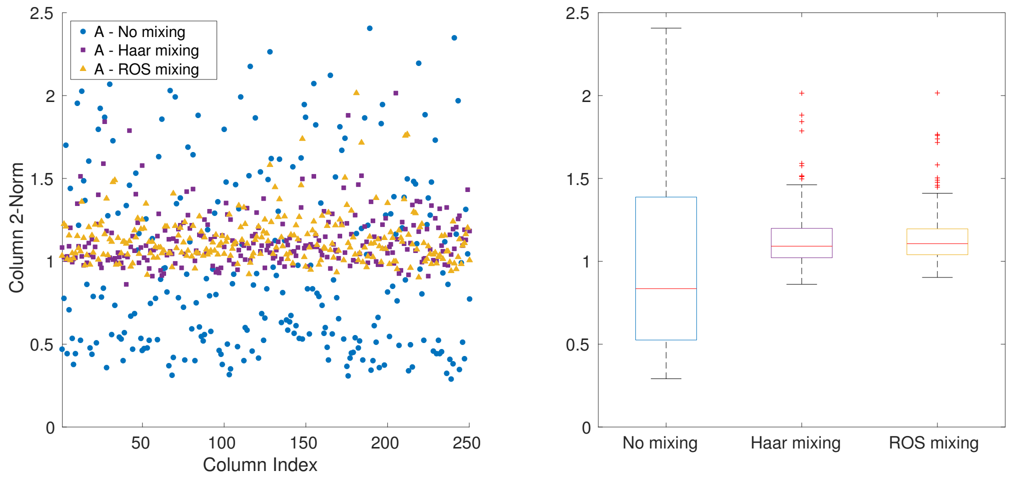


Figure 1.2: Mixing columns of A together with Haar orthogonal matrices and ROS reduces the variance of column norms, while keeping the mean column norm about the same.

via:

$$AV^T = (VA^T)^T = \left(\Pi \prod_{i=1}^{N} (FD_i)\, A^T\right)^T.$$

This "transpose trick" is used in Listing 1.1 for efficiency, and also to cleanly interface with MATLAB's `dct` function, which applies the transform to the columns of its input. Listing 1.1 explicitly returns U and R from the factorization, but returns function handles for V and V^T, which can be used to apply V and V^T, respectively, to the left side of their input.

The implementation used for our experiments is similar, but has performance-critical sections written in C using MATLAB's MEX interface. The mixing step is performed in C using FFTW and the unpivoted QR is performed in C using LAPACK routines from MATLAB's LAPACK [37, 5, 66]. The use of FFTW gives us great control over how the transform is applied (e.g., in blocks, multi-threaded, perhaps not utilizing the "transpose trick", etc.). More details on the use of FFTW for mixing are given in Subsection 1.3.2.

Listing 1.1: A MATLAB implementation of `RURV_ROS`

```
function [U,R,V,Vt] = RURV_ROS(A, n_its)
% RURV_ROS  RURV with ROS mixing for real matrices

  [m,n] = size(A);
  D_diags = sign(rand(n,n_its)-0.5); % diagonals of D_i; i.i.d. uniform +- 1
```

```

   % ROS mixing Ahat = A*V' using transpose trick
   Ahat = apply_V(A',D_diags)';

   % pre-sort
   nrms = sqrt(sum(Ahat.^2,1));
   [nrms,p] = sort(nrms, 2, 'descend');
   p_inv(p) = 1:numel(p);
   Ahat = Ahat(:,p);

   % unpivoted QR factorization
   [U,R] = qr(Ahat,0);

   % Return function handles to apply V and V^T on the left
   V = @(A) apply_V(A,D_diags,p);
   Vt = @(A) apply_Vt(A,D_diags,p_inv);
end

function [Ahat] = apply_V(A, D_diags, p)
% apply_V  Apply ROS mixing: V*A
   Ahat = A;
   for i=1:size(D_diags,2)
      Ahat = bsxfun(@times, Ahat, D_diags(:,i)); % Ahat = D_i*Ahat
      Ahat = dct(Ahat);                          % Ahat = F*Ahat;
   end
   if nargin == 3 % apply sorting
      Ahat = Ahat(p,:);
   end
end

function [Ahat] = apply_Vt(A, D_diags, p_inv)
% apply_Vt  Apply transpose ROS mixing: V^T*A
   if nargin == 3 % apply sorting
      Ahat = A(p_inv,:);
   else
      Ahat = A;
   end
   for i=size(D_diags,2):-1:1
      Ahat = idct(Ahat);                         % Ahat = F^T*Ahat;
      Ahat = bsxfun(@times, Ahat, D_diags(:,i)); % Ahat = D_i*Ahat
   end
end
```

1.3 Applications to Least-Squares Problems

1.3.1 Solving Least-Squares Problems with a URV Factorization

A URV factorization can be used to solve least-squares problems in much the same manner as a QR factorization. Throughout this subsection we assume that A is $m \times n$ and full-rank. We are interested in finding a solution to

$$\min_x \|Ax - b\|_2$$

for both the overdetermined case $m \geq n$ and the underdetermined case $m < n$.

1.3.1.1 Overdetermined Systems

Consider first the case when A is overdetermined. To find the least-squares solution with a QR factorization, we only need a thin QR factorization, where Q is $m \times n$ and R is $n \times n$ [42]. Similarly, the internal QR factorization in `RURV_ROS` can be a thin QR. By computing $A = URV$ and using that U has orthonormal columns,

$$\min_x \|Ax - b\|_2 = \min_x \|URVx - b\|_2 = \min_x \|RVx - U^T b\|.$$

The least-squares problem reduces to the non-singular $n \times n$ upper-triangular system $Ry = U^T b$ in the auxiliary variable $y = Vx$. The system $Ry = U^T b$ is solved for y with backward substitution, and then the least-squares solution is found with $x = V^T y$.

Note that we do not need to explicitly form U to apply U^T to b. When we call LAPACK's `_geqrf` on $\hat{A} = AV^T$, the routine overwrites the upper-triangular part of $\hat{A}$ with R and the Householder reflectors in the strictly lower triangular part of $\hat{A}$. By feeding the Householder reflectors into `_ormqr`, we can implicitly compute $U^T b$ in $\mathcal{O}(mn)$ FLOPs without ever accumulating U [5].

The dominant cost of using `RURV_Haar` to compute least-squares solutions is a mix of generating V, computing $\hat{A} = AV^T$, and the thin QR to find U and R. The latter two operations cost $\mathcal{O}(mn^2)$ FLOPs. The dominant cost of using `RURV_ROS` is also $\mathcal{O}(mn^2)$, but the leading cost term only comes from the unpivoted, thin QR to form U, as the mixing $\hat{A} = AV^T$ is faster, taking only $\mathcal{O}(mn \log n)$ FLOPs.

1.3.1.2 Underdetermined Systems

Now consider the underdetermined case. A full URV factorization $A = URV$ is of the following form:

$$\begin{array}{cc} & n \\ m & \left[\quad A \quad \right] \end{array} = \begin{array}{cc} & m \\ m & U \end{array} \begin{array}{cc} m & n-m \\ \left[R_{11} \right. & \left. R_{12} \right] \end{array} \begin{array}{cc} & n \\ V. & n \end{array} \tag{1.4}$$

Since A is assumed to be full-rank, $\min_x \|Ax - b\|_2 = 0$ and we seek to solve $Ax = b$. As in the overdetermined case, make the change of variable $y = Vx$; we now consider solving the upper-trapezoidal system $Ry = U^T b$. Partitioning y into $m \times 1$ and $(n-m) \times 1$ blocks results in the block system

$$\begin{bmatrix} R_{11} & R_{12} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = U^T b,$$

where R_{11} is upper-triangular and full-rank. A particularly simple solution is found by setting $y_2 = 0$ and performing backward substitution to find y_1. Following [42], we call this the **basic solution**. Note that the basic solution has $m-n$ zeros in y, but after unmixing to find $x_{\text{basic}} = V^T y$, the zeros in y_2 are mixed with the non-zeros in y_1, destroying the sparsity of x_{basic}. While this is less than ideal, mixing and unmixing is fast, and sparsity in the mixed domain might still be applicable in certain applications.

Notice that R_{12} is not used to compute the basic solution. Since R is computed from $\hat{A} = AV^T$, which mixes all the columns of A together, we may compute U and R_{11} from the QR factorization of $\hat{A}(:, 1{:}m)$ (in MATLAB notation). This avoids the computation of R_{12}, leading to a faster solution. Mixing to find $\hat{A} = AV^T$ costs $\mathcal{O}(mn \log n)$; computing R_{11} costs $\mathcal{O}(m^3)$; and applying $U^T b$, backward substitution to find $y = R_{11}^{-1} U^T b$, and unmixing to find $x_{\text{basic}} = V^T y$ all cost a negligible amount for large m and n. This brings the total cost to compute the basic solution to $\mathcal{O}(m^3, mn \log n)$ FLOPs.

Another common solution is the **minimum norm solution**. Since the solution set $\mathcal{X} = \{x \in \mathbb{R}^n \mid Ax = b\}$ is closed and convex, there exists a unique minimum norm solution, which is a principal attraction to the minimum norm solution (a similar statement holds even when A is rank deficient). Finding the minimum norm solution can be expressed as the problem

$$\begin{aligned} \min \quad & \|x\|^2 \\ \text{s.t.} \quad & Ax = b. \end{aligned}$$

Let $\mathcal{L}(x, \nu) = x^T x + \nu^T (Ax - b)$ be the Lagrangian function. Slater's condition for this problem is simply that the problem is feasible, which is of course satisfied since we assume A is full-rank. Therefore, strong duality holds and the KKT conditions,

$$\nabla_x \mathcal{L} = 2x + A^T \nu = 0, \quad Ax - b = 0,$$

give necessary and sufficient conditions for the solution [17]. Solving the KKT conditions gives $x_{\text{mn}} = A^T (AA^T)^{-1} b = A^\dagger b$, where $A^\dagger$ is the (right) pseudoinverse of A. To use this closed-form solution efficiently, it is convenient to perform a QR factorization of A^T. Specifically, if we let $A^T = QR$, then $x_{\text{mn}} = QR^{-T} b$, where $R^{-T} b$ is computed implicitly with forward substitution.

To find the minimum norm solution with mixing, we should mix the columns of A^T in preparation for the unpivoted QR of $\hat{A}^T$. Let $\hat{A}^T = A^T V^T$ (which we may compute via $\hat{A} = VA$) and compute $\hat{A}^T = U^T L^T$ via unpivoted QR. We then have the factorization $A = V^T LU$, where V is our fast ROS mixing matrix, L is $m \times m$ lower triangular, and U is $m \times n$ with orthonormal rows (i.e., U^T is orthonormal). We call the algorithm to compute $A = V^T LU$ `RVLU_ROS` in analogy with `RURV_ROS`. By multiplying $Ax = b$ on the left by V, we find $\hat{A}x = Vb$, and from the discussion above, the minimum norm solution is $x_{\text{mn}} = U^T L^{-1} Vb$. Again note that L^{-1} is applied implicitly using forward substitution. The dominant cost of this approach is again the unpivoted QR factorization of $\hat{A}^T$, which costs $\mathcal{O}(mn^2)$ FLOPs, which can be significantly higher than the $\mathcal{O}(m^3, mn \log n)$ FLOPs for the basic solution.

1.3.2 Timing Experiments

Solving least-squares problems with `RURV_ROS` or `RVLU_ROS` factorizations will be slightly slower than using unpivoted QR; the additional cost comes almost entirely from the mixing steps in Algorithm 2. In our code, we use the DCT-II and DCT-III, as implemented in FFTW [37]. For improved performance, we cache FFTW "wisdom" in a file and load it the next time it is applicable. Finding the solution proceeds in three stages: mixing to find $\hat{A}$, performing unpivoted QR factorization of $\hat{A}$ or $\hat{A}^T$, and computing the final solution vector, which may involve mixing a single vector. For moderately large overdetermined problems, mixing to find $\hat{A}$ takes about 25% of the total runtime; unpivoted QR factorization 75% of the total time; and solving/mixing takes a negligible amount of time, since it is applied to only a single vector.

We compare with BLENDENPIK, which uses mixing across rows and row sampling to find a good preconditioner for LSQR [8, 77]. The authors wrote most of their code in C for efficiency, calling LAPACK and FFTW libraries and providing their own implementation of LSQR. In the underdetermined case, BLENDENPIK computes the minimum norm solution. We use DCT mixing and adjust the size of the preconditioner to balance the cost of building the preconditioner with the cost of LSQR iterations. We otherwise use the default parameters provided in BLENDENPIK's interface.

It is worth noting that the well-known backslash (\) operator in MATLAB solves (rectangular) linear systems in the least-squares sense using a QR-based approach. MATLAB's \ operator tends to be significantly slower than BLENDENPIK and `RURV_ROS`, but \ also supports the case of rank-deficient matrices [66]. LAPACK has a variety of least-squares routines, and can handle full-rank and rank-deficient matrices. The LAPACK routine `_gels` uses a simple rescaling and unpivoted QR or LQ to solve full-rank least-squares problems [5]. For highly overdetermined systems, BLENDENPIK is reported to beat QR-based solvers, including `_gels`, by large factors [8].

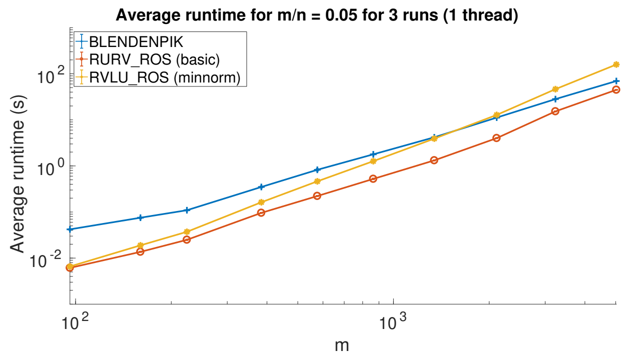
For the following timing experiments, we take A to be a random matrix constructed by

$A = U\Sigma V^T$ where U and V are random orthogonal matrices and Σ is a diagonal matrix of singular values such that $\kappa_2(A) = 10^6$ ($\kappa_2(A) = \|A\|_2\|A^{-1}\|_2$ is the spectral condition number of A). We take a single random right-hand side vector b with entries sampled from $N(0,1)$ and solve the problem $\min_x \|Ax - b\|_2$. We link BLENDENPIK and our code against MATLAB's LAPACK and the standard FFTW library. For timing results, we run each routine a number of times to "warm-up" any JIT-compiled MATLAB code and then run a number of timed samples.

Our code is designed to scale up to multiple threads on a single machine, using multi-threaded versions of LAPACK and FFTW, but BLENDENPIK currently uses only a single thread for their FFTW calls. We note that it would be straightforward to extend BLENDENPIK to use multi-threaded FFTW calls, but mixing is hardly the dominant cost of BLENDENPIK, so one would not expect see a large improvement in runtimes. Nevertheless, we perform the following timing experiments using only a single thread in order to compare BLENDENPIK and `RURV_ROS` fairly.

Figures 1.3 and 1.4 show the average runtime for random A with $\kappa_2(A) = 10^6$ of various sizes. We consider quite underdetermined, slightly underdetermined, slightly overdetermined, and quite overdetermined examples. For underdetermined systems, computing the basic solution with `RURV_ROS` is slightly faster than BLENDENPIK, which computes the minimum norm solution. Using `RVLU_ROS` to compute the minimum norm solution is moderately slower than BLENDENPIK. Note that error bars showing the minimum and maximum runtime in the sample are used in Figures 1.3 and 1.4. Since the variance in runtime is so low, the error bars are barely visible, but this also means that the differences in runtime between methods is significant,

Figure 1.5 shows the ratio of the runtimes for `RURV_ROS` and `RVLU_ROS` for both 1 and 8 threads. For slightly underdetermined systems, the speedup factor approaches 4 (i.e., for large m, using 8 threads runs about 4 times faster than using only 1 thread). For slightly overdetermined systems, the speedup factor approaches 4. Although we see speedup factors of less than the ideal factor of 8, our implementation does parallelize nicely. The speedup factors may very well continue to increase outside of the range of matrices we tested, which was limited by having only 32 GB of RAM on the machine.



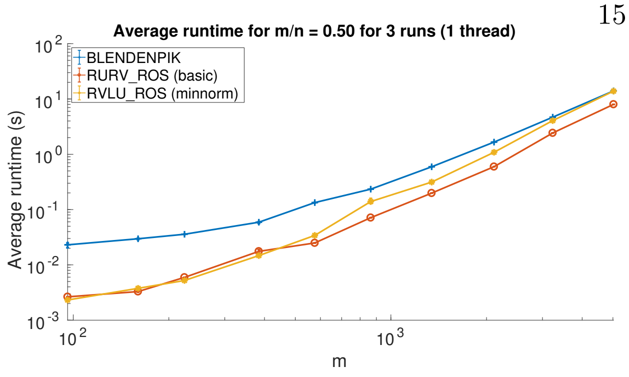
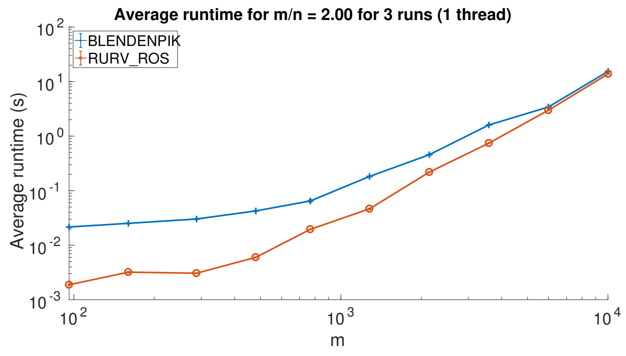


Figure 1.3: Average runtime for BLENDENPIK, `RURV_ROS`, and `RVLU_ROS` approaches on underdetermined systems. The `RVLU_ROS` based minimum norm solution is eventually slower than BLENDENPIK, which also computes the minimum norm solution. The basic solution computed with `RURV_ROS`, being simpler to compute, is a consistently faster than the `RVLU_ROS` based minimum norm solution.



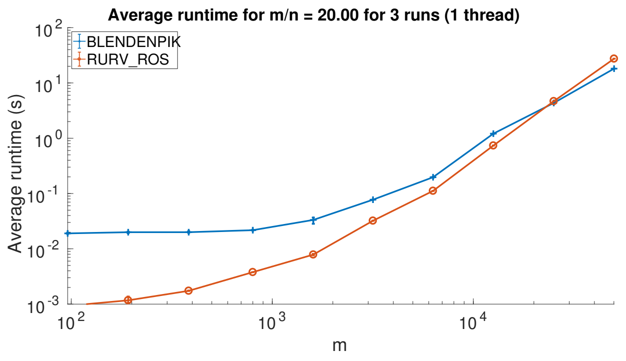
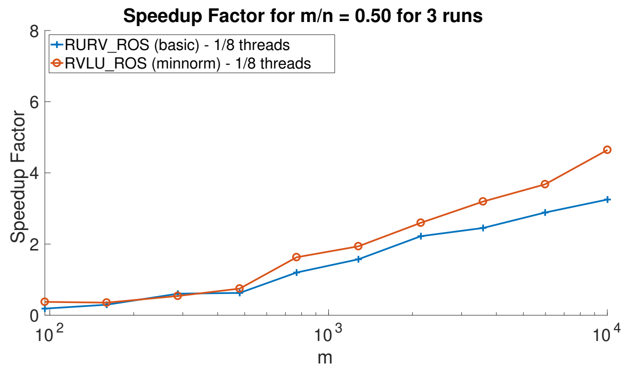


Figure 1.4: Average runtime for BLENDENPIK and the `RURV_ROS` approach on overdetermined systems. `RURV_ROS` compares favorably to BLENDENPIK for matrices of moderate size. For larger m/n (i.e., more highly overdetermined) and larger matrices, BLENDENPIK performs better than `RURV_ROS`, which we expect from [8], which shows that BLENDENPIK tends to outperform QR-based solvers on highly overdetermined systems.

Before we continue with our discussion of our experiments, we make a brief note on implementing `RURV_ROS` on a distributed memory machine. The two major steps of `RURV_ROS` are mixing and the unpivoted QR factorization, which can be handled by FFTW and ScaLAPACK, respectively. FFTW has a distributed memory implementation using MPI, and the interface is very similar to the shared memory interface. ScaLAPACK's routine `p_geqrf` performs unpivoted Householder QR, but uses a suboptimal amount of communication. In [26] communication-avoiding QR (CAQR) was introduced. CAQR sends a factor of $\mathcal{O}(\sqrt{mn/P})$ fewer messages than `p_geqrf`



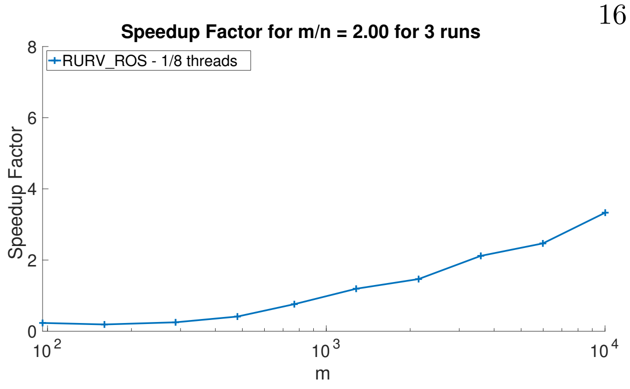


Figure 1.5: Ratios of average runtime for the `RURV_ROS` and `RVLU_ROS` approaches on slightly under- and overdetermined systems using 1 and 8 threads. We plot the ratio of the runtime using 1 thread over the runtime using 8 threads, so speedup factors greater than 1 correspond to an improvement when running in parallel.

(where P is the total number of processors in the grid), and achieves the optimal amount of communication (up to polylogarithmic factors). The reduction in communication is predicted to result in significantly faster factorization runtimes. Using FFTW, ScaLAPACK, or other, existing codes as building blocks, we expect that `RURV_ROS` can be implemented efficiently and straightforwardly for distributed memory environments.

1.3.3 Example - Correlated Columns and the Basic Solution

The dominant cost of using a URV factorization to compute the basic solution to an underdetermined $m \times n$ system ($m < n$) is computing a QR factorization of $\hat{A}(:, 1{:}m)$. Thus, it is asymptotically cheaper than the minimum norm solution, which uses an LQ factorization of the full $\hat{A}$. Specifically, computing the basic solution costs $\mathcal{O}(m^3, mn \log n)$ FLOPs, while computing the minimum norm solution costs $\mathcal{O}(m^2 n)$ FLOPs [42]. We have seen in Figure 1.3 that computing the basic solution is significantly faster than computing the minimum norm solution with `RURV_ROS`, and even slightly faster than BLENDENPIK, which also computes the minimum norm solution.

The following simple example shows that finding the basic solution using unpivoted QR is numerically unstable for some least-squares problems. Consider

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & \epsilon & 1 \end{bmatrix},$$

with $\epsilon \ll 1$ and consider solving $\min_x \|Ax - b\|_2$. Note that A is full-rank and that it can be analytically verified that $\kappa_2(A) \sim 1 + \sqrt{2}$ as $\epsilon \to 0$, so A is very well conditioned for $\epsilon \ll 1$. However, columns 2 and 3 are increasingly correlated as $\epsilon \to 0$. Since A is already upper trapezoidal, an unpivoted QR factorization does not change A when finding R. When finding the basic least-squares solution in this manner, it transpires that we solve the linear system

$$R_{11}x_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & \epsilon \end{bmatrix} \begin{bmatrix} x_{1,1} \\ x_{1,2} \\ x_{1,3} \end{bmatrix} = Q^T b = b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}.$$

However, this system has $\kappa_2(R_{11}) \sim 2/\epsilon$ as $\epsilon \to 0$, and so becomes ill-conditioned as columns 2 and 3 become more correlated. Even for this small system, finding the basic solution $\hat{x}$ with an unpivoted QR factorization leads to a large residual $\|A\hat{x} - b\|_2$.

This can be fixed by using `QRCP` on all of A instead of unpivoted QR on $A(:, 1{:}m)$. Note that using `QRCP` on $A(:, 1{:}m)$ will encounter similar ill-conditioning problems, as doing so will not allow `QRCP` to pivot in column 4. Using a URV factorization which mixes in column 4 of A also leads to a much better conditioned R_{11}, alleviating the issue of correlated columns when using the basic solution. This simple example can be extended to larger matrices, as we show next.

Consider an $m \times (n - p)$ matrix A with $m < n$ such that each element of A is sampled from $N(0, 1)$. We augment A by adding p randomly selected columns of A to the end of A, making A $m \times n$. The augmented A has p perfectly correlated columns, so we add a small amount of $N(0, \sigma^2)$ noise to the augmented A so the correlation is not perfect. We then randomly shuffle the columns. Listing 1.2 gives MATLAB code to generate such a matrix, which tends to be well-conditioned. If the final permutation places a pair of highly correlated columns in the first m columns of A, finding the basic solution $\hat{x}$ with unpivoted QR will produce an ill-conditioned R_{11}, leading to a large residual $\|A\hat{x} - b\|_2$. This can be solved by mixing with RURV_Haar or RURV_ROS, or computing the (more costly) minimum norm solution.

Listing 1.2: MATLAB code to generate a matrix with a few correlated columns

```
m = 1000; n = 1500;
p = 10; sigma = 1e-4;

A = randn(m,n-p);
perm = randperm(n-p,p);
A = [A A(:,perm)];
perm = randperm(n);
A = A(:,perm) + sigma*randn(m,n);
```

Table 1.1 shows the residuals and runtimes on a matrix generated with Listing 1.2. We tested unpivoted QR, QRCP, BLENDENPIK, RURV_Haar, and RURV_ROS. We use unpivoted QR on only the first m columns of A, so it produces a significantly larger residual than the other methods. Note that BLENDENPIK actually computes the minimum norm solution and is included for reference. RURV_Haar and RURV_ROS both compute basic solutions with acceptably small residuals. As expected, RURV_ROS is considerably faster than RURV_Haar, but slightly slower than plain, unpivoted QR.

It is interesting to note that the norm of the mixed basic solution is considerably smaller than the unmixed basic solution. Table 1.2 shows the same comparison for $p = 0$ correlated columns, where we see that the mixed and unmixed basic solutions have norms that are not unreasonably large. The norms of the mixed basic solutions are on the same order for the cases of $p = 10$ and $p = 0$ correlated columns, unlike QR and QRCP.

1.4 Experimental Comparison of RURV_Haar and RURV_ROS

In this section we experiment with a variety of QR and URV factorizations, some of which are known to be rank-revealing. In Subsection 1.4.1 we experiment with how the rank-revealing conditions (1.1) and (1.2) scale with increasing n. Our chief interest here is the comparison of

Method	Residual - $\|A\hat{x} - b\|_2$	Norm - $\|\hat{x}\|_2$	Time (s)
`QR`	3.0×10^{-9}	1.3×10^{5}	0.04
`QRCP`	2.5×10^{-13}	5.8×10^{0}	0.19
BLENDENPIK	1.4×10^{-13}	1.4×10^{0}	0.16
`RURV_Haar`	5.8×10^{-12}	1.5×10^{2}	0.52
`RURV_ROS`	1.4×10^{-12}	4.3×10^{1}	0.10

Table 1.1: Comparison of basic solution residuals for the 1000×1500 matrix from Listing 1.2 with $p = 10$ correlated columns. As expected, unpivoted `QR` has a relatively large residual, while the other methods perform better. Note that BLENDENPIK computes the minimum norm solution.

`RURV_Haar` and `RURV_ROS`, to determine whether `RURV_ROS` may potentially be rank revealing.

We can use `RURV_ROS` to form low-rank approximations by performing the mixing and pre-sort as usual, but only performing k steps of the QR factorization, yielding a rank-k approximation. The mixing step costs $\mathcal{O}(mn \log n)$ FLOPs as usual, but now the partial QR factorization costs only $\mathcal{O}(mnk)$ FLOPs. In Subsection 1.4.3, we investigate pairing QR and URV factorizations with Stewart's QLP approximation to the SVD [86]. One can use the QLP approximation to obtain an improved rank-k approximation by truncating the L factor.

1.4.1 Scaling of Rank-Revealing Conditions

It was shown in [25, 9] that `RURV_Haar` produces a strong rank-revealing factorization with high probability. `RURV_ROS` simply replaces Haar mixing with ROS mixing and adds a pre-sort before the unpivoted QR factorization, so we expect `RURV_ROS` to behave similarly to `RURV_Haar`. Specifically, we hope that `RURV_ROS` obeys the strong rank-revealing conditions (1.1) and (1.2) in a manner similar to `RURV_Haar`.

We experimentally test the scaling of the ratios $\sigma_i(A)/\sigma_i(R_{11})$, $\sigma_j(R_{22})/\sigma_{k+j}(A)$, and the norm $\|R_{11}^{-1}R_{12}\|$ to determine if they appear to be bounded above by a slowly-growing polynomial. In Figure 1.6 we take A to be a random $m \times m$ matrix of rank $k \approx m/2$. The matrix A is formed as $A = U\Sigma V^T$, where U and V are Haar random orthogonal matrices, $\Sigma = \text{diag}(\sigma_1, ..., \sigma_m)$, and the σ_i decay slowly until $\sigma_{m/2}$, where there is a gap of about 10^{-10}, after which the σ_i decay slowly again. We sample sizes m from 10 to 1000; for each m, we generate five instantiations

Method	Residual - $\|A\hat{x} - b\|_2$	Norm - $\|\hat{x}\|_2$	Time (s)
`QR`	5.4×10^{-13}	1.8×10^{1}	0.04
`QRCP`	2.5×10^{-13}	6.1×10^{0}	0.20
BLENDENPIK	1.3×10^{-13}	1.4×10^{0}	0.15
`RURV_Haar`	4.8×10^{-12}	1.2×10^{2}	0.52
`RURV_ROS`	1.3×10^{-12}	3.9×10^{1}	0.10

Table 1.2: Comparison of basic solution residuals for the 1000×1500 matrix from Listing 1.2 with $p = 0$ correlated columns. All methods perform well, and that the two URV-based methods compute mixed basic solutions with norms on the same order as in the previous case with correlated columns.

of the matrix A, perform a variety of factorizations for each A, and compute the conditions (1.1) and (1.2) for each factorization. For plotting, we plot the maximum over the five instantiations of $\max_i \sigma_i(A)/\sigma_i(R_{11})$, $\max_j \sigma_j(R_{22})/\sigma_{k+j}(A)$, and $\|R_{11}^{-1}R_{12}\|$.

We use the LAPACK routine `dgejsv` to compute singular values of the test matrices (when the exact singular values are unknown) and in the computation of the ratios (1.1). `dgejsv` implements a preconditioned Jacobi SVD algorithm, which can be more accurate for small singular values [27, 32, 33]. Specifically, if $A = DY$ (or $A = YD$), where D is a diagonal weighting matrix and Y is reasonably well-conditioned, `dgejsv` is guaranteed to achieve high accuracy. The relative error of the singular values computed with the preconditioned Jacobi method are $\mathcal{O}(\epsilon)\kappa_2(Y)$, whereas the relative errors as computed with a QR-iteration based SVD are $\mathcal{O}(\epsilon)\kappa_2(A)$ [32, 28]. This fact is particularly relevant when we test with the Kahan matrix, which is discussed later in the section. Even when A is not of the form $A = DY$, $A = YD$, or even $A = D_1YD_2$, it is expected that `dgejsv` returns singular values at least as accurate as a QR-iteration based SVD.

We test `QRCP`, `RURV_Haar`, `RURV_ROS`, `HQRRP` from [65], which uses random projections to select blocks of pivots, and `DGEQPX` from [15], which is known to be a rank-revealing QR. Note that `HQRRP` is intended to efficiently produce a column-pivoted Householder QR; it is not a rank-revealing QR, but it tends to be rank-revealing in practice, like `QRCP`.

Figure 1.6 shows the rank-revealing conditions for A a random $m \times m$ matrix of rank $k \approx m/2$. The three QR factorizations we test, `QRCP`, `HQRRP`, and `DGEQPX`, perform very well, meaning that the sampled rank-revealing conditions appear to be bounded above by a slowly growing polynomial.

Note that Figure 1.6 uses a log-log scale, on which polynomial growth appears linear. As we expect, `RURV_ROS` performs about as well as `RURV_Haar`. With the exception of a few points, `RURV_Haar` and `RURV_ROS` appear to be bounded above by a slowly growing polynomial, albeit a significantly larger polynomial than for the three QR factorizations. The exceptions may very well be points where `RURV_Haar` or `RURV_ROS` failed to produce a rank-revealing factorization for at least one of the five sampled A matrices.

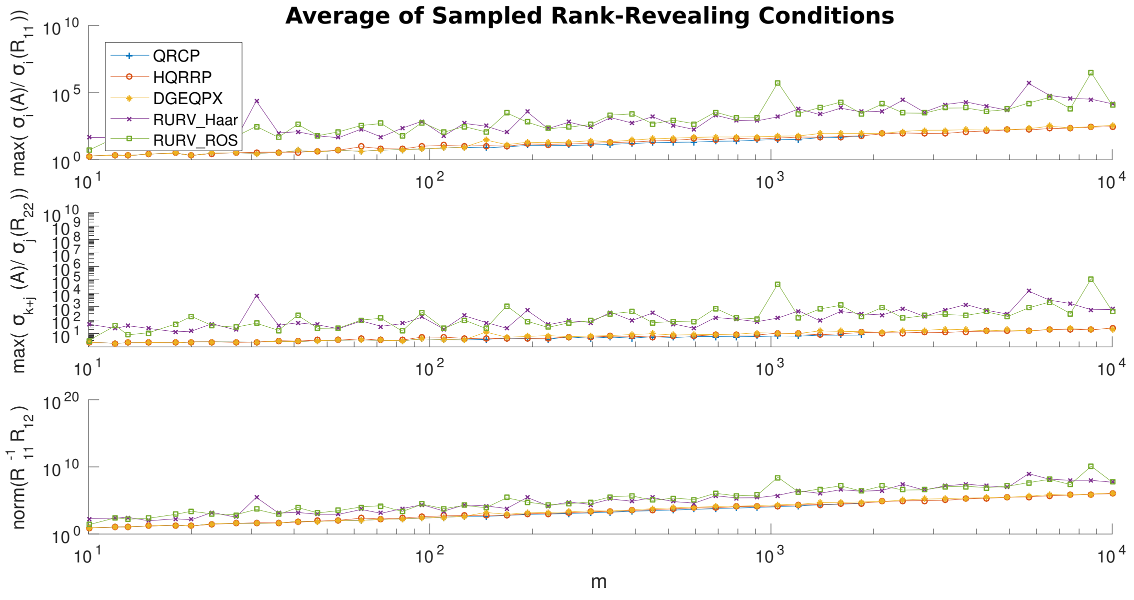


Figure 1.6: Maximum of the sampled values of the rank-revealing conditions from Subsection 1.2.1 for five random $m \times m$ matrices of numerical rank $m/2$. The three QR factorizations exhibit growth that is clearly bounded by a slowly growing polynomial (linear in a log-log plot). `RURV_Haar` and `RURV_ROS` also appear to exhibit bounded growth, with only a few exceptions; recall that `RURV_Haar` produces a strong rank-revealing factorization with high probability, not deterministically.

Figure 1.7 shows the rank-revealing conditions with A the $m \times m$ Kahan matrix and k chosen to be $m-1$. The Kahan matrix is a well-known counterexample on which `QRCP` performs no pivoting in exact arithmetic [31]. We use the Kahan matrix (with perturbation) as described in [28]. The

$m \times m$ Kahan matrix is formed as

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & \cdots & 0 \\ 0 & s & 0 & \cdots & \cdots & 0 \\ 0 & 0 & s^2 & \ddots & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \cdots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & \cdots & 0 & s^{m-1} \end{bmatrix} \begin{bmatrix} 1 & -c & -c & \cdots & \cdots & -c \\ 0 & 1 & -c & \cdots & \cdots & -c \\ 0 & 0 & 1 & \ddots & \cdots & -c \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \cdots & \ddots & \ddots & -c \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}, \tag{1.5}$$

where $s^2 + c^2 = 1$ and $s, c \geq 0$. When using QRCP to compute the factorization

$$A\Pi = QR = Q \begin{bmatrix} R_{11} & R_{12} \\ & R_{12} \end{bmatrix}, \quad R_{11} \in \mathbb{R}^{k \times k}, R_{12} \in \mathbb{R}^{k \times (m-k)}, R_{22} \in \mathbb{R}^{(m-k)\times(m-k)},$$

it is known that $\sigma_k(A)/\sigma_k(R_{11}) \geq \frac{1}{2}c^3(1+c)^{m-4}/s$ for $k = m-1$, and $\sigma_k(R_{11})$ can be much smaller than $\sigma_k(A)$ [45]. That is, QRCP does not compute a rank-revealing factorization, as the first ratio in (1.1) grows exponentially for $i = k = m-1$. To prevent QRCP from pivoting on the Kahan matrix in finite arithmetic, we multiply the jth column by $(1-\tau)^{j-1}$, with $1 \gg \tau \gg \epsilon$ [31, 28]. In our tests, we pick $c = 0.1$ and $\tau = 10^{-7}$.

The most apparent feature of Figure 1.7 is that the rank-revealing conditions for QRCP grow exponentially. This is a known feature of the Kahan matrix, and shows that QRCP is not **strictly speaking** a rank-revealing QR (in practice, however, it is still used as a rank-revealing factorization). Moreover, the Kahan matrix is so bad for QRCP, we believe dgejsv cannot accurately compute the singular values in the ratios $\sigma_i(A)/\sigma_i(R_{11})$ and $\sigma_j(R_{22})/\sigma_{k+j}(A)$. As m grows, the right-hand matrix in (1.5) becomes increasingly ill-conditioned, and we see the exponential growth in Figure 1.7 stop around $m = 10^3$. In infinite precision arithmetic, the exponential growth should continue, so we stop testing at $m \approx 400$. As expected, the rank-revealing conditions for RURV_ROS scale in the same manner as RURV_Haar, giving credence to our thought that RURV_ROS is rank-revealing with high probability.

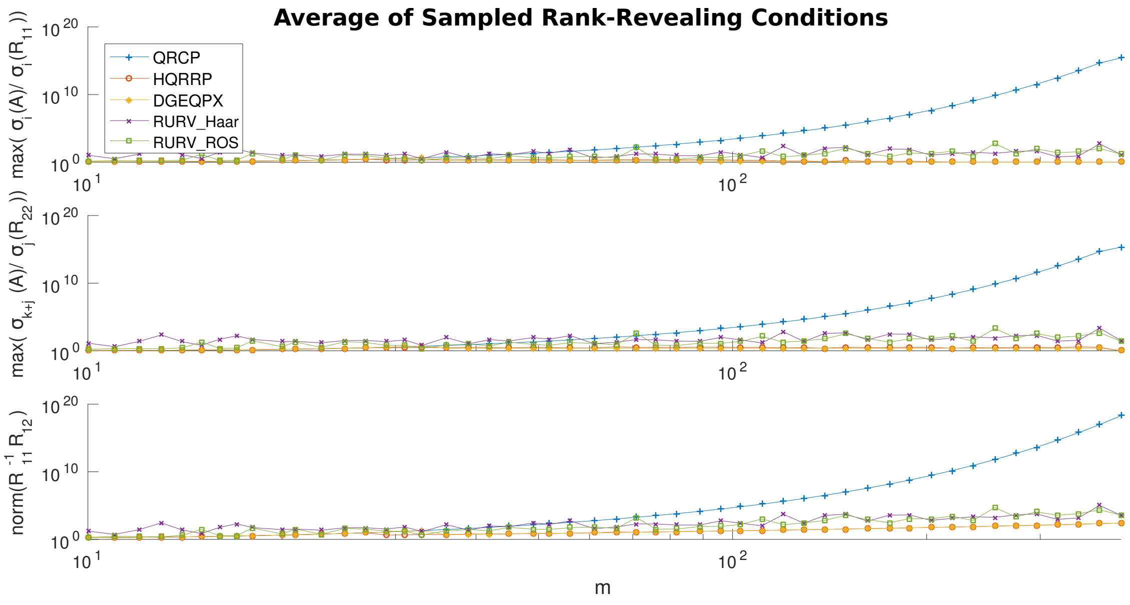


Figure 1.7: Maximum of the sampled values of the rank-revealing conditions from Subsection 1.2.1 on the $m \times m$ Kahan matrix. As expected, `QRCP` performs very poorly, with all conditions scaling exponentially. Again, we see that `RURV_Haar` and `RURV_ROS` behave similarly.

1.4.2 Accuracy of R-Values

Another test we perform involves the accuracy of $|R(i,i)|$ in predicting $\sigma_i(R)$ ($R(i,i)$ is the ith diagonal element of the upper-triangular factor from a QR or URV factorization). Following [28], we call the $|R(i,i)|$ R-values. The R-values can be used as a rough estimate of the singular values. A better approximation is to use Stewart's QLP factorization [86], which we discuss in Subsection 1.4.3. Nevertheless, it is beneficial to investigate the behavior of the R-values.

We test `QRCP`, `RURV_Haar`, and `RURV_ROS` on the first 18 test matrices from Table 2 of [28] (most matrices are from [49, 45]). In Figure 1.8, we plot the minimum, median, and maximum of the ratios $|R(i,i)|/\sigma_i$ for the 18 test matrices. For each matrix, we let `r` and `s` be the vectors of R-values and singular values, respectively; we plot `min(r./s)`, `median(r./s)`, and `max(r./s)` (using MATLAB syntax). We see that `QRCP` produces ratios that are at most just over an order of magnitude away from one. `RURV_Haar` produces slightly worse ratios, which seem to be spread over about two orders of magnitude away from one. `RURV_ROS` with one mixing iteration produces ratios comparable to `RURV_Haar`, with the exception of matrix 15, SPIKES. For matrix 15, the extreme ratios are significantly larger than on the rest of the test set. Adding a second mixing iteration brings the ratios back down to a couple orders of magnitude away from one, but does not improve the ratios for the other matrices beyond what is accomplished with a single mixing. We can also find a bound for the ratios obtained with QR and URV factorizations.

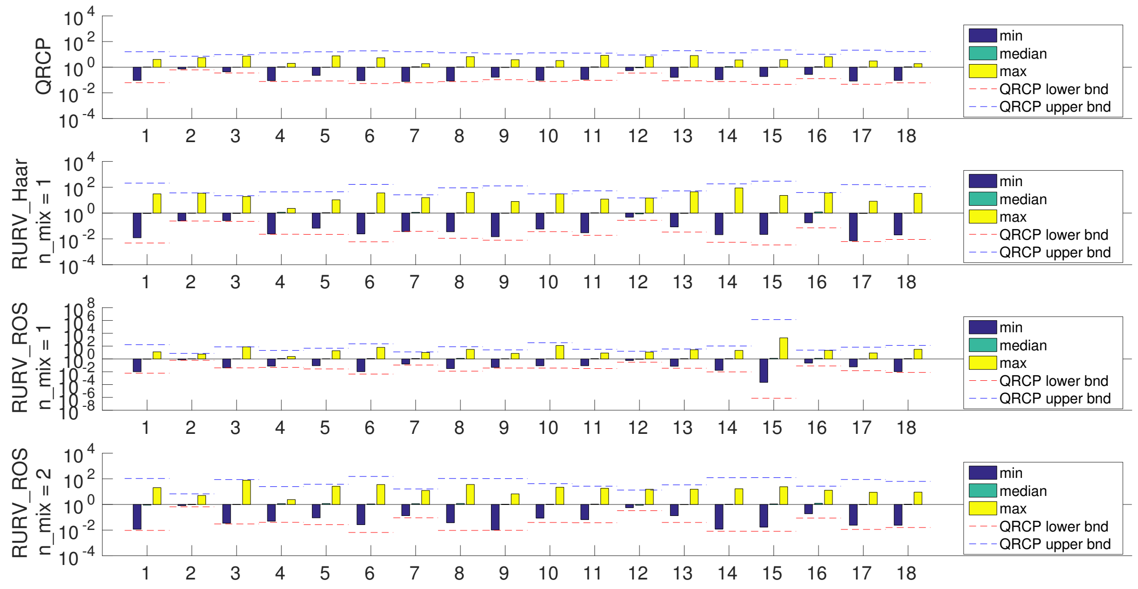


Figure 1.8: Ratios of $|R(i,i)|/\sigma_i(R)$ for the 18 matrices in Table 2 of [28]. The abscissa is the index of the matrix in the test set. For matrix 15 (SPIKES), `RURV_Haar` produces ratios on par with the rest of the test set. For `RURV_ROS`, however, using only 1 mixing step produces very bad max/min ratios; using two two mixing steps produces better ratios, but more mixing steps doesn't appear to yield further improvements. For the other 17 matrices, `RURV_ROS` produces ratios comparable with `RURV_Haar`.

Let D be the diagonal part of R obtained from a QR or URV factorization, and define Y via $R = DY^T$. This results in the factorization $A\Pi = QDY^T$ for `QRCP` and $A = UDY^TV$ for `RURV_Haar` and `RURV_ROS`. For `QRCP`, the diagonal elements of R are non-negative and sorted in decreasing order; this is not guaranteed for `RURV_Haar` or `RURV_ROS`. It follows from the Courant-Fischer minimax theorem [42] that `QRCP` has the bounds

$$\frac{1}{\|Y\|} \leq \frac{R(i,i)}{\sigma_i} \leq \|Y^{-1}\|. \tag{1.6}$$

For `RURV_Haar` and `RURV_ROS`, let ρ_i be the ith largest (in absolute value) diagonal element of R.

For RURV_Haar and RURV_ROS, we have the bounds

$$\frac{1}{\|Y\|} \leq \frac{|\rho_i|}{\sigma_i} \leq \|Y^{-1}\|.$$

In addition to the minimum, median, and maximum values of $|R(i,i)|/\sigma_i$ for each matrix, we plot the bounds (1.6) for both QRCP and the two RURV factorizations. Even though the two RURV factorizations are not guaranteed to be bound by (1.6), since it is a strong rank-revealing URV, we expect the R-values to somewhat closely approximate the singular values and approximately obey the QRCP bounds. With the exception of matrix 12, formed as `A=2*rand(n)-1` in MATLAB, we see this behavior in Figure 1.8, and we again see RURV_ROS behaving similarly to RURV_Haar.

1.4.3 Experiments With the QLP Approximation

The QLP factorization was introduced by G.W. Stewart as an approximation to the SVD in [86]. The idea of the pivoted QLP factorization is to use QRCP to find R-values, and then improve the accuracy (by a surprising amount) by performing another QRCP on R^T. This results in a factorization of the form $A = Q_1 \Pi_2 L Q_2^T \Pi_1$, where L is lower triangular. Following [28], we call the diagonal elements of the L matrix L-values. In Stewart's original experiments, it was found that L-values approximate the singular values significantly more accurately than the R-values. Also, the accuracy seemed intimately tied to using QRCP for the first factorization, but that unpivoted QR could be used in the second QR factorization with only cosmetic differences. It was later shown that the QLP factorization can be interpreted as the first two steps of a QR-style SVD algorithm [53].

We experiment with QLP-style factorizations by performing QR, QRCP, RURV_Haar, or RURV_ROS, and following up with an unpivoted QR to compute the L-values. We denote such a factorization as {factorization}+QLP (e.g., QRCP+QLP). For the RURV factorizations, this QLP-style factorization is of the form $A = ULQ^TV$. Figure 1.9 shows the singular values and L-values for a random matrix of the form $A = U\Sigma V^T$, where U, V are Haar random orthogonal, and the singular values are chosen to decay slowly, have a gap of approximate width 10^{-1}, and decay slowly again. We see that all QLP-style factorizations, including `QR`+QLP, identify both the location and magnitude of the gap quite accurately.

Also shown in Figure 1.9 are the L-values for the Devil's stairs matrix, which is a particularly difficult example for rank-revealing factorizations. The Devil's stairs matrix is discussed in [86, 28], and is formed with $A = U\Sigma V^T$, with U, V Haar random orthogonal and Σ controlling the stair-step behavior. Of all the factorizations, `QRCP`+QLP performs the best, accurately identifying the location and size of the singular value gaps. `QR`+QLP, `RURV_Haar`+QLP, and `RURV_ROS`+QLP all provide evidence for the existence of singular value gaps, but none is able to identify the precise location and size of the gaps.

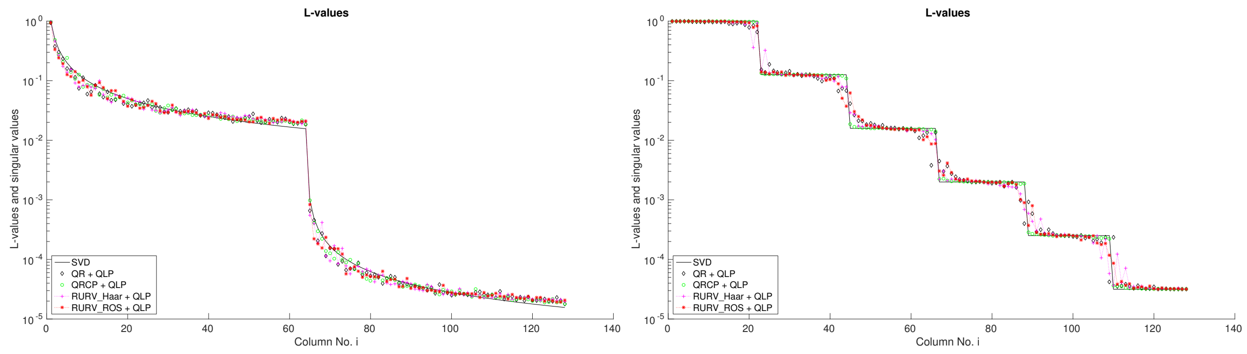


Figure 1.9: L-values from various QLP factorizations on a random 128×128 matrix with slowly decaying singular values and a small gap of approximate size 10^{-1} and the Devil's stairs with gaps of approximate size 10^{-1}. In each case, the legend name is of the form {factorization}+QLP, where the second factorization is always unpivoted `QR`.

Figure 1.10 shows the minimum, median, and maximum L-values for 25 realizations of the Devil's stairs matrix. We again use `QR`, `QRCP`, `RURV_Haar`, and `RURV_ROS`, followed by `QR` to form the QLP factorization. It is clear that `QRCP`+QLP produces the best L-values; `RURV_Haar`+QLP and `RURV_ROS`+QLP generate L-values visually similar to those produced with `QR`+QLP. The L-values are smeared around the jumps for `QR` and the two RURV factorizations, but the L-values have a lower variance around the middle of the flat stairs. The variance of the L-values around the gaps appears visually similar for `QR`+QLP, `RURV_Haar`+QLP, and `RURV_ROS`+QLP. For `QR`+QLP, the variance is explained only by the Haar random orthogonal matrices used to construct the

Devil's stairs matrix; for `RURV_Haar`+QLP and `RURV_ROS`+QLP, the variance is a combination of the random Devil's stairs matrix and the random mixing.

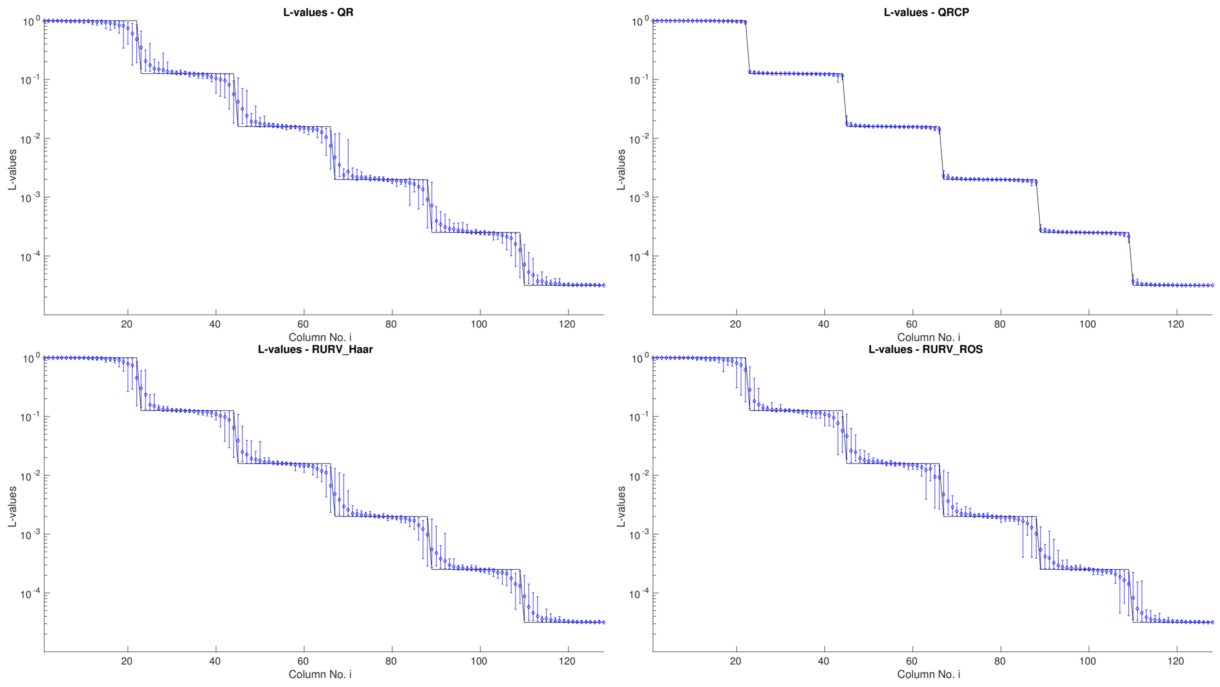


Figure 1.10: Min/Median/Max L-values for 25 runs of a randomly generated 128×128 Devil's stairs matrix with jumps of approximate size 10^{-1}. `QR`, `RURV_Haar`, and `RURV_ROS` appear to predictably show the presence and approximate location of the gaps, but are not accurate enough to estimate the size of the gaps. `QRCP` performs very well, and accurately shows the location and size of the gaps.

1.5 Summary

We have modified `RURV_Haar`, a strong rank-revealing factorization with high probability, to use random orthogonal mixing (ROS) instead of Haar orthogonal matrix mixing. The new algorithm, `RURV_ROS`, applies the mixing matrix implicitly and quickly, as opposed to `RURV_Haar`, where the mixing matrix is generated with an unpivoted `QR` and applied with dense matrix-matrix multiplication. With both randomized URV factorizations, one of the principal attractions is the use of cheaper, unpivoted `QR`, instead of relying on the more expensive `QRCP`. The ansatz is that mixing reduces the variance of the column norms, reducing the effect that column pivoting would have, and so we can forgo pivoting and use a cheaper, unpivoted QR. A URV factorization can be used in many applications that call for a QR, and since the dominant asymptotic cost of `RURV_ROS` is the same as unpivoted `QR`, `RURV_ROS` has the potential to be used as a safer alternative to unpivoted `QR`. We have considered only real matrices, but the extension to complex matrices and transforms is natural.

We experiment with using `RURV_ROS` to solve over- and underdetermined least-squares problems. Using a URV factorization to solve least-squares is very similar to using a QR factorization. Our implementation of `RURV_ROS` even performs comparably to BLENDENPIK, which uses mixing and row sampling to create a preconditioner for LSQR.

When one wants a solution to an underdetermined system, but does not need the minimum norm solution, `RURV_ROS` can be used to find a basic solution slightly faster than BLENDENPIK, which computes the minimum norm solution. Additionally, if even a few of the columns of the A matrix are highly correlated, using unpivoted `QR`, or `QRCP` on the first m columns, can lead to an inaccurate basic solution; using `RURV_ROS` computes a mixed basic solution with an accurate residual and for which the norm of the solution is only an order of magnitude larger than the minimum norm solution.

Finally, we experiment with the possible rank-revealing nature of `RURV_ROS`. We test the scaling of the rank-revealing conditions (1.1) and (1.2) for `RURV_Haar`, `RURV_ROS`, and a few other QR factorizations, one of which is rank-revealing. The prominent feature of the scaling tests is that `RURV_ROS` behaves very similarly to `RURV_Haar`, which leads us to suspect that `RURV_ROS` produces a strong rank-revealing factorization with high probability. We plan to investigate theoretically the apparent rank-revealing nature of `RURV_ROS`.

Chapter 2

Delay-Decimated Cross-Ambiguity Functions

In this chapter we present new algorithms for computing the discrete, narrowband cross-ambiguity function (CAF) on a downsampled grid of delay values for the purpose of quickly detecting the location of peaks in the CAF surface. Due to the likelihood of missing a narrow peak on a downsampled grid of delay values, we propose methods to make our algorithms robust against missing peaks. To identify peak locations to high accuracy, we propose a two-step approach: first identify a coarse peak location using one of our delay-decimated CAF algorithms, then compute the CAF on a fine, but very small, grid around the peak to find its precise location. Runtime experiments with our C++ implementations show that our delay-decimated algorithms can give more than an order of magnitude improvement in overall runtime to detect peaks in the CAF surface when compared against standard CAF algorithms.

2.1 Introduction

Let $x(t)$ and $y(t)$ be two complex baseband signals that share a common, but unknown, signal in the presence of additive noise. It is well know that that if there is no relative frequency shift of the common signal in $x(t)$ and $y(t)$, a relative delay between $x(t)$ and $y(t)$ can be found via the **cross-correlation function**

$$r(\tau) = \int_0^T x(t)y(t+\tau)^* \mathrm{d}t,$$

where τ is a relative delay and T is the integration time. To find potential relative delays between $x(t)$ and $y(t)$, one computes $|r(\tau)|$ over a suitable range of τ and searches for peaks.

When there is a non-zero relative frequency shift of the common signal in $x(t)$ and $y(t)$, the cross-correlation function may no longer produce a peak at a valid relative delay of $x(t)$ and $y(t)$. Under a narrowband signal model [97], the relative frequency shift can be compensated for by shifting the spectrum of $y(t)$ followed by performing the above correlation processing. However, the relative frequency shift is often unknown and must be estimated jointly with the relative time delay, as argued in [85]. A natural generalization of the cross-correlation function for jointly estimating relative delay and relative frequency shift is the **cross-ambiguity function** (CAF)

$$A(\tau, f) = \int_0^T x(t) y(t+\tau)^* e^{-2\pi i f t} \mathrm{d}t,$$

where f parameterizes a relative frequency shift between $x(t)$ and $y(t)$. To find candidate relative delays and relative frequency shifts, one computes $|A(\tau, f)|$ over a suitable range of τ and f and searches for peaks. Note that $A(\tau, f)$ can be interpreted as the cross-correlation of $x(t)$ and $y(t)e^{2\pi i f t}$.

We will work entirely with sampled data, and therefore use a discrete analogue to $A(\tau, f)$. Let $x[k]$ and $y[k]$ be samples, taken at sample rate f_s, of the complex baseband signals $x(t)$ and $y(t)$, respectively. Following [92], we define the discrete, narrowband CAF as

$$A[j, n] = \sum_{k=0}^{K-1} x[k] y[k+j]^* e^{-2\pi i \frac{kn}{K}}, \tag{2.1}$$

where j represents a relative delay and n represents a relative frequency shift. Note that for any fixed j, $A[j, n]$ can be thought of as the discrete Fourier transform of $x[k]y[k+j]^*$. Indeed, the relative frequency shifts n are computed on the standard DFT grid with spacing f_s/K. Note also that the relative delays j are computed at the input sample rate f_s. For convenience, we will refer to the discrete, narrowband CAF as simply the CAF and drop the word "relative" from relative delay and relative frequency shift when the usage is clear from the context.

When computing the CAF for the purpose of identifying relative delays and frequency shifts between $x[k]$ and $y[k]$, it is wasteful to compute values of the CAF outside the range of plausible values of the relative delay and frequency shift. In typical scenarios, the integration time is often quite large compared to the delay search range, as is the signal bandwidth compared to the frequency shift search range [85]. Therefore we are interested in computing $A[j, n]$ over a somewhat short range of j and a narrow range of n (the shape of the array $A[j, n]$ is usually "tall-skinny").

The standard CAF algorithm for this scenario was originally described heuristically by Stein in [85], and later derived more precisely by Tolimieri and Winograd in [92], where it is called the "Transform Method". It computes $A[j, n]$ precisely under the assumption that the plausible range of n is small compared to the number of integration samples K. We review the derivation of this algorithm in Section 2.2, as we use this method as the standard for comparison and as the basis for our delay-decimated CAF algorithms. Due to the data flow in the algorithm (it effectively uses a matrix transpose operation), it is sometimes referred to as the **corner-turn** CAF algorithm, a term we adopt as well.

The corner-turn CAF algorithm computes values of $A[j, n]$ on the grid on which it is defined in (2.1): the delay dimension has spacing $1/f_s$ and the frequency shift dimension has spacing f_s/K. This allows one to directly determine fine estimates of the delay and frequency shift of a detected peak. Furthermore, it can typically provide multiple samples around the sides of the peak that can be used to interpolate the peak location to sub-grid precision.

2.1.1 A Two-Step Strategy for Peak Detection

Peaks in the CAF typically occupy a small portion of the grid on which the CAF is computed. The work spent computing the CAF in bins not nearby the peak is therefore wasted. To reduce the burden of identifying peak locations, we propose a two-step strategy:

(1) Identify peaks via a CAF computed on a grid that has been downsampled in the delay dimension. We refer to such a CAF as a **delay-downsampled** CAF.

(2) For each peak identified, compute a small portion of the non-delay-downsampled CAF on a fine grid around those peaks, providing more precise peak locations.

For this strategy to be useful, the first step cannot miss peaks that the CAF on fine grid would not miss. Of course, it must also provide computational savings over computing the CAF on the fine grid of the same extent. Computing the first step is the subject of Sections 2.3 and 2.4, where new algorithms are derived that efficiently compute the CAF on a delay-downsampled grid while maintaining a measure of safety to missing peaks.

Note that we cannot necessarily decimate $x[k]$ and $y[k]$ to a lower rate, which would serve to downsample the delay dimension of the CAF. We assume that $x[k]$ and $y[k]$ are sampled at a rate commensurate with their bandwidths (i.e., at least satisfying the Nyquist criterion [59]). Were we to lowpass filter and downsample $x[k]$ and $y[k]$, we would lose energy in their spectra that would otherwise correlate and provide useful integration gain. Once a coarse peak location is identified, it is relatively inexpensive to compute a very small CAF (e.g., via the corner-turn algorithm) on a fine grid in order to produce a precise estimate of the peak location. An example will be given in Section 2.5 that shows the cost of such a small CAF on a fine grid is quite inexpensive.

2.1.2 Related Work

One simple way to coarsen the frequency shift grid is to simply integrate over a shorter time period. This decreases the effective processing gain, so this can be undesirable. In [85], Stein proposed forming a coherent sum of short-integration CAFs, which has the effect of coarsening the frequency shift grid while using the same total integration time. This maintains the variance of the delay estimate, but has the side-effect of increasing the variance of the frequency shift estimate. The idea was later expanded upon in [96] to approximate the discrete, wideband CAF. This coherent sum idea can combined with our delay-decimated CAF algorithms, resulting in a coarsened delay and frequency shift grid.

Stein proposed a coarse mode and fine/tracking mode in [85]. The coarse mode processing uses a short integration in order to ease the computational burden when searching for the general location of the peak. The fine mode then dramatically increases the integration time and uses a corner-turn-like algorithm to produce fine estimates of the peak location. Our delay-decimated CAF algorithms can cheaply utilize the full integration time one would use for the fine mode, which gives our algorithms an improvement in SNR over using a shorter integration time.

Auslander and Tolimieri proposed methods for computing decimated CAFs in [6]. Their methods compute CAFs that are decimated in both the delay and frequency shift dimensions, and they propose a procedure that allows for different decimation factors for both the delay and frequency shift dimension. However, a major downside of their algorithms (when used in typical scenarios) is that they compute delays and frequency shifts over the full range of possible values (i.e., all K possible frequency shifts). Since we focus on the scenario of a relatively narrow frequency shift range, we do not consider their method further in this chapter.

2.1.3 Organization and Notation

The remainder of the chapter is devoted to the description of the corner-turn CAF algorithm and our two new algorithms. In Section 2.2 we describe the corner-turn CAF algorithm, which we call `TransformCAF`, as it is called the "Transform Method" in [92]. In Section 2.3 we describe one of our new algorithms, `TransformCAF(delay_dec=`P`)`, which is based on the corner-turn CAF algorithm. We also discuss a simple, but effective, trick to reduce the likelihood of missing a peak caused by computing the CAF on a delay-downsampled grid. When using this trick, or another strategy to increase the robustness of a delay-downsampled CAF, we refer to the CAF as a **delay-decimated** CAF. In Section 2.4 we propose `TransformCAF(delay_comp=`P`, gamma=`γ`)`, which uses a more sophisticated, but costly, method to prevent missing peaks. Finally, Section 2.5 presents a brief runtime experiment with our C++ implementations and some future directions of research.

Throughout the chapter, we will use the notation $u[k]$ to represent a sampled data signal. We overload the notation $u[k]$ to mean both the single element of the signal at sample k and also the vector of data $\{u[k]\}_{0\leq k\leq K-1}$. The particular meaning used should be clear from the context of the usage. Similarly, for two-dimensional arrays, such as $u[j,k]$, we use $u[j,k]$ to refer to either the single element of the array at position (j,k) or the full two-dimensional array.

We will also be slightly lazy with the bounds of the index k, allowing it to be negative or greater than $K-1$. In such a case, we take $u[k]=0$, effectively zero-padded $u[k]$ appropriately.

For example, the convolution (with both initial and final transients)

$$v[k] = \sum_{m=0}^{M-1} h[m]u[k-m], \quad \text{for } 0 \le k \le K+M-1$$

uses this implicit zero padding on both the beginning and end of $u[k]$.

We denote delay-downsampled CAFs as $A[j'P, n]$, where P is the delay-downsampling factor and $A[j, n]$ is the full CAF with all rows computed. We use this notation to keep track of the delay-downsampling factor in a concise manner, instead of introducing a separate notation to keep track of P. It should be noted that when we compute a delay-downsampled CAF $A[j'P, n]$, we don't actually compute the rows between $j'P$ and $(j'+1)P$. Assuming $A[j, n]$ has J rows, we occasionally refer to $A[j'P, n]$ as having J/P rows and reference the row indexed by $j'P$ as row j' of $A[j'P, n]$ (cf. the end of Subsection 2.4.1).

2.2 The Corner-Turn CAF Algorithm

In this section we repeat the derivation of Tolimieri and Winograd's "Transform Method" from [92], also known as the corner-turn CAF. To simplify the derivation, we make minimal assumptions about the lowpass filter $h[m]$ and drop the filter passband correction from the derivation (though we include it in our C++ implementation).

Let us first be more precise about the definition of the CAF and the range of delays and frequency shifts of interest. We seek to compute

$$A[j, n] \overset{\text{def}}{=} \sum_{k=0}^{K-1} x[k]y[k+j]^* e^{-2\pi i \frac{kn}{K}} \tag{2.2}$$

over the range of delays $0 \le j \le J$ and frequency shifts $-N \le n \le N$. We seek to compute frequency shifts only over a narrow band relative to the integration time, meaning $N \ll K$.

Taking the DFT interpretation of $A[j, n]$, we see that we seek only the "central" $2N + 1$ bins of the DFT. The remaining bins, of the K total, are discarded. Since $N \ll K$, we are keeping a small minority of the K bins, so naïvely computing length-K DFTs is very wasteful.

2.2.1 The Zoom FFT

The zoom FFT (also known as the limited range DFT [22], zoom transform, or spectral vernier) is a technique to compute a subset of **contiguous** bins of a large DFT [59]. Due to our use in Section 2.3 of another algorithm that we call the algebraic zoom FFT, we call this zoom FFT the "filter-based" zoom FFT.

Suppose we have complex samples $u[k]$, for $0 \leq k \leq K-1$, and are interested in bins $-N \leq n \leq N$ of the length-K DFT

$$U[n] = \sum_{k=0}^{K-1} u[k] e^{-2\pi i \frac{kn}{K}}.$$

The zoom FFT proceeds by lowpass anti-alias filtering $u[k]$, downsampling the filtered samples, and performing a shorter FFT, which results in the approximate values of $U[n]$ for the bins of interest. The downsampling operation zooms in on the spectrum of $u[k]$ centered on the bin $n = 0$ (hence the name zoom FFT). Without the lowpass anti-alias filter, downsampling would in general produce significant aliasing errors. Finally, the short FFT brings the processed time-domain signal into the frequency domain, giving us the approximate values of $U[n]$. Note that this procedure gives approximate values for $U[n]$ at the exact grid points of the original DFT.

Let us suppose that $K = LR$ with $L, R \in \mathbb{N}$, where the decimation factor L is chosen such that $R \geq 2N+1$. Let $h[m]$, $0 \leq m \leq M-1$, be a lowpass FIR filter with cutoff frequency greater than or equal to $1/L$ (in normalized frequency units from -1 to 1) and unit gain at 0Hz. We form the filtered signal

$$\widetilde{u}[k] = \{h * u\}[k] = \sum_{m=0}^{M-1} h[m] u[k-m], \quad 0 \leq k \leq K-1.$$

We then downsample the filtered signal to form the filtered-and-downsampled signal

$$v[r] = \widetilde{u}[rL] = \sum_{m=0}^{M-1} h[m] u[rL-m], \quad 0 \leq r \leq R-1.$$

We now have that the bins $-N \leq n \leq N$ of the length-R DFT of $v[r]$ are approximately equal to the corresponding bins of $U[n]$ (with an appropriate scaling factor):

$$U[n] \approx LV[n] = L \sum_{r=0}^{R-1} v[r] e^{-2\pi i \frac{rn}{R}} \quad \text{for } -N \leq n \leq N.$$

The approximations involved here are that $h[m]$ is not an ideal lowpass filter and that there are transients involved when forming $\widetilde{u}[k]$.

Forming all K samples of $\widetilde{u}[k]$ just to then downsample by a factor of L when forming $v[r] = \widetilde{u}[rL]$ is itself quite wasteful. As is typical, we implement the filter-and-downsample operation jointly with a polyphase implementation. See [59] and [78] for an introduction to polyphase filtering. If L is sufficiently large or M sufficiently small (certainly if $L \geq M$), computing $v[r]$ directly may also be efficient.

Depending on the requirements of the lowpass filter, the computational complexity of the zoom FFT can be substantially lower than the full length-K DFT. The cost of the naïve, length-K DFT approach is $\mathcal{O}(K \log K)$ FLOPs. The cost of the zoom FFT approach is the cost of the filter-and-downsample operation plus the $\mathcal{O}(R \log R)$ FLOPs for the length-R DFT. For instance, a direct implementation of the filter-and-downsample operation uses $\mathcal{O}(RM)$ FLOPs.

2.2.2 Using the Zoom FFT to Compute the CAF

Let us now use the zoom FFT to compute the CAF over the desired range of frequency shifts $-N \leq n \leq N$. We first define the mixing product

$$u[j,k] \overset{\text{def}}{=} x[k]y[k+j]^*, \quad 0 \leq j \leq J,\ 0 \leq k \leq K-1.$$

As mentioned previously, we can form row j of the CAF by taking the length-K DFT of row j of the mixing product and extracting bins $-N \leq n \leq N$. It should be noted, however, that the mixing produce matrix is of size $(J+1) \times K$, which can be enormous for the large K of typical scenarios. Therefore we would never explicitly form the full mixing product matrix in an implementation. The CAF is of size $(J+1) \times (2N+1)$ which can still be large, but is roughly a factor of L smaller than the mixing product matrix.

As with the zoom FFT, we assume $K = LR$, with the decimation factor L chosen such that $R \geq 2N+1$. We interpret L as controlling the width of the frequency shift search range; larger L results in a narrower search range. The condition $R \geq 2N+1$ states that L must be sufficiently small as to capture the desired range of frequency shifts; typically L is chosen just small enough to satisfy this condition, in which case $R \approx 2N + 1$.

Let $h[m]$, $0 \le m \le M - 1$, be an appropriate lowpass FIR filter. We form the filtered and downsampled mixing product

$$v[j,r] = \sum_{m=0}^{M-1} h[m]u[j,rL-m], \quad 0 \le j \le J,\ 0 \le r \le R-1. \tag{2.3}$$

The filter-and-downsample operation is applied to each row of the mixing product. We finish the computation by computing length-R DFTs across the rows of $v[j,r]$, taking the central bins and multiplying by the appropriate scaling factor:

$$A[j,n] \approx L \sum_{r=0}^{R-1} v[j,r]e^{-2\pi i \frac{rn}{R}}, \quad 0 \le j \le J,\ -N \le n \le N.$$

Before we proceed, let us consider the computational cost of this approach. Consider the case when the filter length M is smaller than L. Since $M < L$, we need not compute all entries of the mixing product $u[j,k]$ in order to compute $v[j,r]$. Computing just the required entries costs $\mathcal{O}(JRM)$ FLOPs. Filtering and downsampling, implemented directly, costs a further $\mathcal{O}(JRM)$ FLOPs. The final FFT across the rows of $v[j,r]$ costs $\mathcal{O}(JR\log R)$ FLOPs, bringing the total cost to

$$\mathcal{O}(JRM) + \mathcal{O}(JRM) + \mathcal{O}(JR\log R) \quad \text{FLOPS}. \tag{2.4}$$

If the filter length M is at least L, then we require all elements of $u[j,k]$, which costs $\mathcal{O}(JK)$ FLOPs to compute. Of course, the filter-and-downsample operation is increasingly costly as M increases, and we may prefer a polyphase implementation with each polyphase component convolved using overlap-scrap convolution [59]. In this case the filter $h[m]$ is split into L polyphase components, each of size approximately M/L. Each component is then used to filter a downsampled version of $u[j,k]$, each of size K/L, starting with one of L offsets. Implementing each filter operation with overlap-scrap convolution will cost about $\mathcal{O}(K/L\log(4M/L))$ FLOPs per polyphase component when using the advice given in Section 13.10 of [59]. To populate $v[j,r]$ with this approach costs

$$\mathcal{O}(JK) + \mathcal{O}\left(JL\frac{K}{L}\log(4M/L)\right) = \mathcal{O}(JK) + \mathcal{O}(JK\log(4M/L)) \quad \text{FLOPs.}$$

By way of comparison, the naïve strategy of computing length-K DFTs over the rows of the mixing product $u[j,k]$ costs $\mathcal{O}(JK\log K)$ FLOPs. Thus the use of the zoom FFT can yield a significant speedup, particularly if M is small. While an improvement over the naïve strategy, a closer examination of (2.3) finds a further improvement.

2.2.3 `TransformCAF` - The Corner Turn

If we write the mixing product in (2.3) in terms of $x[k]$ and $y[k]$, we find

$$\begin{aligned} v[j,r] &= \sum_{m=0}^{M-1} h[m]u[j,rL-m] \\ &= \sum_{m=0}^{M-1} \left(h[m]x[rL-m]\right)\left(y[rL-m+j]^*\right). \end{aligned}$$

We can instead interpret this as populating the columns of $v[j,r]$ via a length-M, un-downsampled convolution of auxiliary vectors. Let us define two sets of R auxiliary vectors, indexed by $r \in \{0,\ldots,R-1\}$:

$$\begin{aligned} f_r[m] &\overset{\text{def}}{=} h[m]x[rL-m], \quad 0 \le m \le M-1; \\ g_r[j] &\overset{\text{def}}{=} y[rL-j]^*, \quad 0 \le j \le J. \end{aligned}$$

To clarify the notation, we note that there are R vectors $f_r[m]$, each of length M, and R vectors $g_r[j]$, each of length $J+1$. In terms of $f_r[m]$ and $g_r[j]$,

$$v[j,r] = \sum_{m=0}^{M-1} f_r[m]g_r[j-m], \tag{2.5}$$

which we view as populating the rth column of $v[j,r]$ with the (truncated) output of the convolution of $f_r[m]$ with $g_r[j]$.

Now that we have $v[j,r]$, we again finish the computation by computing FFTs over the rows of $v[j,r]$, giving

$$A[j,r] \approx L\sum_{r=0}^{R-1} v[j,r]e^{-2\pi i\frac{rn}{R}}, \quad 0 \le j \le J, -N \le n \le N.$$

The array $v[j,r]$ is populated in a column-wise fashion, and then $A[j,n]$ is populated in a row-wise fashion. The data flow is sometimes said to have "turned the corner", when reading from $v[j,r]$ into $A[j,n]$, hence why this approach is called the corner-turn algorithm.

In Tolimieri and Winograd's original derivation, standard FFT-based convolution is used to compute the R convolutions in (2.5). When both M and J are comparable, this is likely the best way to compute the $J+1$ values of the convolution forming each column of $v[j,r]$. However, when J is significantly larger than M, it is often preferable to implement the convolution using overlap-scrap [59]. Indeed, we take this approach for our implementation. Again using the advice given in Section 13.10 of [59], computing each column costs $\mathcal{O}(J\log(4M))$ FLOPs. Computing the CAF costs

$$\mathcal{O}(JR\log(4M)) + \mathcal{O}(JR\log R) \quad \text{FLOPs.} \tag{2.6}$$

In practice L is usually chosen just small enough to ensure $R \geq 2N+1$, so R is approximately equal to $2N+1$ and there are approximately JR elements in the CAF. Recall that there are approximately JK elements in the mixing product array, which is roughly L times more than the CAF. Comparing the computational costs for the direct zoom FFT approach and the corner-turn approach, we see that the corner-turn approach is preferable. It is remarkable to see that the corner-turn CAF algorithm computes an array with about JR elements using only JR times log factors amount of work.

We summarize the computations performed by the corner-turn CAF algorithm in Algorithm 3. Since the algorithm is referred to as the "Transform Method" in [92], we call this algorithm `TransformCAF`.

Figure 2.1 shows a zoomed-in view of an example CAF computed with `TransformCAF`. The two signals used are synthetic examples with realistic, but otherwise arbitrary, parameters. Each signal is polluted with AWGN to an SNR of 10dB. The CAF was computed over a delay search range of ± 0.005s and a frequency shift range of ± 10kHz, which encompasses the true peak location at $(0.001234\text{s}, -5678.9\text{Hz})$. We note that the zoomed-in view that we show in the figure is significantly smaller than the full extent of the CAF that has been computed.

Algorithm 3 `TransformCAF` (The "Transform Method" of [92])

```
Input: Max delay $J$, max frequency shift $N$, integration time $K = LR$, where $L, R$ are integers
  such that $R \geq 2N+1$, complex baseband signals $x[k], y[k]$ of sufficient length, lowpass filter $h[m]$
  of length $M$
Output: $A[j, n]$ for $0 \leq j \leq J,\ -N \leq n \leq N$

  ▷ Compute $v[j, r]$
  for $r = 0, \ldots, R-1$ do
      $f_r[m] = h[m]x[rL - m]$ for $0 \leq m \leq M-1$
      $g_r[j] = y[rL + j]^*$ for $0 \leq j \leq J$
      $v[j, r] = \sum_{m=0}^{M-1} f_r[m] g_r[j-m]$ for $0 \leq j \leq J$        ▷ Use overlap-scrap convolution
  end for

  ▷ Approximate $A[j, n]$ with DFT over rows of $v[j, r]$
  for $j = 0, \ldots, J$ do
      $A[j, n] \approx L \sum_{r=0}^{R-1} v[j, r] e^{-2\pi i \frac{rn}{R}}$ for $-N \leq n \leq N$
  end for
```

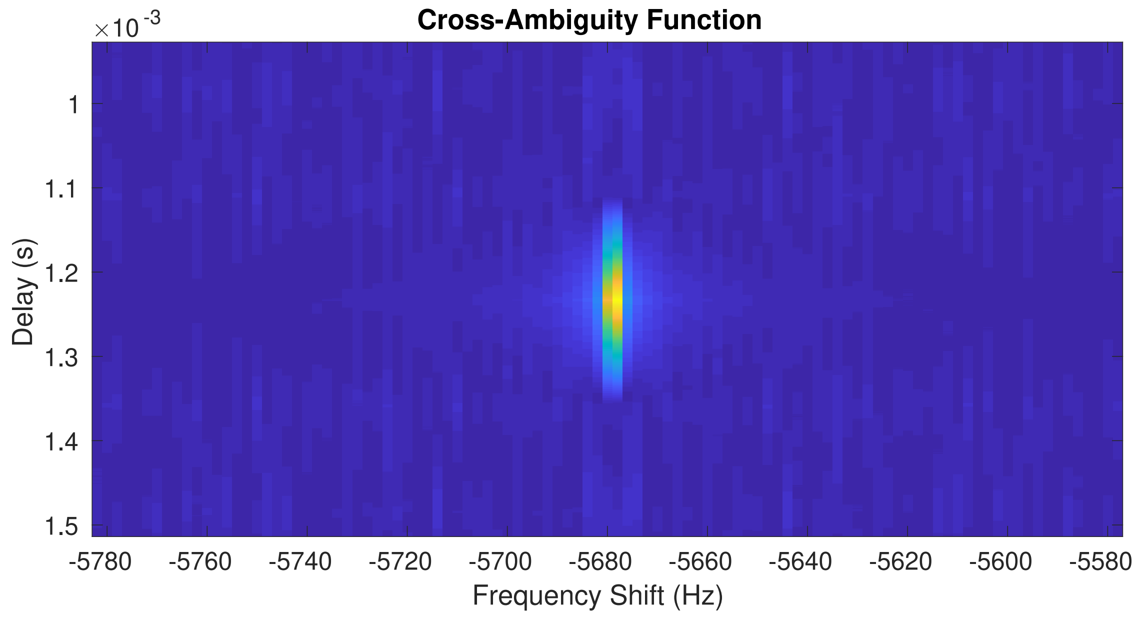


Figure 2.1: An example of a CAF computed with `TransformCAF`.

2.3 Delay-Decimated CAF

In this section we present two algorithms for computing the CAF on a downsampled grid of delays. That is, given some integer delay-downsampling factor P, we seek to compute $A[j'P, n]$ for $0 \leq j' \leq J/P$. Since naïvely downsampling $A[j, n]$ in the delay dimension has a likelihood of missing narrow peaks in the CAF, we also propose a simple, but effective, trick to make the delay-downsampled CAF more robust to missing peaks. When computed with this trick, we call the delay-downsampled CAF a delay-decimated CAF to allude to its safety compared to naïvely downsampling.

2.3.1 Algebraic Zoom FFT

Before we proceed with the discussion of our first new algorithm, we first present an alternative method of computing the DFT on a small set of contiguous bins as described in [78]. We presented the filter-based zoom FFT in Subsection 2.2.1 as an efficient method for this task. While the zoom FFT can be quite efficient, it still involves approximations. The **algebraic** zoom FFT, as we will call it, is an efficient procedure for computing the **exact** DFT on a small set of contiguous bins (it is also known as a pruned FFT [63]). As we will see, the algebraic zoom FFT is basically a single step of a Cooley-Tukey decomposition [23]

Again suppose we have a signal $u[k]$, $0 \leq k \leq K-1$, and are interested in bins $-N \leq n \leq N$ of the length-K DFT

$$U[n] = \sum_{k=0}^{K-1} u[k] e^{-2\pi i \frac{kn}{K}}.$$

As with the filter-based zoom FFT, assume that $K = LR$ with $R \geq 2N+1$. We can decompose the length-K sum into a length-L sum and a length-R sum:

$$\begin{aligned} U[n] &= \sum_{k=0}^{K-1} u[k] e^{-2\pi i \frac{kn}{K}} \\ &= \sum_{l=0}^{L-1} \sum_{r=0}^{R-1} u[rL+l] e^{-2\pi i \frac{ln}{K}} e^{-2\pi i \frac{rn}{R}}. \end{aligned}$$

After rearranging terms, we find

$$U[n] = \sum_{l=0}^{L-1} e^{-2\pi i \frac{ln}{K}} \left(\sum_{r=0}^{R-1} u[rL+l] e^{-2\pi i \frac{rn}{R}} \right). \tag{2.7}$$

Since n is limited to the range $-N \leq n \leq N$, we recognize the term inside the parentheses as a length-R DFT of the downsampled sequence $u[rL+l]$ (where we view l as fixed). From this we see that we can compute bins $-N \leq n \leq N$ of $u[n]$ in two steps. First, compute the L length-R DFTs of the downsampled sequences $u[rL+l]$. Then combine the outputs of the L length-R DFTs using the twiddle factors $e^{-2\pi i \frac{ln}{K}}$ according to (2.7). As Porat notes in [78], the second step cannot be done with an FFT.

The first major difference between the algebraic zoom FFT and filter-based zoom FFT is that the algebraic zoom FFT is exact. The price one pays for this, however, is that the algebraic zoom FFT is typically more expensive than the filter-based zoom FFT. We compute L length-R DFTs via the FFT at a cost of $\mathcal{O}(LR \log R)$ FLOPs, and then combine bins to form $U[n]$ at a cost of $\mathcal{O}(LN)$ FLOPs. The dominant cost is due to the FFTs, so the total cost is $\mathcal{O}(K \log R)$, which is still a gain over the naïve $\mathcal{O}(K \log K)$. Recall from Subsection 2.2.1 that the filter-based zoom FFT computes a single length-R DFT, instead of the L required for the algebraic zoom FFT.

Even though the algebraic zoom FFT appears more expensive than the filter-based zoom FFT, let us look at how the algebraic zoom FFT can be used to compute the CAF.

2.3.2 Using the Algebraic Zoom FFT to Compute the CAF

We begin in much the same manner as Subsection 2.2.2, where we used the filter-based zoom FFT to compute the CAF. Recall that the CAF is defined as

$$A[j,n] = \sum_{k=0}^{K-1} x[k] y[k+j]^* e^{-2\pi i \frac{kn}{K}},$$

and we will work on the mixing product

$$u[j,k] = x[k] y[k+j]^*.$$

In terms of the mixing product, the CAF is

$$A[j,n] = \sum_{k=0}^{K-1} u[j,k] e^{-2\pi i \frac{kn}{K}},$$

Assume $K = LR$ and let us begin to apply the algebraic zoom FFT to the CAF by splitting the length-K sum into two sums:

$$A[j,n] = \sum_{l=0}^{L-1} e^{-2\pi i \frac{ln}{K}} \left(\sum_{r=0}^{R-1} u[j, rL+l] e^{-2\pi i \frac{rn}{R}} \right). \tag{2.8}$$

Note that this gives the CAF exactly; there are no approximations used in this approach. We can interpret (2.8) as forming the desired CAF plane as a weighted sum of L CAFs computed using downsampled mixing products and the shorter integration R. However, this approach is more costly than the corner-turn CAF.

To see this we can switch the order of summation in (2.8):

$$A[j,n] = \sum_{r=0}^{R-1} \left(\sum_{l=0}^{L-1} e^{-2\pi i \frac{ln}{K}} u[j, rL+l] \right) e^{-2\pi i \frac{rn}{R}}.$$

We interpret this as a filter-and-downsample operation on the rows of $u[j,k]$ with filter $h_n[l] = e^{-2\pi i ln/K}$, followed by a length-$R$ DFT of the rows. Compared to the filter-based zoom FFT CAF described in Subsection 2.2.2, this approach is nearly identical. The only difference is that there is a different filter used for each n. When $n = 0$, $h_0[l] = 1$ for $0 \leq l \leq L-1$, and we recognize the filter as a simple, ones-based lowpass filter. When $n \neq 0$, the filter is a ones-based bandpass filter centered on the frequency nf_s/K.

Since the filters $h_n[l]$ are ones-based, they have somewhat slow roll-off from their center frequency. Depending on the particular values, the magnitude of the frequency response of $h_n[l]$ at its center frequency nf_s/K is not substantially far above the response of $h_0[l]$ at nf_s/K. Put another way, the filter impulse response phases $-2\pi i ln/K$ are very slowly varying over the range of l and n used in (2.8), so we can make the approximation $e^{-2\pi i ln/K} \approx 1$. Therefore let us approximate $A[j,n]$ by using just the ones filter $h_0[l]$ instead of using the filter prescribed by the algebraic zoom FFT:

$$A[j,n] \approx \sum_{r=0}^{R-1} \left(\sum_{l=0}^{L-1} u[j, rL+l] \right) e^{-2\pi i \frac{rn}{R}}. \tag{2.9}$$

Comparing to Subsection 2.2.2, we see that this is just a special case of the filter-based zoom FFT applied to the CAF. In particular, the case when a length-L ones filter is used as the lowpass filter. We could once again use the corner-turn approach to find a more efficient algorithm, and it would also result in a special case of the corner-turn CAF.

2.3.3 `AlgebraicCAF` - Delay-Downsampling

Let us now use the algebraic zoom FFT as a basis for computing a delay-downsampled CAF. We start from (2.8), writing the downsampled mixing product in terms of $x[k]$ and $y[k]$:

$$A[j,n] = \sum_{l=0}^{L-1} e^{-2\pi i \frac{ln}{K}} \left(\sum_{r=0}^{R-1} x[rL+l]y[rL+l+j]^* e^{-2\pi i \frac{rn}{R}} \right).$$

Assume that J is divisible by L (we could compute either more or fewer delays if needed). Then we can compute every Lth row of the CAF as

$$A[j'L,n] = \sum_{l=0}^{L-1} e^{-2\pi i \frac{ln}{K}} \left(\sum_{r=0}^{R-1} x[rL+l]y[(r+j')L+l]^* e^{-2\pi i \frac{rn}{R}} \right), \tag{2.10}$$

where the index j' runs from 0 to J/L.

To understand what (2.10) is doing, let us define the L downsampled mixing products

$$u_l[j',r] \overset{\text{def}}{=} x[rL+l]y[(r+j')L+l]^*, \quad 0 \le j' \le L,\ 0 \le r \le R-1,$$

each of which is a $(J/L+1) \times R$ array. Then term inside the parentheses of (2.10) is just

$$\sum_{r=0}^{R-1} u_l[j',r]e^{-2\pi i \frac{rn}{R}},$$

which is the length-R DFT of the rows of u_l. As mentioned in the previous subsection, this can be interpreted as forming L CAF planes with a short integration for a particular input. Finally, we combine the L short-integration CAF planes, forming

$$A[j'L,r] = \sum_{l=0}^{L-1} e^{-2\pi i \frac{ln}{K}} \left(\sum_{r=0}^{R-1} u_l[j',r]e^{-2\pi i \frac{rn}{R}} \right).$$

We therefore have an exact, delay-downsampled CAF. Like (2.8), which computes the CAF exactly, this approach is relatively costly, even though we compute L times fewer rows.

Computing all downsampled mixing products $u_l[j', r]$ directly costs $\mathcal{O}(JR)$ FLOPs. Performing row-wise DFTs over each $u_l[j', r]$ is a further $\mathcal{O}(JR \log R)$ FLOPs. Finally, combining all of the short-integration CAFs costs about $\mathcal{O}(JR)$ FLOPs, assuming $2N + 1 \approx R$. This brings the total cost to

$$\mathcal{O}(JR) + \mathcal{O}(JR \log R) + \mathcal{O}(JR) \text{ FLOPs},$$

which is on par with the cost of the full corner-turn CAF. Since this computes an array of size about $(J/L + 1) \times R$, this cost is difficult to justify, unless the exact values of the delay-downsampled CAF are required.

To speed up this algorithm, we can again use the approximation $e^{-2\pi i l n/K} \approx 1$ as discussed previously. Doing so yields

$$A[j'L, r] \approx \sum_{r=0}^{R-1} \left(\sum_{l=0}^{L-1} u_l[j', r] \right) e^{-2\pi i \frac{rn}{R}}.$$

In using this approximation, we can now bring the "twiddle factor" sum inside the DFT. We still compute all L downsampled mixing products, but now there is a single row-wise DFT. The cost for this algorithm, which we call `AlgebraicCAF`, is

$$\mathcal{O}(JR) + \mathcal{O}\left(\frac{JR}{L} \log R \right) \text{ FLOPs},$$

which is improved, considering we compute about JR/L total elements in the delay-downsampled CAF. We summarize `AlgebraicCAF` in Algorithm 4.

We mention briefly that `AlgebraicCAF` is merely a special case of what would result if we modified the zoom FFT approach of Subsection 2.2.2 to compute every Lth row of the CAF. Since the zoom FFT approach computes $v[j, r]$ in a row-wise manner, it is simple to modify it to compute just $v[j'L, r]$ for $0 \leq j' \leq J/L$. This yields a very similar algorithm, with a similar FLOP count. Further, there is no reason to tie the delay-downsampling factor to the "main" decimation factor L; we could instead compute every Pth row of the CAF, for some integer P. We propose doing this with a modification of the corner-turn algorithm in Subsection 2.3.5, and yield a faster algorithm than `AlgebraicCAF`.

Algorithm 4 AlgebraicCAF

```
Input: Max delay $J$ (divisible by $L$), max frequency shift $N$, integration time $K = LR$, where
    $L, R$ are integers such that $R \geq 2N + 1$, complex baseband signals $x[k], y[k]$ of sufficient length
Output: $A[j'L, n]$ for $0 \leq j' \leq J/L$, $-N \leq n \leq N$

  ▷ Directly compute sum of downsampled mixing products
  $\overline{u}[j', r] \overset{\text{def}}{=} \sum_{l=0}^{L-1} x[rL + l] y[(r + j')L + l]^*$
  ▷ Approximate $A[j'L, n]$ with DFT over rows of $\overline{u}[j', r]$
  for $j' = 0, \ldots, J/L$ do
      $A[j'L, n] \approx \sum_{r=0}^{R-1} \overline{u}[j', r] e^{-2\pi i \frac{rn}{R}}$ for $-N \leq n \leq N$
  end for
```

2.3.4 The Ones Filter Trick

As written, AlgebraicCAF approximates every Lth row of the CAF. Let us be more general (in anticipation of Subsection 2.3.5) and suppose we have computed every Pth row of the CAF. For even moderate values of P (say, about 50), the risk of missing a narrow peak in the CAF between rows that get selected in a delay-downsampled CAF is large.

One way to ameliorate this problem is to average P rows before downsampling. That is, we compute the averaged CAF

$$\overline{A}[j, n] \overset{\text{def}}{=} \frac{1}{P} \sum_{p=0}^{P-1} A[j + p, n],$$

and then delay-downsample by a factor of P. This can be interpreted as treating the CAF surface as an image and performing a ones filter along the delay dimension.

As written, this requires first computing the full CAF $A[j, n]$, then filtering, and then throwing away most of the data. This precludes us from using a delay-downsampled CAF. There is a trick, however. Note that

$$\begin{aligned}\overline{A}[j,n] &= \frac{1}{P}\sum_{p=0}^{P-1} A[j+p,n] \\ &= \frac{1}{P}\sum_{p=0}^{P-1}\sum_{k=0}^{K-1} x[k]y[k+j+p]^* e^{-2\pi i\frac{kn}{K}} \\ &= \sum_{k=0}^{K-1} x[k]\left(\frac{1}{P}\sum_{p=0}^{P-1} y[k+j+p]\right)^* e^{-2\pi i\frac{kn}{K}}.\end{aligned}$$

If we define

$$\overline{y}[k] \overset{\text{def}}{=} \frac{1}{P}\sum_{p=0}^{P-1} y[k+p],$$

we see that $\overline{A}[j,n]$ is just the CAF of $x[k]$ and $\overline{y}[k]$. Therefore we can **implicitly** compute the averaged CAF $\overline{A}[j,n]$ without first forming the (unaveraged) CAF $A[j,n]$ and then averaging. Merely averaging the second signal input to an ordinary CAF algorithm results in the output being averaged.

In the case of a delay-downsampled CAF, this then has the effect of implicitly averaging the full CAF and taking every Pth row of the averaged CAF **without actually computing the extra rows**. This affords us an inexpensive measure of robustness to missing peaks when using delay-downsampled CAFs, and is the final ingredient needed to make them useful for fast, reliable detection.

We note that the trick is not limited to a ones filter. If $b[m]$, $0 \le m \le M-1$ is a preferable filter (where M is not necessarily equal to P), one can implicitly compute

$$\sum_{m=0}^{M-1} b[m]A[j+m,n]$$

by using the signals $x[k]$ and

$$\sum_{m=0}^{M-1} b[m]y[j+m]$$

as the inputs to a CAF algorithm. This of course works with both ordinary and delay-downsampled CAF algorithms.

2.3.5 `TransformCAF` - Adding Delay-Downsampling

We have seen how `AlgebraicCAF` computes the delay-downsampled CAF $A[j'L, n]$. Together with the ones filter trick, it is a useful tool to quickly detect coarse locations of peaks in the CAF. As written, however, the delay decimation is tied to the "main" decimation factor L, which controls the width of the frequency shift search range. Furthermore, it can be viewed as a delay-downsampled version of the zoom FFT CAF algorithm from Subsection 2.2.2. From the FLOP counts of the zoom FFT CAF and corner-turn CAF (see Subsection 2.2.3), we expect that adapting the corner-turn CAF algorithm to handle delay-decimation is the preferable approach.

We propose here to add delay-downsampling by an integer factor P, not necessarily equal to L, to the corner-turn CAF algorithm. We will work with $x[k]$ and $\overline{y}[k]$, the ones filtered version of $y[k]$, so that we compute a delay-decimated CAF that is robust to missing peaks. The main change is that we now compute the delay-downsampled, filtered and downsampled mixing product

$$v[j'P, r] = \sum_{m=0}^{M-1} h[m]x[rL - m]\overline{y}[rL - m + j'P]^*.$$

Following the corner-turn approach, we define the auxiliary sequences $f_r[m]$ and $g_r[j]$ as in Subsection 2.2.3 (using $\overline{y}[k]$ in place of $y[k]$). For each r, the original corner-turn algorithm computes $J + 1$ outputs of the convolution

$$\sum_{m=0}^{M-1} f_r[m]g_r[j - m]$$

at the input sample rate f_s. Instead, we will compute $J/P + 1$ outputs at the decimated rate of f_s/P:

$$v[j'P, r] = \sum_{m=0}^{M-1} f_r[m]g_r[j'P - m].$$

This can be done in a number of ways, such as directly or with a polyphase implementation. If the decimation factor P is relatively large compared to the filter length M, we may prefer a direct implementation, in which case populating $v[j'P, r]$ will cost $\mathcal{O}(JRM/P)$ FLOPs. A polyphase implementation with each component computed using overlap-scrap convolution will use $\mathcal{O}(\frac{JR}{P}\log(4M))$ FLOPs if we continue to follow the advice given in [59].

To finish up, we compute length-R DFTs over the rows of $v[j'P, r]$. This costs $\mathcal{O}(\frac{JR}{P}\log R)$ FLOPs, since there are $J/P + 1$ rows. Using a direct implementation of computing $v[j'P, r]$, the delay-decimated CAF has a total cost of

$$\mathcal{O}\left(\frac{JR}{P}(M + \log R)\right) \text{ FLOPs,}$$

while using a polyphase implementation costs

$$\mathcal{O}\left(\frac{JR}{P}(\log(4M) + \log R)\right) \text{ FLOPs,}$$

Since we are computing an array with approximately JR/P elements, these costs show that the performance improvement of the corner-turn algorithm over the direct zoom FFT CAF algorithm remains when adding delay-decimation. That is to say, the cost of the algorithm is nearly linear in the number of elements computed. Algorithm 5 summarizes the approach, which we call `TransformCAF(delay_dec=`P`)`.

Algorithm 5 `TransformCAF(delay_dec=`P`)`

Input: Max delay J, max frequency shift N, integration time $K = LR$, where L, R are integers such that $R \geq 2N+1$, delay-decimation factor P, complex baseband signals $x[k], y[k]$ of sufficient length, lowpass filter $h[m]$ of length M

Output: $\overline{A}[j'P, n]$ for $0 \leq j' \leq J/P$, $-N \leq n \leq N$

```
  ▷ Use the ones filter trick to implicitly average rows of $A[j, r]$
  $\overline{y}[k] = \frac{1}{P}\sum_{p=0}^{P-1} y[k+p]$
  ▷ Compute $v[j'P, r]$ via the downsampled corner-turn approach
  for $r = 0, \ldots, R-1$ do
      $f_r[m] = h[m]x[rL - m]$ for $0 \leq m \leq M-1$
      $g_r[j] = \overline{y}[rL + j]^*$ for $0 \leq j \leq J$
      $v[j'P, r] = \sum_{m=0}^{M-1} f_r[m] g_r[j'P - m]$ for $0 \leq j' \leq J/P$      ▷ Direct or polyphase implementation
  end for

  ▷ Approximate $\overline{A}[j'P, n]$ with DFT over rows of $v[j'P, r]$
  for $j' = 0, \ldots, J/P$ do
      $\overline{A}[j'P, n] \approx L \sum_{r=0}^{R-1} v[j'P, r] e^{-2\pi i \frac{rn}{R}}$ for $-N \leq n \leq N$
  end for
```

Figure 2.2 shows a zoomed-in view of an example CAF computed with `TransformCAF(delay_dec=250)`, where $P = 250$ is chosen to be equal to the "main" decimation factor $L = 250$. The CAF was computed using the same signals as in Figure 2.1 and zoomed in to the same region. The peak is clearly visible, even though we have used the relatively large delay-decimation factor of $P = 250$.

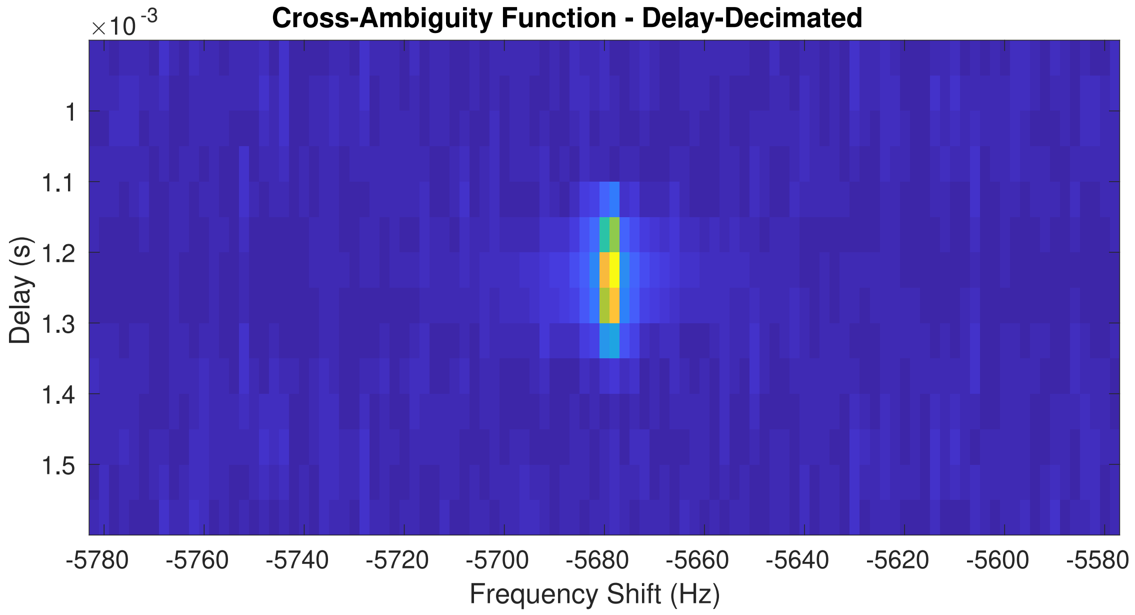


Figure 2.2: An example of a CAF computed with `TransformCAF(delay_dec=250)`.

2.4 Delay-Compressed CAF

We saw in Subsection 2.3.4 a simple, inexpensive trick to allow our delay-downsampled CAF algorithms to be more robust to missing peaks between the rows they compute. In our experiments, only one representative example of which is shown in this chapter, the ones filter trick is quite effective. Since the ones filter trick averages complex samples of the CAF $A[j, n]$, there is a possibility for catastrophic deconstructive interference to occur, causing the peak to be greatly attenuated. Therefore we seek a more robust method to increase the safety of a delay-downsampled CAF.

To remove the possibility of the phase cancellation problem of the ones filter, we can average samples of the magnitude-squared CAF $|A[j, n]|^2$ before delay-downsampling. In terms of our notation, we seek to compute

$$\overline{A}_{\text{mag2}}[j'P, n] \overset{\text{def}}{=} \frac{1}{P} \sum_{p=0}^{P-1} |A[j'P + p, n]|^2. \tag{2.11}$$

Even with averaging magnitude-squared bins of the CAF, there is still a potential reduction in SNR of the peak due to averaging "noisy", or non-peak, bins together with the peak bins. The possible reduction in SNR in this approach is gradual, however, and so it is preferable to the possible catastrophic interference in the ones filter approach. The average of magnitude-squared approach is be more robust than the worst case of the ones filter trick.

The average of magnitude-squared CAF (2.11) is very similar to a standard way of computing spectrograms using the short-time Fourier transform (also known as the time-dependent Fourier transform [76]). Before we study how to cheaply approximate (2.11) by "compressing" the average of magnitude-squared operation, we first study how to compute a "compressed" spectrogram. After discussing how to compute a compressed spectrogram, we will adapt the ideas to form a **delay-compressed** CAF.

2.4.1 The Short-Time Fourier Transform

Consider a single complex signal $x[k]$. From $x[k]$, we first produce an array $s[j, r]$ of size $J \times R$ of overlapped segments of $x[k]$ via the overlapping procedure described by the MATLAB code in Listing 2.1. The size R will be the size of the DFT, and J will be the total number of overlapped segments. We assume that $x[k]$ is sufficiently long for this purpose (we could also zero pad the tail of $x[k]$).

In Listing 2.1, the parameter `overlap` $\in (-\infty, 1)$ controls how much overlap there is between the segments taken from $x[k]$. In particular, the closer `overlap` is to 1, the more overlap there is in the rows of $s[j, r]$. `overlap` can also be negative, in which case samples are skipped/dropped when forming $s[j, r]$.

Listing 2.1: Overlapping Procedure (MATLAB notation)

```
% How many samples to advance to the next segment
n_advance = round( (1 - overlap) * R);

s = zeros(J,R);
for j=0:J-1
    i0 = j*n_advance + 1;
    i1 = j*n_advance + R;

    s(j+1,:) = x(i0:i1);
end
```

Once $s[j,r]$ has been populated, we compute the (unaveraged) complex spectrogram $S[j,n]$ as the length-R DFT of the rows of $s[j,r]$:

$$S[j,n] \overset{\text{def}}{=} \sum_{r=0}^{R-1} w[r]s[j,r]e^{-2\pi i \frac{rn}{R}}, \tag{2.12}$$

where $w[r]$ is an appropriately sized window function (e.g., a Hann window). Finally, we choose a block size P over which to average rows of $|S[j,n]|^2$ (for simplicity, assume that J is divisible by P). It is typical to average blocks of rows of $|S[j,n]|^2$ into a single row of the averaged spectrogram. This serves to decrease the amount of memory used by the spectrogram and allow for faster exploration of its content. In terms of $S[j,n]$, this approach computes the "averaged magnitude-squared" spectrogram

$$\overline{S}_{\text{mag2}}[j'P,n] \overset{\text{def}}{=} \frac{1}{P}\sum_{p=0}^{P-1} |S[j'P+p,n]|^2. \tag{2.13}$$

Figure 2.3 shows $\overline{S}_{\text{mag2}}[j'P,n]$ for a synthetic signal. The signal is constructed as a complex exponential with sinusoidally changing frequency plus AWGN to achieve an SNR of 10dB. We have used blocks of size $P = 16$ for this figure and the spectrogram figures that follow.

For spectrograms, we view (2.13) as the reference equation that we seek to approximate and find a faster algorithm. The procedure for computing $\overline{S}_{\text{mag2}}[j'P,n]$ can be summarized as follows:

(1) Copy overlapping segments of $x[k]$ into rows of the array $s[j,r]$ as in Listing 2.1.

(2) Compute the length-R DFT of each row of $w[r]s[j,r]$, producing $S[j,n]$.

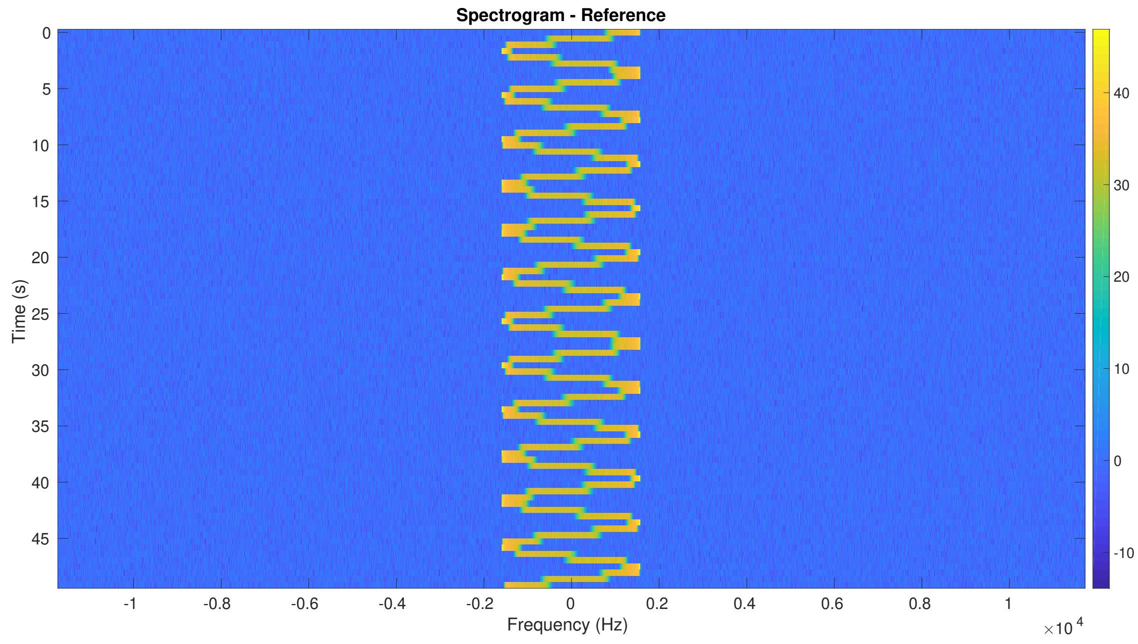


Figure 2.3: Spectrogram of a synthetic signal computed using the reference approach, (2.13).

(3) Average rows $j'P$ to $j'P+P-1$ of $|S[j,n]|^2$ into row j' of $\overline{S}_{\text{mag2}}[j'P,n]$.

2.4.2 Implicit Sum Spectrogram

The magnitude-squared operation used in (2.13) prohibits us from directly using the ones filter trick of Subsection 2.3.4. Let us explore what happens if we simply "force it", by computing the sum before taking the magnitude-squared. That is, we compute

$$\overline{S}_{\text{mag2,IS}}[j'P,n] \overset{\text{def}}{=} \frac{1}{P}\left|\sum_{p=0}^{P-1} S[j'P+p,n]\right|^2 .$$

Defining $\overline{S}_{\text{mag2,IS}}[j'P,n]$ as such allows us to switch the order of the average and DFT:

$$\begin{aligned}\overline{S}_{\text{mag2,IS}}[j'P,n] &= \frac{1}{P}\left|\sum_{p=0}^{P-1}\sum_{r=0}^{R-1} w[r]s[j'P+p,r]e^{-2\pi i\frac{rn}{R}}\right|^2 \\ &= \frac{1}{P}\left|\sum_{r=0}^{R-1}\left(\sum_{p=0}^{P-1} w[r]s[j'P+p,r]\right)e^{-2\pi i\frac{rn}{R}}\right|^2 .\end{aligned}$$

To take advantage of this (i.e., to realize a computational benefit), we define

$$\widetilde{s}_{\text{IS}}[j'P,r] = \sum_{p=0}^{P-1} w[r]s[j'P+p,r].$$

We can then compute $\overline{S}_{\text{mag2,IS}}[j'P, n]$ as the length-R DFT of the rows of $\widetilde{s}_{\text{IS}}$ followed by an element-wise magnitude-squared and scaling by $1/P$. An example of this is shown in Figure 2.4. Note that there are significant artifacts due to phase interference.

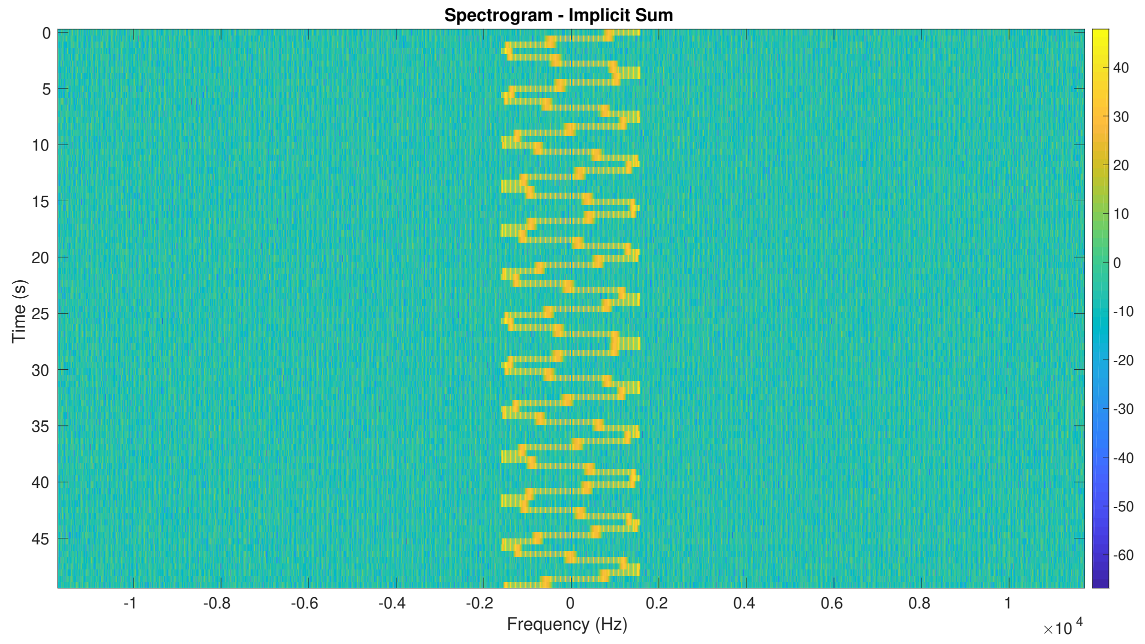


Figure 2.4: Spectrogram of a synthetic signal computed using the implicit sum approach.

This procedure allows us to see a moderate computational benefit over computing the standard averaged spectrogram. Computing $\widetilde{s}_{\text{IS}}[j'P, r]$ takes roughly the same amount of FLOPs as preparing $w[r]s[j, r]$ for step 2 of the standard averaged spectrogram computation. But now computing $\overline{S}_{\text{mag2,IS}}[j'P, n]$ requires a factor of P fewer DFTs, which can yield a significant speedup. However, to achieve this speedup over the standard method, we have made a crude approximation.

2.4.3 A Compressed Spectrogram

A more principled approach is as follows. We treat (2.13) as computing (scaled) column-wise ℓ_2 norms of the appropriate blocks of $S[j, n]$. That is to say, we treat the average of magnitude-squared samples as the (scaled) ℓ_2 norm squared of a length-P vector. We seek to compress these norm computations in a manner that will allow us to compute fewer row-wise DFTs, yielding a computational benefit.

To perform the compression, we seek a low-distortion mapping from $\mathbb{C}^P$ to $\mathbb{C}^{P'}$, where we choose a compression factor $\gamma \in (0,1]$ and take $P' = \lceil \gamma P \rceil$. That such a low-distortion mapping, for a finite set of points, can be chosen to be a particular random linear projection was shown in [54]. The embedding can be represented as a matrix $\Phi \in \mathbb{C}^{P' \times P}$ constructed as follows. Let $U \in \mathbb{C}^{P \times P}$ be a random unitary matrix sampled from the Haar distribution on $\mathrm{U}(P)$, unitary group. Such a matrix U can be generated as described in [69]; note that the procedure for unitary matrices has one extra step from the procedure for orthogonal matrices. The P' rows of Φ are then taken as the first P' columns of U.

Let $B_{j'}[p,n] \overset{\text{def}}{=} S[j'P + p, n],\ 0 \le p \le P-1$, be the j'th block of $S[j,n]$. The standard averaged spectrogram algorithm would take the magnitude-squared of $B_{j'}[p,n]$ and average down the columns, producing a single row of $\overline{S}_{\text{mag2}}[j'P, n]$. We will instead compress the columns of this matrix using Φ:

$$\widetilde{B}_{j'} = \Phi B_{j'},$$

where $\widetilde{B}_{j'}[p',n],\ 0 \le p' \le P'-1$, is the j'th **compressed** block. To compute the "compressed" spectrogram, we average the columns of each $|\widetilde{B}_{j'}[p',n]|^2$:

$$\overline{S}_{\text{mag2,CN}}[j'P, n] = \frac{1}{P} \sum_{p'=0}^{P'-1} \frac{P}{P'} \left| \widetilde{B}_{j'}[p',n] \right|^2 .$$

The factor P/P' is to account for changing the dimension of the ℓ_2 norm. This final computation is nearly identical to the final computation of the standard approach. The difference is that each block has been compressed by a factor $P'/P \approx \gamma$, so the ℓ_2 norms are computed on shorter vectors.

Since we have compressed the blocks **after** computing the row-wise DFT (of which we are trying to reduce the number), we see no real computational benefit from the compression. But let us look more closely. Let $b_{j'}[p,r] \overset{\text{def}}{=} s[j'P + p, r]$ be the j'th block of $s[j,n]$ (i.e., $B_{j'}[p,r]$ before the row-wise DFT). Since the DFTs are taken over the rows of the matrix $b_{j'}$, we can relate the matrices $B_{j'}$ and $b_{j'}$ as $B_{j'} = b_{j'} F_R$, where F_R is the length-R DFT matrix. Due to the associativity of the matrix product, we have

$$\widetilde{B}_{j'} = \Phi B_{j'} = \Phi(b_{j'} F_R) = (\Phi b_{j'}) F_R.$$

This says that we can compress the columns of $b_{j'}$ and then take the row-wise DFTs. In other words, we can now use only P' length-R DFTs instead of the original P DFTs.

We have achieved our goal of decreasing the number of row-wise DFTs while maintaining a degree of accuracy compared to the standard averaged spectrogram. However, applying Φ requires a dense matrix-matrix multiply, requiring $\mathcal{O}(P'PR)$ FLOPs to compress each block. Unless γ is tiny (which shrinks P', increasing the error of the compression), this may be too expensive for our purposes.

2.4.4 A Practical Compressed Spectrogram

Instead of using a dense Φ, let us take a sparse, structured Φ. Ailon and Chazelle in [2] proposed the so-called fast Johnson-Lindenstrauss transform (FJLT), which dramatically improves the speed of applying a compression matrix Φ while maintaining desirable guarantees. Let us now adapt the FJLT into our compressed spectrogram algorithm.

One particular instance of an FJLT-like transform takes Φ of the form

$$\Phi = \Pi F_P D,$$

where Π is a permutation/subsampling matrix, F_P is the length-P DFT matrix, and D is a diagonal matrix of random complex numbers sampled uniformly from the unit circle. This choice of $F_P D$ is inspired by [8], which uses $F_P D$ to lower the coherence of the product $F_P D B$ (for some matrix B) in order to "precondition" random uniform row subsampling. Indeed, picking Π to be the first P' rows of a random permutation matrix will perform uniform row subsampling. One might call this choice of Φ a subsampled randomized Fourier transform (SRFT) [83].

If we apply Φ, as written, to a block B of size $P \times R$, it would cost $\mathcal{O}(PR \log P)$ FLOPs to compress the block. This cost is independent of P' because, in the direct application of Φ, we form the full length-P DFT just to subsample P' of the outputs. For small values of P', this may be more expensive than the Haar-based approach in the previous subsection.

By sacrificing the randomness in the subsampling matrix Π, we can improve this runtime.

We can construct a deterministic Π to subsample P' **contiguous** rows from the output of the DFT. Strictly speaking, Φ will no longer be an SRFT as defined in [83], since Π is no longer random. For simplicity, we will refer to it as an SRFT nonetheless. Due to this simplification, we can now compute just the required output bins with the zoom FFT (see Subsection 2.2.1), instead of computing the full DFT and then subsampling.

For example, let us choose Π to take the rows corresponding to the lowest P' frequencies (in absolute value) from the length-P DFT. To compute just these P' lowest frequency bins with the zoom FFT, we first lowpass filter and decimate the input and then take a length-P' DFT. In our implementation, we choose a length-$\lceil P/P' \rceil$ ones filter and use a direct implementation, which crudely accounts for the fact that we are not decimating by an integer factor. This leads to a cost of approximately $\mathcal{O}(PR)$ FLOPs for the filter-and-decimate operation and $\mathcal{O}(P'R\log P')$ FLOPs for the DFT, and can give a significant savings over the direct approach.

An alternative strategy is to compute the required DFT bins directly using the definition of the DFT. For this to be practical, the number of compressed bins, P', must be sufficiently small. This has the added benefit of allowing for arbitrary subsampling, instead of the contiguous subsampling required by the zoom FFT, and so Φ could form a proper SRFT. Of course, if P' is so small, then using the Haar-based Φ of Subsection 2.4.3 may also be practical.

Figure 2.5 shows the example spectrogram using the zoom FFT approach, with $P = 16$ and $\gamma = -10\text{dB} = 0.1$. When compared to Figure 2.4, we see reduced (but not eliminated) errors. In our implementation, we used a different random diagonal matrix D for each block $b_{j'}$.

2.4.5 A Practical Delay-Compressed CAF

We have seen thus far in this section how to approximate and speed up the computation of averaged magnitude-squared spectrograms. Let us now apply these ideas to the CAF. Exactly as in the case of the spectrograms, we seek to approximate (2.11) and allow us to compute fewer row-wise DFTs. Let us recall the CAF written in terms of the filtered and downsampled mixing

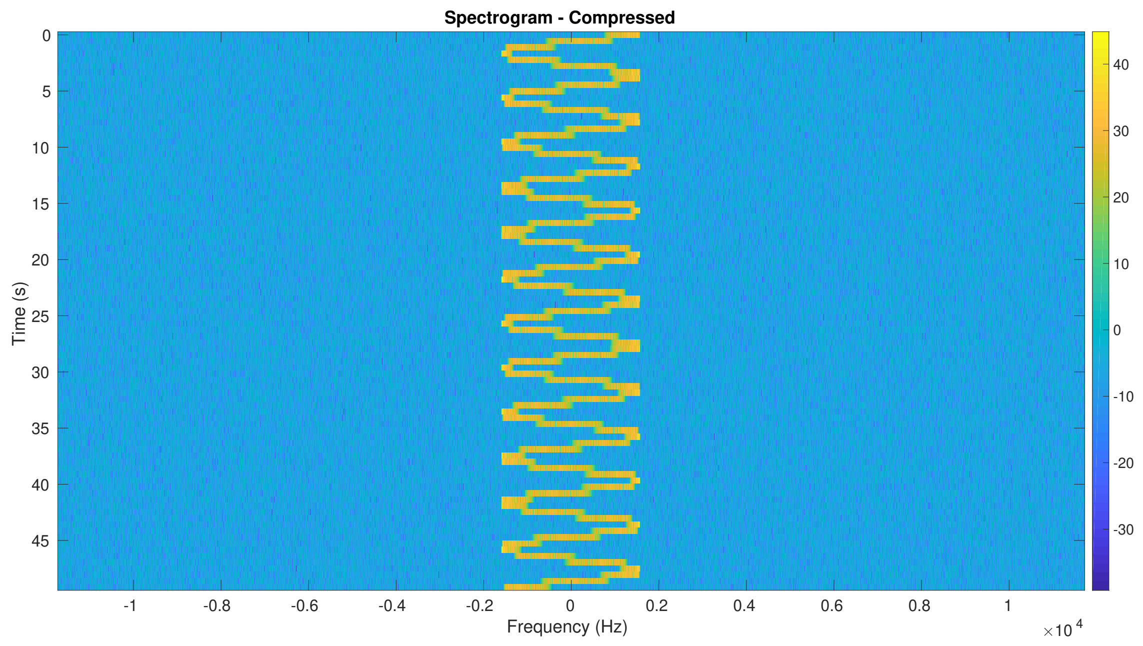


Figure 2.5: Spectrogram of a synthetic signal computed using the compressed norm approach (via the zoom-FFT-based SRFT projection matrix).

product $v[j, r]$:

$$A[j, n] \approx L \sum_{r=0}^{R-1} v[j, r] e^{-2\pi i \frac{rn}{R}}.$$

The unaveraged spectrogram (2.12) and the above formula for the CAF are nearly identical. The CAF merely omits the window function (or has an implicit ones window, if you prefer).

Therefore, once we have the filtered and downsampled mixing product $v[j, r]$, computed with the corner-turn algorithm, we can efficiently compute a delay-compressed CAF using the ideas of the previous section. The j'th row of the output will be an approximation to $\overline{A}_{\text{mag2}}[j'P, n]$, and so we may call it a delay-decimated CAF. We instead prefer the term **delay-compressed** to emphasize the FJLT ideas involved, with the understanding that the output is directly comparable to the output of `TransformCAF(delay_dec=`P`)`, for example.

Algorithm 6 summarizes the procedure for computing the delay-compressed CAF based on the corner-turn algorithm and zoom-FFT-based SRFT. Note that the delay-compressed CAF must still compute the full filtered-and-downsampled mixing product $v[j, r]$. This is still a costly procedure, but the delay-compression will allow us to compute far fewer row-wise DFTs, which are also an asymptotically dominant expense. Delay compression still yields a significant speedup over the corner-turn CAF, though not as much as `TransformCAF(delay_dec=`P`)` with the ones filter trick.

Algorithm 6 `TransformCAF(delay_comp=`P`, gamma=`γ`)`

```
Input: Max delay $J$, max frequency shift $N$, integration time $K = LR$, where $L, R$ are integers
    such that $R \geq 2N+1$, delay-decimation factor $P$, compression factor $\gamma$, complex baseband signals
    $x[k], y[k]$ of sufficient length, lowpass filter $h[m]$ of length $M$
Output: $\overline{A}_{\mathrm{mag2}}[j'P, n]$ for $0 \leq j' \leq J/P$, $-N \leq n \leq N$

  ▷ Compute $v[j, r]$ via the corner-turn approach
  for $r = 0, \ldots, R-1$ do
      $f_r[m] = h[m]x[rL - m]$ for $0 \leq m \leq M-1$
      $g_r[j] = y[rL + j]^*$ for $0 \leq j \leq J$
      $v[j, r] = \sum_{m=0}^{M-1} f_r[m] g_r[j-m]$ for $0 \leq j \leq J$          ▷ Use overlap-scrap convolution
  end for

  ▷ Compute the compressed blocks $\widetilde{B}_{j'}$
  $P' = \lceil \gamma P \rceil$
  for $j' = 0, \ldots, J/P$ do
      $b_{j'}[p, r] = v[j'P + p, r]$ for $0 \leq p \leq P-1$, $0 \leq r \leq R-1$.
      $\widetilde{b}_{j'} = \Phi b_{j'}$                                        ▷ Compress columns of $b_{j'}$
      $\widetilde{B}_{j'}[p', n] = \sum_{r=0}^{R-1} \widetilde{b}_{j'}[p', r] e^{-2\pi i \frac{rn}{R}}$          ▷ $P'$ row-wise DFTs
  end for

  ▷ Approximate $\overline{A}_{\mathrm{mag2}}[j'P, n]$ by computing column-wise norms of $\widetilde{B}_{j'}$
  for $j' = 0, \ldots, J/P$ do
      $\overline{A}_{\mathrm{mag2}}[j'P, n] \approx \frac{L^2}{P'} \sum_{p'=0}^{P'-1} \left| \widetilde{B}_{j'}[p', n] \right|^2$ for $-N \leq n \leq N$.
  end for
```

Figure 2.6 shows a zoomed-in view of an example CAF computed with the delay-compressed CAF (using the zoom-FFT-based SRFT projection matrix). The CAF was computed using the same signals as in Figure 2.1 and zoomed in to the same region. We also took the elementwise square-root of the CAF to make the plot comparable to the other CAF figures. In order to allow the comparison of Figures 2.6 and 2.2, we again choose $P = 250$. The compression factor γ is set to $-15\text{dB} \approx 0.03$.

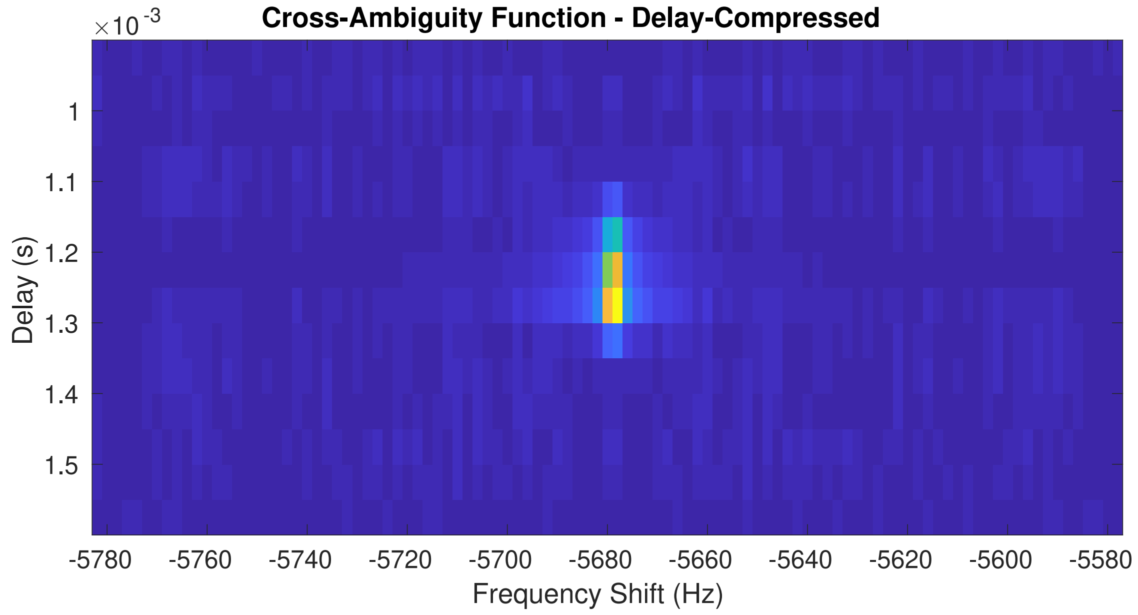


Figure 2.6: An example of a CAF computed with `TransformCAF(delay_comp=`250`,` `gamma=`$-$15dB`)`.

2.5 Discussion

2.5.1 A Runtime Experiment

A substantial part of this project was the construction of C++ implementations of the algorithms discussed in this chapter. We created a simple vector operations library for computations such as multiplying two complex vectors. The library is written to directly utilize SIMD intrinsics, though not necessarily optimally for any particular CPU or sequence of instructions. We use OpenMP to parallelize various outer loops in our codes, for example the two main loops in Algorithm 3. FFTs are computed either with FFTW3 [37] or Intel's MKL (using the FFTW3 interface).

Table 2.1 shows the runtime of the algorithms used to generate the three example CAFs shown in Figures 2.1, 2.2, and 2.6. As one can clearly see, the delay-decimated CAFs run substantially faster than the standard corner-turn CAF, `TransformCAF`. `TransformCAF(delay_dec=`250`)` runs about 35 times faster than the corner-turn CAF, which is a massive speedup. The delay-compressed algorithm, `TransformCAF(delay_comp=`250`,` `gamma=`$-$15dB`)` also sees a significant speedup, though not as dramatic. For comparison, we also show the runtime of `AlgebraicCAF`, which has a fixed delay-decimation factor of $P = L = 250$, and `TransformCAF(delay_dec=25)`, which uses $P = 25$.

Algorithm (Figure Number)	Runtime (seconds)
`TransformCAF` (2.1)	3.49
`TransformCAF(delay_dec=250)` (2.2)	0.10
`TransformCAF(delay_comp=250, gamma=`-15dB`)` (2.6)	1.32
`AlgebraicCAF`	0.56
`TransformCAF(delay_dec=25)`	0.82

Table 2.1: Runtime to compute example CAFs

`TransformCAF(delay_dec=`P`)` reduces the cost of `TransformCAF` in two interconnected ways. First, it reduces the number of rows of the filtered-and-downsampled mixing product that need to be computed. Second, since there are fewer rows, the work spent computing row-wise DFTs is reduced. When increasing the delay-decimation factor P, the amount of work is reduced for both portions of the computation.

`TransformCAF(delay_comp=`P`, gamma=`γ`)`, on the other hand, computes the full filtered and downsampled mixing product, regardless of P. It then does a bit more work to compress blocks of the mixing product. The number of row-wise DFTs is then reduced, which is the primary manner in which the amount of work is reduced (there are also fewer magnitude-squared computations, but that is insignificant compared to the DFTs). That is, when increasing the delay-decimation factor P, the amount of work is only reduced for the row-wise DFT computation. So while the algorithm is effective, the speedup is limited.

The CAF peak is clearly visible for all examples, so we expect that a peak detection algorithm will correctly identify the presence and coarse location of the peak in all cases. To find a more precise peak location, we run the standard corner-turn CAF algorithm over a small region enclosing the peak. For example, a region of size 2×10^{-4}s by ± 20Hz, with FFT-based interpolation to decrease the frequency shift grid spacing to less than 0.01Hz, runs in 0.018s (not counting the cost of time shifting and mixing the input signals). As long as there are not a large number of peaks in the CAF that each require fine location estimates, the costs of computing fine peak locations should not pose too great a burden.

These simple runtime experiments show that our delay-decimated CAF algorithms bring a potentially large speedup to the problem of CAF peak detection. Our algorithms give a massive speed up by computing a coarse peak location, which can then be refined for a small additional cost. The simple ones filter trick and delay-compression proved to be effective for the synthetic example shown in this chapter and also for many other examples not shown here.

2.5.2 Future Directions

One expectation of the averaged magnitude-squared spectrogram (2.13) is the reduction in noise variance as P, the length of the average, increases. This has the benefit of possibly increasing the SNR of the spectrogram by a small amount. Indeed, we observe this in exploratory experiments. We also see that the compressed spectrogram also enjoys an increase in SNR, though by a slightly smaller amount. The implicit sum spectrogram sees a decrease in SNR, as one might expect from the crude approximation used. Characterizing the SNR improvement for the compressed spectrogram, and also for the CAF, could be useful.

It would also be beneficial to understand when the ones filter trick will fail. Clearly, if `TransformCAF(delay_dec=`P`)` fails to produce a peak when `TransformCAF` does, then it cannot be used for coarse peak detection. For instance, suppose cancellation occurs for a particular value of P. Does decreasing the delay decimation factor mitigate the cancellation? What role does the SNR of the input signals play?

We have worked exclusively with the narrowband CAF in this chapter. Adapting our ideas to algorithms for the wideband CAF would be valuable as well. One straightforward approach is to use sums of short-time, narrowband CAFs as in [96].

Our definition of $\overline{A}_{\text{mag2}}$ in (2.11) set us up to use FJLT-like ideas to compress the ℓ_2 norm. It may be preferable to use the ℓ_∞ norm instead, in which case we would compute

$$\overline{A}_{\max}[j'P, n] = \max_{0 \le p \le P-1} |A[j'P + p, n]|.$$

Can we modify the delay-compressed CAF to allow for use of the ℓ_∞ norm?

Our implementation `TransformCAF(delay_dec=`P`)` saw a large speedup over `TransformCAF`, but the improvement is actually not optimal. Recalling the FLOP counts in Subsection 2.2.3 and 2.3.5, we should see a reduction in runtime that scales with $1/P$. Almost all of the runtime taken by `TransformCAF(delay_dec=`P`)` is spent computing $v[j'P, r]$ with a filter-and-downsample subroutine, so future work to speed up the implementation should begin there.

2.5.3 Acknowledgements

James Folberth and Stephen Becker thank Northrop Grumman for their generous support throughout this project.

Chapter 3

Algorithms for Focused-Spot Fluorescence Microscopy

In this chapter we study non-negative least-squares (NNLS) problems arising from a new technique in super-resolution fluorescence microscopy. Our technique involves scanning a focused illumination spot across the object and collecting an image for each illumination spot. To form a super-resolved image, we solve an NNLS problem for each collected image to recover intensities of atoms in a dictionary of point spread functions (PSF). The recovered intensities are then assembled into the final super-resolved image. Details on the specifics of the fluorescence microscope our collaborators built, as well as experiments on how and when this method achieves super-resolution, are in our paper [98].

The image formation process results in a large number of moderate-sized NNLS problems, each with the same PSF dictionary. We take advantage of this special structure by adapting an optimal first-order method to efficiently solve many NNLS problems simultaneously. The PSF dictionary matrix is extremely ill-conditioned, so we also experiment with using a block-diagonal preconditioner and the alternating direction method of multipliers (ADMM) to improve convergence.

With the goal of certifying the uniqueness of the recovered PSF intensities from the NNLS problems, we develop a safe feature elimination strategy for NNLS problems that eliminates PSFs from the solution that are guaranteed to have zero intensity at an optimal point. If sufficiently many PSFs are eliminated, this allows us to certify the uniqueness of the solution to the original NNLS problem. We take inspiration from recent works in the literature for ℓ_1-regularized least-squares (e.g, safe feature elimination for LASSO [38]), though our method instead uses an inexact, but accurate, primal-dual point pair, making it more robust and able to be used on our extremely ill-conditioned NNLS problems.

Though we focus on our microscopy application, our strategy is applicable to general NNLS problems and has applications beyond certification of solution uniqueness. For example, our method can always give a certificate of how far away the objective value is from the optimal value, by providing a duality gap estimate. When we have certified a solution as being unique, we can also bound the distance away from the optimal solution. The method can be extended to other problems with a non-negativity constraint; for example, we sketch how to extend the method to Poisson maximum likelihood.

3.1 Introduction

3.1.1 Super-Resolution Fluorescence Microscopy

Fluorescence microscopy is a technique in optical microscopy that uses fluorescence instead of the reflection/absorption of the illumination light to create images. In short, a fluorescence microscope works by exciting fluorophores with higher-frequency excitation light, filtering out the excitation light, and detecting the lower-frequency light emitted by the fluorophores. To image a (biological) sample with fluorescence microscopy, the sample is treated with fluorescent stains which bind to certain parts of the sample. This allows researchers to image specific portions (e.g., actin filaments, DNA in cell nuclei) of the sample with minimal interference from reflected light from other parts of the sample.

A useful resolution limit for incoherent imaging is the Rayleigh distance

$$d_R \approx \frac{0.6\lambda}{\mathrm{NA}},$$

where λ is the wavelength of light and NA is the numerical aperture, a dimensionless measure of angles at which the microscope objective lens can gather light. Note that the numerical aperture $\mathrm{NA} = n\sin(\alpha)$, where n is the index of refraction of the medium surrounding the lens and α is the lens central half-angle. For example, a lens working in air has $n \approx 1$ and a lens working in an immersion oil has $n \approx 1.5$ for the optical wavelengths we consider. There are practical limits on increasing the lens numerical aperture, so $\mathrm{NA} \leq 1.5$ is reasonable upper bound.

The Rayleigh criterion states that two point sources of equal intensity are just resolved when the maximum of the point spread function of one source is aligned with the first minimum of the other. This is quantified by d_R: two point sources are just resolved if they are separated by a distance d_R. While the Rayleigh limit is a useful estimate of the resolution of an incoherent imaging system, it can be broken under certain circumstances. Indeed, the 2014 Nobel Prize in Chemistry was awarded "for the development of super-resolved fluorescence microscopy" [75].

Almost all current techniques in super-resolution fluorescence microscopy utilize properties of special fluorophores to limit the size of the emitting region or otherwise induce sparsity in the imaged object. Notable examples of this include fluorescence stimulated emission depletion [51] and fluorescing state transitions [12]. Stimulated emission depletion artificially creates much smaller region of emitted light, even though both the illumination spot and detector system are diffraction limited. Through knowledge of the location of this smaller region (it is in the center of the original illumination spot), super-resolution is achieved.

The main exception to the trend of using special fluorophore properties is structured illumination [48], where mixing between periodic illumination patterns directly gives an up-to two-fold increase in resolution. To surpass the two-fold increase with structured illumination, fluorescing state transitions have been used [81].

Donoho argues in [30] that spatial extent/sparsity of the imaged object ("near blackness") as well as non-negativity are crucial to achieving super-resolution in the presence of noise. In fluorescence microscopy, it is not the full spatial extent of the object that is important; rather, it is the spatial extent of the region that is emitting light. Previous works like [51] can be interpreted as using fluorophore properties to artificially limit the spatial extent of the object.

3.1.2 Focused-Spot Illumination Microscopy

We propose in [98] a different route to limit the spatial extent of the imaged object and achieve super-resolution. In widefield fluorescence microscopy, the entire sample is illuminated by a widefield excitation light source. Instead, we artificially limit the spatial extent of the object by focusing the excitation light down to a single, diffraction-limited illumination spot. To cover the entire field of view of the camera, the focused-spot is scanned (with overlap) across the object and an image is collected for each location of the illumination spot. Since the illumination spot is quite small relative to the camera's field of view, almost all of the image is nearly black. Therefore we crop each image around the illumination spot, producing a "subimage" for each illumination spot center position.

We numerically model point spread functions (PSFs) of the microscope for points on a 2D grid that spans the center of a subimage. The grid is constructed to be an integer factor finer than the effective camera pixel size, which is typically a bit smaller than the Rayleigh limit. We call the collection of all the modeled PSF the PSF dictionary. Figure 3.1 shows a diagram of this setup for a grid that is two-times finer than the effective camera pixel size. Note that the PSF grid is truncated outside a circle, since the illumination intensity decreases away from the center of the illumination spot.

Once we have scanned the illumination spot over the sample and collected all subimages, we then solve an optimization problem for each subimage to recover the intensity associated with each PSF in the dictionary. In the example shown in Figure 3.2, there are 40000 such optimization problems. This then gives us the location and intensity of point sources on the fine grid used to the construct the PSF dictionary. These intensities are then shifted according to the scanning pattern and summed to form the image of the full object on the fine grid of PSF center locations. Note that we use **the same PSF dictionary for each optimization problem**; this is a key structure that we exploit in Section 3.2 to create fast solvers.

For the remainder of the chapter, we focus on the subimage optimization problem. Define

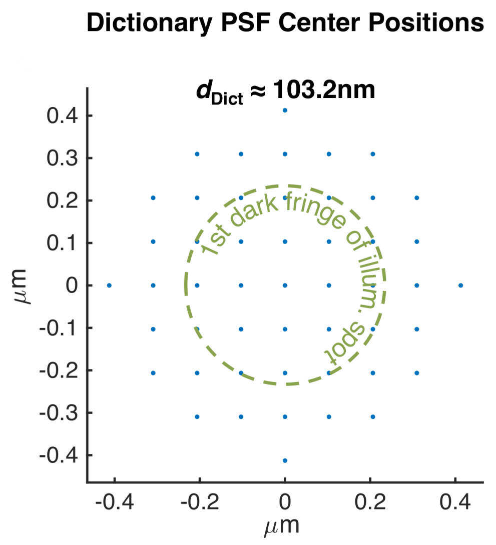


Figure 3.1: Example PSF dictionary grid. PSFs are centered on grid points.

the PSF dictionary matrix $A \in \mathbb{R}^{m \times n}$ where each column is a "vectorized" PSF from the PSF dictionary. PSFs in the dictionary are originally formed as $\sqrt{m} \times \sqrt{m}$ matrices, where $\sqrt{m}$ is the number of camera pixels in a subimage; these matrices are then unrolled into the columns of A. Note that the PSF is the magnitude of the complex PSF, not the complex PSF itself, hence we cannot write A as a convolution. Doing this for all n PSFs in the dictionary gives us A.

To recover the intensities of each PSF in a single subimage b (appropriately vectorized), we solve the following non-negative least-squares (NNLS) problem:

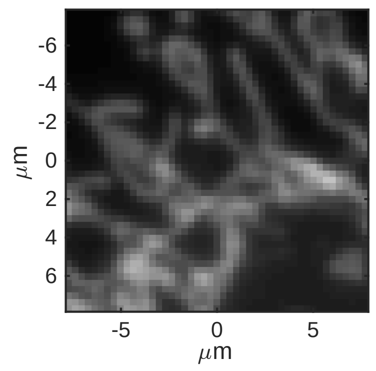
$$\begin{aligned} \min_x \quad & \frac{1}{2}\|Ax - b\|^2 \\ \text{s.t.} \quad & x \geq 0. \end{aligned} \tag{3.1}$$

The NNLS problem seeks to reconstruct the subimage b by finding non-negative intensities x for the PSFs in the dictionary (i.e., the columns of A). The final image is formed by solving (3.1) for each subimage, and then shifting and summing the recovered intensities appropriately. Note again the key structure that the same PSF matrix A is used for each NNLS problem; only the collected subimage b changes.

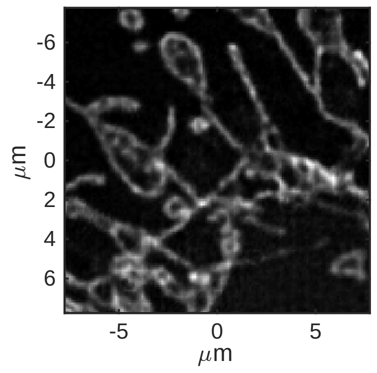
Figure 3.2 shows the result of one such recovery as well as two reference images. Figure 3.2a shows an effective widefield illumination image with a 0.5NA lens. The effective widefield image is a control image formed by directly summing all the focused-spot illumination subimages, without solving any of the PSF intensity recovery problems. Figure 3.2c shows the widefield image using a 1.4NA lens; one can see a large resolution improvement, in accordance with the Rayleigh criterion. Note that 0.5NA effective widefield image is more “pixelated” because the effective camera pixel size is approximately three times larger than with the 1.4NA objective. What we view in Figures 3.2a and 3.2c is effectively what the camera would collect in a widefield illumination setup.

Figure 3.2b shows the result of focused-spot illumination and NNLS-based PSF intensity recovery (FS-NNLS). What we view in the figure is the result of assembling all of the recovered PSF intensities from the NNLS problems, not the view from the camera pixels, as was the case in Figures 3.2a and 3.2c. There is a clear improvement in resolution compared to the 0.5NA effective widefield image.

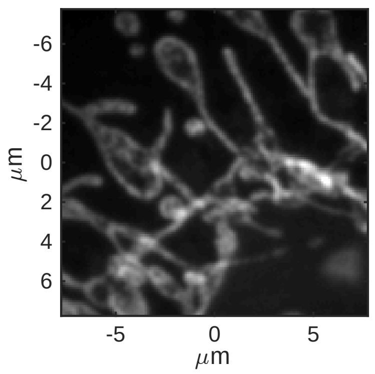
In fact, the details in Figure 3.2b are comparable to, if not better than, those in Figure 3.2c, which uses a 1.4NA objective (vs. the 0.5NA objective used in Figure 3.2b). An example where our approach is better than the 1.4NA objective is the triangular structure in the bottom right. Since the 1.4NA objective has a shallow depth of field, the object appears blurry. The 0.5NA objective used for the FS-NNLS image formation has a much longer depth of field, and FS-NNLS appears to have resolved the structure cleanly, whereas the 1.4NA objective does not.



(a) 0.5NA effective widefield



(b) 0.5NA FS-NNLS



(c) 1.4NA widefield

Figure 3.2: Example 0.5NA effective widefield image, 0.5NA focused-spot NNLS recovery, and 1.4NA widefield image. Our NNLS post-processing is applied only to Figure 3.2b, not Figures 3.2a and 3.2c.

Figure 3.2b uses a PSF dictionary grid spacing three times finer than the camera pixel size. This results in $A \in \mathbb{R}^{m \times n}$ of size $m = 1681$ and $n = 1010$ (i.e., 1010 PSFs of size 41×41). There are 40000 subimages collected, so we must solve the NNLS problem (3.1) with the same A for 40000 different right-hand sides (RHSs). In other scenarios, we use an even finer PSF grid spacing, upwards of 12 times finer, which results in an "underdetermined" A, where $m < n$. In all cases, PSF dictionary matrix is extremely ill-conditioned (e.g., $\kappa_2(A) \approx 10^{20}$), a complicating factor which motivates much of our work. Table 3.1 shows the sizes and condition numbers of some of our PSF dictionary matrices (condition numbers are computed using `dgesvj` compiled to use quadruple precision [5]).

PSF Grid Spacing	Size of A (# Camera Pixels $\times$ # PSFs)	$\kappa_2(A)$
$1\times$ finer	1681×114	2.1×10^{5}
$2\times$ finer	1681×442	7.6×10^{16}
$3\times$ finer	1681×1010	1.1×10^{20}
$4\times$ finer	1681×1794	2.3×10^{21}
$5\times$ finer	1681×2822	2.4×10^{20}

Table 3.1: Sizes and condition numbers of PSF dictionaries.

Our paper [98] has many more details on the operation of the microscope, as well as further experiments and simulations.

3.1.3 Contributions

In this chapter we focus on quickly and reliably solving the NNLS problem (3.1) with the same A for many RHS b. To solve the problems quickly, we adapt an accelerated first-order method from [7] with improvements from TFOCS [11] to run more effectively on blocks of many RHS. Since first-order methods for NNLS use A as a linear operator (e.g., computing matrix-vector products Ax and $A^T r$), we implement the method on a GPU, which gives much improved performance. We also explore using a block diagonal preconditioner with accelerated first-order methods and implementing ADMM efficiently for our highly ill-conditioned NNLS problems.

It is natural to ask whether the PSF intensities recovered by NNLS are unique. Motivated by this question, we develop a safe feature elimination strategy for a general NNLS problem. Inspired by safe feature elimination (SAFE) for LASSO [38], we construct a method that safely eliminates features/columns/PSFs from A when the corresponding intensity $x_i = 0$ at an optimal point. The method is safe in the sense that it will only eliminate a column of A if the intensity x_i is certified to be zero at any optimal point. We make a number of departures from SAFE for LASSO, in part due to the differences in the dual geometries of LASSO and NNLS. Perhaps the most significant is that our strategy uses an inexact primal-dual point pair, making it robust enough to be used for our highly ill-conditioned NNLS problems. Under reasonable assumptions, we prove that our feature elimination strategy will eventually eliminate all zero features from the problem.

If we eliminate sufficiently many features from an underdetermined NNLS problem, we can form a reduced problem (i.e., an NNLS problem with just the non-eliminated columns) that is overdetermined. If the problem has full-rank, then by strong convexity it has a unique optimal point. Since our feature elimination strategy is safe, this implies that the original underdetermined NNLS problem also has a unique optimal point. When used on the image reconstruction problem for Figure 3.2b, for example, we certify that all 40000 NNLS problems have a unique solution, meaning that the reconstructed image is unique.

Our feature elimination strategy has applications beyond our FS-NNLS problems, as it is applicable to general NNLS problems. For any NNLS problem, we can bound the objective suboptimality, $f(\hat{x}) - f(x^*)$, for some feasible point $\hat{x} \geq 0$. If we have reduced an underdetermined NNLS problem to a strongly-convex reduced NNLS problem, we can also bound the distance from $\hat{x}$ to the unique optimal point x^*, which is a much stronger guarantee. We also give a short discussion on how our strategy can be incorporated into first-order and active set methods to improve convergence speed (see Subsection 3.5.3).

3.1.4 Organization and Notation

In Section 3.2 we discuss various methods to solve our moderately sized NNLS problem for a large number of RHS. We first explore accelerated first-order methods in Section 3.2.1. We then introduce a block diagonal preconditioner in Section 3.2.2, though it has limitations. In Section 3.2.3 we describe issues encountered when implementing ADMM for our highly ill-conditioned NNLS problems.

We introduce a basic version of our safe feature elimination procedure for NNLS problems in Section 3.3. We prove various properties of the method, notably that it will "eventually work", in a certain sense to be made precise later. We discuss strengthened versions of the procedure in Section 3.4, as well as certifying the uniqueness of all NNLS problems in an example focused-spot illumination problem.

In Section 3.5 we offer a discussion of the optimization methods we studied. We discuss further applications of our safe feature elimination strategy beyond the certification of unique solutions. Finally, we offer a number of future directions, including extending the strategy to other non-negativity constrained optimization problems.

We denote the ith column of the matrix A by a_i. For a subset $\mathcal{I} \subseteq \{1, \dots, n\}$, we let $A_{\mathcal{I}}$ denote the matrix formed by the columns a_i for $i \in \mathcal{I}$. For all other vectors, we use x_i to denote the ith element of the vector. We use $\langle x, y \rangle$ and $x^T y$ to denote the Euclidean inner product, and $\|x\|$ to denote the Euclidean norm. We will typically use λ, ν to represent dual variables/vectors/points and x^* to denote the value of the quantity x at an optimal point. In particular, we denote the primal optimal value of a problem by p^* and the dual optimal value by d^*.

3.1.5 Optimization Background

In this subsection we given an overview of the standard definitions and results of convex analysis and optimization that we will use in this chapter. See the popular text [17] or Beck's text [10] for a detailed presentation.

We will work entirely in the Euclidean space $\mathbb{R}^n$ equipped with the standard inner product $\langle x, y \rangle \overset{\text{def}}{=} \sum_{i=1}^n x_i y_i = x^T y$ and the usual induced norm $\| \cdot \|$. A set C is said to be convex if for all $x, y \in C$ and $t \in [0, 1]$, it holds that $tx + (1-t)y \in C$. Two examples of convex sets:

- The halfspace $\{x : \langle a, x\rangle \geq b\}$ for fixed $a \in \mathbb{R}^n$, $b \in \mathbb{R}$. A halfspace is the set of all points on one side of the plane $\langle a, x\rangle = b$, including the plane itself.
- The non-negative orthant $\mathbb{R}^n_+ \overset{\text{def}}{=} \{x : x \geq 0\}$ is the set of all points with non-negative entries.

An extended real-valued function is a function that can take on any real value as well as the infinite values $-\infty$ and ∞. The domain of a function f is the set $\operatorname{dom}(f) \overset{\text{def}}{=} \{x : f(x) < \infty\}$. We adopt the convention that $f(x) = \infty$ for x outside of $\operatorname{dom}(f)$. The indicator function of a set C is

$$\iota_C(x) \overset{\text{def}}{=} \begin{cases} 0 & x \in C \\ \infty & x \notin C. \end{cases}$$

The epigraph of an extended real-valued function is defined by

$$\operatorname{epi}(f) \overset{\text{def}}{=} \{(x, p) : f(x) \leq p, x \in \mathbb{R}^n, p \in \mathbb{R}\}.$$

Note that $\operatorname{epi}(f)$ is a subset of $\mathbb{R}^n \times \mathbb{R}$. A function is said to be proper if it does not attain the value $-\infty$ and there exists an $x \in \mathbb{R}^n$ such that $f(x) < \infty$. A function is said to be closed if its epigraph is a closed set; equivalently, a function is closed iff it is lower semicontinuous.

A function is defined to be convex if its epigraph is a convex set. For a proper function f with convex domain, convexity is equivalent to the usual inequality

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y) \quad \forall x, y \in \operatorname{dom}(f), \forall t \in [0, 1].$$

We will mainly consider continuously differentiable functions, specifically functions with L-Lipschitz continuous gradients:

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\| \quad \forall x, y \in \operatorname{dom}(f).$$

We use the subdifferential $\partial f(x)$ in a non-essential manner in Subsection 3.2.3, so we simply point the reader to either of [10, 82] for its definition and properties.

A function is said to be μ-strongly convex ($\mu > 0$) if $f(\cdot) - \mu/2\|\cdot\|^2$ is convex. This states that the function minus a scaled quadratic remains convex, and is a stronger form of convexity, as the name implies. If f is twice continuously differentiable, then f is μ-strongly convex if $\nabla^2 f - \mu I$ is positive semidefinite. An important property of a strongly convex function is that it has a unique minimizer and the strong convexity constant μ gives a bound on how far away a point is from the optimal point:

Theorem 1 (Theorem 5.25 of [10])**:** *Let f be a proper, closed, and μ-strongly convex function. Then*

(1) f has a unique minimizer x^*,

(2) $\frac{\mu}{2}\|x - x^*\|^2 \leq f(x) - f(x^*)$ for all $x \in \operatorname{dom}(f)$.

Our safe feature elimination strategy for NNLS makes use of Theorem 1, as well as Lagrangian duality, which we discuss next. All of the duality material comes from [17], where it is discussed in more detail. Consider the general convex problem

$$\begin{aligned} \min \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \leq 0, \quad i = 1, \ldots, m \\ & a_i^T x = b_i, \quad i = 1, \ldots, l, \end{aligned} \tag{3.2}$$

where $f_i(x)$, $i = 0, \ldots, m$, are convex and the equality constraints $a_i^T x = b_i$, $i = 1, \ldots, l$ are affine. Note that we use a_i to represent the ith equality constraint vector, not the ith element of the vector a. Let $\mathcal{D}$ be the intersection of the domains of all the f_i; we assume $\mathcal{D}$ is non-empty and f_0 is proper so the problem is feasible. Let p^* be the optimal value.

The Lagrangian function is defined as

$$\mathcal{L}(x, \lambda, \nu) \overset{\text{def}}{=} f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{l} \nu_i (a_i^T x - b_i).$$

The Lagrangian can be interpreted as an augmented form of the objective $f_0(x)$ that takes into account the constraints. The vectors λ and ν are called dual variables. The dual function is defined as the infimum of the Lagrangian over x:

$$g(\lambda, \nu) \overset{\text{def}}{=} \inf_{x \in \mathcal{D}} \left(f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{l} \nu_i (a_i^T x - b_i) \right).$$

Note that the dual function is always concave, since it is the pointwise infimum of a family of affine functions of (λ, ν).

An important property of the dual function is that it provides a lower bound for the primal optimal value p^*:

$$g(\lambda, \nu) \leq p^* \quad \forall \lambda \geq 0, \forall \nu.$$

The Lagrange dual problem seeks the best such lower bound:

$$\begin{aligned} \max \quad & g(\lambda, \nu) \\ \text{s.t.} \quad & \lambda \geq 0. \end{aligned} \tag{3.3}$$

A pair (λ, ν) where $\lambda \geq 0$ and $g(\lambda, \nu) > -\infty$ is called dual feasible. Note that the dual problem is (equivalent to) a convex problem, since the dual objective $g(\lambda, \nu)$ is always concave and the dual feasible set is convex.

Let d^* be the dual optimal value (i.e., the optimal value of (3.3)). Since $g(\lambda, \nu) \leq p^*$ for all dual feasible (λ, ν), we have the inequality

$$d^* \leq p^*.$$

This important relation is called weak duality. If the equality $d^* = p^*$ holds, it is said that strong duality holds. Note that strong duality does not always hold, though weak duality always holds.

There are many results that establish conditions on the primal problem, called constraint qualifications, under which strong duality holds. Slater's condition is one simple constraint qualification: if the primal problem (3.2) is convex and there exists an $x \in \operatorname{relint} \mathcal{D}$ such that

$$f_i(x) < 0 \quad \forall i = 1, \ldots, m \tag{3.4}$$

$$a_i^T x = b_i \quad \forall i = 1, \ldots, l, \tag{3.5}$$

then strong duality holds (see Section 2.1.3 of [17] for the definition of relint, the relative interior of a set) and a dual optimal point exists, meaning that the dual optimal value is achieved. Such a point x is called strictly feasible. Slater's constraint qualification can be relaxed to change (3.4) to $f_i(x) \leq 0$ for any i for which $f_i(x)$ is affine.

Let x be a primal feasible point, (λ, ν) be a dual feasible pair, and assume strong duality holds. Without knowledge of the value of p^*, we can bound how far $f_0(x)$ is from p^* via

$$f_0(x) - p^* \leq f_0(x) - g(\lambda, \nu),$$

where the quantity $f_0(x)-g(\lambda, \nu)$ is called the duality gap. The duality gap is often used in stopping criteria for optimization algorithms. We will use it in our safe feature elimination strategy.

Let us now assume that the objective f_0 and constraint functions $f_1, \ldots, f_m$ are differentiable. Let x^* and (λ^*, ν^*) be any primal and dual optimal points with zero duality gap (so strong duality holds). The Karush-Kuhn-Tucker (KKT) conditions are

$$\begin{aligned}
\nabla f_0(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla f_i(x^*) + \sum_{i=1}^{l} \nu_i^* a_i &= 0 \\
f_i(x^*) &\leq 0, \quad i = 1, \ldots m \\
a_i^T x^* &= b_i, \quad i = 1, \ldots l \\
\lambda_i^* &\geq 0, \quad i = 1, \ldots, m \\
\lambda_i^* f_i(x^*) &= 0, \quad i = 1, \ldots, m.
\end{aligned}$$

In order, the constraints are first-order optimality of x^* minimizing $\mathcal{L}(x, \lambda^*, \nu^*)$ over $x \in \mathcal{D}$, primal feasibility, dual feasibility, and complementary-slackness. The KKT conditions are necessary for x^* and (λ^*, ν^*) to be primal and dual optimal with zero duality gap. For convex problems, the KKT conditions are also sufficient to show optimality with zero duality gap. That is, for f_i convex, if x and (λ, ν) satisfy the KKT conditions, then they are primal and dual optimal with zero duality gap. If Slater's condition holds, then strong duality holds, the dual maximum is attained, and the KKT conditions are necessary and sufficient for x and (λ, ν) to be optimal. Note that Slater's condition holding for the primal problem does not imply that the primal optimal value is attained; for that, we must have Slater's condition holding for both the primal and dual problems.

3.2 Optimization Methods

A major part of the image formation process in our approach is the solution of many moderate-sized NNLS problems. When using a single illumination spot, collecting all subimages takes approximately 100 seconds. By using multiple, well-separated illumination spots, the full sample can be scanned in just a few seconds. Rapidly solving the NNLS problem for each subimage is of great importance, as it will allow users of the microscope to effectively explore many samples.

As Figure 3.3 shows, there are significant differences between a "low" and "high" accuracy solution. The "low" accuracy (which was also used in Figure 3.2) is visually smoother, though still super-resolved; the "high" accuracy solution appears more crisp. However, the "high" accuracy solution does appear to have artificial sparsity, which is an artifact of the NNLS model, not an inaccurate solution. So we must have both fast and accurate solutions to our thousands of NNLS problems.

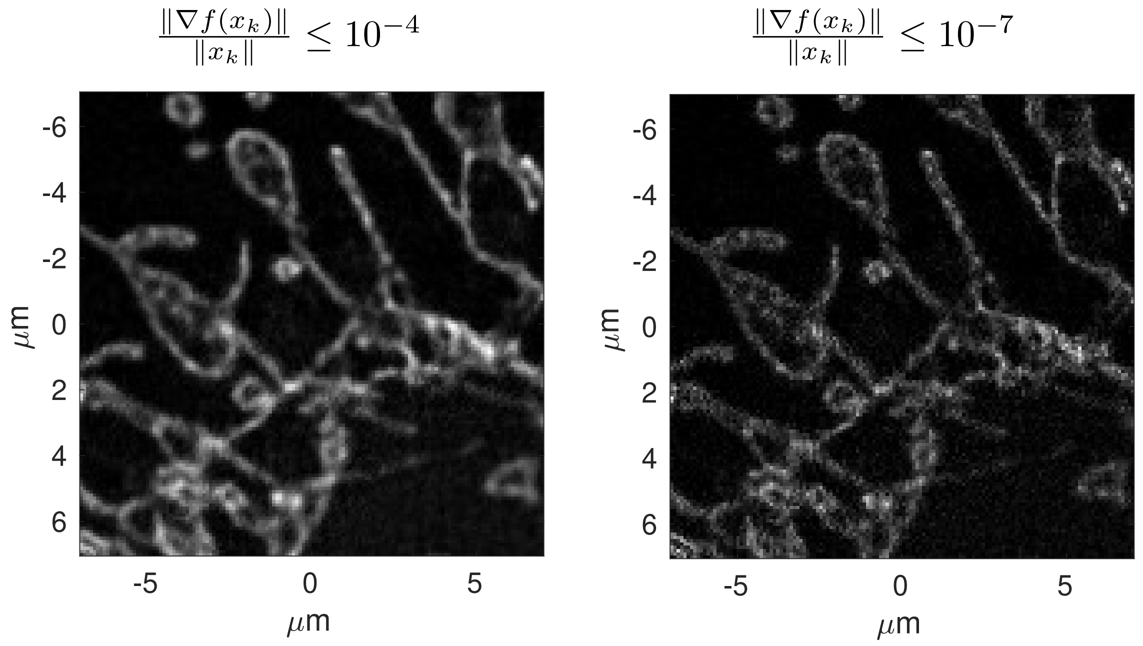

$\frac{\|\nabla f(x_k)\|}{\|x_k\|} \leq 10^{-4}$
$\frac{\|\nabla f(x_k)\|}{\|x_k\|} \leq 10^{-7}$
-6
-4
-2
0
2
4
6
μm
-5
0
5
μm

Figure 3.3: A higher accuracy solution appears more sharp than a lower accuracy solution.

Interior point methods (IPM), a generalization of Newton's method to inequality constrained problems, are both fast and accurate [17]. Internally, IPMs must solve a linear system of equations at each step of the iteration. This linear system changes at each iteration and is different for each RHS, so IPMs are limited to working on a single problem/RHS at a time. But our 40000 NNLS problems are completely independent of each other, giving us an “embarrassingly parallel” situation. For example, we can spread the problems over a large number of nodes in a supercomputer. To give a concrete example, consider the $3\times$ oversampled PSF dictionary matrix A, which is 1681×1010. Solving each of the 40000 NNLS problems with MATLAB’s `quadprog`, which uses an interior point method, requires 90 core hours on one of our workstation computers (dual socket Intel® Xeon™ E5-2630).

But recall that the PSF dictionary A is identical for all the NNLS problems; only the right-hand side (RHS) b changes between problems. This is a special structure that we can exploit using first-order methods, which can be adapted to efficiently work on multiple RHS simultaneously. In a sense, implementations of first-order methods are actually more computationally efficient when working on multiple RHS, as it amortizes costs associated with applying A as a linear operator on multiple vectors. Our GPU implementation of an accelerated projected gradient method solves the same 40000 problems in 26 minutes on a single NVIDIA K40c GPU installed on the same workstation computer. While certainly much slower than the few seconds it takes to collect the subimages, this is a massive speedup over the interior point method.

In Subsection 3.2.1 we adapt an accelerated first-order method from [7] with the adaptive step size scheme and a few other improvements from [11] to run efficiently on multiple RHS simultaneously. The dominant cost of the method involves matrix-matrix products, for which the parallelism of GPUs is particularly well-suited. We discuss implementing the method on a GPU, which yields a major improvement in runtime in our experiments. In Subsection 3.2.2 we explore using a block diagonal preconditioner with first-order methods to improve runtime for ill-conditioned problems. In Subsection 3.2.3 we discuss implementing the alternating directions method of multipliers (ADMM), which brings in second-order information in an inexpensive manner that extends to multiple RHS. ADMM can also be easily implemented on a GPU, and in our experiments, it can converge slightly faster than accelerated first-order methods.

3.2.1 Accelerated First-Order Methods

3.2.1.1 Projected Gradient Method

The projected gradient method (PGM) is the archetypal first-order method one can use to solve NNLS. The essential idea of projected gradient is to take a step to decrease the objective function (i.e., a step in the negative gradient direction) and then project onto the feasible set, as the gradient step may have gone outside the feasible set. This produces the iteration

$$x^{k+1} = P_C\left(x^k - t_k \nabla f(x^k)\right),$$

where P_C is orthogonal projection onto the feasible set C, t_k is a step size, and $\nabla f(x)$ is the gradient of the objective function $f(x)$. Note that we use t_k to denote the scalar step size at the kth iteration, not the k element of a vector. In NNLS, the feasible set is the non-negative orthant $C = \mathbb{R}^n_+ \overset{\text{def}}{=} \{x \in \mathbb{R}^n : x \geq 0\}$, in which case the projection operator is

$$P_C(x) = \max\{0, x\},$$

with the max performed elementwise. Of course, if x is already feasible (i.e., $x \in C = \mathbb{R}^n_+$), the projection returns x: $P_C(x) = x$.

Note that PGM requires computing the projection operation at every iteration. In order for PGM to run quickly, the projection operation must be relatively inexpensive to compute. For NNLS, the projection operation is nearly trivial, so PGM is a good candidate; for problems where the projection is expensive to compute, other methods may be preferred (e.g., interior point methods, which handle the constraint differently).

An important consideration when using PGM is the choice of step size t_k, which must be chosen in order to guarantee convergence, but also produce large enough steps to make substantial progress. If $\nabla f(x)$ is globally Lipschitz continuous with constant L, $t_k = 1/L$ is typically used (in convergence analysis, at least), though any $t_k < 2/L$ will do [10]. Because L is the global Lipschitz constant, if there is a smaller L that applies in a local region, using $t_k = 1/L$ is overly conservative. We will return shortly to selecting step sizes in practice.

A standard result is that PGM produces a sequence with $\mathcal{O}(1/k)$ convergence in function value for a wide class of convex problems.

Theorem 2 (Theorem 10.21 of [10])**:** *Assume the following:*

- *f is proper, closed, has L-Lipschitz continuous gradient, and $C \subset \operatorname{dom}(f)$,*
- *C is convex and non-empty,*
- *the optimal set X^* is non-empty, and the optimal value is p^*.*

Assume further that f is convex. Let x^k be the sequence of points produced by PGM using step size $t_k = 1/L$. Then for any optimal point $x^* \in X^*$ and $k \geq 0$,

$$f(x^k) - p^* \leq \frac{L}{2k}\|x^0 - x^*\|^2.$$

The gradient of the NNLS objective $f(x) = \frac{1}{2}\|Ax - b\|^2$ is

$$\nabla f(x) = A^T(Ax - b).$$

Combined with the projection onto the non-negative orthant, the PGM iteration for NNLS is

$$x^{k+1} = \max\{0, x^k - t_k A^T(Ax^k - b)\}. \tag{3.6}$$

The computational burden of this iteration is in computing the matrix-vector products Ax and $A^T r$, where $r = Ax - b$.

Recall that we have many NNLS problems with the same A but a different RHS for each problem. This allows us to group multiple gradient evaluations together to utilize matrix-matrix products with A and A^T. To do this, we group a number of RHS as columns of the matrix B and group the corresponding block of PSF intensity vectors into the matrix X. We can then evaluate the gradient for all columns in X efficiently via

$$A^T(AX - B),$$

which involves the matrix-matrix products AX and $A^T R$, where $R = AX - B$.

On modern computer architectures with a hierarchical memory architecture, a high-performance matrix-matrix product implementation is generally much faster than repeated use of a matrix-vector product implementation. For example, see Figure 3.4, which shows the runtime of evaluating the NNLS gradient with many matrix-vector products versus using matrix-matrix products for 1681×2822 A. When using a block of 1000 RHS (so X is 2822×1000), computing the gradient using matrix-matrix products is nearly 9 times faster than using many matrix-vector products. The reason for the speedup is the more efficient memory access pattern of the matrix-matrix product implementation. We refer the reader to [42] for a brief discussion of blocking matrix-matrix multiplication.

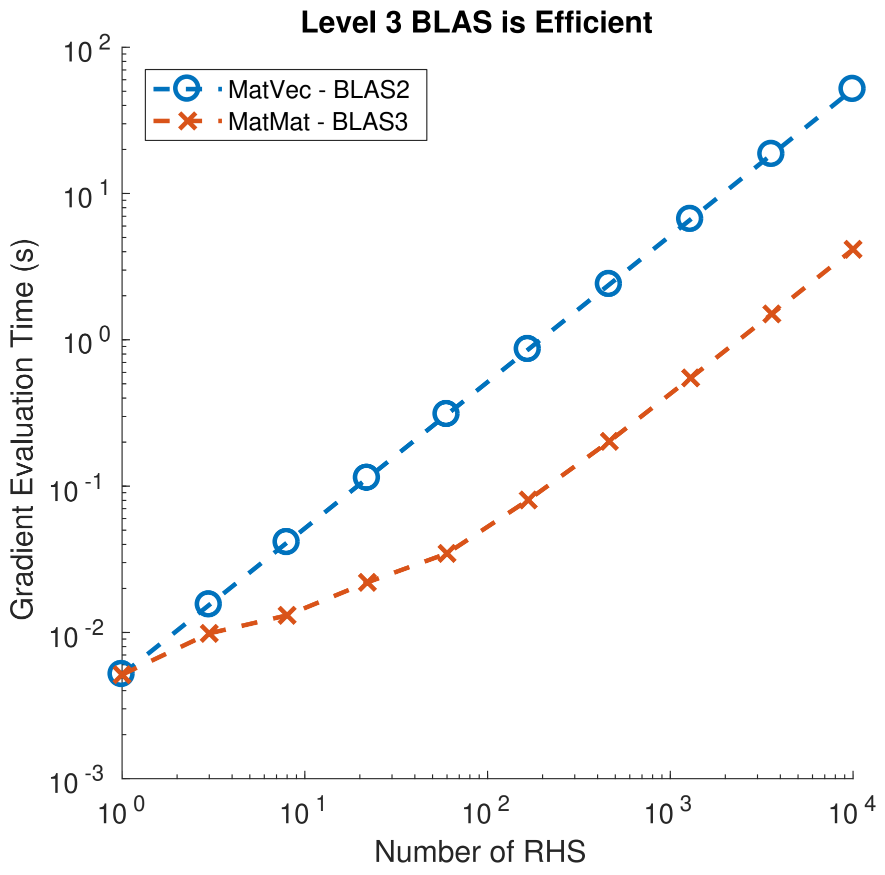


Figure 3.4: Cost of evaluating the NNLS gradient for multiple RHS by utilizing many matrix-vector products or matrix-matrix products.

Projection onto the non-negative orthant $\mathbb{R}^n_+$ trivially extends to a block of columns, as the operation is elementwise separable. In terms of these matrix variables, the PGM iteration for NNLS is

$$X^{k+1} = \max\{0, X^k - t_k A^T(AX^k - B)\},$$

which is, of course, very similar to the single RHS iteration (3.6). Additionally, the objective values for each column in X can be easily computed from $R = AX - B$, which is computed as part of the gradient computation.

3.2.1.2 An Accelerated Projected Gradient Method

We have seen that the PGM offers a computationally efficient method to solve in parallel many NNLS problems with the same A but different RHS. In the past few decades, a great deal of research has been performed on so-called optimal first-order methods. The first such method was discovered by Nesterov in 1983 [72], where it was shown that, for a large class of convex problems, the objective values $f(x^k)$ converge to the optimal value p^* with rate $\mathcal{O}(1/k^2)$. The projected gradient method exhibits $\mathcal{O}(1/k)$ convergence, hence Nesterov's and the many subsequent methods are called accelerated first-order methods. Prior work showed that the optimal rate is $\mathcal{O}(1/k^2)$ [71], hence Nesterov's method is called optimal. For more details on the sense in which such methods are optimal, see the text [73]; Section 5.2 of [11] lists a number of popular optimal first-order methods in a common format that allows for easy comparison of the methods.

That accelerated first-order methods are optimal is itself remarkable. More interesting is the fact that many optimal methods utilize a single gradient evaluation and projection operation per iteration, just like PGM. Indeed, it is typical that the cost of accelerated methods is only slightly higher than PGM, making them generally preferable in practice.

To be concrete, an accelerated method from [7], which we will abbreviate AT, is shown in Algorithm 7. The method introduces auxiliary sequences, giving the iteration a particular form of momentum. The gradient and projection are evaluated using the auxiliary sequences. The role that momentum plays in accelerating convergence is explored in [40]. We can see that the computational expense of AT is very similar to PGM: there is a single gradient and projection computation per iteration, with just two additional convex combinations.

Algorithm 7 AT: Accelerated Projected Gradient Method from [7]

```
Require: Starting point $x^0 \in \operatorname{dom} f$
  $z^0 \leftarrow x^0$, $\theta_0 = 1$
  for $k = 0, 1, 2, \dots$ do
    $y^k \leftarrow (1-\theta_k)x^k + \theta_k z^k$
    $z^{k+1} \leftarrow P_C\left(z^k - \frac{t_k}{\theta_k}\nabla f(y^k)\right)$
    $x^{k+1} \leftarrow (1-\theta_k)x^k + \theta_k z^{k+1}$
    $\theta_{k+1} \leftarrow \frac{2}{1+\sqrt{1+4/\theta_k^2}}$
  end for
```

The choice of step size is of critical importance to both PGM and accelerated methods. While the constant step size $t_k = 1/L$ is used often in theory, as it guarantees convergence for functions with L-Lipschitz continuous gradient, the steps it produces are often too conservative as the iterates approach an optimal point. In other words, the Lipschitz bound

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x-y\| \quad \forall x, y \in \operatorname{dom} f$$

is no longer tight when x, y are near an optimal point x^*. Taking larger steps by increasing t_k can improve performance considerably, though care must be taken to ensure convergence.

3.2.1.3 Improvements from TFOCS

It is customary to use a backtracking line search accomplish this. We use the hybrid backtracking line search with local Lipschitz constant estimation from TFOCS [11], which takes steps $t_k = 1/L_k$, where L_k is a local Lipschitz estimate at step k. The condition used to ensure convergence is

$$f(x^{k+1}) \leq f(y^k) + \left\langle \nabla f(y^k), x^{k+1} - y^k \right\rangle + \frac{L_k}{2}\|x^{k+1} - y^k\|^2. \tag{3.7}$$

With standard backtracking, we would iteratively increase L_k (equivalently, we decrease t_k) until the condition (3.7) is satisfied. This can be done by updating $L_k \leftarrow 2L_k$, for example, then recomputing x^{k+1} and checking the condition. Once the condition is satisfied, we accept the step, giving us the values z^{k+1} and x^{k+1}, and move on to the next iteration.

TFOCS uses the hybrid strategy taking $L_k \leftarrow \max\{2L_k, \hat{L}\}$, where $\hat{L}$ is the smallest value of L_k that satisfies (3.7). It is noted that the condition (3.7) exhibits cancellation errors when $f(x^{k+1}) \approx f(y^k)$, which occurs as the iteration converges. To avoid these errors, TFOCS uses the alternate condition

$$\left|\left\langle x^{k+1} - y^k, \nabla f(x^{k+1}) - \nabla f(y^k)\right\rangle\right| \leq \frac{L_k}{2}\|x^{k+1} - y^k\|^2 \tag{3.8}$$

when $f(y^k) - f(x^{k+1}) \geq \gamma f(x^{k+1})$, where γ is a small positive constant. Additionally, a slight correction to the θ_k sequence is used to account for the changing step size.

One final improvement implemented in TFOCS to make efficient use of applications of the linear operator A. To highlight how A is used, let us reformulate NNLS slightly. Define the function

$$\overline{f}(z) \overset{\text{def}}{=} \frac{1}{2}\|z - b\|^2.$$

Using $\overline{f}$, we can write NNLS as

$$\begin{aligned} \min \quad & \overline{f}(Ax) \\ \text{s.t.} \quad & x \geq 0. \end{aligned}$$

The gradient of the original NNLS objective is $\nabla f(x) = A^T(Ax - b) = A^T \nabla \overline{f}(Ax)$. The gradient $\nabla \overline{f}(z) = z - b$ is much cheaper to compute (given z) than $\nabla f(x) = A^T(Ax - b)$ (given x). By defining the auxiliary sequences $x_A^k = Ax^k$ and $z_A^k = Az^k$, we can form $y_A^k = Ay^k$ without actually computing Ay^k. This allows us to save a matrix multiply when computing the gradient: $\nabla f(y^k) = A^T \nabla \overline{f}(Ay^k) = A^T \nabla \overline{f}(y_A^k)$. Furthermore, the auxiliary sequences allow us to evaluate the convergence condition (3.8) without forming the full gradient $\nabla f(x^{k+1})$. By using definition of the adjoint operator, an identical condition is

$$\left|\left\langle x_A^{k+1} - y_A^k, \nabla \overline{f}(x_A^{k+1}) - \nabla \overline{f}(y_A^k)\right\rangle\right| \leq \frac{L_k}{2}\|x^{k+1} - y^k\|^2.$$

Algorithm 8 lists the accelerated projected gradient method AT (see Algorithm 7) with all of these improvements. Note that there is a single application of A^T, P_C, and A per iteration of the backtracking loop. In our experiments, it is typical that more than 90% of backtracking iterations are accepted, resulting in few wasted applications of A and A^T. If we were to forego the modification of θ_k when L_k is changed, the computation of g_y could be moved outside the backtracking loop; in such a case, there would be a single application of A^T per "main" iteration, and one each of P_C and A per backtracking iteration. However, our experiments reveal that the θ_k modification is quite worthwhile, resulting in fewer iterations and less work overall than would be saved by undoing the θ_k modification and moving the A^T application outside the backtracking loop.

Algorithm 8 AT: Algorithm 7 with Improvements from TFOCS [11]

```
**Require:** Starting point $x^0 \in \operatorname{dom} f$, $\alpha \in (0,1]$, $\beta \in (0,1)$, $\hat{L} > 0$
  $z^0 \leftarrow x^0$, $x_A^0 \leftarrow Ax^0$, $z_A^0 \leftarrow Ax^0$, $L_{-1} = \hat{L}$, $\theta_{-1} = \infty$
  for $k = 0, 1, 2, \ldots$ do
      $L_k = \alpha L_{k-1}$
      loop
          $\theta_k \leftarrow \dfrac{2}{1 + \sqrt{1 + 4L_k/\theta_{k-1}^2 L_{k-1}}}$          ▷ $(\theta_0 \overset{\text{def}}{=} 1)$
          $y^k \leftarrow (1-\theta_k)x^k + \theta_k z^k$
          $y_A^k \leftarrow (1-\theta_k)x_A^k + \theta_k z_A^k$
          $g_y \leftarrow A^T \nabla \overline{f}(y_A^k)$
          $z^{k+1} \leftarrow P_C\left(z^k - \frac{1}{\theta_k L_k} g_y\right)$
          $z_A^{k+1} \leftarrow Az^{k+1}$
          $x^{k+1} \leftarrow (1-\theta_k)x^k + \theta_k z^{k+1}$
          $x_A^{k+1} \leftarrow (1-\theta_k)x_A^k + \theta_k z_A^{k+1}$
          $\widehat{L} \leftarrow 2\left|\left\langle x_A^{k+1} - y_A^k, \nabla\overline{f}(x_A^{k+1}) - \nabla\overline{f}(y_A^k)\right\rangle\right| \Big/ \|x^{k+1} - y^k\|^2$
          if $L_k \geq \hat{L}$ then
              break
          end if
          $L_k \leftarrow \max\{L_k/\beta, \hat{L}\}$
      end loop
  end for
```

3.2.1.4 Modifications to Better Support Multiple RHS

We initially used TFOCS [11] to solve the many NNLS problems arising from our microscopy application. While TFOCS gives an improvement over using an interior point method for each problem individually, there are a couple drawbacks when used for our specific scenario. The first is the manner in which TFOCS handles solving multiple NNLS problems. When an NNLS problem is solved individually, the step sizes are adjusted for that specific problem. When working on multiple RHS, TFOCS treats a block of columns as a matrix variable and performs a single Lipschitz estimate for the block, not the individual columns. This results in a single step sized being used for all columns of the matrix, which unnecessarily slows convergence.

Our implementation fixes this by computing function values and gradients, performing the backtracking line search, Lipschitz estimation, and checks for convergence on a per-column basis. The major computational benefit of using matrix-matrix products instead of multiple matrix-vector products is retained by grouping most operations into large array operations (e.g., we still compute the gradients via $A^T(AX-B)$). This allows us not only to use a different step size for each column, which speeds convergence, but furthermore to prune columns that have already converged. The column pruning allows us to remove columns that have met a convergence criterion from the block of columns, which decreases the cost of subsequent iterations. All of these improvements come at a measurable overhead cost, but we have found a significant net speedup when they are used.

The second downside is that TFOCS is designed for the CPU only. We have access to an NVIDIA K40c GPU, which can greatly speed up the computation of the moderately sized matrix-matrix products AX and A^TR used in the gradient computation. However, there are other operations (e.g., convergence checks, step size adaptation) that are rather costly on the GPU. By using MATLAB's Parallel Computing toolbox, our implementation can carefully blend GPU and CPU operations to see a further improvement in runtime over CPU-only operation.

Specifically, the iteration variables x^k, x_A^k, etc. are held on the GPU, where the large array operations are performed. The Lipschitz estimation procedure requires a significant amount of branching (when both Lipschitz estimates are computed with various checks for numerical round off), so various quantities are computed on the GPU and then transferred to the CPU, where branching is less costly, for final computation of the Lipschitz estimate. The step size decisions and convergence checks are then transferred to the GPU for use in the next iteration.

3.2.1.5 Warm Starting

When using a first-order algorithm, or any algorithm that allows one to specify the initial iterate x^0, the choice of starting point x^0 can be used to our advantage. If we have an $\hat{x}$ that is nearby an optimal point, we can "warm start" the solver with $x^0 = \hat{x}$ to reach an optimal point in fewer iterations. To see this, recall the convergence bound for PGM from Theorem 2:

$$f(x^k) - p^* \leq \frac{L}{2k}\|x^0 - x^*\|^2,$$

for any optimal x^*. Warm starting seeks to shrink $\|x^0 - x^*\|^2$ by picking a more accurate x^0.

Since we have control over the construction of the PSF dictionary, we use the following warm start procedure. We first construct a PSF dictionary on a coarse grid, say $A_{1/3}$ which uses $3\times$ oversampling relative to the effective camera pixel size. This results in $A_{1/3}$ of size 1681×1010. We start with a generic initial guess, such as $x^0_{1/3} = 0$ or $x^0_{1/3} = \max\{0, (A^TA)^{-1}A^Tb\}$, the least-squares solution projected onto $\mathbb{R}^n_+$. We then solve the NNLS problem to a low accuracy, giving $\hat{x}_{1/3}$. We then construct a finer PSF dictionary, say $A_{1/5}$ of size 1681×2822, and interpolate $\hat{x}_{1/3}$ to the new PSF grid and use the interpolated values as the initial guess $x^0_{1/5}$. If $A_{1/5}$ is the final PSF grid we are interested in using, we then solve high accuracy; otherwise we solve to low accuracy and interpolate the solution to the next PSF grid and repeat the process.

This warm start procedure allows us to compute a coarse solution on a smaller PSF dictionary, where gradient evaluations are cheaper and the objective has nicer curvature properties (for small enough dictionaries, the PSF dictionary matrix is strongly convex). We then transfer the coarse solution up to a finer grid, where gradient evaluations are more expensive and convergence is slower. Using this warm start allows allows us to save a small number of iterations on the finer PSF dictionary, speeding up the runtime to reach high accuracy on fine PSF dictionaries.

3.2.1.6 Convergence for a Strongly Convex Objective Function

There are two significant differences between "overdetermined" NNLS ($m \geq n$) and "underdetermined" NNLS ($m < n$). The first is that, in the overdetermined case, we can rewrite an overdetermined NNLS problem as a quadratic program (QP) for cheaper gradient evaluations. Define the $n \times n$ matrix $P = A^T A$, the vector $q = -A^T b$, and the scalar $s = \frac{1}{2}\|b\|^2$. The quadratic program

$$\begin{aligned} \min \quad & \tfrac{1}{2}\langle x, Px\rangle + \langle q, x\rangle + s \\ \text{s.t.} \quad & x \geq 0, \end{aligned} \tag{3.9}$$

is equivalent to NNLS. Note that in the gradient computation for NNLS, we compute the matrix-vector products Ax and $A^T r$, each at a cost of $\mathcal{O}(mn)$ FLOPs. In the QP formulation, the gradient is

$$\nabla f(x) = Px + q,$$

which costs $\mathcal{O}(n^2)$ FLOPs. When $m \geq n/2$, in particular when A is overdetermined, we see that the QP formulation reduces the cost of each gradient computation. The cost of computing $P = A^T A$, $q = -A^T b$, and $s = \frac{1}{2}\|b\|^2$ is not considered, since the cost of repeated gradient evaluations typically outweighs this initialization cost by a large factor, especially when working on multiple RHS. Note that the QP formulation, by explicitly forming $P = A^T A$, has worse ill-conditioning and may be less accurate than the natural NNLS formulation that uses Ax and $A^T r$.

The second difference in the overdetermined, full-rank case is that the NNLS objective is strongly convex. Theorem 2 showed that PGM converges with rate $\mathcal{O}(1/k)$ in function value. For a μ-strongly convex function, this rate bound can be improved. We repeat the following theorem from [10].

Theorem 3 (Theorem 10.29 of [10])**:** *Suppose f satisfies the conditions of Theorem 2 and is also μ-strongly convex ($\mu > 0$). Let x^k be the sequence generated by PGM with constant step size $t_k = 1/L$. Then for all $k \geq 0$,*

$$f(x^{k+1}) - p^* \leq \frac{L}{2}\left(1 - \frac{\mu}{L}\right)^{k+1} \|x^0 - x^*\|^2.$$

The quantity L/μ is called the condition number of the problem, as it controls the rate of convergence of PGM. For NNLS, the condition number $L/\mu = \kappa_2(A^T A) = \sigma_{\max}(A^T A)/\sigma_{\min}(A^T A)$, the usual condition number from numerical linear algebra. Theorem 3 shows that PGM achieves linear convergence for strongly convex problems. Furthermore, there is no change to the method required to achieve such convergence; the improved convergence comes automatically in the presence of strong convexity. Note that if the condition number L/μ is very large, the convergence bound given by Theorem 3 may be weaker for a given k than the $\mathcal{O}(1/k)$ bound given by Theorem 2. Theorem 2 still applies, so convergence will be bounded by whichever bound is tighter.

Accelerated methods, such as AT (see Algorithms 7 and 8), require modification to achieve linear convergence. There is a variant of FISTA, another popular accelerated first-order method, that achieves accelerated linear convergence with improved rate $\mathcal{O}((1-\sqrt{\mu/L})^k)$ [10]. Unlike PGM, however, this variant does not automatically achieve linear convergence, as it requires knowledge of the strong convexity constant μ.

The strong convexity constant can be difficult to estimate in practice. A popular approach that also can give accelerated linear convergence is to iteratively restart the acceleration of an accelerated first-order method (i.e., set $\theta_k = 1$ every Kth iteration). The number K is typically manually tuned to the problem at hand, a process which does not require knowledge of the strong convexity constant. It is typical that iteratively restarted accelerated methods with tuned K achieve improved linear convergence compared to PGM. Optimal values of K are known for certain methods, though they require the strong convexity constant [10, 46].

3.2.2 Block Diagonal Preconditioner

In this subsection we consider overdetermined, full-rank NNLS problems, which have a strongly convex objective function. Our PSF dictionary matrices are full-rank and overdetermined for grid spacing 1× to 3× finer than the camera pixel sizes. The matrix sizes and condition numbers are given in Table 3.1. The associated NNLS problems have condition number $L/\mu = \kappa_2(A^TA)$. As long as the condition number is not too high, we expect to see linear convergence for PGM and AT (with tuned iterative restarts) with rate depending on $\kappa_2(A^TA)$. Through the use of a preconditioner to decrease $\kappa_2(A^TA)$, we seek to improve the speed of linear convergence.

Remark: *Our PSF dictionaries are extremely ill-conditioned ($\kappa_2(A) \approx 10^{20}$), and so we expect the* *convergence of PGM and AT to follow the $\mathcal{O}(1/k)$ and $\mathcal{O}(1/k^2)$ rates, respectively. Nevertheless, we explore preconditioned first-order methods applied to the QP* (3.9) *with "random" ill-conditioned problems with more moderate condition numbers.*

For our real PSF dictionaries, we noticed that the NNLS Hessian $P = A^T A$ concentrates along the main diagonal. This is due to PSFs with nearby centers having larger inner products than PSFs that are further separated. The concentration along the main diagonal can be further strengthened by a permutation $\Pi P \Pi^T$ that organizes PSFs with nearby grid centers to have similar indexes in A. Such a permutation simply amounts to re-indexing the columns of A and elements of x, which does not change the NNLS problem, but can allow us to find a stronger preconditioner. For the remainder of this subsection, we will consider the permutation to have been applied.

Let M be the preconditioner, which of course must be invertible. In order to guide the construction of M, let first see how to use it to precondition NNLS. We insert M into NNLS by performing a change of variables:

$$\begin{array}{ll} \min & \frac{1}{2}\|Ax - b\|^2 \\ \text{s.t.} & x \geq 0 \end{array} \iff \begin{array}{ll} \min & \frac{1}{2}\|AMM^{-1}x - b\|^2 \\ \text{s.t.} & x \geq 0 \end{array} \iff \begin{array}{ll} \min & \frac{1}{2}\|AMz - b\|^2 \\ \text{s.t.} & Mz \geq 0 \end{array}$$

The Hessian of the rightmost problem is $M^T A^T A M$, but note that the constraint is now $Mz \geq 0$. We will solve the rightmost problem with PGM or AT, which requires that we project onto the set $\{z : Mz \geq 0\}$. This projection returns the (unique) optimal value of

$$\begin{array}{ll} \min & \frac{1}{2}\|\zeta - z\|^2 \\ \text{s.t.} & M\zeta \geq 0. \end{array}$$

After the change of variables $M\zeta = \xi$, the problem is

$$\begin{array}{ll} \min & \frac{1}{2}\|M^{-1}\xi - z\|^2 \\ \text{s.t.} & \xi \geq 0. \end{array} \tag{3.10}$$

This projection subproblem is itself an NNLS problem of of size $n \times n$, which is comparable in size to the original NNLS problem. We must therefore pick M with some structure that allows for quickly computing the projection, otherwise the cost of projection may outweigh the reduction in iterations granted by improving the conditioning of the original NNLS problem. We discuss these and other structural constraints in Subsection 3.5.1.

A simple choice of M is a diagonal matrix, which is considered in [74]. For diagonal M, the projection subproblem (3.10) can be solved in closed form:

$$\xi_i = \max\{0, M_{ii} z_i\},$$

from which we find $\zeta_i = \xi_i / M_{ii}$. While this allows for fast projection, the improvement in conditioning is nearly negligible in our experiments.

We instead consider a block diagonal preconditioner to increase the improvement in conditioning, while allowing the projection subproblem (3.10) to be block-separable, which aids in its fast solution. We will use square blocks of size $N \times N$, where we assume for simplicity that n is divisible by N. In practice, we can account for blocks of different sizes and this does not adversely affect the improvement in conditioning.

In order to decrease the condition number $\kappa_2(M^T A^T A M)$, we populate blocks of M with the inverse matrix square root of the corresponding block of $A^T A$. See Figure 3.5 for an example of $A^T A$ and a sketch of how we form M. Increasing the block size N will generally decrease $\kappa_2(M^T A^T A M)$, leading to fewer iterations of the first-order method.

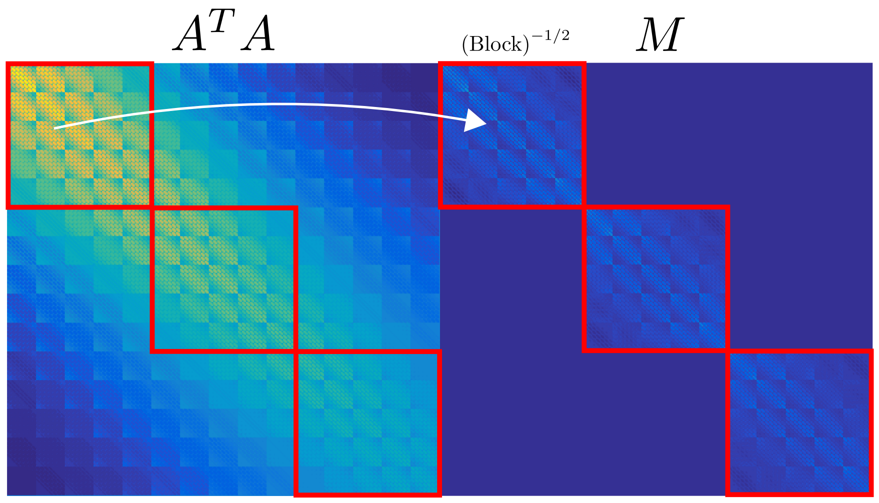


Figure 3.5: Forming the block diagonal preconditioner M from the NNLS Hessian $P = A^T A$.

To solve the projection subproblem (3.10), we use that the problem is block separable. Let M_i be the ith block of M, z_i the ith block of z, and ξ_i the ith block of ξ. We can project the ith block independently from the other blocks with the block subproblem

$$\begin{array}{ll} \min & \frac{1}{2}\|M_i^{-1}\xi_i - z_i\|^2 \\ \text{s.t.} & \xi_i \geq 0, \end{array}$$

which is itself a smaller, $N \times N$ NNLS problem. To project z onto $\{z : Mz \geq 0\}$, we solve all block subproblems and then assemble the blocks, giving us the projection.

Since the block subproblem is small, active set and interior point methods are computationally efficient. We use SPAMS [61], which provides a fast implementation of LARS [35], which can be used to solve NNLS problems. The complexity of LARS for an $N \times N$ NNLS problem is roughly $\mathcal{O}(N^3)$ FLOPs, depending on the solution path (see [62] for a discussion of this, where it is shown that the solution path has exponentially many segments in the worst case). There are $\lceil n/N \rceil$ blocks, so the total cost of projecting z onto $\{z : Mz \geq 0\}$ is roughly $\mathcal{O}(nN^2)$. Thus we see that increasing the block size leads to increased projection costs when using the preconditioner. There is therefore a trade-off between increasing the block size N to improve the problem condition number and decreasing the block size to speed up each iteration.

We run a simple experiment to test this trade-off on synthetic problems with a more moderate condition number. We generate a random matrix $A = Q\Sigma^{1/2}$, where $Q \in \mathbb{R}^{m\times n}$ is a random orthogonal matrix [69] and $\Sigma \in \mathbb{R}^{n\times n}$ is a random "covariance" matrix with entries

$$\Sigma_{ij} = e^{-\gamma|i-j|}.$$

This particular choice of A and Σ is meant to concentrate larger values along the diagonal of A^TA. The parameter γ is chosen so that $\kappa_2(A) = 10^6$.

We pick a random RHS b and find an optimal point using `lsqnonneg` in MATLAB, which implements an active-set method from [57]. Alternatively, we can generate a random NNLS solution by first picking a solution $x^* \in \mathbb{R}^n$; for example, we could populate x^* with standard normal entries and then threshold $x^* \leftarrow \max\{0, x^*\}$. We can then find an RHS b that will result in x^* being an

NNLS solution according to Appendix C. Once we have the NNLS problem with known solution, we then run AT with and without preconditioning until $(f(x^k) - f(x^*))/f(x^*) \leq 5 \times 10^{-4}$, measuring the runtime in each case.

Figure 3.6 shows the runtime of preconditioned AT relative to the runtime of AT without the preconditioner for two problems. There is a strong dependence on block size, which is indicative of the trade-off between improving problem conditioning and the cost of the projection. We do see an improvement in runtime at the best block size, although it is rather mild.

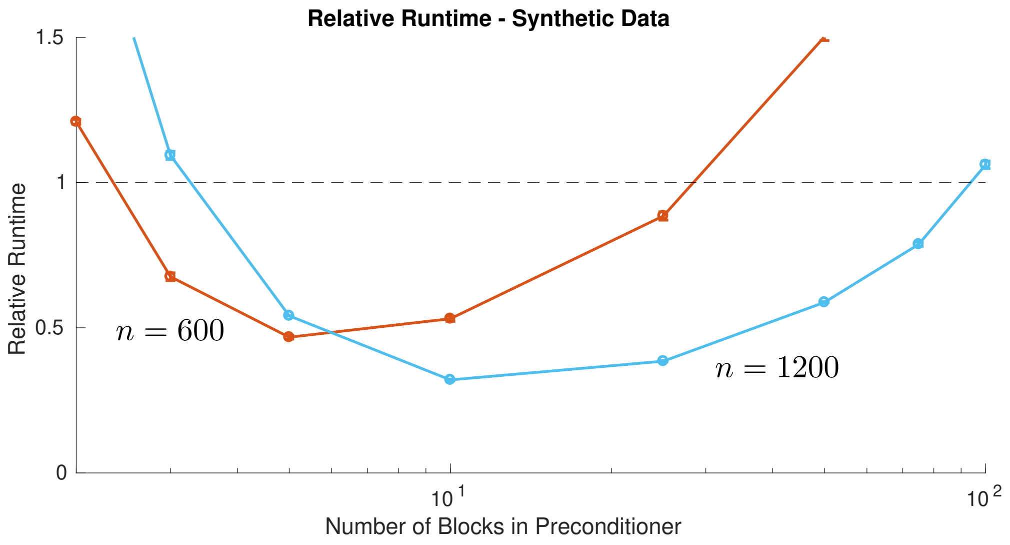


Figure 3.6: Relative runtime versus number of blocks in preconditioner (number of blocks $= n/N$).

There is room for possible improvement in this preconditioned approach. The discussion in Subsection 3.5.1 highlights a more effective way to construct a block diagonal preconditioner. Note that this discussion is limited to running on a CPU since we use SPAMS, which is limited to the CPU. As it currently stands, the improvement gained by running unpreconditioned AT on the GPU far outweighs preconditioned AT on the CPU. To efficiently utilize a GPU, we need to avoid communicating between the GPU and host as much as possible. For this reason we would likely need to implement LARS, or another fast solver for small NNLS problems, on the GPU.

3.2.3 Alternating Direction Method of Multipliers

First-order methods, like those discussed in Subsection 3.2.1, by definition are limited to using only up to first-order information. The use of a block diagonal preconditioner can be though of as introducing second-order information (i.e., information related to the Hessian A^TA) in a somewhat inexpensive manner. However, introducing the preconditioner adversely affects the projection subproblem, ultimately limiting the performance of the method.

Interior-point methods utilize the Hessian directly by solving a saddle-point system of linear equations at each iteration. Note that this linear system depends on the current iterate, say x^k, and thus changes at each iteration (and per RHS). While very efficient for finding a high-accuracy solution for a relatively small number of RHS, the costs associated with solving many different saddle-point systems becomes very costly. Indeed, this is why we do not consider interior-point methods for solving the large number of NNLS problems arising from our microscopy application.

The alternating direction method of multipliers (ADMM) when applied to NNLS offers a simple method that efficiently utilizes a simple form of second-order information. The review article [16] provides a nice introduction and survey for those readers unfamiliar with ADMM.

3.2.3.1 The Basic Version of ADMM

To begin, we define the following problem:

$$\begin{array}{ll} \min_{x,y} & \frac{1}{2}\|Ay-b\|^2 + \frac{\rho}{2}\|x-y\|^2 \\ \text{s.t.} & x \geq 0 \\ & x = y, \end{array} \tag{3.11}$$

where $\rho > 0$ is a penalty parameter. Note that (3.11) is equivalent to NNLS, due to the constraint $x = y$ and $\rho > 0$. The motivation behind using this objective is that we have separated the least-squares objective (notice that it is now in terms of y) from the non-negativity constraint $x \geq 0$. The augmentation $\frac{\rho}{2}\|x-y\|^2$ serves to penalize $x \neq y$, and can be thought of as a remnant of the augmented Lagrangian method applied to NNLS; see [16] or [10] to see how ADMM comes from the augmented Lagrangian method.

The ADMM iteration for (3.11) is as follows.

$$\begin{aligned}
x^{k+1} &= \operatorname{argmin}_{x\geq 0} \quad \tfrac{1}{2}\|Ay^k - b\|^2 + \tfrac{\rho}{2}\|x - y^k\|^2 - \langle \lambda^k, x - y^k\rangle \\
y^{k+1} &= \operatorname{argmin}_{y} \quad \tfrac{1}{2}\|Ay - b\|^2 + \tfrac{\rho}{2}\|x^{k+1} - y\|^2 - \langle \lambda^k, x^{k+1} - y\rangle \\
\lambda^{k+1} &= \lambda^k - \rho(x^{k+1} - y^{k+1}).
\end{aligned}$$

The variables λ^k can be interpreted as dual variables. With ADMM's connection to the augmented Lagrangian method and dual ascent, we can interpret ρ as a step size [16]. Note that the update for y^{k+1} involves the updated variable x^{k+1}, akin to Gauss-Seidel for solving linear systems of equations [42].

The subproblem for computing x^{k+1} is equivalent to

$$\min_x \iota_{\mathbb{R}^n_+}(x) + \frac{\rho}{2}\|x - y^k\|^2 - \langle \lambda^k, x - y^k\rangle.$$

The optimum of this ρ-strongly convex (but not differentiable) problem satisfies

$$0 \in \partial\left(\iota_{\mathbb{R}^n_+}(x) + \frac{\rho}{2}\|x - y^k\|^2 - \langle \lambda^k, x - y^k\rangle\right),$$

By Theorem 3.40 of [10] (on when the subdifferential distributes across addition), this condition simplifies to

$$0 \in \partial\iota_{\mathbb{R}^n_+}(x) + \rho(x - y^k) - \lambda^k.$$

From this, we see that the solution is

$$x^{k+1} = \max\left\{0, y^k + \frac{1}{\rho}\lambda^k\right\}.$$

The subproblem for computing y^{k+1} is unconstrained and smooth, so we use the first-order optimality conditions

$$0 = A^T(Ay - b) - \rho(x^{k+1} - y) + \lambda^k.$$

Solving for y, we have the update

$$y^{k+1} = (A^TA + \rho I)^{-1}(A^Tb + \rho x^{k+1} - \lambda^k).$$

Algorithm 9 shows the two updates together, along with the update for λ^k. We typically choose $y^0 \leftarrow x^0$ and $\lambda^0 \leftarrow 0$ to initialize the algorithm. Of course, we are not the first to apply ADMM to NNLS: see [13, 87] for some recent examples. There is also an accelerated version of ADMM [41], which is based on accelerated first-order methods. In our experiments, it offers a modest improvement without much additional computation, as is typically the case with accelerated first-order methods.

Algorithm 9 ADMM for NNLS (3.11)

```
Require: Initial points $x^0$, $y^0$, $\lambda^0$, step size $\rho > 0$.
1: for $k = 0, 1, 2, \ldots$ do
2:     $x^{k+1} \leftarrow \max\left\{0, y^k + \frac{1}{\rho}\lambda^k\right\}$
3:     $y^{k+1} \leftarrow \left(A^TA + \rho I\right)^{-1}\left(A^Tb + \rho x^{k+1} - \lambda^k\right)$
4:     $\lambda^{k+1} \leftarrow \lambda^k - \rho(x^{k+1} - y^{k+1})$
5: end for
```

Lines 2 and 4 of Algorithm 9 are very cheap to compute. The second step involves $(A^TA + \rho I)^{-1}$, which is the simple form of second-order information to which we have previously alluded. As written, Algorithm 9 uses a fixed parameter ρ for all iterations. Therefore we need only "precompute $(A^TA + \rho I)^{-1}$" at the start of the algorithm, and can "multiply by the inverse" at each iteration.

The final new ingredient for ADMM is the choice of the step size ρ. There are strategies for varying ρ, for example the simple scheme described in Section 3.4.1 of [16], originally from [50]. In practice, we find that a manually tuned, fixed ρ can often yield much faster convergence than the simple varying scheme.

3.2.3.2 Cholesky Variant

In order to implement Algorithm 9, we need to apply $(A^TA + \rho I)^{-1}$ as a linear operator. Naïvely, one could precompute the inverse matrix $(A^TA + \rho I)^{-1}$ and use matrix-vector products. We instead solve the linear system

$$\left(A^TA + \rho I\right) y^{k+1} = w, \quad \text{where } w \overset{\text{def}}{=} A^Tb + \rho x^{k+1} - \lambda^k.$$

This provides a more numerically stable implementation of the inverse operator [42, 52]. Furthermore, this leads to relatively cheap iterations, as we can precompute a factorization of $A^TA + \rho I$ in order to implement the linear system solve.

Suppose A is overdetermined ($m \geq n$). We compute the Cholesky factorization of $A^TA+\rho I$, which is guaranteed (in infinite precision arithmetic) to be positive definite since $\rho > 0$. That is, we compute the factorization $A^TA + \rho I = R^TR$, with R upper triangular. The linear system solve (i.e., the update for y^{k+1}) is then

$$y^{k+1} = R^{-1}R^{-T}w, \tag{3.12}$$

where R^{-T} and R^{-1} are applied using forward/backward substitution.

In the underdetermined case, $A^TA + \rho I$ is $n \times n$ and may be quite large compared to A. We can avoid computing this product by using the matrix inversion lemma [52]:

$$\left(A^TA + \rho I\right)^{-1} = \frac{1}{\rho}I - \frac{1}{\rho}A^T\left(AA^T + \rho I\right)^{-1}A, \tag{3.13}$$

where $AA^T + \rho I$ is $m \times m$, which is smaller than $A^TA + \rho I$. We then compute the Cholesky factorization $AA^T + \rho I = R^TR$, giving the update

$$y^{k+1} = \frac{1}{\rho}w - \frac{1}{\rho}A^TR^{-1}R^{-T}Aw. \tag{3.14}$$

Note that in both the over- and underdetermined cases, we can apply ADMM to a block of RHS simultaneously. Just like the discussion in Subsection 3.2.1.1, this allows us to use more efficient matrix-matrix operations, ultimately speeding up runtime by a significant factor. We can also move the computations onto a GPU for further improved performance.

Unlike Algorithm 8 however, Algorithm 9 does not adjust the step size ρ either per RHS or as the iteration proceeds. If we were to change ρ, we would find that ρ is embedded in the Cholesky factor R, and a new Cholesky factorization would need to be computed in order to change the step size. We return to this topic in Subsection 3.2.3.4.

3.2.3.3 Comparison with Accelerated First-Order Method

Here we compare the performance of Algorithm 8, an accelerated projected gradient method with improvements from TFOCS, versus Algorithm 9, ADMM. We test the methods on $A \in \mathbb{R}^{1681\times 1010}$, which uses $3\times$ oversampling relative to the camera pixel size, and run on a block of 50 RHS. Note that this problem is strongly convex, so we use Algorithm 8 with iterative restarts at every 500 iterations. Algorithm 8 uses per-RHS step sizes that are adjusted with the backtracking line search. In contrast, Algorithm 9 uses a fixed step size ρ that was manually tuned to achieve the fastest convergence for the entire block of RHS; for this example, we use $\rho = 2 \times 10^{-5}$ (with A normalized to have $\|A\|_2 = 1$). ADMM uses the y^{k+1} update with (3.12).

Figure 3.7 shows the convergence of each algorithm. Note that the quantity labeled as "Relative Norm of Gradient" uses the norm using only the indexes where the constraint $x \geq 0$ is inactive. The sharp jumps downward for AT occur when the acceleration is restarted. We see that ADMM initially converges slower than AT, but eventually overtakes AT and ends up with more than an order of magnitude more accuracy after 25000 iterations.

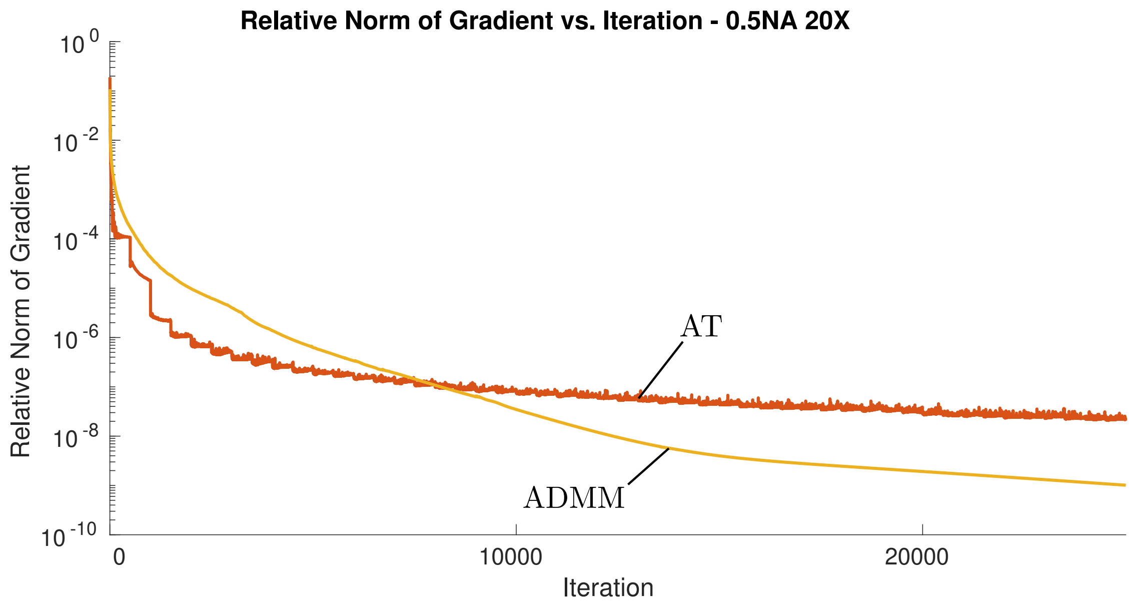


Figure 3.7: AT (Algorithm 8) with per-RHS variable step sizes and iterative restarts vs. ADMM (Algorithm 9) with a manually tuned, fixed ρ.

Note that Figure 3.7 shows convergence based on iteration number, so we calculate the per-iteration cost of each method to show that for NNLS, a single step of AT is comparable to a single step of ADMM. In AT, the dominant costs is the matrix-matrix product with $P = A^T A$ (with P precomputed in the QP-formulation; see (3.9)). In ADMM for overdetermined problems with the Cholesky-based update (3.12), the dominant cost is forward-backward substitution with $R^{-1}R^{-T}$. With A of size 1681×1010, we find that a step of ADMM is very nearly the same cost as a step of AT when using our GPU implementations. Therefore, comparing by iteration count is fair for this case, and we see that ADMM with a manually tuned ρ is preferable to AT if we need a relative norm of the gradient less than, say, 10^{-7}.

For ADMM with underdetermined A, the Cholesky-based update (3.14) is more expensive than a step of AT. A step of AT costs $\mathcal{O}(2mn)$ FLOPs whereas the Cholesky-based update (3.14) costs $\mathcal{O}(2mn + m^2)$ FLOPs. Notice that for large n, the difference is less severe. In our GPU implementation, the $5\times$ oversampled PSF dictionary matrix of size 1681×2822, a step of ADMM is roughly 1.5 times as expensive as a step of AT. Unlike the overdetermined, strongly convex example shown in Figure 3.7, the convergence of ADMM is on par with AT, even with a tuned ρ. Thus the added cost of ADMM with update (3.14) is not justified for our underdetermined NNLS problems. However, the update (3.14) is not the only way to implement the operation $(A^T A + \rho I)^{-1} w$, as we explore in the next subsection.

3.2.3.4 SVD Variant

We saw in Subsection 3.2.3.2 how to use the Cholesky factorization and matrix inversion lemma to implement the solution to

$$\left(A^T A + \rho I\right) y = w.$$

There are a few disadvantages with this approach. One we have already seen is that for an underdetermined NNLS problem, an iteration of ADMM (see (3.14)) is more expensive than an iteration of AT. We could precompute the product $A^T R^{-1} R^{-T} A$, or just $R^{-T} A$, from (3.14), but we expect this to incur worse round off errors than applying each operation in order from right to left [52].

Another possible issue is that the numerical round off error in the Cholesky factorization scales with $\kappa_2(A^TA+\rho I)$ (for the overdetermined case) [52]. Though ρ does serve to regularize the system $A^TA+\rho I$, we tune ρ to achieve fast convergence of ADMM, not necessarily to ease the ill-conditioning of $A^TA+\rho I$. Therefore in the process of tuning ρ, we may inadvertently make the condition number quite large (e.g., if ρ is small), leading to an inaccurate implementation of $(A^TA+\rho I)^{-1}w$. Moreover, the Cholesky factorization might even fail. The tuned value of ρ in our practical problems usually results in moderate conditioning, e.g. $\kappa_2(A^TA+\rho I)=10^5$. In synthetic ill-conditioned problems the tuned value of ρ typically results in poor conditioning of $A^TA+\rho I$, however; we discuss this further in Subsection 3.2.3.5.

Finally, the step size ρ is embedded in the Cholesky factorizations used in (3.12) and (3.14). If we wanted to change ρ, we would need to recompute the factorization. This also impedes us from using a different ρ for each RHS, as we would need a different factorization for each ρ we want to use.

An alternative approach is to precompute the "thin" singular value decomposition $A=U\Sigma V^T$. For an overdetermined problem, can straightforwardly find the update

$$y^{k+1}=V\left(\Sigma^2+\rho I\right)^{-1}V^Tw. \tag{3.15}$$

Note that $\Sigma^2+\rho I$ is diagonal, so implementing the inverse operation is trivial. Moreover, since ρ is not coupled to the factorization, this allows us to change ρ as we please, perhaps even using a different ρ for each RHS. Observe that the Cholesky-based update (3.12) is approximately one half the cost of the SVD-based update (3.15), since R is $n\times n$ triangular and V is $n\times n$ dense.

In the underdermined case, we must take note that U is orthogonal but V is only orthonormal, so $V^TV=I$ but $VV^T\neq I$. Using the matrix inversion lemma (3.13), we have the update

$$y^{k+1}=\frac{1}{\rho}w-\frac{1}{\rho^2}V\left(\Sigma^2+\frac{1}{\rho}I\right)^{-1}V^Tw \tag{3.16}$$

This update can also be used with a changing ρ. The dominant operations in this update are the multiplications by V^T and V, which means that this update is cheaper than the Cholesky-based update (3.14).

In fact, the cost of this update is asymptotically equal to the cost of evaluating the NNLS gradient. In a sense, ADMM allows us to use second-order information for free, compared to gradient-based methods. Indeed, in our GPU implementation of ADMM with this update, a single iteration takes the same amount of time as an iteration of Algorithm 8 implemented on the GPU. However, recall that the convergence of ADMM is not that much faster than AT for our problems, as shown in Figure 3.7. Since we manually tune the step size ρ, we generally prefer Algorithm 8, with its automatic step size selection.

One lingering detail is the accuracy of computing the SVD of A. With the Cholesky variant, we computed the Cholesky factorization of $A^TA + \rho I$. For our NNLS problems (with very ill-conditioned A), the step size ρ was such that the condition of $A^TA + \rho I$ was quite moderate, making the Cholesky factorization quite accurate. However, for the SVD variant just discussed, we have no such regularization and must pay “full freight” on the conditioning of A. For our microscopy NNLS problems, the SVD variant still allows us to achieve sufficiently high accuracy solutions, even though $\kappa_2(A)$ is very large.

In experiments with random ill-conditioned matrices, however, both the Cholesky variant and the SVD variant were not accurate enough to produce very high accuracy solutions. In the next subsection we explore a ways to increase the accuracy of the updates, thereby allowing ADMM to achieve very high accuracy.

3.2.3.5 Iterative Refinement

In this subsection we consider using ADMM to achieve very high accuracy on an underdetermined, random, ill-conditioned NNLS problem. We take $A \in \mathbb{R}^{100\times 150}$ formed as

$$A = U\Sigma V^T$$

where U is a random 100×100 orthogonal matrix (i.e., sampled from the Haar distribution on orthogonal matrices [69]) and V is a random 150×150 orthogonal matrix. The singular value matrix $\Sigma = \mathrm{diag}(1, \sigma^1, \ldots, \sigma^{99})$ is chosen so $\kappa_2(A) = 10^6$. This gives us a random ill-conditioned A. We generate a random solution x with half of its entries equal to 0 and the other half sampled from the standard normal distribution. The RHS b is generated according to the procedure in Appendix C.

Figure 3.8 shows the convergence of AT, ADMM, and ADMM with iterative refinement (to be described shortly; see Algorithm 10). We plot the relative norm of the gradient on the set of indices where the constraint $x \geq 0$ is inactive. Note that we use a combined linear and logarithmic scale, with the transition between scales occurring at iteration 5000.

AT and ADMM both converge rapidly for the first 1000 iterations, but then AT slows dramatically. ADMM continues converging rapidly, until around iteration 2000 when it stops converging abruptly and makes no further progress. With iterative refinement, ADMM continues its fast convergence past iteration 2000, stopping around iteration 4000 at a relative norm of 10^{-11}, which corresponds to about 16 orders of magnitude improvement in the relative norm of the gradient.

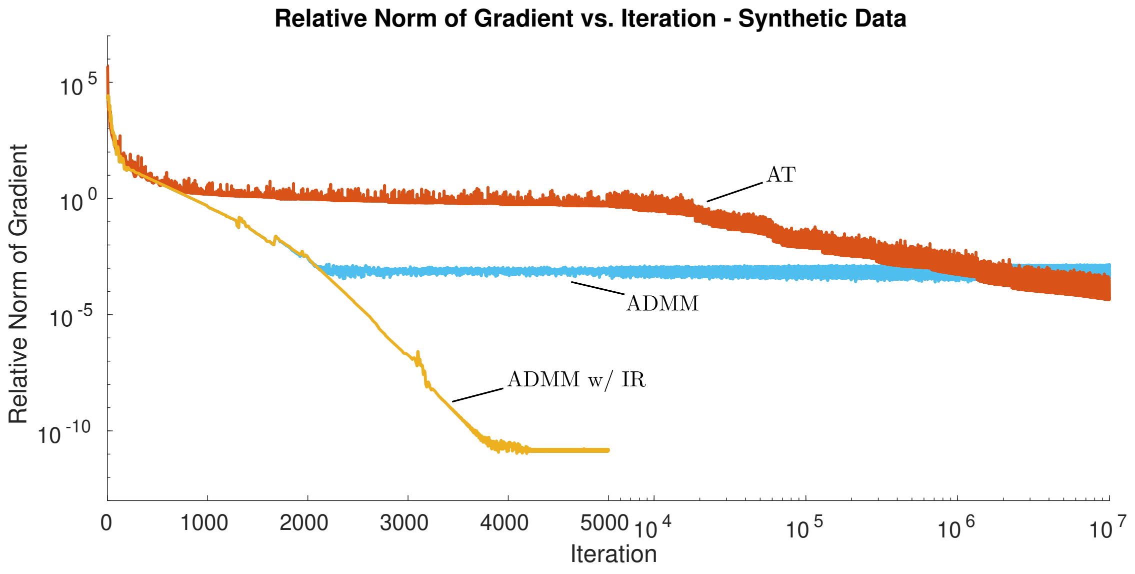


Figure 3.8: AT (Algorithm 8), ADMM (Algorithm 9), and ADMM with iterative refinement. Note the combined linear and logarithmic scales.

It is clear that ADMM converges significantly faster than AT for this synthetic problem. What limits ADMM's convergence after 2000 iterations is numerical round off in the implementation of the y^{k+1} update. For the underdetermined problem we consider, we manually tuned the step size, finding $\rho = 0.1$ to be the best. We can use either the Cholesky update (3.14) or the SVD update (3.16). In the example, we use the SVD update, as the relevant condition number $\kappa_2(A) = 10^6$ is lower than the relevant condition number $\kappa_2(A^T A + \rho I) = 10^7$ for the Cholesky update. Indeed, using the Cholesky update causes the convergence floor for ADMM to rise from 10^{-3} (SVD update) to 10^{-1} (Cholesky update).

To increase the accuracy, we could compute all steps of ADMM in extended precision, but that would likely be extremely slow. An alternative approach is to precompute the required factorization (either Cholesky or SVD) in extended precision, cast the factors back to double precision, and carry out the ADMM iterations in double precision. This uses extended precision quite sparingly, as the factorization is computed only once, at the start of the algorithm. An alternative method to possibly achieve a higher accuracy SVD using double precision is to use a Jacobi SVD algorithm, e.g. `dgesvj` in LAPACK [5]. `dgesvj` is sometimes able to compute small singular values and vectors more accurately than algorithms which first tridiagonalize A [27].

To test this, we cast A to 128-bit floats, compute the SVD with `dgesvj` compiled to use 128-bit floats instead of double precision (64-bit floats), and then cast the results back to double precision for use with the double precision ADMM iterations. We run the SVD variant and observe the convergence floor drops from 10^{-3} to 10^{-4}. So simply increasing accuracy of the factorization in this manner is not enough to fix the convergence floor.

Iterative refinement is a simple technique to enhance the accuracy of a linear system solve, like those used in the y^{k+1} updates. Fixed precision iterative refinement for a system $Rx = b$, where R is some non-singular matrix, is as follows [42]:

```
1: Solve $R\hat{x} = b$ for $\hat{x}$
2: Compute the residual $r \leftarrow b - R\hat{x}$
3: Solve $R\delta = r$ for $\delta$
4: Update $\hat{x} \leftarrow \hat{x} + \delta$
```

Steps 2 to 4 can be repeated to possibly improve accuracy further. Note that all steps are carried out in a fixed precision (e.g., double precision). In our experiments, fixed precision iterative refinement using double precision did not improve convergence.

A more accurate approach is to use extended precision for the residual computation. This is called mixed precision iterative refinement:

```
1: Solve $R\hat{x} = b$ for $\hat{x}$
2: Compute the residual $r \leftarrow b - R\hat{x}$          ▷ Compute in extended precision
3: Solve $R\delta = r$ for $\delta$
4: Update $\hat{x} \leftarrow \hat{x} + \delta$
```

Note that only step 2 is computed using extended precision. Specifically, the quantities b, R, and $\hat{x}$ are cast to an extended precision, the computation is carried out, and the result is cast back to standard precision, giving r. The mixed precision version is typically slower than the fixed precision version, due to the use of extended precision computations.

We implement this strategy for the SVD update (3.16) in Algorithm 10. We use 64-bit floats as the standard precision (double precision) and 128-bit floats as the extended precision (quadruple precision). We denote extended precision quantities with a subscript 128, and denote casting between standard and extended precision quantities through the subscript. Note that we write the algorithm to use a single iteration of iterative refinement, as only a single iteration was needed for our experiments. Steps 10-14 could be repeated (with minor modification) to perform multiple iterations.

Algorithm 10 was used to achieve high accuracy in Figure 3.8. ADMM without iterative refinement takes about 0.2 seconds to run 5000 iterations. ADMM with mixed precision iterative refinement takes about 7.2 seconds to run 5000 iterations. While that is certainly an increase in runtime, it is still much better than attempting to reach the same accuracy with AT. In comparison, AT takes 20 minutes to run 10 million iterations, and only achieves an accuracy of 10^{-4}.

In Algorithm 10, each one of the extended precision computations are actually necessary to achieve very high accuracy. If any extended precision step is removed, the convergence floor is raised by many orders of magnitude. Note in particular that we compute the “right-hand side”

Algorithm 10 ADMM with Iterative Refinement for Underdetermined NNLS

```
Require: Initial points $x^0$, $y^0$, $\lambda^0$, step size $\rho > 0$.
 1: Cast $A_{128} \leftarrow A, b_{128} \leftarrow b$                    ▷ Cast to extended precision
 2: Compute SVD $A_{128} = U_{128}\Sigma_{128}V_{128}^T$              ▷ Compute in extended precision
 3: Cast $U \leftarrow U_{128}$, $\Sigma \leftarrow \Sigma_{128}$, $V \leftarrow V_{128}$       ▷ Cast to standard precision
 4: for $k = 0, 1, 2, \ldots$ do
 5:     $x^{k+1} \leftarrow \max\{0, y^k + \frac{1}{\rho}\lambda^k\}$
 6:     Cast $x_{128}^{k+1} \leftarrow x^{k+1}, \lambda_{128}^k \leftarrow \lambda^k$
 7:     $w_{128} \leftarrow A_{128}^T b_{128} + \rho x_{128}^{k+1} - \lambda_{128}^k$              ▷ Precompute $A_{128}^T b_{128}$
 8:     Cast $w \leftarrow w_{128}$
 9:     $\hat{y} \leftarrow \frac{1}{\rho}w - \frac{1}{\rho^2}V\left(\Sigma^2 + \frac{1}{\rho}I\right)^{-1}V^T w$
10:     Cast $\hat{y}_{128} \leftarrow \hat{y}$
11:     $r_{128} \leftarrow w_{128} - A_{128}^T A_{128}\hat{y}_{128} - \rho\hat{y}_{128}$
12:     Cast $r \leftarrow r_{128}$
13:     $\delta \leftarrow \frac{1}{\rho}r - \frac{1}{\rho^2}V\left(\Sigma^2 + \frac{1}{\rho}I\right)^{-1}V^T r$
14:     $y^{k+1} \leftarrow \hat{y} + \delta$
15:     $\lambda^{k+1} \leftarrow \lambda^k - \rho(x^{k+1} - y^{k+1})$
16: end for
```

w_{128} in extended precision in step 7. The standard precision quantity (cast down from extended precision) is used in the linear system solve in step 9, but the original extended precision quantity is used to compute the residual in step 11. If we compute w in standard precision or use w instead of w_{128} in step 11, we do not achieve very high accuracy.

3.3 Safe Feature Elimination for NNLS - Theory

El Ghaoui et al. in [38] proposed the safe feature elimination (SAFE) method to safely eliminate features from the ℓ_1-regularized least-squares problem

$$\min_x \frac{1}{2}\|Ax - b\|^2 + \mu\|x\|_1. \tag{3.17}$$

In [38], they call SAFE applied to the above problem SAFE for LASSO. The salient feature of SAFE for LASSO is that the eliminated features (i.e., columns of A) are **guaranteed** to not be present in a solution. This is unlike other feature elimination strategies, e.g., [89] which have more aggressive feature elimination criteria, but at the cost of potentially eliminating a feature that should be present in a true solution. Inspired by ideas from SAFE for LASSO, we formulate a safe feature elimination strategy for NNLS problems.

One of our ultimate goals of using feature elimination is to certify the uniqueness of an NNLS solution. NNLS problems with overdetermined, full-rank A are strongly convex and therefore have a unique solution. For NNLS problems with underdetermined A, we have no such guarantee. Eliminating features from an underdetermined NNLS problem allows us to form a reduced problem involving A_{red} with fewer columns. If the reduced problem forms an overdetermined, full-rank NNLS problem, then the reduced problem is strongly convex and has a unique solution. Since the feature elimination strategy is safe, this then implies that the original, underdetermined NNLS problem also has a unique solution.

3.3.1 NNLS Dual Problem and Primal KKT Conditions

Before we proceed with our discussion of feature elimination, let us derive an NNLS dual problem and KKT conditions for the primal problem. To derive a dual, let us introduce the auxiliary variable $z = Ax - b$ and work with the equivalent problem

$$\begin{aligned} \min \quad & \frac{1}{2}\|z\|^2 \\ \text{s.t.} \quad & z = Ax - b \\ & x \geq 0. \end{aligned}$$

The Lagrangian function for this equivalent problem is

$$\mathcal{L}(x, z, \lambda, \nu) = \frac{1}{2}\|z\|^2 - \langle \lambda, x \rangle + \langle \nu, Ax - b - z \rangle .$$

The Lagrange dual problem is defined to be the infimum over x, z of the Lagrangian function:

$$g(\lambda, \nu) \overset{\text{def}}{=} \inf_{x,z} \mathcal{L}(x, z, \lambda, \nu).$$

The Lagrangian is minimized over $x \in \mathbb{R}^m, z \in \mathbb{R}^m$ when

$$\nabla_{x,z}\mathcal{L} = \begin{bmatrix} -\lambda + A^T\nu \\ z - \nu \end{bmatrix} = 0.$$

Using $z - \nu = 0$, we find that the dual objective is

$$g(\lambda, \nu) = -\frac{1}{2} \left\| \nu \right\|^2 - \langle \nu, b \rangle .$$

We can eliminate λ by using $\lambda = A^T \nu$, but we must still enforce $\lambda \geq 0$. Thus the dual problem is

$$\begin{aligned} \max \quad & -\frac{1}{2} \left\| \nu \right\|^2 - \langle \nu, b \rangle \\ \text{s.t.} \quad & A^T \nu \geq 0. \end{aligned} \tag{3.18}$$

The dual objective $g(\nu) = -\frac{1}{2}\|\nu\|^2 - \langle \nu, b \rangle$ is 1-strongly concave, and so the dual problem has a unique optimal point (note that the constraint set $\{\nu : A^T \nu \geq 0\}$ is non-empty for any A, so the optimal point always exists).

Observe that Slater's condition holds for both the NNLS primal problem (3.1) and the dual problem (3.18). Slater's condition holding for the primal implies that the dual optimal value is attained, meaning that ν^* exists (which we knew from above). Slater's condition holding for the dual implies that the primal optimal value is attained, meaning that x^* exists.

Let x^* be any primal optimal point and ν^* be the unique dual optimal point. The KKT conditions for the NNLS primal problem (3.1) are

$$A^T \left(Ax^* - b \right) - A^T \nu^* = 0 \tag{3.19}$$

$$x^* \geq 0 \tag{3.20}$$

$$A^T \nu^* \geq 0 \tag{3.21}$$

$$x_i^* \left\{ A^T \nu^* \right\}_i = 0, \quad \forall i = 1, \ldots, n. \tag{3.22}$$

Since Slater's condition holds for both the NNLS primal and dual, the KKT conditions are both necessary and sufficient for optimality: if x and ν satisfy (3.19)-(3.22), then they are optimal with zero duality gap [17].

It is occasionally more convenient to use the KKT conditions written in terms of $\lambda^* = A^T\nu^*$:

$$\begin{aligned} A^T(Ax^* - b) - \lambda^* &= 0 \\ x^* &\geq 0 \\ \lambda^* &\geq 0 \\ x_i^*\lambda_i^* &= 0, \quad \forall i = 1, \ldots, n. \end{aligned}$$

3.3.2 Safe Feature Elimination With an Inexact Solution

In Appendix A we adapt SAFE for LASSO somewhat directly to NNLS problems. Like SAFE for LASSO, the feature elimination problem assumes knowledge of an exact NNLS solution. Having knowledge of an exact solution is reasonable for SAFE for LASSO, due to the way it takes advantage of LASSO's regularization path. But the assumption of having an exact solution is quite fragile for our highly ill-conditioned microscopy problems. Even when using an extremely high accuracy solution, the feature elimination strategy of Appendix A proved to be too weak to eliminate many features in our experiments. In this subsection we take a different approach in order to relax the assumption of having access to an exact NNLS solution. We will instead use a feasible point that is accurate, but not necessarily exactly optimal.

Let $f(x) = \frac{1}{2}\|Ax - b\|^2$ be the primal objective and $p^* = f(x^*)$ be the primal optimal value. We let a_i denote the ith column of A. As noted previously, strong duality holds for all primal-dual optimal pairs: $d^* \overset{\text{def}}{=} g(\nu^*) = -\frac{1}{2}\|\nu^*\|^2 - \langle \nu^*, b\rangle = p^*$. Assume we solve the primal NNLS problem (3.1) inexactly, giving the primal feasible point $\hat{x}$, which is not necessarily optimal. Let us further assume we have access to a dual feasible $\hat{\nu}$, also not necessarily optimal. This gives us the duality gap $\epsilon = f(\hat{x}) - g(\hat{\nu}) \geq 0$. Note that $\hat{\nu} = A\hat{x} - b$ is not necessarily dual feasible, though this will be the case if $\hat{x}$ is primal optimal. The important problem of constructing a suitable $\hat{\nu}$ from the inexact primal point $\hat{x}$ will be discussed in Subsection 3.3.3.

Just like in SAFE for LASSO, we will use complementary slackness (3.22) to determine if $x_i^* = 0$. Specifically, if $\{A^T\nu^*\}_i = \langle a_i, \nu^*\rangle > 0$, then complementary slackness implies $x_i^* = 0$. Of course, we do not have access to ν^*, only an accurate, dual feasible point $\hat{\nu}$. But if we can bound $\langle a_i, \nu^* \rangle$ to be strictly greater than 0, then we can still guarantee $x_i^* = 0$.

We will use $\hat{\nu}$ to construct a set N of dual points that is guaranteed to contain the dual optimal point ν^*. We then solve the feature elimination subproblem

$$\begin{array}{ll} \min_\nu & \langle a_i, \nu \rangle \\ \text{s.t.} & \nu \in N. \end{array} \tag{3.23}$$

If the optimal value of this subproblem is strictly positive, we have guaranteed that $\langle a_i, \nu^* \rangle = \lambda_i^* > 0$ and thus $x_i^* = 0$ by complementary slackness. If the lower bound is not strictly positive, then the search is inconclusive and we could have either $x_i^* = 0$ or $x_i^* > 0$. To find features to eliminate, we solve this subproblem for each column a_i of A.

Let us derive a simple set N for use in this section; in Section 3.4 we will describe additional constructions of N that lead to stronger subproblems. Define the dual feasible set $\mathcal{N} = \{\nu \,:\, A^T\nu \geq 0\}$. Note that $\mathcal{N}$ is convex and non-empty, as $\nu = 0$ is feasible for any A. Define

$$\widetilde{g}(\nu) \overset{\text{def}}{=} g(\nu) - \iota_{\mathcal{N}}(\nu) = -\frac{1}{2}\|\nu\|^2 - \langle \nu, b \rangle - \iota_{\mathcal{N}}(\nu),$$

where $\iota_{\mathcal{N}}(\nu)$ is the indicator function for the set $\mathcal{N}$:

$$\iota_{\mathcal{N}}(\nu) = \begin{cases} 0 & \nu \in \mathcal{N} \\ \infty & \text{otherwise.} \end{cases}$$

Since $-\frac{1}{2}\|\nu\|^2 - \langle \nu, b \rangle$ is 1-strongly concave and $-\iota_{\mathcal{N}}(\nu)$ is concave, $\widetilde{g}(\nu)$ is 1-strongly concave. Since ν^* and $\hat{\nu}$ are both dual feasible, $g(\nu^*) = \widetilde{g}(\nu^*)$ and $g(\hat{\nu}) = \widetilde{g}(\hat{\nu})$. By Theorem 5.25 of [10], we have $\widetilde{g}(\nu^*) - \widetilde{g}(\hat{\nu}) \geq \frac{1}{2}\|\hat{\nu} - \nu^*\|^2$. We can then conclude

$$g(\nu^*) - g(\hat{\nu}) \geq \frac{1}{2}\|\hat{\nu} - \nu^*\|^2. \tag{3.24}$$

Since strong duality holds, $g(\nu^*) = f(x^*) \leq f(\hat{x})$, so $g(\nu^*) - g(\hat{\nu}) \leq \epsilon$. We therefore define the search set

$$N \overset{\text{def}}{=} \left\{\nu \,:\, \|\nu - \hat{\nu}\|^2 \leq 2\epsilon\right\},$$

which comes from the inequality (3.24) and the duality gap. The set N is guaranteed to contain the optimal point ν^*, so the feature elimination subproblem (3.23) using this search set is safe.

Written out, this feature elimination subproblem is

$$\begin{aligned} \min_\nu \quad & \langle a_i, \nu \rangle \\ \text{s.t.} \quad & \|\nu - \hat{\nu}\|^2 \leq 2\epsilon. \end{aligned} \tag{3.25}$$

This problem minimizes a linear objective over a ball, which means it is convex. The closed-form solution is easily found (see Subsection 3.4.1), giving the optimal value

$$\langle a_i, \hat{\nu} \rangle - \sqrt{2\epsilon}\|a_i\|.$$

If this value is strictly positive, then we can certify that $x_i^* = 0$ and the feature a_i can be eliminated from the problem. Note that given $\hat{x}$ and $\hat{\nu}$, it is easy to compute ϵ by evaluating the primal and dual objectives. The simple form of the close-form solution allows us to quickly test for elimination of features.

3.3.3 Finding an Accurate, Dual Feasible $\hat{\nu}$ From an Accurate, Primal Feasible $\hat{x}$

In order to use lower-bound subproblem (3.25), we must have a primal feasible $\hat{x}$ and a dual feasible $\hat{\nu}$ that achieve a reasonably small duality gap $\epsilon = f(\hat{x}) - g(\hat{\nu})$. When using first-order methods, we typically only have access to a primal feasible variable $\hat{x}$. To use our feature elimination strategy, we need to find an accurate, dual feasible $\hat{\nu}$ from the accurate, primal feasible $\hat{x}$ given by a first-order solver.

To leverage the accuracy of $\hat{x}$, we form $\nu' = A\hat{x} - b$, since if $\hat{x}$ were optimal, then $A\hat{x} - b$ would be the dual optimal point. But note that ν' is not guaranteed to be dual feasible (i.e., $A^T\nu' \ngeq 0$ is possible), since $\hat{x}$ is not necessarily optimal. To fix this, perhaps the "best" approach is to solve the orthogonal projection problem

$$\begin{aligned} \min_{\hat{\nu}} \quad & \frac{1}{2}\|\hat{\nu} - \nu'\|^2 \\ \text{s.t.} \quad & A^T\hat{\nu} \geq 0, \end{aligned}$$

which finds $\hat{\nu}$ that is dual feasible point that is closest to ν'. This projection subproblem is essentially the same as solving the NNLS dual problem (3.18), and is therefore just as difficult to solve as the dual problem. Using a first-order method to solve the problem is unlikely to be fast, since projecting onto the dual feasible set is expensive. If we had fewer RHS, the use of an interior point method for this task would be more practical.

Instead, let us assume we have access to a strictly dual feasible point $\tilde{\nu}$. We will give a few simple methods to find such a strictly dual feasible $\tilde{\nu}$ in Subsection 3.3.3.1, and in Subsection 3.3.3.2 we will show that $\tilde{\nu}$ being strictly dual feasible (instead of just dual feasible) is necessary for our feature elimination problem to work as we desire. We can use $\tilde{\nu}$ to construct $\hat{\nu}$ nearby ν' that is also dual feasible using a simple line search. Specifically, we find the closest point to ν' along the line segment between ν' and $\tilde{\nu}$ via the dual line search

$$\begin{array}{ll} \min & t \\ \text{s.t.} & A^T((1-t)\nu' + t\tilde{\nu}) \geq 0 \\ & 0 \leq t \leq 1. \end{array}$$

Once we have solved the line search, we form $\hat{\nu} = (1-t^*)\nu' + t^*\tilde{\nu}$. We can view this line search as a not-necessarily-orthogonal projection onto the dual feasible set.

See Figure 3.9 for a diagram of this line search in two dimensions. The coordinate axes are the first and second elements of the dual variable ν. The boundary of the dual feasible set is given by two hyperplanes $\langle a_1, \nu\rangle = 0$ and $\langle a_2, \nu\rangle = 0$. We can see $\langle a_2, \nu'\rangle < 0$, so ν' is not dual feasible. The dashed circle is the level curve of the dual objective at the optimal value: $\{\nu : g(\nu) = g(\nu^*)\}$.

The constraint $t \geq 0$ is used in the line search only so that $\hat{\nu} = \nu'$ in the case when ν' is already dual feasible. Additionally, the optimal value is never greater than 1, since the point $\tilde{\nu}$ is assumed to be dual feasible. By precomputing $A^T\nu'$ and $A^T\tilde{\nu}$, the optimal value of the line search and the resulting dual feasible point $\hat{\nu}$ can be found in closed form, which is given in the next subsection.

When A is elementwise non-negative, i.e. $A_{ij} \geq 0$, as is the case for our microscopy problems, there is a simple way of generating a dual feasible $\tilde{\nu}$ from ν'. We project ν' into the non-negative orthant, forming

$$\tilde{\nu} = \max\{0, \nu'\}.$$

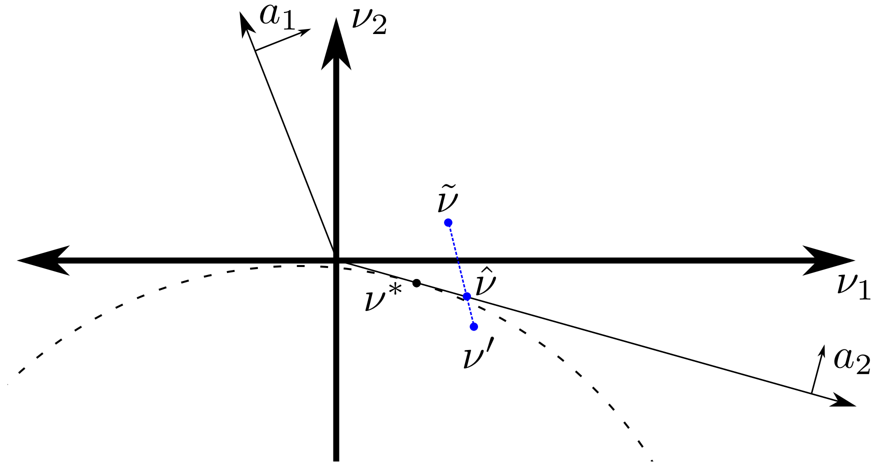


Figure 3.9: Finding $\hat{\nu}$ from ν' and $\tilde{\nu}$ via a line search.

Since $A_{ij} \geq 0$, we then have that $A^T\tilde{\nu} \geq 0$ so $\tilde{\nu}$ is guaranteed to be dual feasible. In our microscopy problems A is elementwise positive, and $\tilde{\nu} = \max\{0, \nu'\}$ is almost always strictly dual feasible. Since $\tilde{\nu}$ may be rather far from ν', we still use the line search to find a point closer to ν'. This approach typically achieves $\hat{\nu}$ with tighter duality gaps than when using the dual line search with a fixed strictly dual feasible point.

3.3.3.1 Finding a Strictly Dual Feasible $\tilde{\nu}$

For a given A, we can search for a strictly dual feasible point $\tilde{\nu}$ via the linear program

$$\begin{array}{ll} \max_{\tilde{\nu},t} & t \\ \text{s.t.} & A^T\tilde{\nu} \geq t \\ & \mathbb{1}^T A^T \tilde{\nu} = 1. \end{array} \tag{3.26}$$

This problem maximizes the entries of $A^T\tilde{\nu}$ while the constraint $\mathbb{1}^T A^T\tilde{\nu} = 1$ serves to keep $\tilde{\nu}$ bounded. Note that this linear program may fail to find a strictly dual feasible $\tilde{\nu}$. But it will do so only when A does not admit any strictly feasible points (e.g., $A = 0$ does not does not admit any strictly dual feasible points). The cost of this LP is not a concern, as it needs to be solved just once to find $\tilde{\nu}$; once we have a strictly dual feasible $\tilde{\nu}$, we can use it for any NNLS problem with the same A, regardless of the RHS b.

If the sum of each row of A is positive, the point $\tilde{\nu} = \mathbb{1}$ is strictly dual feasible. In particular, if A is elementwise positive (as is the case in our microscopy problems), $\tilde{\nu} = \mathbb{1}$ is strictly dual feasible. Furthermore, if $\tilde{\nu} = \max\{0, \nu'\}$ has at least one positive entry, it is strictly dual feasible for A elementwise positive. In our microscopy problems, A is elementwise positive and we find in our experiments that using $\tilde{\nu} = \max\{0, \nu'\}$ quite reliably produces strictly dual feasible points (see Subsections 3.4.2 and 3.4.3). We use $\tilde{\nu} = \mathbb{1}$ only if $\tilde{\nu} = \max\{0, \nu'\}$ is not strictly dual feasible.

3.3.3.2 When is the Dual Line Search Continuous?

In Subsection 3.3.4 we will show how $\hat{\nu}$ from the dual line search converges to the dual optimal point ν^* as $\hat{x}$ from a first-order solver converges to a primal optimal point. This will then be used to show that, under reasonable conditions, our dual line search and feature elimination strategy will eventually eliminate all zero features from the problem. This means that if we perform sufficiently many iterations of PGM/AT, we can eliminate all zero features from the problem. To enable that analysis, we find a closed-form solution to the line search and prove a lemma on the continuity of the mapping from $\nu' = A\hat{x} - b$ to $\hat{\nu}$ found via the line search.

We find the closed-form solution to the line search by identifying two cases:

(1) If ν' is dual feasible, $t = 0$ is the minimum feasible value, which leads to $\hat{\nu} = \nu'$.

(2) Otherwise, there is at least one index i such that $\langle a_i, \nu' \rangle = \{A^T\nu'\}_i < 0$. In this case, we must increase t until the all coordinates of $A^T((1-t)\nu' + t\tilde{\nu})$ are non-negative.

Let $\mathcal{I} \overset{\text{def}}{=} \left\{ i : a_i^T\nu' < 0 \right\}$ be the set of all indices causing ν' to be dual infeasible. The feasible values of t for which $a_i^T((1-t)\nu' + t\tilde{\nu}) \geq 0$, $i \in \mathcal{I}$, are

$$\max\left\{0, \frac{a_i^T\nu'}{a_i^T\nu' - a_i^T\tilde{\nu}}\right\} \leq t \leq 1.$$

Since all coordinates of $a_i^T((1-t)\nu' + t\tilde{\nu})$ must be non-negative, we have

$$t^* = \max \left\{ 0, \max_{i \in \mathcal{I}} \frac{a_i^T \nu'}{a_i^T \nu' - a_i^T \tilde{\nu}} \right\}.$$

We then form $\hat{\nu} = (1 - t^*)\nu' + t^* \tilde{\nu}$.

Note that there appears to be a pathological case if there is an i such that $a_i^T \nu' - a_i^T \tilde{\nu} = 0$. If this were to occur, then any value of t would produce a point where the ith coordinate is zero, i.e. $a_i^T((1-t)\nu + t\tilde{\nu}) = 0$. Geometrically, the condition $a_i^T \nu' - a_i^T \tilde{\nu} = 0$ states that the vector a_i is orthogonal to $\nu' - \tilde{\nu}$, indicating that the line segment between ν' and $\tilde{\nu}$ is contained in the hyperplane $\{\nu : a_i^T \nu = 0\}$. From this we see that $i \notin \mathcal{I}$ by construction, since $a_i^T \nu' = 0$, and such a pathological case is avoided.

Let us reformulate the expression for t^* slightly. Define $\lambda' = A^T \nu'$ and $\tilde{\lambda} = A^T \tilde{\nu}$. Then

$$t^* = \max_i \begin{cases} 0 & \lambda_i' \geq 0 \\ \dfrac{\lambda_i'}{\lambda_i' - \tilde{\lambda}_i} & \lambda_i' < 0. \end{cases}$$

If we define the function

$$t(\lambda_i'; \tilde{\lambda}_i) \overset{\text{def}}{=} \begin{cases} 0 & \lambda_i' \geq 0 \\ \dfrac{\lambda_i'}{\lambda_i' - \tilde{\lambda}_i} & \lambda_i' < 0, \end{cases}$$

we can write

$$t^* = \max_i \; t(\lambda_i'; \tilde{\lambda}_i).$$

We again have $\hat{\nu} = (1 - t^*)\nu' + t^* \tilde{\nu}$.

In Section 3.3.4, we will use the fact that this mapping from ν' to $\hat{\nu}$ is continuous in ν' when $\tilde{\nu}$ is **strictly** dual feasible.

Lemma 4: *If $\tilde{\nu}$ is strictly dual feasible (i.e., $A^T \tilde{\nu} > 0$), then the dual line search mapping ν' to $\hat{\nu}$ is continuous in ν'.*

Proof. In terms of $\tilde{\lambda}$, the strict dual feasibility assumption states that $\tilde{\lambda}_i > 0$ for each i. The dual line search produces the point

$$\hat{\nu} = (1 - t^*)\nu' + t^* \tilde{\nu}.$$

To show continuity of the mapping $\nu' \mapsto \hat{\nu}$, it is sufficient to show t^* is continuous in ν'. Since we have assumed $\tilde{\lambda}_i > 0$ for each i,

$$t(\lambda_i'; \tilde{\lambda}_i) = \begin{cases} 0 & \lambda_i' \geq 0 \\ \dfrac{\lambda_i'}{\lambda_i' - \tilde{\lambda}_i} & \lambda_i' < 0, \end{cases}$$

is continuous in λ_i' for each i (the only point of concern is $\lambda_i' = 0$; see Figure 3.10 for an illustration). Finally, $t^* = \max_i t(\lambda_i'; \tilde{\lambda}_i)$ is therefore continuous in $\lambda' = A^T\nu'$, and so the dual line search mapping ν' to $\hat{\nu}$ is continuous in ν'. $\square$

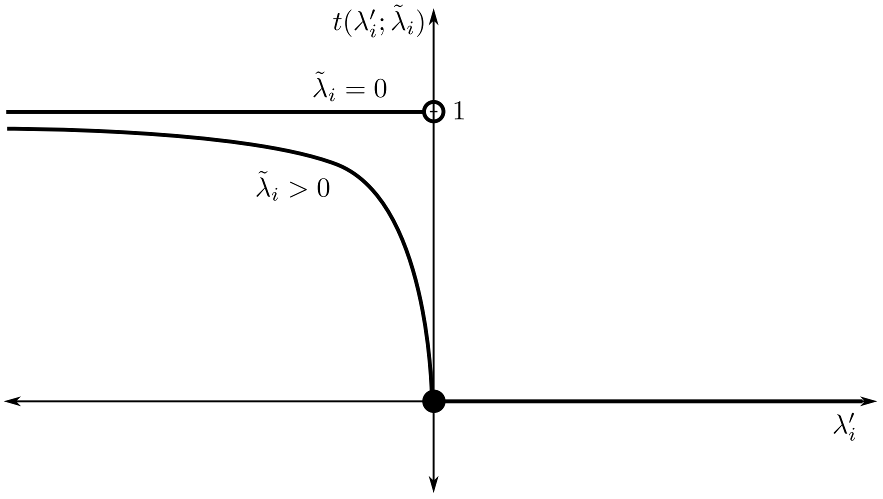


Figure 3.10: $t(\lambda_i'; \tilde{\lambda}_i)$ is not continuous at $\lambda_i' = 0$ when $\tilde{\lambda}_i = 0$.

The strict dual feasibility assumption in Lemma 4 is necessary for the continuity of the mapping. To see this, assume ν' and $\tilde{\nu}$ are such that $\lambda_i' = a_i^T\nu' < 0$ and $\tilde{\lambda}_i = a_i^T\tilde{\nu} = 0$ for some a_i. This means that ν' is not dual feasible and $\tilde{\nu}$ is dual feasible, but not strictly. Since ν' is dual infeasible, so we must use the line search to find a dual feasible point. The function $t(\lambda_i'; \tilde{\lambda}_i) = t(\lambda_i'; 0)$ is discontinuous at $\lambda_i' = 0$, which causes the mapping $\nu' \mapsto \hat{\nu}$ to be discontinuous. Requiring $\tilde{\nu}$ to be strictly dual feasible, so $\tilde{\lambda}_i = a_i^T\tilde{\nu} > 0$ for all i prevents this discontinuity.

Let us now look at the dual geometry when $\tilde{\nu}$ is not strictly dual feasible. The non-strictly dual feasible point $\tilde{\nu}$ is on the boundary of the dual feasible set, since it satisfies $a_i^T\tilde{\nu} = 0$. When ν' is not dual feasible, the only dual feasible point on the line segment between ν' and $\tilde{\nu}$ is $\tilde{\nu}$, so the dual line search returns $\hat{\nu} = \tilde{\nu}$. Thus, when $\tilde{\nu}$ is not strictly dual feasible, the dual line search will return one of two values: when ν' is not feasible, the line search returns $\tilde{\nu}$; when ν' is feasible, the line search returns ν'.

Figure 3.11 illustrates what would happen if ν' converges to ν^* while remaining dual infeasible. The line search is "stuck", returning $\hat{\nu} = \tilde{\nu}$ for all ν'. So as $\nu' \to \nu^*$, $\hat{\nu} = \tilde{\nu}$ is "stuck" and does not converge to ν^*. Picking $\tilde{\nu}$ to be strictly dual feasible "unsticks" the dual line search, allowing the returned value $\hat{\nu}$ to converge to ν^* as $\nu' \to \nu^*$.

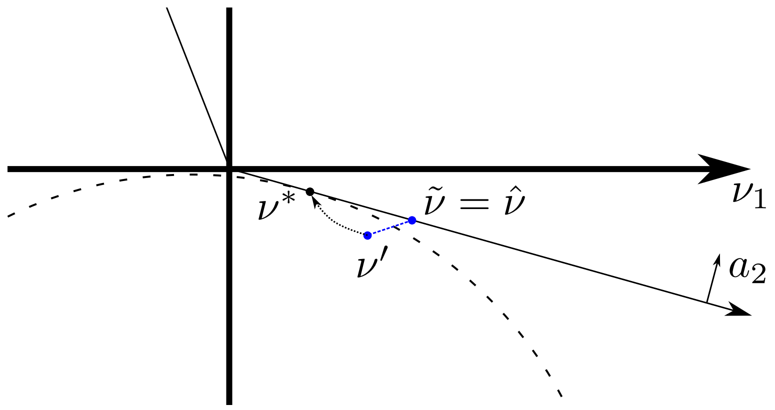


Figure 3.11: Pathological case for the continuity of the dual line search when $\tilde{\nu}$ is on the boundary of the dual feasible set (i.e., not strictly dual feasible).

3.3.4 Convergence of Dual Sequence Given Primal Sequence

Using a first-order method to solve NNLS typically provides a sequence x^k of primal feasible points that converge to a primal optimal point x^* (which may not be unique). For example, Theorem 10.24 of [10] shows that the proximal gradient method produces such a sequence. In Subsection 3.3.3, we discussed a dual line search that allows us to produce a dual feasible point $\hat{\nu}^k$ from $\nu'^k \overset{\text{def}}{=} Ax^k - b$ and a separate dual feasible point $\tilde{\nu}$.

In Subsection 3.3.2 we saw a simple feature elimination subproblem (3.25) that utilized an accurate, but inexact primal-dual pair. The strength of the subproblem depends on the size of the duality gap ϵ. In other words, as the duality gap shrinks, the lower bound produced by the subproblem increases, possibly eliminating the feature. In order for the duality gap ϵ to shrink to zero as x^k converges, we must also have that the dual line search produces $\hat{\nu}^k$ that converges to the dual optimal point ν^*. If $\tilde{\nu}$ is strictly dual feasible, we will see that $\hat{\nu}^k \to \nu^*$ as $x^k \to x^*$, thus giving $\epsilon \to 0$ as desired.

First, we give a lemma that states $\nu^* = Ax^* - b$ for any primal optimal x^*. This comes from the KKT conditions for the dual problem.

Lemma 5: *Let x^* be a primal optimal point. Then $\nu = Ax^* - b$ is the unique dual optimal point.*

Proof. We first write the dual problem (3.18) as a minimization problem:

$$\begin{aligned} \min \quad & \frac{1}{2}\|\nu\|^2 + \langle \nu, b\rangle \\ \text{s.t.} \quad & A^T\nu \geq 0. \end{aligned}$$

The KKT conditions for this problem are

$$\begin{aligned} \nu^* + b - A\xi^* &= 0 \\ \xi^* &\geq 0 \\ A^T\nu^* &\geq 0 \\ \{\xi^*\}_i\{A^T\nu^*\}_i &= 0 \quad \forall i = 1, \dots, n, \end{aligned}$$

where ξ^* are the variables associated with the constraint $A^T\nu \geq 0$.

By comparison with the primal KKT conditions (3.19)-(3.22), we can see that if we define $\nu = Ax^* - b$, then $\nu^* = \nu$ and $\xi^* = x^*$ will satisfy the KKT conditions. Since the constraint $A^T\nu \geq 0$ is linear, strong duality holds and satisfying the KKT conditions implies optimality. Therefore $\nu = Ax^* - b$ is dual optimal. Since the dual objective $\frac{1}{2}\|\nu\|^2 + \langle \nu, b\rangle$ is strongly convex, the dual optimal point is unique. Therefore we have that $\nu = Ax^* - b$ is the unique dual optimal point. □

Now we show that if $\tilde{\nu}$ is strictly dual feasible, then $\hat{\nu}^k \to \nu^*$ as the sequence x^k of primal feasible points converge to a primal optimal point x^*.

Theorem 6: *Let x^k be a sequence of primal feasible points that converge to a primal optimal point x^*, and let $\tilde{\nu}$ be a strictly dual feasible point (i.e., $A^T\tilde{\nu} > 0$). For each k, define the dual feasible* *point $\hat{\nu}^k$ by performing the dual line search of Subsection 3.3.3.2 using $\nu'^k = Ax^k - b$ and $\tilde{\nu}$. Then the sequence $\hat{\nu}^k$ of dual feasible points converges to the unique dual optimal point ν^*.*

Proof. Note that the affine map $x^k \mapsto \nu'^k = Ax^k - b$ is continuous. By continuity and Lemma 5, we have that $\nu'^k \to \nu^*$ as $x^k \to x^*$. We seek to show that the dual feasible sequence $\hat{\nu}^k \to \nu^*$ as $x^k \to x^*$. Note that

$$\|\hat{\nu}^k - \nu^*\| \leq \|\hat{\nu}^k - \nu'^k\| + \|\nu'^k - \nu^*\|$$

by the triangle inequality. We already have that $\|\nu'^k - \nu^*\| \to 0$, so to complete the proof it remains to show $\|\hat{\nu}^k - \nu'^k\| \to 0$.

Since $\hat{\nu}^k$ is computed using the dual line search using ν'^k and $\tilde{\nu}$, we have that

$$\|\hat{\nu}^k - \nu'^k\| = \|(1-t^k)\nu'^k + t^k\tilde{\nu} - \nu'^k\| = t^k\|\nu'^k - \tilde{\nu}\|,$$

where t^k is the optimal value of t from the dual line search for that particular ν'^k. Since $\nu'^k \to \nu^*$, we know that $\|\nu'^k - \tilde{\nu}\|$ is eventually bounded above by a constant. Therefore, we just need to show $t^k \to 0$.

Since $\nu'^k \to \nu^*$, we see that ν'^k converges to a dual feasible point. The proof of Lemma 4 shows that t^k is continuous in ν'^k. As ν'^k approaches a dual feasible point, t^k approaches zero continuously, since the line search returns a point closer to ν'^k as ν'^k becomes closer to being dual feasible. Therefore we have that $t^k \to 0$ as ν'^k converges, which shows that $\|\hat{\nu}^k - \nu'^k\| \to 0$, completing the proof. $\square$

3.3.5 When Will the Screening Rule Eliminate all Zero Features?

Here we show that the feature elimination subproblem (3.25) will, under certain assumptions and with sufficiently small duality gap ϵ, eliminate all zero features from the solution. Coupled with the use of a first-order method and the dual line search of Subsection 3.3.3, this means that our feature elimination strategy will eventually eliminate all features that can be eliminated.

Theorem 7: *Assume there exists a unique NNLS solution x^* and that strict complementary slackness holds (i.e., $x_i^* = 0$ iff $\langle a_i, \nu^* \rangle > 0$). Define $\mathcal{I} \overset{\text{def}}{=} \{i : \langle a_i, \nu^* \rangle > 0\}$ to be the indexes of zero features. Let the pair $\hat{x}, \hat{\nu}$ produce a duality gap estimate ϵ such that*

$$\sqrt{\epsilon} < \frac{1}{2\sqrt{2}} \min_{i \in \mathcal{I}} \frac{\langle a_i, \nu^* \rangle}{\|a_i\|}.$$

Then the feature elimination problem (3.25) *will eliminate all zero features.*

Proof. Since we have assumed strict complementary slackness, a zero feature $x_i^* = 0$ correspond to the dual variable $\langle a_i, \nu^* \rangle > 0$. If we assumed non-strict complementary slackness, then a zero feature $x_i^* = 0$ may be associated with $\langle a_i, \nu^* \rangle = 0$, which could not be eliminated by the subproblem (3.25). Therefore strict complementary slackness implies that all zero features are associated with a strictly positive dual variable, and we must show that the subproblem produces a strictly positive lower bound for $\langle a_i, \nu^* \rangle$ for each $i \in \mathcal{I}$.

For the subproblem to produce a strictly positive bound for the ith feature, the search set N must be contained strictly in the interior of the halfspace $\langle a_i, \nu \rangle \geq 0$. The search set N is a (closed) ball of radius $\sqrt{2\epsilon}$ centered at $\hat{\nu}$. From the 1-strong concavity of the dual objective $g(\nu)$, we know that $\|\hat{\nu} - \nu^*\| \leq \sqrt{2\epsilon}$. Therefore the closest point in N to the hyperplane $\langle a_i, \nu \rangle = 0$ (i.e., the optimal point of the subproblem) satisfies

$$\|\hat{\nu} - \nu^*\| + \sqrt{2\epsilon} \leq 2\sqrt{2\epsilon}.$$

Figure 3.12 illustrates this geometry.

We require that this distance is strictly less than the distance from ν^* to the hyperplane, so that N is strictly separated from the hyperplane. By strict complementary slackness, we know that ν^* is strictly separated from the hyperplane, with signed distance $\langle a_i, \nu^* \rangle / \|a_i\|$. Therefore we must have

$$2\sqrt{2\epsilon} < \frac{\langle a_i, \nu^* \rangle}{\|a_i\|}.$$

By assumption, ϵ is small enough to satisfy this condition for each $i \in \mathcal{I}$, so we have that (3.25) will eliminate all zero features. $\square$

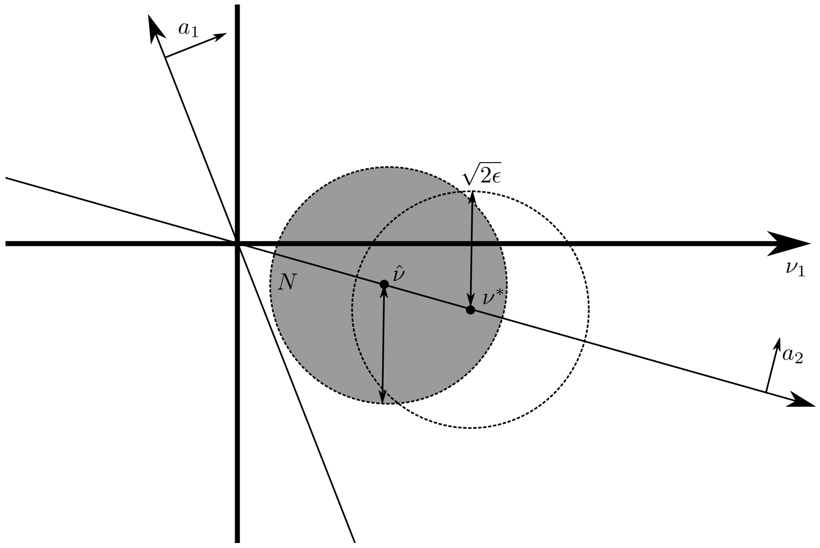


Figure 3.12: The duality gap ϵ must be less than half the distance from ν^* to the nearest hyperplane corresponding to a zero feature. The figure is drawn such that $x_1^* = 0$ and $x_2^* > 0$.

Note that Theorem 7 requires knowledge of the true dual optimal point ν^*, though its use in the index set $\mathcal{I}$ and the bound on ϵ. The following theorem modifies the proof of Theorem 7 to give a bound on ϵ that involves $\hat{\nu}$ instead of ν^*. Note that the bound still uses knowledge of ν^*, but now only through the index set $\mathcal{I}$.

Theorem 8: *Assume there exists a unique NNLS solution x^* and that strict complementary slackness holds (i.e., $x_i^* = 0$ iff $\langle a_i, \nu^* \rangle > 0$). Define $\mathcal{I} \overset{\text{def}}{=} \{i : \langle a_i, \nu^* \rangle > 0\}$ to be the indexes of zero features. Let the pair $\hat{x}, \hat{\nu}$ produce a duality gap estimate ϵ such that*

$$\sqrt{\epsilon} < \frac{1}{\sqrt{2}} \min_{i \in \mathcal{I}} \frac{\langle a_i, \hat{\nu} \rangle}{\|a_i\|}.$$

Then the feature elimination problem (3.25) *will eliminate all zero features.*

Proof. The proof is a modification of the proof of Theorem 7, so we discuss only the required modification. The proof of Theorem 7 shows that ϵ sufficiently small ensures that the search set N is strictly separated from each hyperplane $\langle a_i, \nu \rangle = 0$ for each $i \in \mathcal{I}$ by using the fact that ν^* is strictly separated from those hyperplanes. We can instead show that ϵ sufficiently small ensures that the optimal value of the feature elimination subproblem (3.25) is strictly positive for each $i \in \mathcal{I}$.

Recall from Subsection 3.3.2 that the optimal value is $\langle a_i, \hat{\nu} \rangle - \sqrt{2\epsilon}\|a_i\|$ for the ith feature. Up to a scaling factor, the subproblem finds the minimum distance between the search set N and the hyperplane $\langle a_i, \nu \rangle = 0$. Therefore the search set N is strictly separated from the hyperplane precisely when the optimal value is strictly positive:

$$\langle a_i, \hat{\nu} \rangle - \sqrt{2\epsilon}\|a_i\| > 0 \iff \sqrt{\epsilon} < \frac{1}{\sqrt{2}} \frac{\langle a_i, \hat{\nu} \rangle}{\|a_i\|}.$$

By assumption on the size of ϵ, this condition is satisfied for each $i \in \mathcal{I}$, so feature elimination will again eliminate all zero features. $\square$

Theorem 7 and 8 show that, under certain conditions and if the duality gap ϵ is sufficiently small, then feature elimination will eliminate all zero features from the problem. We used knowledge of the the exact dual optimal point to quantify how small ϵ must be in order to imply that feature elimination will work. But even without knowledge of the dual optimal point, we still know that if ϵ is sufficiently small, then feature elimination will have worked. Indeed, by combining Theorems 6 and 7, we have the following:

Corollary 9: *Assume there exists a unique NNLS solution x^* and that strict complementary slackness holds (i.e., $x_i^* = 0$ iff $\langle a_i, \nu^* \rangle > 0$). Let x^k be a sequence of primal feasible points that converges to x^* (e.g., from PGM/AT) and let $\hat{\nu}^k$ be the sequence of dual feasible points produced as in Theorem 6. Then the duality gap $\epsilon = f(x^k) - g(\hat{\nu}^k) \to 0$ as $k \to \infty$ and Theorem 7 will eventually apply. This means that the feature elimination subproblem* (3.25) *will eventually eliminate all zero features.*

This tells us that if we do enough iterations of PGM/AT, perform the dual line search, and then do feature elimination, we will eliminate all possible features. But we don't know how many iterations are sufficient (without knowledge of the dual optimal point, that is). Nevertheless, we can still use feature elimination very effectively in practice, including certifying that underdetermined NNLS problems have unique solutions.

3.4 Safe Feature Elimination for NNLS - Use in Practice

In Section 3.3 we analyzed safe feature elimination for NNLS. The feature elimination subproblem (3.25), though simple, is strong enough to provably eliminate all zero features (under certain conditions). In experiments, we use a first-order method to find an accurate $\hat{x}$ and use the dual line search to find an accurate $\hat{\nu}$. Though the subproblem (3.25) will eventually work, a stronger subproblem may allow us to expend less work increasing the accuracy of $\hat{x}$. We begin this section by deriving a few stronger, but still safe, feature elimination subproblems.

In the beginning of Section 3.3, we motivated feature elimination by discussing its use in certifying the uniqueness of NNLS solutions. We use our strengthened feature elimination subproblems to successfully certify uniqueness of solutions to an example microscopy problem in Subsection 3.4.3.

We also discuss an alternative approach to certifying uniqueness of NNLS solutions. The alternative approach is simpler, in a sense, though it relies on strong assumptions on A that can be infeasible to check in practice. Furthermore the alternative approach only certifies uniqueness, unlike feature elimination which guarantees features to be zero, which is useful in its own right.

3.4.1 Stronger Feature Elimination Subproblems

In this subsection we strengthen the basic feature elimination subproblem (3.25). While (3.25) does have theoretical guarantees (see Theorem 7 and 8), a stronger subproblem is desirable in practice, as it the potential to eliminate features at a larger value of the duality gap ϵ, for example. Subproblem strength cannot come at the cost of safety, however, so we still require that the subproblems do not eliminate non-zero features.

3.4.1.1 Strong Concavity

We begin with a discussion of (3.25), the basic subproblem used for Section 3.3. This subproblem is based on the 1-strong concavity of the dual objective:

$$g(\nu^*) - g(\hat{\nu}) \geq \frac{1}{2}\|\hat{\nu} - \nu^*\|^2.$$

From the pair $\hat{x}, \hat{\nu}$ we have the duality gap $\epsilon = f(\hat{x}) - g(\hat{\nu})$ and the bound $g(\nu^*) - g(\hat{\nu}) \leq \epsilon$. This gives rise to the subproblem

$$\begin{aligned} \min_\nu \quad & \langle a_i, \nu \rangle \\ \text{s.t.} \quad & \|\nu - \hat{\nu}\|^2 \leq 2\epsilon, \end{aligned} \tag{3.27}$$

which is of course identical to (3.25). The search set $N = \{\nu : \|\nu - \hat{\nu}\|^2 \leq 2\epsilon\}$ is a ball of radius $\sqrt{2\epsilon}$ centered at $\hat{\nu}$. Since $\nu^* \in N$ (by construction), this subproblem is guaranteed to be safe.

The unique (if $a_i \neq 0$) solution to this subproblem is

$$\hat{\nu} - \sqrt{2\epsilon}\frac{a_i}{\|a_i\|},$$

and the optimal value is

$$\langle a_i, \hat{\nu} \rangle - \sqrt{2\epsilon}\|a_i\|.$$

3.4.1.2 Strong Concavity and Dual Objective Constraint

Define N_1 to be the strong concavity search set from the previous subsection:

$$N_1 = \{\nu : \|\nu - \hat{\nu}\|^2 \leq 2\epsilon\}.$$

We will form the search set $N = N_1 \cap N_2$ to shrink the search set, resulting in a stronger subproblem. Note that N must contain the dual optimal point ν^* in order to remain safe.

Let $\hat{d} \overset{\text{def}}{=} g(\hat{\nu}) = -\frac{1}{2}\|\hat{\nu}\|^2 - \langle \hat{\nu}, b \rangle$ be the dual objective at the point $\hat{\nu}$. Since $\hat{\nu}$ is not necessarily dual optimal, we have that $g(\nu^*) \geq g(\hat{\nu}) = \hat{d}$. After simplification, this becomes

$$\|\nu^* + b\|^2 \leq \|b\|^2 - 2\hat{d}.$$

We therefore define the set

$$N_2 = \left\{\nu : \|\nu + b\|^2 \leq \|b\|^2 - 2\hat{d}\right\}.$$

By construction, we have $\nu^* \in N_2$, so using $N = N_1 \cap N_2$ is safe.

We now have the feature elimination subproblem

$$\begin{aligned} \min_\nu \quad & \langle a, \nu \rangle \\ \text{s.t.} \quad & \|\nu - \hat{\nu}\|^2 \leq 2\epsilon \\ & \|\nu + b\|^2 \leq \|b\|^2 - 2\hat{d}. \end{aligned} \tag{3.28}$$

Figure 3.13 shows the dual geometry for the subproblem (3.28) when the duality gap estimate ϵ is too large to eliminate a_1. The black, dashed circle is the level curve of the dual objective at the optimal value: $\{\nu : g(\nu) = g(\nu^*) = d^*\}$. The red, dotted circle is the boundary of the region of the first constraint (strong concavity); the blue, dotted circle is the boundary of the region of the second constraint (dual objective). The shaded region is the search set $N = N_1 \cap N_2$ for the subproblem; note that it extends past the halfspace given by $\langle a_1, \nu \rangle \geq 0$ so the subproblem fails to eliminate the feature a_1. But $\langle a_1, \nu^* \rangle > 0$, so for ϵ sufficiently small, the subproblem will eliminate a_1. Note also that $\hat{\nu}$ is contained in the hyperplane given by $\langle a_2, \nu \rangle = 0$, so a_2 is also not eliminated. This is to be expected (under strict complementarity), since $\langle a_2, \nu^* \rangle = 0$ and so a_2 will **never** be eliminated from the problem, as $x_2^* > 0$.

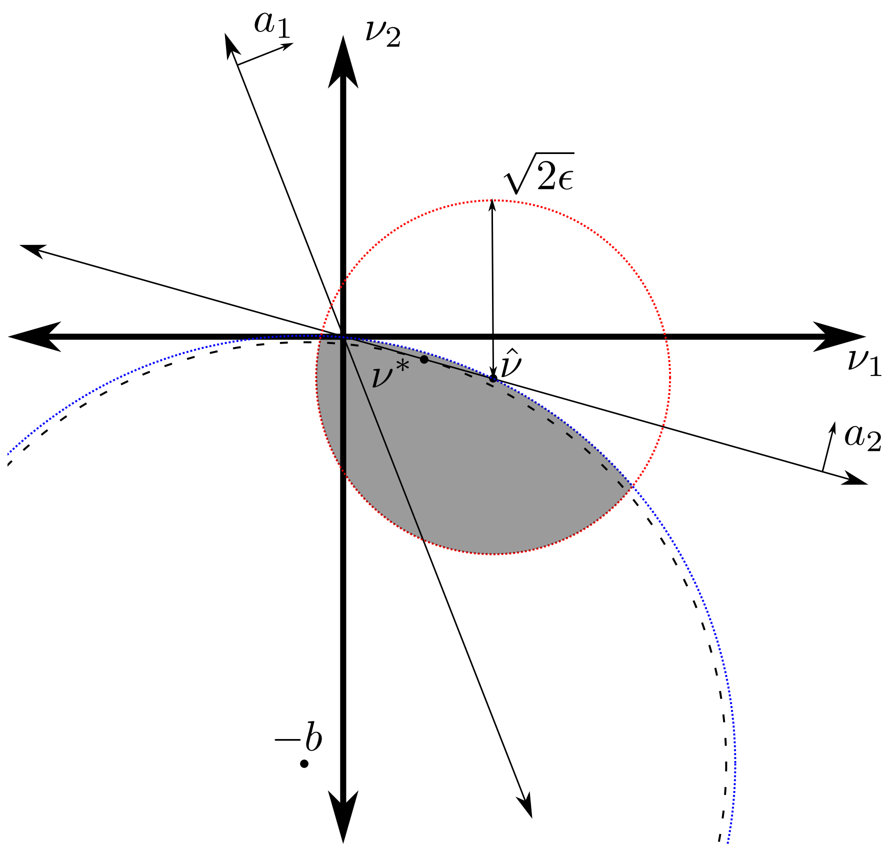


Figure 3.13: Feature elimination with subproblem (3.28) doesn't work with a too large duality gap bound ϵ.

Figure 3.14 shows the dual geometry when the duality gap estimate ϵ is sufficiently small to eliminate the feature a_1. Note that the search set is contained in the interior of the halfspace given by $\langle a_1, \nu \rangle \geq 0$, so the subproblem eliminates a_1. Again a_2 is not, and never will be, eliminated.

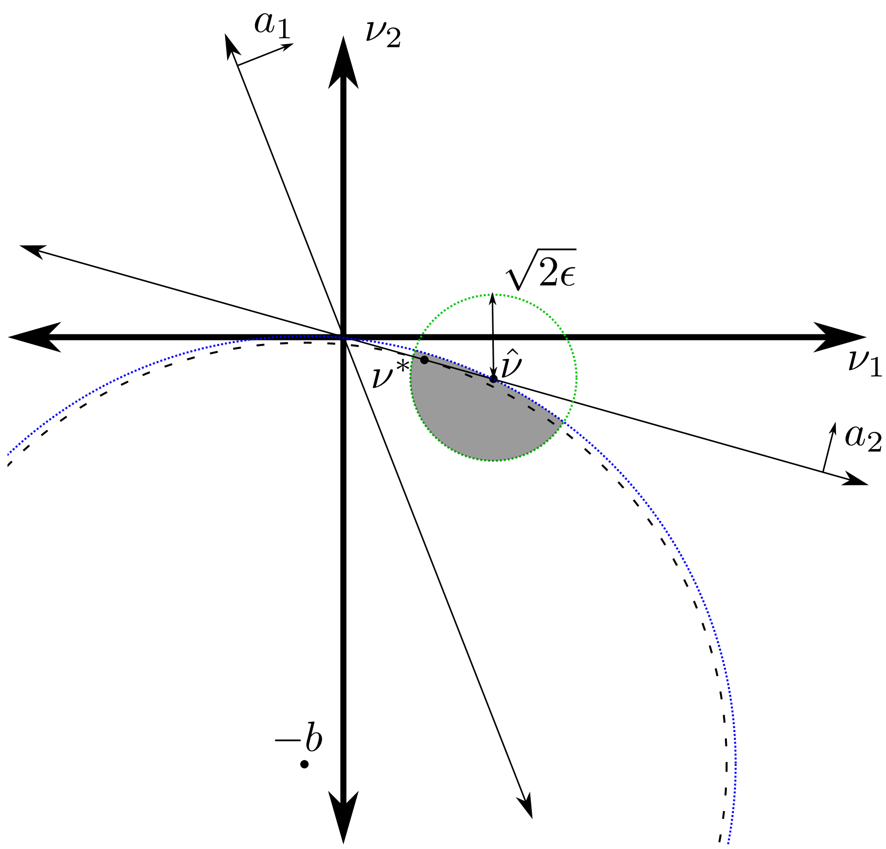


Figure 3.14: Feature elimination with subproblem (3.28) eliminates the feature a_1 when the duality gap bound ϵ is sufficiently small.

3.4.1.3 Strong Concavity and Weak Duality

Note that Figure 3.14 is drawn with $A \geq 0$ and $b \geq 0$, which is the case for our microscopy problems. The dual objective constraint cuts off the "top right" (towards the non-negative orthant) of the strong duality constraint ball. However, we really want to cut off the "bottom left" (towards the non-positive orthant) of the ball, as that side is closest to the boundary of the hyperplane defined by a_1 when $A \geq 0$. By adding a weak duality constraint, we can cut off the "bottom left" part of the strong concavity ball.

Define $\hat{p} \overset{\text{def}}{=} f(\hat{x})$ and note that weak duality states that $\hat{p} \geq g(\hat{\nu})$. Writing out this inequality yields $\|\hat{\nu}+b\|^2 \geq \|b\|^2 - 2\hat{p}$. Since ν^* must also satisfy this inequality, we can use this as a constraint on ν in a feature elimination subproblem:

$$\|\nu + b\|^2 \geq \|b\|^2 - 2\hat{p}.$$

Note that this says that ν must be **outside** the interior of the ball centered at $-b$ with radius $\|b\|^2 - 2\hat{p}$. The dual objective constraint stated that ν must be **inside** a ball of slightly larger radius.

This leads us to the subproblem based on both the strong concavity constraint and this weak duality constraint:

$$\begin{aligned} \min_\nu \quad & \langle a, \nu \rangle \\ \text{s.t.} \quad & \|\nu - \hat{\nu}\|^2 \leq 2\epsilon \\ & \|\nu + b\|^2 \geq \|b\|^2 - 2\hat{p}. \end{aligned} \tag{3.29}$$

The feasible set of this subproblem is the intersection of a ball and the (closure of) the outside of a ball. Thus, the problem is non-convex in general.

Figure 3.15 shows the dual geometry for the subproblem (3.29). We can see that the weak duality constraint does indeed clip off the "bottom left" of the strong concavity constraint ball.

A relaxation of the subproblem (3.29) relaxes the feasible set to its convex hull, thus resulting in a convex problem. The convex hull is equivalently formed as the intersection of the strong duality ball and the halfspace that contains the intersection of the two spheres. The boundary of the halfspace (a hyperplane) is given by

$$\|\nu - \hat{\nu}\|^2 - 2\epsilon = \|\nu + b\|^2 - \|b\|^2 + 2\hat{p},$$

which, after simplification, is equivalent to

$$\langle \nu, \hat{\nu} + b \rangle = \frac{1}{2}\|\hat{\nu}\|^2 - \epsilon - \hat{p}.$$

This leads to the "dome subproblem"

$$\begin{aligned} \min_\nu \quad & \langle a, \nu \rangle \\ \text{s.t.} \quad & \|\nu - \hat{\nu}\|^2 \leq 2\epsilon \\ & \langle \nu, \hat{\nu} + b \rangle \geq \frac{1}{2}\|\hat{\nu}\|^2 - \epsilon - \hat{p}, \end{aligned} \tag{3.30}$$

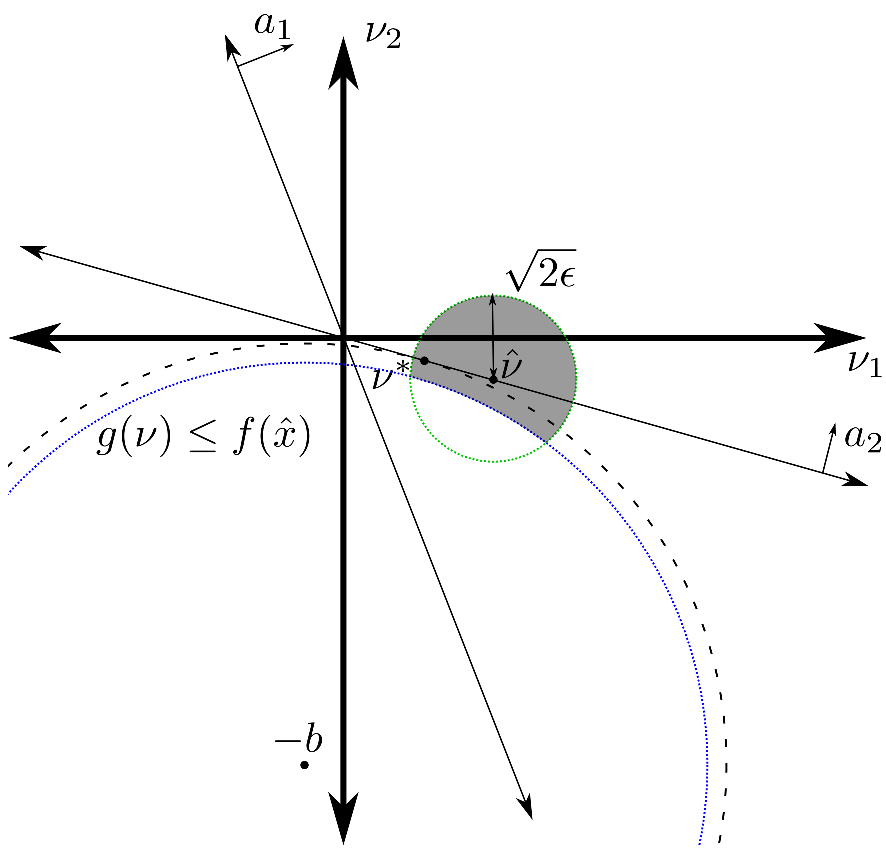


Figure 3.15: Feature elimination with subproblem (3.29) eliminates the feature a_1 when the duality gap bound ϵ is sufficiently small.

where the search set is the intersection of a ball and a halfspace (i.e., a dome).

In our microscopy problems, b represents photon counts, and so $\|b\|$ is usually quite large relative to ϵ. Thus the curvature of the weak duality sphere is small and the halfspace approximation is good. Moreover, "dome subproblems" are convex and have a closed-form solution, which we derive in Appendix B.

3.4.1.4 Strong Concavity and Partial Dual Feasibility

Another way to remove parts of the strong duality constraint ball, thereby strengthening the subproblem, is to enforce dual feasibility for a select few columns of A. That is, we add the constraints $\langle a_j, \nu \rangle \geq 0$ for $j \in \mathcal{J}$, where $\mathcal{J}$ is a set of indexes for the columns of A. One possible way to construct $\mathcal{J}$ is $\mathcal{J} = \{j : \langle a_j, \hat{\nu} \rangle \leq \text{tolerance}\}$, the intuition being that these constraints are active (or nearly active) near $\hat{\nu}$, where the strong duality constraint ball is centered.

This results in the subproblem

$$\begin{aligned} \min_\nu \quad & \langle a, \nu \rangle \\ \text{s.t.} \quad & \|\nu - \hat{\nu}\|^2 \leq 2\epsilon \\ & \langle a_j, \nu \rangle \geq 0 \quad \forall j \in \mathcal{J}. \end{aligned} \tag{3.31}$$

This subproblem is convex, since the objective is linear and the feasible set is the intersection of a ball and multiple halfspaces. This can be considered a generalization of a "dome subproblem", like (3.30). However, finding a closed-form solution seems difficult.

To solve (3.31) efficiently, IPMs are a natural choice. Specifically, CVXGEN [67] produces an optimized IPM for a specific, fixed problem structure (which must be transformable to a QP). The IPM that it generates can be re-used to rapidly solve the problem for different problem data. Still, we must solve the problem a very large number of times, which could become impractical.

Since a simple closed-form solution is preferred from a practical standpoint, we choose to relax (3.31). What is preventing us from finding a closed-form solution is enforcing $\langle a_j, \nu \rangle \geq 0$ jointly for all $j \in \mathcal{J}$. Let us relax the problem by splitting those $|\mathcal{J}|$ halfspace constraints into $|\mathcal{J}|$ problems with a single halfspace constraint and then take the maximum optimal value of the split problems:

$$\begin{aligned} \max_{j \in \mathcal{J}} \quad \min_\nu \quad & \langle a_i, \nu \rangle \\ \text{s.t.} \quad & \|\nu - \hat{\nu}\|^2 \leq 2\epsilon \\ & \langle a_j, \nu \rangle \geq 0. \end{aligned} \tag{3.32}$$

We recognize the inner problem as a dome subproblem, for which we derive a closed-form solution in Appendix B. To solve (3.32), we call the dome subproblem solution $|\mathcal{J}|$ times, recording the optimal value for each subproblem. We then take the best (i.e., maximum) optimal value to use as the lower bound for $\langle a_i, \nu^* \rangle$.

Another choice of the index set $\mathcal{J}$ is to simply take $\mathcal{J} = \{1, \ldots, n\}$, so that we enforce dual feasibility, but only one feature at a time. Doing so for each feature a_i of A will require the use of the dome subproblem solution n^2 times. This can be made much more efficient by precomputing all the required dot products and norms used by the closed-form solution, Algorithm 11, in Appendix B. We discuss this further in Subsection 3.5.3.

3.4.2 An Additional Dual Line Search

In Subsection 3.3.3 we described two dual line search methods that allow us to find an accurate, dual feasible $\hat{\nu}$ from an accurate, primal feasible $\hat{x}$. The first dual line search relied on a fixed, strictly dual feasible point $\tilde{\nu}$. While we have guarantees that this line search is sufficient (i.e., will enable feature elimination to work eventually), we desire a strengthened dual line search to use in practice.

One approach for elementwise non-negative A was also proposed in Subsection 3.3.3. It uses $\tilde{\nu} = \max\{0, \nu'\}$, where $\nu' = A\hat{x} - b$, which is not necessarily strictly dual feasible, and so is not guaranteed to be safe. However, we can compute this point and check if it is strictly dual feasible, a computation that merely amounts to checking if $A^T\tilde{\nu} > 0$. If $\tilde{\nu}$ is strictly dual feasible, we use $\tilde{\nu}$ in a line search; if not, then we revert to a precomputed, strictly dual feasible point to use in the line search (see Subsection 3.3.3.1). In experiments, we find that using $\tilde{\nu} = \max\{0, \nu'\}$ almost always results in a strictly dual feasible point, and when it does, it results in $\hat{\nu}$ with tighter duality gap estimates than using a fixed $\tilde{\nu}$.

Here we describe an additional approach that can work for a general A. We note that the gradient $\nabla g(\nu')$ may point toward the dual feasible set. We can then find a dual feasible point by searching along the line starting from ν' in the direction of $\nabla g(\nu')$. Figure 3.16 shows a diagram of this line search. This line search tries to find a scalar $t \geq 0$ such that the point

$$\nu' + t\nabla g(\nu')$$

is dual feasible, meaning

$$A^T(\nu' + t\nabla g(\nu')) \geq 0.$$

Noting that $\nabla g(\nu') = \nu' + b$, we find that $A^T\nabla g(\nu') = A^T(\nu' + b) > 0$ ensures that we can find such a $t \geq 0$. But note that this is not a necessary condition: we may still find a suitable t if

$A^T \nabla g(\nu') \not> 0$, but such a t is not guaranteed to exist.

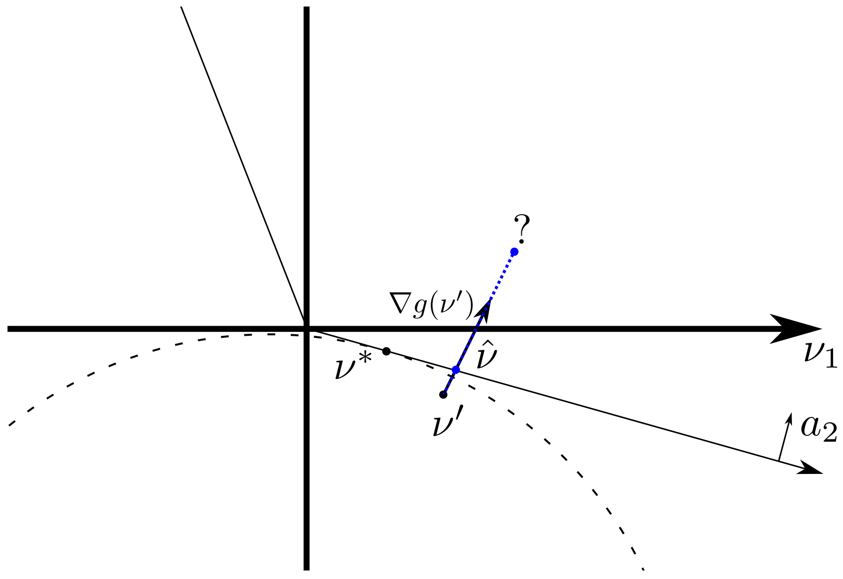


Figure 3.16: Dual line search to find $\hat{\nu}$ based on $\nabla g(\nu')$.

As with the line search based on $\tilde{\nu} = \max\{0, \nu'\}$ for elementwise non-negative A, if this line search fails, we merely revert to the line search based on a fixed, strictly dual feasible $\tilde{\nu}$. If we have multiple successful line searches, we will have multiple candidate points $\hat{\nu}$ to use. To select just one, we pick the point that produces the smallest duality gap ϵ, thereby strengthening the feature elimination subproblems.

3.4.3 Example - Certifying Uniqueness of a Super-Resolved Image

Here we consider certifying the uniqueness of the recovered PSF intensities for an instance of the focused-spot NNLS problem discussed in Section 3.1. If A is overdetermined and full-rank, the NNLS objective is strongly convex, and so has a unique optimal point by the standard result Theorem 1. We therefore only consider the case of underdetermined A.

In this example, we consider the data used to reconstruct the image in Figure 3.2b. Note that Figure 3.2b uses a PSF grid spacing that is $3\times$ finer than the effective camera pixel size. From Table 3.1, that PSF dictionary is overdetermined and full-rank, so the recovered image is unique.

We consider here the $5\times$ finer PSF grid, where the matrix A is of size 1681×2822, which is underdetermined. The matrix A is strictly positive: $A > 0$. There are 40000 right-hand sides b, with entries corresponding to subimage pixel values measured by the camera. While the matrix A is full-rank, the condition number $\kappa_2(A) = 2.4 \times 10^{20}$ (computed using `dgesvj` compiled to use quadruple precision [5]). We rescale the problem data so that $\|A\|_2 = 1$; the scaling is reversed after running the optimization method.

We solve the primal NNLS problems with our GPU implementation of Algorithm 8, the accelerated projected gradient method discussed in Subsection 3.2.1. Each RHS uses its own backtracking line search loop and local Lipschitz constant estimation. Computations are carried out in double precision, with a majority of the work (e.g., large matrix-matrix multiplies) performed on an NVIDIA K40c GPU. For this experiment, we run a fixed number of iterations instead of using more sophisticated stopping criteria.

For a given number of iterations, once we have all 40000 primal feasible points $\hat{x}$, we then must find a strictly dual-feasible point $\hat{\nu}$ for each $\hat{x}$. Since A is elementwise positive, we can use $\tilde{\nu} = \max\{0, \nu'\}$, with $\nu' = A\hat{x} - b$, with the dual line search of Subsection 3.3.3. In experiments, we find that $\tilde{\nu}$ constructed in this manner is always strictly dual feasible. We also compute a second (likely) dual feasible point using the additional line search described in Subsection 3.4.2.

For each line search, we require that $A^T\hat{\nu} \geq 10^{-14}$ to give a measure of robustness to finite-precision arithmetic. After computing the dual feasible point $\hat{\nu}$ for all RHS, we estimate the duality gap $\epsilon = f(\hat{x}) - g(\hat{\nu})$. We find that the second line search typically produces tighter duality gaps, except for very high accuracy $\hat{x}$. In either case, we use the point with the tightest duality gap for feature elimination.

By eliminating features, we can form the reduced NNLS problem

$$\begin{array}{ll} \min & \frac{1}{2}\|A_{\text{red}}x_{\text{red}} - b\|^2 \\ \text{s.t.} & x_{\text{red}} \geq 0, \end{array}$$

where A_{red} contains the columns of A that have not been eliminated. The reduced problem is strongly convex (and therefore has a unique optimal point) if A_{red} is overdetermined and full-rank. Since our feature elimination procedure is safe, there is a direct correspondence between the solution of the reduced problem and the original NNLS problem. This correspondence means that uniqueness of the solution to the reduced problem implies uniqueness of the solution to the original problem. Therefore to certify uniqueness of a solution we must eliminate at least $2822-1681 = 1141$ features and verify that the reduced matrix A_{red} is full-rank.

Failure to eliminate sufficiently many features can come in many forms. If the problem does have a unique solution that is sufficiently sparse but $\hat{x}$, $\hat{\nu}$ do not produce a tight enough duality gap ϵ, we need only increase the accuracy of $\hat{x}$ and $\hat{\nu}$ to certify uniqueness. The solution could be sparse enough but not satisfy strict complementary slackness, so the feature elimination problems will not eliminate the zero features associated with zero dual variables. The problem may also have a solution (possibly unique) that is simply not sparse enough to certify uniqueness via feature elimination. Even if we had access to the exact points x^* and ν^*, feature elimination would not eliminate sufficiently many features. In such a case, the alternative method in Subsection 3.4.4 may be used, assuming the strong condition on A is satisfied. And of course, we cannot certify uniqueness if the problem does not have a unique solution.

Table 3.2 shows the results of using the strong concavity subproblem (3.27). We show the number of iterations of AT (Algorithm 8), the total number of solutions certified to be unique, and the average number of features eliminated across all NNLS problems. There are 40000 NNLS problems, each with 2822 features, giving approximately 113 million total features. As the accuracy of the primal feasible point $\hat{x}$ increases, the duality gap closes and more features are eliminated from the problem. Even at 500K iterations, not all problems are certified to have a unique solution.

Iterations	Number Certified Unique	Average Features Eliminated
10000	7237	20.0%
50000	18339	46.2%
100000	23512	57.9%
500000	34462	81.6%

Table 3.2: Feature elimination using the strong concavity subproblem (3.27).

Table 3.3 shows the analogous results when using the strong concavity and partial dual feasibility subproblem (3.32). We see a marked improvement in the number of solutions certified to be unique, though we still fall a bit short of certifying uniqueness for all 40000 problems. This appears to be due to a few particularly slow-to-converge problems where the accuracy of $\hat{x}$ is still quite low. We fix this by computing a high-accuracy solution for the remaining 1996 problems using MATLAB's `lsqnonneg`, which implements an active set method from [57]. This results in a sufficiently accurate $\hat{x}$ and we certify the remaining problems as having unique solutions. Thus we have successfully used feature elimination to certify that all 40000 problems have unique solutions in one of our microscopy scenarios.

Iterations	Number Certified Unique	Average Features Eliminated
10000	9510	24.8%
50000	23174	56.9%
100000	28826	68.3%
500000	38094	87.8%
500000 + `lsqnonneg`	**40000**	91.0%

Table 3.3: Feature elimination using the strong concavity subproblem with partial dual feasibility (3.32).

3.4.4 An Alternative Method to Certify Uniqueness

We have seen how to use feature elimination for NNLS problems to certify uniqueness of solutions by eliminating sufficiently many features from the solution. Our microscopy problems exhibit quite sparse solutions, which allows us to certify uniqueness for all 40000 subimages. If the solution is not sufficiently sparse, however, the feature elimination strategy will fail to certify uniqueness.

Slawski and Hein, as part of their analysis of NNLS problems in [84], prove a lemma on the uniqueness of NNLS solutions. The lemma relies on a strong assumption on the columns of A, but provides a simple condition to certify uniqueness of a solution. We discuss this condition first, state their lemma, and finally discuss how to use their lemma to certify uniqueness in practice.

The columns of the matrix $A \in \mathbb{R}^{m \times n}$ are said to be in **general linear position** (GLP) in

$\mathbb{R}^m$ if the following condition holds:

$$\forall \mathcal{J} \subseteq \{1, \ldots, n\},\ |\mathcal{J}| = \min\{m, n\},\ \forall x \in \mathbb{R}^{|\mathcal{J}|}, A_{\mathcal{J}} x = 0 \implies x = 0. \tag{3.33}$$

In other words, every subset of $\min\{m, n\}$ columns is linearly-independent. For brevity, we will say "A is in GLP" to mean "the columns of A are in GLP". It is easy to see that A in GLP implies that A is full-rank, but the converse is not true; GLP is strictly a stronger criterion than full-rank. A being in GLP is also related to the spark of A, where $\operatorname{spark}(A)$ is defined in [29] to be minimum number of columns that form a linearly dependent set. If $m < n$, then A is in GLP iff $\operatorname{spark}(A) = m + 1$.

Unlike the rank of a matrix, determining the spark of A and determining if A is in GLP may be prohibitively difficult in the worst case. The straightforward computation to determine if A is in GLP requires computing combinatorially many determinants. In [55] it is shown that determining if A is not in GLP is NP-complete. Determining if A is in GLP (equivalently, if $\operatorname{spark}(A) = m + 1$) is coNP-complete [3, 91]. Computing $\operatorname{spark}(A)$ is NP-hard in general [91]. So computationally verifying that A is in GLP is likely intractable.

Note that these are worst-case results, and there are matrices that are known to be in GLP or have known spark. For example, take $A \in \mathbb{R}^{m \times n}$ with each entry drawn from the standard normal distribution (i.e., A is taken from the Gaussian ensemble). Then A is in GLP with probability one (proof comes from the union bound and non-singularity of the Gaussian ensemble). Though it is complex, another example is $A = \begin{bmatrix} I_n & F_n \end{bmatrix}$ where I_n is the $n \times n$ identity matrix and F_n is the $n \times n$ discrete Fourier transform matrix. When n is a perfect square, the spark is known to be exactly $2\sqrt{n}$ [29].

There are also lower bounds for $\operatorname{spark}(A)$ [29, 93]. One such bound is $\operatorname{spark}(A) > 1/\mu(A)$, where

$$\mu(A) = \max_{i \neq j} \frac{|\langle a_i, a_j \rangle|}{\|a_i\| \|a_j\|}$$

is called the coherence parameter of A. For our $A \in \mathbb{R}^{1681 \times 2822}$, we have $\mu(A) \approx 0.99$ which gives the uninformative bound $\operatorname{spark}(A) \geq 2$.

Assuming we know that A is in GLP, the following lemma from [84] gives a simple condition implying the uniqueness of the NNLS solution. The result is akin to a result on the uniqueness of the solution to ℓ_1-regularized least-squares [90].

Lemma 10 (Lemma 5 from [84])**:** *Let the columns of $A \in \mathbb{R}^{m\times n}$, $m < n$, be in GLP. If the NNLS optimal value is strictly positive,*

$$p^* = \min_{x\geq 0} \frac{1}{2}\|Ax - b\|^2 > 0,$$

then the NNLS problem has a unique solution.

For underdetermined NNLS problems with A in GLP, we can certify uniqueness simply by certifying $p^* > 0$. Assuming we know that A is in GLP, this is simple to check in practice. We can produce a dual feasible point $\hat{\nu}$ from a primal feasible point $\hat{x}$ as in Subsection 3.3.3 or in Subsection 3.4.3. If we have that $g(\hat{\nu}) > 0$, then $p^* > 0$ by weak duality and the solution is certified to be unique.

Note that this method can certify uniqueness even if the NNLS solution is not sufficiently sparse to produced an overdetermined reduced problem. That is to say, this can be used even when certifying uniqueness via feature elimination fails. However, we must know that A is in GLP, which may be infeasible to check directly. When feature elimination fails to certify uniqueness, it still provides certificates that features are not present in the solution. This is a positive result, whereas the Lemma 10 approach provides no additional benefit when it fails to certify uniqueness.

3.5 Discussion

We have seen a number of interesting challenges presented by focused-spot illumination microscopy. Each individual NNLS problem involved in image formation is moderately sized, and so is individually a good candidate for an IPM or active-set method. However, the sheer number of NNLS problems, all with the same A, is more efficiently solved by first-order methods, which can naturally be adapted to work on multiple RHS simultaneously.

The PSF dictionary used in our examples is extremely ill-conditioned ($\kappa_2(A) \approx 10^{20}$), though we still see acceptable performance from AT (Algorithm 8) in practice. We have seen that only a few thousands of iterations of AT are required to give lower accuracy images (see Figure 3.3). On our NVIDIA K40c GPU, running AT until all 40000 RHS have a relative norm of the gradient less than 10^{-5} takes about an hour to run. However, this is quite slow compared to the, say, 100 seconds taken for image acquisition. There is obviously the desire for further improvements in runtime.

Preconditioning AT offers one practical way forward on this front. We discuss fundamental obstacles when using preconditioners with projected/proximal methods and a possible way to improve our results in Subsection 3.5.1. ADMM offers another route to improved runtime by bringing in second-order information in an inexpensive manner.

3.5.1 Discussion of Preconditioning

The obstacles faced when preconditioning first-order methods applied to NNLS are not specific to NNLS. Since projected gradient methods are typically used when the projection operator can be applied quickly, introducing a preconditioner must not interfere too severely with the projection. This notion generalizes directly to proximal gradient methods.

If the proximal function is separable, a diagonal preconditioner is a natural choice that will not interfere with the application of the proximal operator. However, a diagonal preconditioner offers negligible improvement in problem conditioning for our microscopy problems. If $MM^T = \alpha I$ with $\alpha > 0$, then, in a sense, M and the proximal function commute (see Proposition 11 of [20] or Table 1 of [21]). However, such an M has $\kappa_2(M) = 1$ and may not improve conditioning.

In [70], a graph-based approach is used to build a block diagonal preconditioner for the primal-dual hybrid gradient method. The structure of a block diagonal preconditioner, together with its graph-based construction, allows for a fast solution of the proximal subproblem.

Giselsson and Boyd in [39] use preconditioning to tremendous effect for a model predictive control problem. While extremely effective, they were able to construct a preconditioner with minimal structural constraints due to their particular problem. However, they offer a fairly general approach to constructing preconditioners. Adapted to the QP (3.9), we could construct M by solving the problem

$$\begin{aligned} \min \quad & \frac{\sigma_{\max}(M^T A^T A M)}{\sigma_{\min>0}(M^T A^T A M)} \\ \text{s.t.} \quad & M \in \mathcal{M}, \end{aligned}$$

where $\mathcal{M}$ is a structure imposing set. This problem can be cast as a semi-definite program and solved for moderate sized A [39].

While this may be expensive, it is a one-time cost for our NNLS problems, as the PSF dictionary is the same for all subimages (and any further subimages, as long as the optical system is not changed). We could use this approach with $\mathcal{M}$ designed to restrict M to be block diagonal, which results in the same cost of solving the projection subproblem (3.10). Moreover constructing M by minimizing the condition number should result in a more effective preconditioner than the block inverse matrix square-root we used in Subsection 3.2.2.

3.5.2 Discussion of ADMM

ADMM offers an inexpensive way of bringing in second-order information to a simple iterative method. The dominant cost of ADMM (Algorithm 9) is the application of $(A^T A + \rho I)^{-1}$ to a vector. We offer both a Cholesky-based and an SVD-based implementation of this operation. For overdetermined problems, the Cholesky-based approach is the cheapest, being on par with the cost of AT using the QP formulation (3.9). However, the step size ρ is embedded in the Cholesky factor; to change ρ, the Cholesky factorization must be recomputed. The SVD-based approach allows for changing the step size ρ at-will. Moreover, for underdetermined problems, the cost of an iteration of ADMM with the SVD-based implementation is the same as a step of AT.

We saw in Figure 3.7 that ADMM with a manually tuned ρ can converge faster than AT for our NNLS problems arising from our microscopy application. We also tried the automatic ρ adaptation strategy from [16], but it did not produce convergence nearly as fast as the manually tuned, fixed ρ. Since manually tuning ρ is a somewhat crude, interactive procedure, we have generally preferred AT, due to its automatic step size selection.

In experiments with a random ill-conditioned A (Subsection 3.2.3.5), we saw that ADMM with a manually tuned ρ could achieve convergence that is much faster than AT. It is unclear why ADMM with a random NNLS problem achieves such fast convergence, while ADMM with our microscopy NNLS problems is only slightly faster than AT. One possible explanation is that convergence depends not just on the structure and conditioning of A (e.g., as used in Theorem 3 for PGM), but also on the RHS b. That is to say, convergence bounds based solely on the condition number of A are overly pessimistic compared to the convergence we see in practice. It is possible that the random RHS for a random problem results in an "easier" problem for ADMM than the real-data NNLS problems from our microscopy application.

It was shown in [24] that the Douglas-Rachford (DR) splitting for basis pursuit eventually exhibits linear convergence, with rate depending on the principal angle between the nullspace of A and a subspace depending on zeros in the solution. This rate is notably independent of the condition number of A. Since ADMM is a special case of the DR splitting [16], it may be possible to find a similar theoretical explanation for the convergence of ADMM for NNLS problems.

3.5.3 Discussion of Safe Feature Elimination

In Sections 3.3 and 3.4 we developed a new safe feature elimination strategy for NNLS problems. Our strategy is applicable to any NNLS problem, though we do make a few (optional) adaptations for the NNLS problems arising our microscopy application. We showed how to use the strategy to certify the uniqueness of solutions for underdetermined NNLS problems.

Let us discuss briefly some further applications of our dual line search and feature elimination strategies. Let us suppose we have performed some number of iterations of a first-order NNLS solver, giving us a primal feasible point $\hat{x}$. Using $\hat{x}$, we we can perform our dual line search (in closed-form) to find a dual feasible $\hat{\nu}$. The duality gap $\epsilon = f(\hat{x}) - g(\hat{x})$ directly gives us a bound on how far away the objective value is from the optimal value:

$$f(\hat{x}) - f(x^*) \leq \epsilon.$$

Once we have $\hat{x}$ and $\hat{\nu}$, we can then compute the solution to a number of feature elimination subproblems to eliminate features from the problem. Doing so will then allow us to form a reduced matrix, A_{red}, with fewer columns. This has the benefits of a faster gradient computation and possibly accelerated convergence (e.g., by decreasing $\kappa_2(A)$).

We saw in Subsection 3.4.3, that if A_{red} was underdetermined and full-rank, we can certify that the solution to the original NNLS problem is unique. In such a case, the reduced objective is $\sigma_{\min}(A)^2$-strongly convex. We can use this to bound the distance between $\hat{x}$ and the unique optimal point x^*, using Theorem 1. Let $f_{\text{red}}(x_{\text{red}}) = \frac{1}{2}\|A_{\text{red}}x_{\text{red}} - b\|^2$ and $\hat{x}_{\text{red}}$, x^*_{red} be equal to $\hat{x}$, x^*, respectively, with the eliminated coordinates removed. We then have that

$$\tfrac{\sigma_{\min}(A)^2}{2}\|\hat{x} - x^*\|^2 = \tfrac{\sigma_{\min}(A)^2}{2}\|\hat{x}_{\text{red}} - x^*_{\text{red}}\|^2 \le f_{\text{red}}(\hat{x}_{\text{red}}) - f_{\text{red}}(x^*_{\text{red}}) = f(\hat{x}) - f(x^*) \le \epsilon,$$

allowing us to bound the distance from $\hat{x}$ to x^* in terms of the duality gap ϵ.

Performing the dual line search with a fixed, strictly dual feasible $\tilde{\nu}$ requires only a single matrix-vector product $A^T\hat{\nu}$. When using $\tilde{\nu} = \max\{0, \nu'\}$ (for A elementwise non-negative) or $\tilde{\nu} = \nu' + t\nabla g(\nu')$, we must check that $A^T\tilde{\nu} > 0$, which requires a matrix-vector product. The computation of the optimal value of the basic feature elimination subproblem (3.25) for all columns of A can re-use $A^T\hat{\nu}$. Roughly speaking, this means that feature elimination costs on the order of a gradient evaluation. It is therefore rather inexpensive to stop a first-order method to try to eliminate features.

Using the stronger subproblem (3.32), for instance, is more expensive than the basic subproblem (3.25). However, many of the required values in the closed-form solution are independent of b and $\hat{\nu}$ and can be precomputed. The "dome subproblem" closed-form solution, Algorithm 11, uses these values plus $\mathcal{O}(1)$ additional work per use. So, it costs roughly a gradient evaluation plus constant-time additional work.

Note that this pruning technique may not apply so well to the scenario arising from our microscopy application. We have many medium-sized NNLS problems with the same A, but each with different RHS b. The features eliminated from each problem are generally different, with a limited number of shared eliminated features. Therefore, the reduced A for each problem may be quite different, and we can no longer compute many gradients simultaneously using matrix-matrix products (as was the case with A fixed for all RHS). The gain from using a smaller A for each problem may be offset by the increased cost of computing gradients using matrix-vector products instead of a matrix-matrix product.

For a single very large-scale NNLS problem, the pruning technique is likely advantageous. First-order methods also enjoy an advantage due to their use of simple matrix-vector operations. At this scale, matrix-vector products are expensive due to the sheer size of A. Using feature elimination to reduce the problem size as a first-order solver progresses offers a way to decrease the cost of the gradient computation for all future iterations.

Though we have focused on first-order methods, safe feature elimination can be applied to other methods. For instance, one could incorporate our strategy into an active set method to certify coordinates will be zero at a solution and can therefore be removed from further consideration by the solver.

3.5.4 Future Directions

Here we list a few assorted further directions for our work.

- NNLS uses a Gaussian noise model, though a Poisson noise model is more appropriate. We sketch the Poisson maximum likelihood problem and how safe feature elimination can be adapted in Subsection 3.5.5.
- In Appendix A.2 we show that NNLS can be thought of as an extremal case of a regularized least-squares problem. This may allow us to derive a homotopy method for solving NNLS, akin to LARS for LASSO [35]. The regularization path would start with penalty $\mu = 0$, which results in the ordinary least-squares solution, and increase μ until an NNLS solution is reached.
- Suppose we construct a random NNLS problem with underdetermined $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$, each filled standard normal entires. In preliminary numerical experiments, it

appears that if $m > n/2$, then the NNLS solution is unique with high probability as $m, n \to \infty$. Conversely, if $m < n/2$, then the NNLS solution is non-unique with high probability as $m, n \to \infty$. It would be interesting to quantify when such a random NNLS problem has a unique solution. Work in this direction would likely utilize previous work from [4, 18].

3.5.5 Safe Feature Elimination for Poisson Maximum Likelihood

A Gaussian noise model, which is implicit in the formulation of NNLS, is an approximation to the noise the camera actually detects. Nevertheless, NNLS provides a useful model, in particular because it is convex with (globally) Lipschitz continuous gradient. It is more accurate to model the camera pixel counts with a Poisson distribution. In terms of the PSF dictionary matrix A and PSF intensities x, we expect the camera pixel values to follow

$$p(b_i|x) = \frac{\{Ax\}_i^{b_i} e^{-\{Ax\}_i}}{b_i!},$$

where $\{\cdot\}_i$ denotes the ith element of a vector. Note that the camera pixel values are photon counts (i.e., integers), though the “rate” $\{Ax\}_i$ need only be positive. We assume that the elements of b are conditionally independent given x, so

$$p(b|x) = \prod_{i=1}^{m} p(b_i|x)$$

The negative log-likelihood of the PSF intensities x, after some simplification, is,

$$-\langle \log(Ax), b\rangle + \langle Ax + \log(b!), \mathbb{1}\rangle,$$

where the log and factorial are taken elementwise and $\mathbb{1}$ is the vector of all ones. The maximum likelihood estimate of the non-negative PSF intensities is then

$$\begin{aligned} \min \quad & -\langle \log(Ax), b\rangle + \langle Ax + \log(b!), \mathbb{1}\rangle \\ \text{s.t.} \quad & x \geq 0. \end{aligned} \tag{3.34}$$

Note that the Poisson ML objective does not have a globally Lipschitz continuous gradient; as an element of ν approaches 1 from the left, the gradient blows up.

After finding an appropriate introduction of a variable (akin to that used in Subsection 3.3.1), we find that a dual to Poisson ML is

$$\begin{aligned} \max \quad & -\left\langle \tfrac{b}{1-\nu}, b \right\rangle + \langle b + \log(b!), \mathbb{1} \rangle \\ \text{s.t.} \quad & A^T \nu \geq 0. \end{aligned} \tag{3.35}$$

Note that there is the implicit constraint $\nu \leq 1$, which comes from the domain of the log in the Poisson ML primal problem. Otherwise, the dual constraint $A^T\nu \geq 0$ is identical to the NNLS dual (3.18). The dual objective is not globally strongly concave, which is in accordance with the conjugate correspondence theorem (Theorem 5.26 of [10]). That means we cannot directly use strong-concavity-based feature elimination subproblems, like those discussed Subsection 3.4.1.

However, there appears to be an alternate approach. In our initial experiments the dual feasible set

$$\mathcal{N} = \{\nu \,:\, \nu \leq 1, A^T\nu \geq 0\}$$

is bounded, a property of the PSF dictionary matrix A and does not depend on the RHS b. If we have a dual feasible point $\hat{\nu}$ (e.g., from a primal feasible point $\hat{x}$), this allows us to bound the dual optimal point according to $g(\nu^*) \geq g(\hat{\nu})$, which in turn allows us to bound ν^* away from $\mathbb{1}$. The bounded dual feasible set that is also bounded away from $\nu = \mathbb{1}$ then gives us a locally strongly convex dual objective. If we can find the local strong convexity constant on this bounded domain (which must contain the dual optimal point ν^*), we can then use feature elimination as we have for NNLS.

3.5.6 Acknowledgements

Stephen Becker acknowledges the donation of a Tesla K40c GPU from NVIDIA. This work utilized the RMACC Summit supercomputer, which is supported by the National Science Foundation (awards ACI-1532235 and ACI-1532236), the University of Colorado Boulder, and Colorado State University. The Summit supercomputer is a joint effort of the University of Colorado Boulder and Colorado State University.

Bibliography

[1] N. Ailon and B. Chazelle, Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform, in Proceedings of the thirty-eighth annual ACM symposium on Theory of Computing, New York, NY, USA, 2006, ACM, ACM, pp. 557–563.

[2] ——, The fast Johnson–Lindenstrauss transform and approximate nearest neighbors, SIAM Journal on Computing, 39 (2009), pp. 302–322.

[3] B. Alexeev, J. Cahill, and D. G. Mixon, Full spark frames, Journal of Fourier Analysis and Applications, 18 (2012), pp. 1167–1194.

[4] D. Amelunxen, M. Lotz, M. B. McCoy, and J. A. Tropp, Living on the edge: Phase transitions in convex programs with random data, Information and Inference: A Journal of the IMA, 3 (2014), pp. 224–294.

[5] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, LAPACK Users' Guide, Society for Industrial and Applied Mathematics, Philadelphia, PA, third ed., 1999.

[6] L. Auslander and R. Tolimieri, Computing decimated finite cross-ambiguity functions, IEEE Transactions on Acoustics, Speech, and Signal Processing, 36 (1988), pp. 359–364.

[7] A. Auslender and M. Teboulle, Interior gradient and proximal methods for convex and conic optimization, SIAM Journal on Optimization, 16 (2006), pp. 697–725.

[8] H. Avron, P. Maymounkov, and S. Toledo, BLENDENPIK: Supercharging LAPACK's least-squares solver, SIAM Journal on Scientific Computing, 32 (2010), pp. 1217–1236.

[9] G. Ballard, J. Demmel, and I. Dumitriu, Minimizing communication for eigenproblems and the singular value decomposition, Tech. Rep. EECS-2011-14, Electrical Engineering and Computer Sciences, University of California, Berkeley, 2010.

[10] A. Beck, First-Order Methods in Optimization, vol. 25, SIAM, 2017.

[11] S. R. Becker, E. J. Candès, and M. C. Grant, Templates for convex cone problems with applications to sparse signal recovery, Mathematical Programming Computation, 3 (2011), p. 165.

[12] E. Betzig, G. H. Patterson, R. Sougrat, O. W. Lindwasser, S. Olenych, J. S. Bonifacino, M. W. Davidson, J. Lippincott-Schwartz, and H. F. Hess, Imaging intracellular fluorescent proteins at nanometer resolution, Science, 313 (2006), pp. 1642–1645.

[13] J. M. Bioucas-Dias and M. A. Figueiredo, Alternating direction algorithms for constrained sparse regression: Application to hyperspectral unmixing, in Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS), 2010 2nd Workshop on, IEEE, 2010, pp. 1–4.

[14] C. Bischof and C. Van Loan, The WY representation for products of Householder matrices, SIAM Journal on Scientific and Statistical Computing, 8 (1987), pp. s2–s13.

[15] C. H. Bischof and G. Quintana-Ortí, Algorithm 782: codes for rank-revealing QR factorizations of dense matrices, ACM Transactions on Mathematical Software (TOMS), 24 (1998), pp. 254–257.

[16] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, et al., Distributed optimization and statistical learning via the alternating direction method of multipliers, Foundations and Trends in Machine Learning, 3 (2011), pp. 1–122.

[17] S. Boyd and L. Vandenberghe, Convex Optimization, Cambridge University Press, Cambridge, United Kingdom, 2004.

[18] J. J. Bruer, J. A. Tropp, V. Cevher, and S. R. Becker, Designing statistical estimators that balance sample size, risk, and computational cost, IEEE Journal of Selected Topics in Signal Processing, 9 (2015), pp. 612–624.

[19] W. Cheney and A. A. Goldstein, Proximity maps for convex sets, Proceedings of the American Mathematical Society, 10 (1959), pp. 448–450.

[20] P. L. Combettes and J.-C. Pesquet, A Douglas–Rachford splitting approach to nonsmooth convex variational signal recovery, IEEE Journal of Selected Topics in Signal Processing, 1 (2007), pp. 564–574.

[21] ——, Proximal splitting methods in signal processing, in Fixed-Point Algorithms for Inverse Problems in Science and Engineering, Springer, 2011, pp. 185–212.

[22] J. Cooley and S. Winograd, A limited range discrete Fourier transform algorithm, in IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 5, IEEE, 1980, pp. 213–217.

[23] J. W. Cooley and J. W. Tukey, An algorithm for the machine calculation of complex Fourier series, Mathematics of Computation, 19 (1965), pp. 297–301.

[24] L. Demanet and X. Zhang, Eventual linear convergence of the Douglas-Rachford iteration for basis pursuit, Mathematics of Computation, 85 (2016), pp. 209–238.

[25] J. Demmel, I. Dumitriu, and O. Holtz, Fast linear algebra is stable, Numerische Mathematik, 108 (2007), pp. 59–91.

[26] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, Communication-optimal parallel and sequential QR and LU factorizations, SIAM Journal on Scientific Computing, 34 (2012), pp. A206–A239.

[27] J. Demmel and K. Veselić, Jacobi's method is more accurate than QR, SIAM Journal on Matrix Analysis and Applications, 13 (1992), pp. 1204–1245.

[28] J. W. Demmel, L. Grigori, M. Gu, and H. Xiang, Communication avoiding rank revealing QR factorization with column pivoting, SIAM Journal on Matrix Analysis and Applications, 36 (2015), pp. 55–89.

[29] D. L. Donoho and M. Elad, Optimally sparse representation in general (nonorthogonal) dictionaries via ℓ^1 minimization, Proceedings of the National Academy of Sciences, 100 (2003), pp. 2197–2202.

[30] D. L. Donoho, I. M. Johnstone, J. C. Hoch, and A. S. Stern, Maximum entropy and the nearly black object, Journal of the Royal Statistical Society. Series B (Methodological), (1992), pp. 41–81.

[31] Z. Drmač and Z. Bujanović, On the failure of rank-revealing QR factorization software–a case study, ACM Transactions on Mathematical Software (TOMS), 35 (2008), p. 12.

[32] Z. Drmač and K. Veselić, New fast and accurate Jacobi SVD algorithm. I, SIAM Journal on Matrix Analysis and Applications, 29 (2008), pp. 1322–1342.

[33] ——, New fast and accurate Jacobi SVD algorithm. II, SIAM Journal on Matrix Analysis and Applications, 29 (2008), pp. 1343–1362.

[34] J. A. Duersch and M. Gu, Randomized QR with column pivoting, SIAM Journal on Scientific Computing, 39 (2017), pp. C263–C291.

[35] B. Efron, T. Hastie, I. Johnstone, R. Tibshirani, et al., Least angle regression, The Annals of statistics, 32 (2004), pp. 407–499.

[36] M. P. Friedlander and G. Goh, Efficient evaluation of scaled proximal operators, Electronic Transactions on Numerical Analysis, 46 (2017), pp. 1–22.

[37] M. Frigo and S. G. Johnson, The design and implementation of FFTW3, Proceedings of the IEEE, 93 (2005), pp. 216–231.

[38] L. E. Ghaoui, V. Viallon, and T. Rabbani, Safe feature elimination for the LASSO and sparse supervised learning problems, arXiv preprint arXiv:1009.4219, (2010).

[39] P. Giselsson and S. Boyd, Preconditioning in fast dual gradient methods, in IEEE 53rd Annual Conference on Decision and Control (CDC), IEEE, 2014, pp. 5040–5045.

[40] G. Goh, Why momentum really works, Distill, (2017).

[41] T. Goldstein, B. O'Donoghue, S. Setzer, and R. Baraniuk, Fast alternating direction optimization methods, SIAM Journal on Imaging Sciences, 7 (2014), pp. 1588–1623.

[42] G. H. Golub and C. F. Van Loan, Matrix computations, vol. 3, JHU Press, Baltimore, MD, USA, 1998.

[43] M. Grant and S. Boyd, CVX: Matlab software for disciplined convex programming, version 2.1, Mar. 2014.

[44] L. Grigori, S. Cayrols, and J. W. Demmel, Low rank approximation of a sparse matrix based on LU factorization with column and row tournament pivoting, SIAM Journal on Scientific Computing, 40 (2018), pp. C181–C209.

[45] M. Gu and S. C. Eisenstat, Efficient algorithms for computing a strong rank-revealing QR factorization, SIAM Journal on Scientific Computing, 17 (1996), pp. 848–869.

[46] M. Gu, L.-H. Lim, and C. J. Wu, ParNes: a rapidly convergent algorithm for accurate recovery of sparse and approximately sparse signals, Numerical Algorithms, 64 (2013), pp. 321–347.

[47] L. Gubin, B. Polyak, and E. Raik, The method of projections for finding the common point of convex sets, USSR Computational Mathematics and Mathematical Physics, 7 (1967), pp. 1–24.

[48] M. G. Gustafsson, Surpassing the lateral resolution limit by a factor of two using structured illumination microscopy, Journal of Microscopy, 198 (2000), pp. 82–87.

[49] P. C. Hansen, Regularization tools version 4.0 for matlab 7.3, Numerical Algorithms, 46 (2007), pp. 189–194.

[50] B. He, H. Yang, and S. Wang, Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities, Journal of Optimization Theory and Applications, 106 (2000), pp. 337–356.

[51] S. W. Hell and J. Wichmann, Breaking the diffraction resolution limit by stimulated emission: stimulated-emission-depletion fluorescence microscopy, Optics Letters, 19 (1994), pp. 780–782.

[52] N. J. Higham, Accuracy and stability of numerical algorithms, SIAM, 2002.

[53] D. A. Huckaby and T. F. Chan, On the convergence of Stewart's QLP algorithm for approximating the SVD, Numerical Algorithms, 32 (2003), pp. 287–316.

[54] W. B. Johnson and J. Lindenstrauss, Extensions of Lipschitz mappings into a Hilbert space, Contemporary Mathematics, 26 (1984), p. 1.

[55] L. Khachiyan, On the complexity of approximating extremal determinants in matrices, Journal of Complexity, 11 (1995), pp. 138–153.

[56] S.-J. Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky, An interior-point method for large-scale ℓ_1-regularized least squares, IEEE Journal of Selected Topics in Signal Processing, 1 (2007), pp. 606–617.

[57] C. L. Lawson and R. J. Hanson, Solving least squares problems, vol. 15, SIAM, 1995.

[58] D. A. Lorenz, Constructing test instances for basis pursuit denoising, IEEE Transactions on Signal Processing, 61 (2013), pp. 1210–1214.

[59] R. G. Lyons, Understanding Digital Signal Processing, Pearson Education, 2010.

[60] M. W. Mahoney et al., Randomized algorithms for matrices and data, Foundations and Trends in Machine Learning, 3 (2011), pp. 123–224.

[61] J. Mairal, F. Bach, J. Ponce, et al., Sparse modeling for image and vision processing, Foundations and Trends in Computer Graphics and Vision, 8 (2014), pp. 85–283.

[62] J. Mairal and B. Yu, Complexity analysis of the lasso regularization path, arXiv preprint arXiv:1205.0079, (2012).

[63] J. Markel, FFT pruning, IEEE Transactions on Audio and Electroacoustics, 19 (1971), pp. 305–311.

[64] P.-G. Martinsson, Blocked rank-revealing QR factorizations: How randomized sampling can be used to avoid single-vector pivoting, arXiv preprint arXiv:1505.08115, (2015).

[65] P.-G. Martinsson, G. Quintana-Orti, N. Heavner, and R. van de Geijn, Householder QR factorization: Adding randomization for column pivoting. FLAME working note# 78, arXiv preprint arXiv:1512.02671, (2015).

[66] Mathworks, Inc., MATLAB 8.6, 2016.

[67] J. Mattingley and S. Boyd, CVXGEN: A code generator for embedded convex optimization, Optimization and Engineering, 13 (2012), pp. 1–27.

[68] X. Meng, M. A. Saunders, and M. W. Mahoney, LSRN: a parallel iterative solver for strongly over- or underdetermined systems, SIAM Journal on Scientific Computing, 36 (2014), pp. C95–C118.

[69] F. Mezzadri, How to generate random matrices from the classical compact groups, Notices of the American Mathematical Society, 54 (2007), pp. 592–604.

[70] T. Möllenhoff, Z. Ye, T. Wu, and D. Cremers, Combinatorial preconditioners for proximal algorithms on graphs, in Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics, 2018.

[71] A. S. Nemirovsky and D. B. Yudin, Problem complexity and method efficiency in optimization, (1983).

[72] Y. Nesterov, A method of solving a convex programming problem with convergence rate $\mathcal{O}(1/k^2)$, in Soviet Mathematics Doklady, vol. 27, 1983, pp. 372–376.

[73] ——, Introductory lectures on convex optimization: A basic course, vol. 87, Springer Science & Business Media, 2013.

[74] D. K. Nguyen and T. B. Ho, Anti-lopsided algorithm for large-scale nonnegative least square problems, arXiv preprint arXiv:1502.01645, (2015).

[75] Nobel Foundation, The Nobel prize in chemistry, 2014.

[76] A. V. Oppenheim, J. R. Buck, and R. W. Schafer, Discrete-time signal processing, (1989).

[77] C. C. Paige and M. A. Saunders, LSQR: An algorithm for sparse linear equations and sparse least squares, ACM Transactions on Mathematical Software (TOMS), 8 (1982), pp. 43–71.

[78] B. Porat, A course in digital signal processing, vol. 1, Wiley New York, 1997.

[79] J. Poulson, B. Marker, R. A. Van de Geijn, J. R. Hammond, and N. A. Romero, Elemental: A new framework for distributed memory dense matrix computations, ACM Transactions on Mathematical Software (TOMS), 39 (2013), p. 13.

[80] G. Quintana-Ortí, X. Sun, and C. H. Bischof, A BLAS-3 version of the QR factorization with column pivoting, SIAM Journal on Scientific Computing, 19 (1998), pp. 1486–1494.

[81] E. H. Rego, L. Shao, J. J. Macklin, L. Winoto, G. A. Johansson, N. Kamps-Hughes, M. W. Davidson, and M. G. Gustafsson, Nonlinear structured-illumination microscopy with a photoswitchable protein reveals cellular structures at 50-nm resolution, Proceedings of the National Academy of Sciences, 109 (2012), pp. E135–E143.

[82] R. T. Rockafellar, Convex analysis, Princeton university press, 2015.

[83] V. Rokhlin and M. Tygert, A fast randomized algorithm for overdetermined linear least-squares regression, Proceedings of the National Academy of Sciences, 105 (2008), pp. 13212–13217.

[84] M. Slawski, M. Hein, et al., Non-negative least squares for high-dimensional linear models: Consistency and sparse recovery without regularization, Electronic Journal of Statistics, 7 (2013), pp. 3004–3056.

[85] S. Stein, Algorithms for ambiguity function processing, IEEE Transactions on Acoustics, Speech, and Signal Processing, 29 (1981), pp. 588–599.

[86] G. Stewart, The QLP approximation to the singular value decomposition, SIAM Journal on Scientific Computing, 20 (1999), pp. 1336–1348.

[87] A. Szlam, Z. Guo, and S. Osher, A split Bregman method for non-negative sparsity penalized least squares with applications to hyperspectral demixing, in Image Processing (ICIP), 2010 17th IEEE International Conference on, IEEE, 2010, pp. 1917–1920.

[88] R. Tibshirani, Regression shrinkage and selection via the lasso, Journal of the Royal Statistical Society. Series B (Methodological), (1996), pp. 267–288.

[89] R. Tibshirani, J. Bien, J. Friedman, T. Hastie, N. Simon, J. Taylor, and R. J. Tibshirani, Strong rules for discarding predictors in lasso-type problems, Journal of the Royal Statistical Society: Series B (Statistical Methodology), 74 (2012), pp. 245–266.

[90] R. J. Tibshirani et al., The lasso problem and uniqueness, Electronic Journal of Statistics, 7 (2013), pp. 1456–1490.

[91] A. M. Tillmann and M. E. Pfetsch, The computational complexity of the restricted isometry property, the nullspace property, and related concepts in compressed sensing, IEEE Transactions on Information Theory, 60 (2014), pp. 1248–1259.

[92] R. Tolimieri and S. Winograd, Computing the ambiguity surface, IEEE Transactions on Acoustics, Speech, and Signal Processing, 33 (1985), pp. 1239–1245.

[93] J. A. Tropp, Greed is good: Algorithmic results for sparse approximation, IEEE Transactions on Information Theory, 50 (2004), pp. 2231–2242.

[94] J. A. Tropp, Topics in Sparse Approximation, PhD thesis, 2004.

[95] J. A. Tropp, Improved analysis of the subsampled randomized Hadamard transform, Advances in Adaptive Data Analysis, 3 (2011), pp. 115–126.

[96] R. Ulman and E. Geraniotis, Wideband TDOA/FDOA processing using summation of short-time CAF's, IEEE Transactions on Signal Processing, 47 (1999), pp. 3193–3200.

[97] L. G. Weiss, Wavelets and wideband correlation processing, IEEE Signal Processing Magazine, 11 (1994), pp. 13–32.

[98] J.-Y. Yu, S. R. Becker, J. Folberth, B. F. Wallin, S. Chen, and C. J. Cogswell, Achieving superresolution with illumination-enhanced sparsity, Optics Express, 26 (2018), pp. 9850–9865.

Appendix A

Directly Adapting SAFE to NNLS

In this appendix we construct a somewhat direct adaptation of SAFE for LASSO to NNLS problems. Just like SAFE for LASSO, the strategy we find depends on knowledge of an **exact** solution. However, the NNLS problems considered in Chapter 3 are highly ill-conditioned and even the use of high-accuracy solutions proved to be problematic in experiments. Though we ultimately opted for the use of **inexact** solutions in Subsection 3.3.2, the feature elimination strategy discussed in this appendix could be useful for more well-conditioned problems.

In A.1 we summarize the basic feature elimination procedure in SAFE for LASSO from [38]. Section A.2 presents a regularized least-squares problem that provably recovers the NNLS solution for sufficiently strong regularization. Finally, Section A.3 presents the exact-solution based safe feature elimination strategy for NNLS problems.

A.1 Summary of SAFE for LASSO

Consider the ℓ_1-regularized LS problem

$$\min \quad \frac{1}{2}\|Ax-b\|^2+\mu\|x\|_1. \tag{A.1}$$

This problem is sometimes called LASSO, though the lasso, as introduced in [88], is an ℓ_1-constrained LS problem. Nevertheless, we will still call (A.1) LASSO to follow [38].

A dual problem corresponding to (A.1) is

$$\begin{aligned} \max \quad & g(\nu) \\ \text{s.t.} \quad & |\langle a_i, \nu \rangle| \le \mu, \quad i = 1, ..., n, \end{aligned}$$

where

$$g(\nu) = -\frac{1}{2}\|\nu + b\|^2 + \frac{1}{2}\|b\|^2.$$

Note that this dual problem is a strongly convex problem, so the solution is unique. For convenience, define $\lambda_i^* = \{A^T \nu^*\}_i = \langle a_i, \nu^* \rangle$. By observing the complementary slackness condition for the dual problem at the optimal point, we see that if $|\langle a_i, \nu^* \rangle| < \mu$ for some coordinate i, then $x_i^* = 0$. We can therefore eliminate x_i from the problem.

At a high level, SAFE for LASSO constructs a set N of dual points that is guaranteed to contain the dual optimal point ν^*. If $|\langle a_i, \nu \rangle| < \mu$ for all $\nu \in N$, then we can conclude that $|\langle a_i, \nu^* \rangle| < \mu$ and therefore $x_i^* = 0$. To determine if $|\langle a_i, \nu \rangle| < \mu$ for all $\nu \in N$, we can solve

$$\begin{aligned} \max \quad & |\langle a_i, \nu \rangle| \\ \text{s.t.} \quad & \nu \in N. \end{aligned}$$

To remove the non-linearity in the objective, we solve

$$\begin{aligned} \max \quad & \langle a, \nu \rangle \\ \text{s.t.} \quad & \nu \in N. \end{aligned} \tag{A.2}$$

twice: once for $a = a_i$ and once more for $a = -a_i$. This then allows us to find the maximum value of $|\langle a_i, \nu \rangle|$ over $\nu \in N$, which is an upper bound on $|\langle a_i, \nu^* \rangle|$. If this upper bound is strictly less than μ, we can eliminate the ith feature. The rest of the derivation of SAFE for LASSO involves the construction of the set N and solving the above upper bound subproblem.

Assume we have access to an **exact** solution x_0^* for some $\mu_0 \ge \mu$ (in this subsection we use the subscript "0" to indicate a variable corresponds to penalty parameter μ_0). For instance, we may take μ_0 sufficiently large that $x_0^* = 0$. It is well known that this is possible, and a sufficient condition for μ_0 is given in [56].

The search space N is constructed as the intersection of two sets, $N = N_1 \cap N_2$. For N_1, we use the fact that the dual feasible set for penalty parameter $\mu_0 \geq \mu$ is a superset of the dual feasible set with penalty parameter μ. In other words, $|\langle a_i, \nu\rangle| \leq \mu$ implies $|\langle a_i, \nu\rangle| \leq \mu_0$ when $\mu_0 \geq \mu$. Since the dual objective does not depend on μ, we have that $g(\nu^*) \geq g(\nu_0^*)$, where $\nu_0^* = Ax_0^* - b$. Since we know ν_0^* exactly, we have a computable lower bound on $g(\nu^*)$:

$$g(\nu^*) = -\frac{1}{2}\|\nu^* + b\|^2 + \frac{1}{2}\|b\|^2 \geq g(\nu_0^*) = -\frac{1}{2}\|\nu_0^* + b\|^2 + \frac{1}{2}\|b\|^2.$$

If we define $\gamma_0 = g(\nu_0^*)$, this inequality reduces to $\|\nu^* + b\|^2 \leq \|b\|^2 - 2\gamma_0$. Thus, we define

$$N_1 \overset{\text{def}}{=} \left\{\nu \,:\, \|\nu + b\|^2 \leq \|b\|^2 - 2\gamma_0\right\}.$$

For N_2, we use a first-order characterization of the optimality of ν_0^*: ν_0^* is optimal iff ν_0^* is dual feasible and $\langle \nabla g(\nu_0^*), \nu_0 - \nu_0^*\rangle \leq 0$ for all dual feasible ν_0 (see Section 4.2.3 of [17], noting that $g(\nu)$ is concave). This states that $+\nabla g(\nu_0^*)$ defines a supporting hyperplane to the dual feasible set at ν_0^*. The dual optimal point ν^* for penalty parameter $\mu \leq \mu_0$ is dual feasible for penalty parameter μ_0, therefore it too must be supported by the hyperplane. Thus, we define

$$N_2 \overset{\text{def}}{=} \{\nu \,:\, \langle -\nu_0^* - b, \nu - \nu_0^*\rangle \leq 0\}.$$

With these definitions of $N = N_1 \cap N_2$, the upper bound subproblem (A.2) becomes

$$\begin{aligned} \max \quad & \langle a, \nu\rangle \\ \text{s.t.} \quad & \|\nu + b\|^2 \leq \|b\|^2 - 2\gamma_0 \\ & \langle -\nu_0^* - b, \nu - \nu_0^*\rangle \leq 0. \end{aligned} \tag{A.3}$$

The feasible set for this problem is the intersection of a ball and a halfspace. A closed-form solution to this "dome subproblem" is given in Appendix B.

We solve the upper bound subproblem (A.3) for $a = a_i$ and $a = -a_i$, for each column a_i of A. For each column where where the optimal value is strictly less than μ, we conclude that $x_i^* = 0$ and that a_i can be eliminated from the problem. We can then form and solve the reduced problem, giving us another exact solution. We may then repeat the feature elimination problem for a lower value of the penalty parameter. See [38] for applications of this iterative procedure.

A.2 NNLS Through Regularization

SAFE for LASSO uses two values of the penalty parameter μ to construct nested dual feasible sets. The dual feasible set for some penalty parameter μ_0 satisfying $\mu_0 \geq \mu$ is a superset of the dual feasible set for penalty μ. This fact is used in the construction of the search set $N = N_1 \cap N_2$.

To adapt SAFE to NNLS, we first derive a regularized least-squares problem that recovers NNLS for sufficiently strong regularization. Specifically, we consider the problem

$$\min_x \frac{1}{2}\|Ax - b\|^2 + \mu \sum_{i=1}^{n} \max\{0, -x_i\}, \tag{A.4}$$

with penalty parameter $\mu \geq 0$. In this subsection we show that for μ sufficiently large, the "max-regularized" least-squares problem recovers the NNLS solution (i.e., a solution of one problem is a solution of the other).

The non-differentiable penalty term

$$\max\{0, -x_i\} = \begin{cases} -x_i & x_i < 0 \\ 0 & x_i \geq 0 \end{cases}$$

serves to penalize negative x_i. The intuition behind this formulation is that the regularizer penalizes x_i that are negative, while not penalizing those x_i that are non-negative. When $\mu = 0$, the problem becomes ordinary least-squares. As μ increases from 0, the problem increasingly penalizes negative x_i. Note that $\|x\|_1 = \sum_{i=1}^{n} \max\{x_i, -x_i\}$. With this in mind, it is reasonable to expect (A.4) to have properties analogous to the ℓ_1-regularized least-squares problem (A.1).

Recall that we derived the primal KKT conditions for NNLS in Subsection 3.3.1. The KKT conditions apply to problems with differentiable objective and constraint functions, which is not the case for (A.4). We can still derive a first-order optimality condition, however.

Define $f(x) = \frac{1}{2}\|Ax - b\|^2$ and $h(x) = \sum_{i=1}^{n} \max\{0, -x_i\}$. Since $f(x) + \mu h(x)$ is a proper, convex function with full domain,

$$0 \in A^T(Ax^*_\mu - b) + \mu \partial h(x^*_\mu) \tag{A.5}$$

gives a first-order condition for optimality for x_μ^* to be a solution to (A.4) [10]. The subdifferential of $h(x)$ is given elementwise by

$$\{\partial h(x)\}_i \in \begin{cases} \{0\} & x_i > 0 \\ \{-\mu\} & x_i < 0 \\ [-\mu, 0] & x_i = 0. \end{cases}$$

We can identify conditions on μ under which a solution to NNLS is a solution to the regularized problem (A.4). These conditions are a special case of Theorem 3.72 of [10], for instance, and are simple enough that we prove them directly here.

Lemma 11: *Let x^* be a solution to the NNLS problem* (3.1) *and $\mu \geq \|A^T(Ax^* - b)\|_\infty$. Then x^* also solves the regularized problem* (A.4)*.*

Proof. From the NNLS first-order optimality condition (3.19), we know that $\lambda^* = A^T(Ax^* - b)$. For x^*, λ^* to satisfy the first-order optimality conditions (A.5) for the regularized problem (A.4), we must have

$$\lambda_i^* \in \begin{cases} \{0\} & x_i^* > 0 \\ \{-\mu\} & x_i^* < 0 \\ [-\mu, 0] & x_i^* = 0. \end{cases}$$

By complementary slackness (3.22), the first case is satisfied. The second case will not occur for optimal x_i^*, since $x^* \geq 0$. Again by complementary slackness, the third case is satisfied if $\mu \geq \lambda_i^*$, which occurs by assumption since $\lambda_i^* \leq \|\lambda^*\|_\infty = \|A^T(Ax^* - b)\|_\infty \leq \mu$. □

From this we have that it is necessary (we don't yet know if it is sufficient) that $\mu \geq \|A^T(Ax^* - b)\|_\infty$ for a solution to the max-regularized least-squares (A.4) to solve NNLS (3.1). Analogously to the previous lemma, we can determine conditions under which a solution x_μ^* of the regularized problem (A.4) is a solution to NNLS (3.1).

Lemma 12: *Let x_μ^* be an optimal point of the regularized problem* (A.4)*. If $x_\mu^* \geq 0$, then it is an optimal point of NNLS.*

Proof. Define $\lambda = A^T(Ax_\mu^* - b)$. From the first-order conditions (A.5) we know that

$$-\lambda_i \in \begin{cases} \{0\} & \{x_\mu^*\}_i > 0 \\ \{-\mu\} & \{x_\mu^*\}_i < 0 \\ [-\mu, 0] & \{x_\mu^*\}_i = 0. \end{cases}$$

Since $\{x_\mu^*\}_i \geq 0 \quad \forall i = 1, ..., n$, we see that complementary slackness (3.22) is satisfied and that λ is dual feasible. Therefore x_μ^* and λ satisfy the KKT conditions and, since strong duality holds for NNLS, are primal and dual optimal points, respectively. $\square$

The above lemma gives a condition under which we can determine if a given solution x_μ^* is an NNLS solution. We would prefer a condition on μ that implies the solution x_μ^* is an NNLS solution. Another "implicit" condition is given next.

Lemma 13: *Let x_μ^* be a solution to the regularized problem* (A.4)*. If it so happens that $\|A^T(Ax_\mu^* - b)\|_\infty < \mu$, then $x_\mu^* \geq 0$.*

Proof. This follows immediately from the first-order conditions (A.5). Note that x_μ^* also solves NNLS, by Lemma 12. $\square$

Note that Lemma 13 doesn't allow us to *a priori* determine μ such that $x_\mu^* \geq 0$. To make this lemma more useful, we seek an upper bound on $\|A^T(Ax_\mu^* - b)\|_\infty$ that is independent of x_μ^*. To do this, we first find a Lagrangian dual problem to the regularized least-squares problem (A.4). An equivalent primal problem is

$$\begin{aligned} \min \quad & \frac{1}{2}\|z\|^2 + \mu h(x) \\ \text{s.t.} \quad & z = Ax - b, \end{aligned}$$

which has Lagrangian

$$\mathcal{L}_\mu(x, z, \nu) = \frac{1}{2}\|z\|^2 + \mu h(x) + \langle \nu, Ax - b - z \rangle .$$

The dual objective is given by $g_\mu(\nu) = \inf_{x,z} \mathcal{L}_\mu(x, z, \nu)$, which occurs when

$$z = \nu, \quad \{-A^T\nu\}_i \in \mu\{\partial h(x)\}_i = \begin{cases} \{0\} & x_i > 0 \\ \{-\mu\} & x_i < 0 \\ [-\mu, 0] & x_i = 0. \end{cases}$$

From this we see that if any $\{A^T\nu\}_i < 0$ or $\{A^T\nu\}_i > \mu$, then $\mathcal{L}_\mu$ is not bounded below and $g_\mu(\nu) = -\infty$. We are therefore interested in the case when $0 \le A^T\nu \le \mu$. In this case, the dual objective is

$$\begin{aligned} g_\mu(\nu) &= \inf_x \mathcal{L}_\mu(x, \nu, \nu) \\ &= \inf_x \frac{1}{2}\|\nu\|^2 + \mu h(x) + \langle \nu, Ax - b - \nu\rangle \\ &= -\frac{1}{2}\|\nu\|^2 - \langle \nu, b\rangle + \inf_x \Big(\mu h(x) + \langle A^T\nu, x\rangle\Big) \end{aligned}$$

The term inside the infimum is

$$\mu h(x) = \langle A^T\nu, x\rangle = \sum_{i=1}^{n} \Big(\mu \max\{0, -x_i\} + \{A^T\nu\}_i x_i\Big).$$

If $0 \le A^T\nu \le \mu$, then $A^T\nu \in \mu\partial h(x)$ and the ith term in the sum is

$$\mu \max\{0, -x_i\} + \{A^T\nu\}_i x_i = \begin{cases} 0 + \cancelto{0}{\{A^T\nu\}_i x_i} & x_i > 0 \\ -\mu x_i + \cancelto{\mu}{\{A^T\nu\}_i x_i} & x_i < 0 \\ 0 + 0 & x_i = 0 \end{cases} = 0.$$

We therefore have the dual function

$$g_\mu(\nu) = \begin{cases} -\frac{1}{2}\|\nu\|^2 - \langle \nu, b\rangle & 0 \le A^T\nu \le \mu \\ -\infty & \text{otherwise.} \end{cases}$$

Finally, a Lagrangian dual to the regularized least-squares problem (A.4) is

$$\begin{aligned} \max \quad & -\frac{1}{2}\|\nu\|^2 - \langle \nu, b\rangle \\ \text{s.t.} \quad & 0 \le A^T\nu \le \mu. \end{aligned} \tag{A.6}$$

Note that this dual is very similar to the NNLS dual problem (3.18). Using this dual problem and strong duality, we can find an upper bound on $\|A^T(Ax_\mu^* - b)\|_\infty$. Our bound uses the $(2,\infty)$ operator norm from [94], which is defined as

$$\|A\|_{2,\infty} = \sup_x \frac{\|Ax\|_\infty}{\|x\|_2}.$$

Thankfully, $\|A\|_{2,\infty}$ is easily determined to be the maximum ℓ_2 norm of the rows of A.

Theorem 14: *Let x^* be a solution to NNLS and $\mu > \|A^T\|_{2,\infty}\|Ax^* - b\|$. Let x_μ^* solve the regularized problem* (A.4)*. Then x_μ^* also solves NNLS.*

Proof. It is sufficient to show that $\mu > \|A^T(Ax_\mu^* - b)\|_\infty$, because Lemmas 13 and 12 then imply that x_μ^* solves NNLS (3.1).

Note that strong duality holds for both NNLS and its dual (3.18), and the regularized problem (A.4) and its dual (A.6), since all problems are convex, feasible, and involve only affine constraints (if present) [17, 10]. Let x^*, ν^* be primal and dual optimal points for NNLS and its dual. Define $p^* = \frac{1}{2}\|Ax^* - b\|^2$ and $d^* = -\frac{1}{2}\|\nu^*\|^2 - \langle \nu^*, b\rangle$ to be the primal and dual optimal values. Since strong duality holds, $p^* = d^*$. Observe that $\|Ax^* - b\| = \sqrt{2p^*}$.

Let x_μ^* be a solution to the regularized problem (A.4) and define

$$p_\mu^* = \frac{1}{2}\|Ax_\mu^* - b\|^2 + \mu \sum_{i=1}^{n} \max\{0, -\{x_\mu^*\}_i\}$$

to be the primal optimal value. Observe that $\|Ax_\mu^* - b\| \leq \sqrt{2p_\mu^*}$. Since strong duality holds, $p_\mu^* = d_\mu^*$, where d_μ^* is the optimal value for the dual (A.6). Furthermore, the dual optimal value $d_\mu^* \leq d^*$, since the dual feasible set for the regularized problem is a subset of the dual feasible set for NNLS:

$$\left\{\nu \,:\, 0 \leq A^T\nu \leq \mu\right\} \subseteq \left\{\nu \,:\, 0 \leq A^T\nu\right\}.$$

Now let us begin assembling inequalities to arrive at the desired bound $\|A^T(Ax_\mu^* - b)\|_\infty < \mu$. Using the definition of the $(2,\infty)$ operator norm of A^T, we can bound

$$\|A^T(Ax_\mu^* - b)\|_\infty \leq \|A^T\|_{2,\infty}\|Ax_\mu^* - b\|,$$

Using a few of the previously mentioned relations, we have

$$\|Ax_\mu^* - b\| \leq \sqrt{2p_\mu^*} = \sqrt{2d_\mu^*} \leq \sqrt{2d^*} = \sqrt{2p^*} = \|Ax^* - b\|.$$

Chaining everything together, we have

$$\|A^T(Ax_\mu^* - b)\|_\infty \leq \|A^T\|_{2,\infty}\|Ax^* - b\|.$$

Since we assumed μ satisfies $\|A^T\|_{2,\infty}\|Ax^* - b\| < \mu$, we have $\|A^T(Ax_\mu^* - b)\|_\infty < \mu$ as desired. $\square$

Finally, we have a simple method to select a suitable μ that will ensure that x_μ^* also solves NNLS.

Corollary 15: *Given any $x \geq 0$, $\mu > \|A^T\|_{2,\infty}\|Ax - b\|$, a solution x_μ^* to the regularized problem* (A.4) *also solves the NNLS problem* (3.1)*.*

Proof. Let x^* be an optimal point of NNLS. Since $x \geq 0$ is primal feasible, but not necessarily optimal,

$$\|Ax - b\| \geq \|Ax^* - b\|.$$

This then implies that $\mu > \|A^T\|_{2,\infty}\|Ax^* - b\|$ and so Theorem 14 can be invoked. $\square$

A.3 SAFE for NNLS

We now have a regularized least-squares problem that recovers an NNLS solution when the penalty parameter μ is sufficiently large. This is analogous to LASSO, which has the zero solution for sufficiently large penalty parameter. SAFE for LASSO uses the fact that the dual feasible set for penalty parameter μ_0, with $\mu_0 \geq \mu$, is a superset of the dual feasible set for penalty μ. Let us now adapt SAFE to max-regularized least-squares (A.4). When the penalty parameter μ is sufficiently large (see Corollary 15), we will have SAFE for NNLS, instead of SAFE for max-regularized least-squares (A.4).

Define the dual feasible set for NNLS to be

$$\mathcal{N} \overset{\text{def}}{=} \left\{\nu \, : \, 0 \leq A^T\nu\right\},$$

and the dual feasible set for the max-regularized LS problem, parameterized by penalty parameter μ, to be

$$\mathcal{N}(\mu) \overset{\text{def}}{=} \left\{\nu \,:\, 0 \leq A^T\nu \leq \mu\right\}.$$

Let μ be sufficiently large so that Theorem 14 applies. Assume we have an **exact** primal-dual solution pair x_0^*, ν_0^* to the max-regularized problem (A.4) for some $\mu_0 \leq \mu$. We will use this solution at penalty μ_0 to eliminate features for the problem with penalty μ.

Just like in SAFE for NNLS as described in Section 3.3, we want to solve the lower bound subproblem

$$\begin{aligned} \min \quad & \langle a_i, \nu\rangle \\ \text{s.t.} \quad & \nu \in N, \end{aligned}$$

for some search set N that is guaranteed to contain the dual optimal point ν^*. We again form N as the intersection of two sets, $N = N_1 \cap N_2$.

For N_1, we follow SAFE for LASSO and use a relation between the dual optimal value at penalty parameters μ_0 and μ. Specifically, we have $\mu_0 \leq \mu$, where μ was chosen sufficiently large that a dual optimal point of the max-regularized least-squares problem is dual optimal for the NNLS dual problem (3.18). This implies that $\mathcal{N}(\mu_0) \subseteq \mathcal{N}(\mu)$. Therefore, we know $g(\nu_0^*) \leq g(\nu^*)$, where $g(\nu) = -\frac{1}{2}\|\nu\|^2 - \langle \nu, b\rangle$ is the dual objective. Defining $\gamma_0 = g(\nu_0^*)$, we define

$$N_1 \overset{\text{def}}{=} \{\nu \,:\, g(\nu) \geq g(\nu_0^*)\} = \left\{\nu \,:\, \|\nu + b\|^2 \leq \|b\|^2 - 2\gamma_0\right\}.$$

This is the same as in SAFE for LASSO (Section A.1), since the dual objectives are the same in both cases.

For N_2 we must deviate slightly from SAFE for LASSO. We again use the first-order characterization of optimality that ν_0^* is dual optimal iff ν_0^* is dual feasible and $\langle \nabla g(\nu_0^*), \nu_0 - \nu_0^*\rangle \leq 0$ for all dual feasible ν_0. If $\nabla g(\nu_0^*) \neq 0$, this can be understood to mean $\nabla g(\nu_0^*)$ defines a supporting hyperplane to the dual feasible set at ν_0^*.

In SAFE for LASSO, the dual feasible set for penalty μ_0 is a superset of the dual feasible set at penalty μ (since $\mu \leq \mu_0$). Thus the hyperplane defined by $\nabla g(\nu_0^*)$ at ν_0^* also supports the dual feasible set at penalty parameter μ. For NNLS however, we pick $\mu_0 \le \mu$ and have that $\mathcal{N}(\mu_0)$ is a subset of $\mathcal{N}(\mu)$. This means that the hyperplane defined by $\nabla g(\nu^*)$ at ν^* supports $\mathcal{N}(\mu_0)$. Writing this out, we have $\langle \nabla g(\nu^*), \nu_0 - \nu^* \rangle \le 0$ for all ν_0 in $\mathcal{N}(\mu_0)$. We use this condition to build a set that must contain ν^*:

$$\nu^* \in \{\nu \,:\, \langle \nabla g(\nu), \nu_0 - \nu \rangle \quad \forall \nu_0 \in \mathcal{N}(\mu_0)\}.$$

It is impractical to test all $\nu_0 \in \mathcal{N}(\mu_0)$, so we instead test just $\nu_0^* \in \mathcal{N}(\mu_0)$. Accordingly, we define the search set

$$N_2 \overset{\text{def}}{=} \{\nu \,:\, \langle -\nu - b, \nu_0^* - \nu \rangle \le 0\}\,.$$

The lower bound subproblem for λ_i^* is

$$\begin{aligned} \min \quad & \langle a_i, \nu \rangle \\ \text{s.t.} \quad & \|\nu + b\|^2 \le \|b\|^2 - 2\gamma_0 \\ & \langle \nabla g(\nu), \nu_0^* - \nu \rangle \le 0. \end{aligned}$$

In experiments with both `rand`/`randn` problems and for our PSF dictionaries, this test did not perform very well. We took μ such that Theorem 14 applies and $\mu_0 = \mu/10$. We solve the regularized problem at level μ_0 with CVX [43] to get a high-accuracy, but not exact, solution. Unfortunately, it was almost always the case for the 2D PSF dictionaries that the lower bound subproblem produced negative lower bounds, which does not allow us to eliminate many features.

Appendix B

Dome Subproblem - Closed-Form Solution

Here we follow [38] to derive a closed-form solution to the generic "dome subproblem"

$$\begin{aligned} \min_\nu \quad & \langle a, \nu \rangle \\ \text{s.t.} \quad & \|\nu - \hat{\nu}\|^2 \leq \delta \\ & \langle \nu, b \rangle \geq \gamma. \end{aligned} \tag{B.1}$$

Breaking our usual convention, we treat (B.1) as the primal problem and will find its optimal value via its dual. Here ν will be a primal point and λ_1, λ_2 will be dual variables associated with the two constraints.

We assume that $\delta > 0$, so the constraint $\|\nu - \hat{\nu}\|^2 \leq \delta$ can be satisfied strictly (e.g., by $\nu = \hat{\nu}$). The constraint $\langle \nu, b \rangle \geq \gamma$ is affine. We can therefore apply Slater's condition, which implies that strong duality holds [17]. Hence we will find the primal optimal value (e.g., the bound used for feature elimination) by computing the dual optimal value.

The Lagrangian for (B.1) is

$$\mathcal{L}(\nu, \lambda_1, \lambda_2) = \langle a, \nu \rangle + \lambda_1 \left(\|\nu - \hat{\nu}\|^2 - \delta \right) + \lambda_2 \left(\gamma - \langle \nu, b \rangle \right).$$

Assuming $\lambda_1 > 0$, $\inf_\nu \mathcal{L}$ is attained when

$$\nabla_\nu \mathcal{L} = a + 2\lambda_1(\nu - \hat{\nu}) - \lambda_2 b = 0.$$

which occurs when

$$\nu = \hat{\nu} + \frac{\lambda_2}{2\lambda_1} b - \frac{1}{2\lambda_1} a. \tag{B.2}$$

After simplifying, the dual objective is

$$g(\lambda_1, \lambda_2) = \inf_{\nu} \mathcal{L} = \langle a, \hat{\nu} \rangle - \lambda_1 \delta + \lambda_2 \gamma - \lambda_2 \langle \hat{\nu}, b \rangle - \frac{1}{4\lambda_1} \|\lambda_2 b - a\|^2 \qquad \text{if } \lambda_1 > 0.$$

If $\lambda_1 = 0$, we must have $a = \lambda_2 b$ to have a finite $\inf_\nu \mathcal{L}$. In this case, $\inf_\nu \mathcal{L} = \lambda_2 \gamma$. If $\lambda_1 = 0$ and $a \neq \lambda_2 b$, then $\inf_\nu \mathcal{L} = -\infty$. The dual objective is therefore

$$g(\lambda_1, \lambda_2) = \begin{cases} \langle a, \hat{\nu} \rangle - \lambda_1 \delta + \lambda_2 \gamma - \lambda_2 \langle \hat{\nu}, b \rangle - \frac{1}{4\lambda_1} \|\lambda_2 b - a\|^2 & \lambda_1 > 0 \\ \lambda_2 \gamma & \lambda_1 = 0, a = \lambda_2 b \\ -\infty & \lambda_1 = 0, a \neq \lambda_2 b. \end{cases}$$

Let us work with the branch $\lambda_1 > 0$. We have

$$\frac{\partial g}{\partial \lambda_2} = \gamma - \langle \hat{\nu}, b \rangle - \frac{1}{2\lambda_1} \langle \lambda_2 b - a, b \rangle.$$

Setting $\partial g / \partial \lambda_2 = 0$ and solving for $\lambda_2 \geq 0$, we find

$$\lambda_2 = \max \left\{ 0, \frac{\langle a, b \rangle}{\|b\|^2} + \frac{2\lambda_1}{\|b\|^2} \left(\gamma - \langle \hat{\nu}, b \rangle \right) \right\}.$$

If $\lambda_2 = 0$, then

$$g(\lambda_1, 0) = \langle a, \hat{\nu} \rangle - \lambda_1 \delta - \frac{1}{4\lambda_1} \|a\|^2$$

is minimized when $\partial g / \partial \lambda_1 \big|_{\lambda_2 = 0} = 0$, which occurs when $\lambda_1 = \|a\| / (2\sqrt{\delta})$. We use this value of λ_1 to check if $\lambda_2 = 0$: $\lambda_2 = 0$ if

$$\frac{\langle a, b \rangle}{\|b\|^2} + \frac{2}{\|b\|^2} \frac{\|a\|}{2\sqrt{\delta}} \left(\gamma - \langle \hat{\nu}, b \rangle \right) \leq 0$$

and $\lambda_2 > 0$ otherwise.

In the case that $\lambda_2 = 0$, it is simple to find the optimal value d^* and to compute ν^* via (B.2).

In the case that $\lambda_2 > 0$, we know from above that

$$\lambda_2 = \frac{\langle a, b \rangle}{\|b\|^2} + \frac{2\lambda_1}{\|b\|^2} \left(\gamma - \langle \hat{\nu}, b \rangle \right).$$

To find λ_1, we substitute the above expression for λ_2 into

$$\frac{\partial g}{\partial \lambda_1} = -\delta + \frac{1}{4\lambda_1^2} \|\lambda_2 b - a\|^2 = 0.$$

and solve for $\lambda_1 \geq 0$ After simplifying the resulting equation for λ_1, we define

$$\tau \overset{\text{def}}{=} \frac{\dfrac{\langle a, b\rangle^2}{\|b\|^2} - \|a\|^2}{\dfrac{4\left(\gamma - \langle \hat{\nu}, b\rangle\right)^2}{\|b\|^2} - 4\delta}.$$

If $\tau > 0$, we have $\lambda_1 = \sqrt{\tau} > 0$, and we again find ν^* via (B.2) and d^* via $g(\lambda_1, \lambda_2)$. If $\tau \leq 0$, we set $\lambda_1 = 0$ and $\lambda_2 = \langle a, b\rangle / \|b\|^2$. If $a = \lambda_2 b$, we can find an optimal ν^* (since we assumed $\langle \hat{\nu}, b\rangle > \gamma$) by projecting $\hat{\nu}$ onto the plane $\langle \nu, b\rangle = \gamma$:

$$\nu^* = \hat{\nu} - \frac{\langle \hat{\nu}, b\rangle - \gamma}{\|b\|^2} b.$$

The optimal value in this case is $d^* = \lambda_2 \gamma$. If $a \neq \lambda_2 b$, then the optimal value is $d^* = -\infty$.

We summarize the above solution in Algorithm 11.

In our implementation of Algorithm 11, we make a couple simplifications. First, if $\langle \hat{\nu}, b\rangle \leq \gamma$, we simply report the bound $d^* = -\infty$. The dome subproblem (B.1) may in actuality be feasible or infeasible, but we report $d^* = -\infty$ for convenience. Second, if $\tau \leq 0$, we again report $d^* = -\infty$ instead of checking if $a = \lambda_2 b$. These simplifications are merely for convenience, and they still result in correct bounds on the optimal value (i.e., they are safe to use with feature elimination).

Algorithm 11 Solution to Dome Subproblem (B.1)

```
Given $0 \neq a \in \mathbb{R}^m$, $0 \neq b \in \mathbb{R}^m$, $\hat{\nu} \in \mathbb{R}^m$, $\delta > 0$, $\gamma \in \mathbb{R}$ such that $\langle \hat{\nu}, b \rangle > \gamma$.
if $\frac{\langle a, b \rangle}{\|b\|^2} + \frac{2}{\|b\|^2} \frac{\|a\|}{2\sqrt{\delta}} (\gamma - \langle \hat{\nu}, b \rangle) > 0$ then
    $\tau = \left( \frac{\langle a, b \rangle^2}{\|b\|^2} - \|a\|^2 \right) \Big/ \left( \frac{4\left(\gamma - \langle \hat{\nu}, b \rangle\right)^2}{\|b\|^2} - 4\delta \right)$.
    if $\tau \leq 0$ then
        $\lambda_1^* = 0$.
        $\lambda_2^* = \frac{\langle a, b \rangle}{\|b\|^2}$.
        if $a = \lambda_2 b$ then
            $\nu^* = \hat{\nu} - \frac{\langle \hat{\nu}, b \rangle - \gamma}{\|b\|^2} b$        ▷ Projection of $\hat{\nu}$ onto the plane $\langle \nu, b \rangle = \gamma$
            $d^* = \frac{\langle a, b \rangle}{\|b\|^2} \gamma$.
        else
            $d^* = -\infty$.
        end if
    else
        $\lambda_1^* = \sqrt{\tau}$
        $\lambda_2^* = \frac{\langle a, b \rangle}{\|b\|^2} + \frac{2\lambda_1^*}{\|b\|^2} \left(\gamma - \langle \hat{\nu}, b \rangle\right)$.
        $\nu^* = \hat{\nu} + \frac{\lambda_2^*}{2\lambda_1^*} b - \frac{1}{2\lambda_1^*} a$.
        $d^* = \langle a, \hat{\nu} \rangle - \lambda_1^* \delta + \lambda_2^* \gamma - \lambda_2^* \langle \hat{\nu}, b \rangle - \frac{1}{4\lambda_1^*} \left\| \lambda_2^* b - a \right\|^2$.
    end if
else
    $\lambda_1^* = \frac{\|a\|}{2\sqrt{\delta}}$.
    $\lambda_2^* = 0$.
    $\nu^* = \hat{\nu} - \sqrt{\delta} \frac{a}{\|a\|}$.
    $d^* = \langle a, \hat{\nu} \rangle - \|a\| \sqrt{\delta}$.

end if
```

Appendix C

Generating NNLS Problems With a Known Solution

We discuss here how to generate RHSs for NNLS problems given a matrix A and desired exact solution x^*. The methods are inspired by [58, 36].

Strictly Overdetermined Problems

We consider first $A \in \mathbb{R}^{m\times n}$ that is strictly overdetermined (i.e., where $m > n$). The KKT conditions for NNLS are

$$\begin{aligned} A^T(Ax^* - b) - \lambda^* &= 0 \\ x^* &\geq 0 \\ \lambda^* &\geq 0 \\ x_i^*\lambda_i^* &= 0, \quad \forall i = 1, ..., n. \end{aligned}$$

We can generate an NNLS problem given A and x^* by forming $b = Ax^*$. However, this RHS doesn't have any noise, which is likely undesirable. From the KKT conditions, we see that if we form $\hat{b} = b + y$ where $A^T y = 0$, the new NNLS problem $(A, \hat{b})$ has x^* as a solution.

For full-rank, strictly-overdetermined problems, $\ker(A^T)$ is non-trivial, so it is sufficient to select any $y \in \ker(A^T)$ with the desired noise level. We can generate such a y with ℓ_2 noise level `noise` with the following MATLAB code.

Listing C.1: Strictly-Overdetermined NNLS RHS Generation

```
[Q,R] = qr(A,0);
y = randn(m,1);
y = y - Q*(Q'*y);
y = noise*y/norm(y);
b_hat = A*x + y;
```

One downside to this approach is that $\lambda^* = A^T(Ax^* - b) = 0$, so complementary slackness is not satisfied strictly.

Square and Underdetermined Problems

The above approach does not work for square or underdetermined full-rank problems, as $\ker(A^T)$ is trivial. We again desire to form $\hat{b} = b + y$ without changing x^* an optimal solution. The first-order conditions with RHS $\hat{b}$ are $A^T(Ax^* - b) - A^T y - \lambda^* = 0$. We therefore must find a y such that $A^T y$ results in a valid $\hat{\lambda}^* = \lambda^* + A^T y$, which means it must be non-negative and satisfy complementary-slackness.

We therefore want to find a vector $v \in \operatorname{ran}(A^T) \cap \mathcal{I}$, where $\mathcal{I}$ is defined via

$$v \in \mathcal{I} \Leftrightarrow \begin{cases} v_i = 0 & x_i > 0 \\ v_i \leq 0 & x_i = 0. \end{cases}$$

To do this, we can use projection onto convex sets (POCS) [19, 47]. The idea is to alternatingly project onto $\operatorname{ran}(A^T)$ and $\mathcal{I}$. To project onto $\operatorname{ran}(A^T)$, we form $A^T = QR$ and compute $w = Q(Q^T v)$. To project onto $\mathcal{I}$, we set $v_i = 0$ if $x_i = 0$ or $w_i > 0$, and $v_i = w_i$ otherwise. Note that this can be quite slow to converge, so [58] proposes a QP as an alternative formulation (for basis pursuit denoising problems).

Once we have $v \in \operatorname{ran}(A^T) \cap \mathcal{I}$, we find y by solving $A^T y = v$. Note that the overdetermined system $A^T y = v$ does have an exact solution, as v is in the range of A^T. Listing C.2 shows a simple implementation of this procedure.

Listing C.2: Underdetermined NNLS RHS Generation

```
[Q,R] = qr(A',0);
v = randn(n,1);
w = zeros(n,1);
atol = 1e-14;

% Find v in ran(A') \cap I with POCS
while true
    w_old = w;

    % Project onto ran(A')
    w = Q*(Q'*v);

    % Project onto inequality set
    v = w .* (x == 0);
    v(v>0) = 0;

    % Check for convergence
    adiff = max(norm(v - w_old), norm(v - w));
    if adiff < atol
        break
    end
end

% Find y s.t. v = A'*y
y = A'\v;
b_hat = A*x + y;
```