ARL-TR-7610 • Mar 2016

US Army Research Laboratory

Parallel Nonnegative Least Squares Solvers for Model Order Reduction

by James P Collins

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-7610 ● MAR 2016

US Army Research Laboratory

Parallel Nonnegative Least Squares Solvers for Model Order Reduction

by James P Collins
Computer and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
March 2016	Final	August 2014–September 2015

4. TITLE AND SUBTITLE
Parallel Nonnegative Least Squares Solvers for Model Order Reduction

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
James P Collins

5d. PROJECT NUMBER
AHPCRC

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-CIH-C
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-7610

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
primary author's email: <james.p.collins106.civ@mail.mil>.

14. ABSTRACT
Parallel nonnegative least squares (NNLS) solvers are developed specifically for NNLS problems that arise when the Energy Conserving Sampling and Weighting hyper-reduction procedure is used when constructing a reduced-order model (ROM) from high-fidelity, finite element calculations. With this approach, nonzero entries in the NNLS solution define a subset of the finite element mesh where the nonlinear terms need to be evaluated when integrating the ROM, and their values are weighting factors for each element; hence, a sparse solution is sought to reduce the computational work of integrating the ROM. The NNLS solution does not have to be optimal but it must satisfy some criteria associated with the accuracy of the ROM. The goal, therefore, is to produce approximate NNLS solutions that are sparse but meet some tolerance. Two algorithms are considered and scalable codes are developed: one uses the Lawson and Hanson active set method and the other the projected quasi-Newton iterative method. The latter does not inherently produce sparse solutions but can be modified to promote this. Both codes are parallelized using ScaLAPACK and performance results are presented.

15. SUBJECT TERMS nonnegative least squares, model order reduction, hyper-reduction, Energy Conserving Sampling and Weighting, Lawson and Hanson, projected quasi-Newton, ScaLAPACK.

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			James P Collins
Unclassified	Unclassified	Unclassified	UU	42	19b. TELEPHONE NUMBER (Include area code) 410-278-5061

Contents

List of Figures

List of Tables

Executive Summary

Imposing nonnegative constraints on least squares problems arises naturally in many fields, usually to avoid nonphysical solutions, such as negative chemical concentrations in chemical compositions or negative pixel intensities in computer graphics. The motivation for this work comes from a particular method for constructing nonlinear reduced-order models (ROMs) where nonnegative constraints on associated least squares problems are imposed to maintain the stability of the ROM. This methodology was developed for finite element models at the Army High Performance Computing Research Center.[1] The nonnegative least squares (NNLS) problem comes from the embedded hyper-reduction step referred to as Energy Conserving Sampling and Weighting (ECSW). ECSW reduces the complexity of a ROM by choosing a minimal subset of the finite element mesh, referred to as a reduced mesh, on which to evaluate the nonlinear function when integrating the ROM. Choosing this mesh requires the solution of a NNLS problem where the sparsity of the solution is associated with the reduced mesh. Each nonzero entry in the solution vector defines the reduced mesh and is a weighting factor for evaluating the nonlinear function at that mesh point; hence, the overall complexity of the ROM is tied to the sparsity of the NNLS solution. For this case the NNLS solution does not have to be optimal, but it must satisfy some tolerance associated with the accuracy of the ROM.

Solving large NNLS problems is computationally intensive and on a single processor can sometimes take days. In many cases, as with problems that arise from ECSW, they are too big to fit on a single node so a parallel implementation is necessary. As will be shown, ECSW NNLS problems are not solved to completion but to a tolerance to achieve the goal of a minimal, reduced mesh with acceptable accuracy. Thus, a stopping criteria needs to be imposed, and the method must produce a sparse solution. Unfortunately, there are no readily available parallel NNLS software packages that do this.

There are a number of algorithms that solve NNLS problems.[2] In this work, 2 algo-

[1] Farhat C, Avery P, Chapman T and Cortiel J. Dimensional reduction of nonlinear finite element dynamic models with finite rotations and energy-based mesh sampling and weighting for computational efficiency. Int J Numer Meth Engng. 2014;98:625–662.

[2] Chen D, Plemmons R. Nonnegativity constraints in numerical analysis. In proceedings of the 2007 Symposium on the Birth of Numerical Analysis; World Scientific Publishinc Co., 2010, 109–140.

rithms are implemented and parallelized using ScaLAPACK. They are the Lawson and Hanson active set method and the projected quasi-Newton iterative method. The projected quasi-Newton method does not, in general, produce a sparse solution, so modifications to the basic method are made to promote this. These methods and their implementations are discussed in Section 3. Performance results for both are presented in Section 4. Both methods were integrated into the "aeros" finite element structures code developed at the Army High Performance Computing Research Center at Stanford University by the Farhat Research Group. This code is equipped with model order reduction capabilities and ECSW hyper-reduction. Results using the 2 methods when constructing a reduced-order model of a general V-Hull vehicle subjected to an under-body blast are presented. For technical details of the code and its model order reduction capabilities, see the works of Farhat et al.[1]

INTENTIONALLY LEFT BLANK.

1. Introduction

Nonlinear model order reduction methods are being developed at the Army High Performance Computing Research Center (AHPCRC) at Stanford University by the Farhat Research Group for the purpose of enabling parametric studies of complex physical processes, such as under-body blasts, in a reasonable amount of time. Because of nonlinearity, the complexity of the reduced-order model (ROM) remains on the order of the high-dimensional model since evaluation of the nonlinear function cannot be computed off-line during construction of the ROM; thus, a second-tier approximation, known in the literature as hyper-reduction, is necessary for fast evaluation of the nonlinear function when integrating the ROM. One such hyper-reduction approach is the Energy Conserving Sampling and Weighting (ECSW) method developed at the AHPCRC. It is designed specifically for finite element models and derived from an energy conservation argument that seeks to maintain the total energy in the reduced model as in the high-dimensional model. This procedure requires a sparse, approximate solution to a nonnegative least squares (NNLS) problem, where the sparsity represents the element set used to evaluate the nonlinear terms, and the values are the weights applied to these elements. The sparsity reduces the overall complexity of the nonlinear reduced-order model, thus allowing parametric studies to be done in a reasonable amount of time.

NNLS problems that emerge from the ECSW hyper-reduction procedure are generally large, ill-conditioned matrices whose column dimension is equal to the number of elements in the finite element model and whose row dimension is the number of basis vectors in the reduced-order model times the number of snapshots used to produce the model. By way of construction, $\mathbf{x} = \mathbf{1}$ is always a dense solution to these systems; however, the objective is to find a sparse, approximate solution that is sufficiently accurate. The solution to such a problem on a single processor can take days and so the objective of this work is to develop scalable solution techniques that are compatible with the ECSW procedure and produce solutions quickly.

Two NNLS solution methods are considered: the Lawson and Hanson active set method and a projected quasi-Newton (PQN) iterative method. Active set methods are a natural choice for use with the ECSW hyper-reduction procedure since they typically remove one or more variables from the active set (i.e., the set of optimization variables that are constrained) so that a sparse, approximate solution can be

achieved with a simple stopping criterion. Iterative methods on the other hand, typically do not do this. The PQN method presented here is modified to control the size of the active set, thereby making it appropriate for hyper-reduction. Both methods are implemented and parallelized using ScaLAPACK.

The general NNLS problem and some details about ECSW are presented in Section 2. Section 3 discusses the Lawson and Hanson (LH) and PQN algorithms and their parallel implementation. Results and conclusions are presented in Sections 4 and 5, respectively.

2. Nonnegative Least Squares

The NNLS problem[1] is a constrained least squares problem where all components of the solution vector must be greater than or equal to zero. Given a matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$ and a right-hand side $\mathbf{b} \in \mathbb{R}^{M}$, the problem is to find an $\mathbf{x} \in \mathbb{R}^{N}$ that satisfies

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) = \frac{1}{2} \left\| \mathbf{A}\mathbf{x} - \mathbf{b} \right\|_2^2 \\ \text{subject to} \quad & \mathbf{x} \geq 0. \end{aligned} \tag{1}$$

It is well known that the solution $\mathbf{x}$ must satisfy the Karush-Kuhn-Tucker (KKT) optimality conditions given by

$$\begin{aligned} \mathbf{x} &\geq 0 \\ \nabla f(\mathbf{x}) &\geq 0 \\ \nabla f(\mathbf{x})^T \mathbf{x} &= 0. \end{aligned} \tag{2}$$

(For proof of the KKT conditions, see Nocedal and Wright,[2] chapter 12.) Associated with the solution, and indeed any feasible vector $\mathbf{x}$, is a set of indices where $x_i = 0$. This is referred to as the active set.[2] If the active set at the optimal solution is known a priori, then solving Eq. 1 is equivalent to solving an equality constrained least squares problem.

When the matrix $\mathbf{A}$ comes from the ECSW hyper-reduction procedure, it consists of the unassembled elemental contributions to the reduced vector of forces to be approximated, and the right-hand side $\mathbf{b}$ is the assembled reduced vector of forces,

see Farhat et al.[3] Each component of $\mathbf{b}$ is then the sum over the columns of $\mathbf{A}$ for the associated row. That is,

$$b_i = \sum_{j=1}^{N} a_{ij}. \tag{3}$$

Problems constructed in this way obviously have a dense solution given by $\mathbf{x} = \mathbf{1}$; however, the ECSW objective is to find a sparse solution $\mathbf{x}$ that is accurate enough for approximating the nonlinear functions in the reduced model. The sparsity represents the element set with which to evaluate the nonlinear terms and the values the weights applied to these elements. As discussed in Farhat et al.,[3] the accuracy of the desired solution is controlled by a parameter τ that controls the residual. The corresponding approximate problem is

$$\begin{aligned} &\min_{\mathbf{x}\in\Phi} \quad f(\mathbf{x}) = \frac{1}{2}\left\|\mathbf{A}\mathbf{x}-\mathbf{b}\right\|_2^2 \\ &\Phi = \{\mathbf{x}\in\mathbb{R}^N \mid \ \left\|\mathbf{A}\mathbf{x}-\mathbf{b}\right\|_2 \leq \tau \left\|\mathbf{b}\right\|_2, \ \ \mathbf{x}\geq 0\}. \end{aligned} \tag{4}$$

The ECSW matrix generally has a large condition number making it difficult to solve. As a result, both methods are implemented with an option to scale the matrix columns so that the $\mathcal{L}^2$ norm of each column is 1.

3. Methods and Implementations

This section describes the LH and the PQN algorithms and their parallel implementations. The implementations are not limited to ECSW hyper-reduction problems; they are both capable of solving the usual NNLS problem Eq. 1. A simple stopping criterion, based on a user-supplied input parameter, is added to both methods to promote sparse solutions. This is sufficient for the LH method but not for the PQN method. For the latter method the size of the active set is controlled to promote sparse solutions. This is described in Section 3.2.1.

3.1 Lawson and Hanson

The LH algorithm is an active set method that solves Eq. 1 in a finite number of iterations by constructing a sequence of active sets and associated feasible vectors, which converge to the optimal solution. This iterative procedure consists of 2 loops: an outer iteration loop and a sub-iteration loop referred to as the downdate procedure. Each outer iteration decreases the current active set by one and checks its validity by solving an equality constrained optimization problem. If the solution is

feasible, then the new active set is valid and the next outer loop iteration begins. If not, the downdate procedure is invoked where selected optimization variables are added back to the active set, and an equality constrained problem associated with the new, larger active set, is solved. If this produces a feasible vector, the next outer loop iteration begins, otherwise, the downdate procedure is repeated until the solution becomes feasible again. The method terminates when the active set cannot be decreased without producing an infeasible solution, or by some other criteria, such as the tolerance on the residual.

The implementation is best described by reformulating the optimality conditions Eq. 2 in terms of the active set as

$$\begin{aligned} &\mathbf{w} = -\nabla f(\mathbf{x}) = \mathbf{A}^T (\mathbf{b} - \mathbf{A}\mathbf{x}) \\ &x_i = 0 \text{ for } i \in \boldsymbol{\mathcal{Z}}, \quad x_i > 0 \text{ for } i \in \boldsymbol{\mathcal{P}} \\ &w_i \leq 0 \text{ for } i \in \boldsymbol{\mathcal{Z}}, \quad w_i = 0 \text{ for } i \in \boldsymbol{\mathcal{P}}, \end{aligned} \tag{5}$$

where the set $[1, 2, \ldots, N]$ is partitioned into 2 sets

$$\begin{aligned} \boldsymbol{\mathcal{P}} &= \{i \mid x_i > 0\} \\ \boldsymbol{\mathcal{Z}} &= \{i \mid x_i = 0\} = \{1, 2, \ldots, N\} \setminus \boldsymbol{\mathcal{P}}. \end{aligned} \tag{6}$$

The active set is $\boldsymbol{\mathcal{Z}}$ and is the complement of $\boldsymbol{\mathcal{P}}$. If $\mathbf{x},\ \mathbf{w} \in \mathbb{R}^N$ satisfy Eq. 5 then $\mathbf{x}$ satisfies Eq. 2.

The algorithm produces a sequence of active sets and solve the associated equality constrained minimization problem

$$\begin{aligned} \min_{\mathbf{x}} \ & f(\mathbf{x}) = \tfrac{1}{2} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 \\ \text{subject to } \ & x_i = 0,\ i \in \boldsymbol{\mathcal{Z}} \end{aligned} \tag{7}$$

for each set. The solution to Eq. 7 is obtained by partitioning $\mathbf{A}$ and $\mathbf{x}$ according to $\boldsymbol{\mathcal{P}}$ and solving the unconstrained least squares problem for the portion of $\mathbf{A}$ and $\mathbf{x}$ associated with $\boldsymbol{\mathcal{P}}$. If $\boldsymbol{\P}_p \in \mathbb{R}^{N \times N}$ is a permutation matrix that permutes the p columns of $\mathbf{A}$ whose indices are in set $\boldsymbol{\mathcal{P}}$ to the first p columns of the matrix, then

the desired partition is given by

$$\mathbf{A}\,\P_p = [\,\mathbf{A}_p\;\mathbf{A}_z\,], \quad \P_p^T\,\mathbf{x} = \mathbf{y} = \begin{bmatrix} \mathbf{y}_p \\ \mathbf{y}_z \end{bmatrix}. \tag{8}$$

There is no required ordering of the columns of $\mathbf{A}_p$ so $\P_p$ is not unique. The permutation matrix defines an ordering on the set $\mathcal{P}$ denoted by $\widehat{\mathcal{P}} = (\mathcal{P}, \P_p)$.

With this partition problem Eq. 7 is equivalent to the unconstrained least squares problem

$$\min_{\mathbf{y}_p} \quad \tfrac{1}{2}\,\|\mathbf{A}_p\mathbf{y}_p - \mathbf{b}\|_2^2 \tag{9}$$

with $\mathbf{y}_p \in \mathbb{R}^p$. It is solved using a $\mathbf{QR}$ factorization of $\mathbf{A}_p$ written as

$$\mathbf{A}_p = \mathbf{Q}_p \begin{bmatrix} \mathbf{R}_p \\ \mathbf{0} \end{bmatrix}, \quad \mathbf{Q}_p = \mathbf{H}_1\,\mathbf{H}_2 \ldots\,\mathbf{H}_p, \tag{10}$$

where $\mathbf{Q}_p$ is a composition of elementary reflectors $\mathbf{H}_i$. If

$$\mathbf{Q}_p^T\mathbf{b} = \mathbf{g} = \begin{bmatrix} \mathbf{g}_p \\ \mathbf{g}_z \end{bmatrix}, \tag{11}$$

then the solution to Eq. 9 is

$$\mathbf{y}_p = \mathbf{R}_p^{-1}\,\mathbf{g}_p. \tag{12}$$

The solution exists if $\mathbf{A}_p$ is linearly independent. The solution to Eq. 7 is then

$$\mathbf{x} = \P_p \begin{bmatrix} \mathbf{y}_p \\ \mathbf{y}_z = \mathbf{0} \end{bmatrix} \tag{13}$$

and the residual $\mathbf{r}$ is

$$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{y} = \mathbf{Q}_p \begin{bmatrix} \mathbf{0}_p \\ \mathbf{g}_z \end{bmatrix}. \tag{14}$$

Algorithm 1 presents the ScaLAPACK implementation where the main parts are the outer iteration loop starting at line 4, the downdate procedure starting at line 13, and the $\mathbf{QR}$ update procedure starting at line 27. Some comments on the right of each line contain the equation number for reference back to the text and, if appropriate, the ScaLAPACK routine used at that point. Inputs are the matrix $\mathbf{A}$, the right-hand

side $\mathbf{b}$, optional stopping criteria τ the controls the residual and p^{max} that limits the size of set $\boldsymbol{\mathcal{P}}$, a flag for scaling the matrix, and various ScaLAPACK parameters that control the processor grids and block sizes (not shown). Parameter τ stops the calculation when the $\mathcal{L}^2$ norm of the residual drops below $\tau \left\|\mathbf{b}\right\|_2$ and p^{max} stops the calculation if the size of set $\boldsymbol{\mathcal{P}}$ reaches p^{max}. If scaling is selected, then the matrix $\mathbf{A}$ is postmultiplied by a diagonal scaling matrix Λ_s whose diagonal entries are the inverse of the $\mathcal{L}^2$ norm of each column vector. The Output is the solution vector $\mathbf{x}$.

After initialization, the outer iteration loop is entered where $\mathbf{r}$, $\mathbf{w}$, and $w_{i_{max}}$, the maximum component of $\mathbf{w}$ over indices in set $\boldsymbol{\mathcal{Z}}$, are computed (lines 5, 6 and 7). If $w_{i_{max}} > 0$, then there is a potential for finding a new feasible vector that reduces the objective function $f(\mathbf{x})$ so index i_{max}, the most likely candidate for removing from the active set, is moved to $\boldsymbol{\mathcal{P}}$. If $w_{i_{max}} \leq 0$ then the calculation is complete. Equation 9 is then solved for $\mathbf{y}_p$, and if the minimum component of $\mathbf{y}_p$ is strictly greater then zero, then $\mathbf{y} = [\mathbf{y}_p, \mathbf{0}]^T$ is feasible and the current active set is valid.

If, on the other hand, any component of $\mathbf{y}_p$ is less than or equal to zero, the current active set is not valid and the downdate procedure is invoked. First, the previous solution $\mathbf{y}$ is updated using the new but infeasible $\mathbf{y}_p$ as follows:

$$\begin{aligned} &\mathbf{y} = \mathbf{y} + \alpha\left(\bar{\mathbf{y}}_p - \mathbf{y}\right), \quad \bar{\mathbf{y}}_p = \left[\mathbf{y}_p, \mathbf{0}_{(N-p)}\right]^T \\ &\alpha = \min\left\{ \frac{y_i}{y_i - \bar{y}_{p_i}} \mid \bar{y}_{p_i} \leq 0, i \in \{1, 2, ..., p\} \right\}. \end{aligned} \tag{15}$$

This choice of α ensures $\mathbf{y}$ remains feasible and has at least one component equal to zero with an index in $\boldsymbol{\mathcal{P}}$. This component is moved from set $\boldsymbol{\mathcal{P}}$ to $\boldsymbol{\mathcal{Z}}$ along with any others that are zero. Equation 9 is solved again and the process repeats until a new active set is found that produces a feasible $\mathbf{y}_p$.

The $\mathbf{QR}$ factorization of $\mathbf{A}_p$ is required in both the outer iteration loop and the downdate procedure. The outer loop constructs $\mathbf{A}_p$ from $\mathbf{A}_{p-1}$ by adding a new column vector at index p and the downdate procedure removes one or more columns. Because it is stored compactly consistent with ScaLAPACK, the factorization can be incrementally updated by computing and applying new elementary reflectors where appropriate. If the only change is a new column vector at index p, then only $\mathbf{H}_p$ in Eq. 10 needs to be computed. If one or more column vectors are removed and q_{min} is the smallest index in the ordered set $\widehat{\boldsymbol{\mathcal{P}}}$ of the vectors that are to be

removed, then only elementary reflectors $\mathbf{H}_{q_{min}}, \mathbf{H}_{q_{min}+1}, ..., \mathbf{H}_p$ need to be recomputed. Maximizing q_{min}, therefore, reduces the work associated with the downdate procedure.

Updating the $\mathbf{QR}$ factorization is done in procedure UpdateQR(k). It starts by copying the columns of $\mathbf{A}$ associated with indices k through p of $\mathbf{A}_p$ to the appropriate sub-matrix of the ScaLAPACK matrix that holds the factorization represented by $[\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_p]$ in Algorithm 1. The ScaLAPACK routine pdormqr is called at line 32 to multiply the sub-matrix $[\mathbf{q}_k, \mathbf{q}_{k+1}, \ldots, \mathbf{q}_p]$ by $\mathbf{Q}_{k-1}$, followed by a call to pdgeqrf at line 33 to complete the factorization. The vector $\mathbf{g}$ is either incrementally updated at line 35 or recomputed from $\mathbf{b}$ at line 37. It is worth noting that the storage requirement for holding the factorization is $O(M \times M)$; however, if the optional stopping criteria p^{max} is specified, then the storage requirement is reduced to $O(M \times p^{max})$.

When the downdate procedure increases the active set, or equivalently removes column vectors from $\mathbf{A}_p$, the new ordering imposed on the columns, although not unique, could have an impact on performance through the value of q_{min} at each iteration. Two orderings were implemented. The first fills in the holes with column vectors that were added most recently, and the second shifts all columns to the left. The shift ordering performed better most likely because any column vector that is associated with the optimal active set tends to migrate to the lower column indices of matrix $\mathbf{A}_p$, hence maximizing q_{min} on subsequent iterations.

The processor grid used by ScaLAPACK to distribute the matrices and parallelize the computations can have a large influence on performance. Using the assumption that $\mathbf{A}_p \in \mathbf{R}^{M\,x\,p}$ where $p \ll M$, the most efficient processor grid for the $\mathbf{QR}$ factorization is $N_{procs} \times 1$, where N_{procs} is the number of processors. This assumption is true for the early part of any calculation when p is small and is also true for the optimal solution of many problems; however, it is not true in all cases. But, since the goal of ECSW hyper-reduction step is to produce a sufficiently optimal solution with $p \ll M$, the assumption is valid here.

For large matrices the most expensive part of the algorithm is computing $\mathbf{w}$. Because the optimal processor grid for the $\mathbf{QR}$ factorization is not necessarily the best for computing $\mathbf{w}$, 2 separate processor grids, or BLACS contexts, were implemented. One context is used for the $\mathbf{QR}$ factorization and all computations aligned

with it, while the other is used for computations aligned with the matrix $\mathbf{A}$. It was found that the overhead associated with multiple BLACS contexts generally exceeded the benefit, so, in general, the single processor grid $N_{procs} \times 1$ is used for both.

Not shown in Algorithm 1, but implemented in the code, is a check on linear independence of $\mathbf{A}_p$ as each new column vector is added. It closely follows the original implementation in Lawson and Hanson.[4] It is rarely executed so it was not included in Algorithm 1.

Algorithm 1 Parallel LH (PLH) NNLS Algorithm

Input:

$\mathbf{A} \in \mathbb{R}^{M \times N}$, $\mathbf{b} \in \mathbb{R}^{M}$, τ, p^{max}, $scale$

Output:

$\mathbf{x}$

Initialize: $\mathbf{x} = \mathbf{y} = \mathbf{0}$, $\widehat{\boldsymbol{\mathcal{P}}} = \varnothing$, $p = 0$, $\mathbf{g} = \mathbf{b}$, $\mathbf{Q}_0 = \mathbf{I}$
if $scale$ **then**
    $\mathbf{A} \leftarrow \mathbf{A}\,\Lambda_s$ ▷ pdnrm2
repeat
    $\mathbf{r} \leftarrow \mathbf{Q}_p \left[\mathbf{0}_p,\ \mathbf{g}_{(N-p)}\right]^T$ ▷ Eq. 14, pdormqr
    $\mathbf{w} \leftarrow -\nabla f(\mathbf{x}) = \mathbf{A}^T\,\mathbf{r}$ ▷ Eq. 5, pdgemv
    $w_{i_{max}} = \max\{w_i \mid i = 1, 2, \ldots N,\ i \notin \widehat{\boldsymbol{\mathcal{P}}}\}$
    if $w_{i_{max}} > 0$ **then**
        $p \leftarrow p + 1,\ \ \widehat{\boldsymbol{\mathcal{P}}}(p) = i_{max}$
        UpdateQR(p)
        Solve $\mathbf{R}_p\,\mathbf{y}_p = \mathbf{g}_p$ ▷ Eq. 12, pdtrsv
        $y_{min} = \min\{y_{p_i} \mid i = 1, 2, \ldots p\}$
        while $y_{min} \leq 0$ **do** ▷ Downdate
            $\alpha = \min\left\{\frac{y_i}{y_i - y_{p_i}} \mid y_{p_i} \leq 0, i \in \{1, 2, \ldots, p\}\right\}$
            $\mathbf{y} \leftarrow \mathbf{y} + \alpha(\bar{\mathbf{y}}_p - \mathbf{y}),$ where $\bar{\mathbf{y}}_p = \left[\mathbf{y}_p, \mathbf{0}_{(N-p)}\right]^T$
            $\boldsymbol{\mathcal{P}}_0 = \{\widehat{\boldsymbol{\mathcal{P}}}(i) \mid y_i = 0,\ i = 1, 2, \ldots, p\}$
            $q_{min} = \min i \in \boldsymbol{\mathcal{P}}_0$
            $\widehat{\boldsymbol{\mathcal{P}}} = \widehat{\boldsymbol{\mathcal{P}}} \setminus \boldsymbol{\mathcal{P}}_0,\ \ p \leftarrow \mid \widehat{\boldsymbol{\mathcal{P}}} \mid$
            UpdateQR(q_{min})
            Solve $\mathbf{R}_p\,\mathbf{y}_p = \mathbf{g}_p$ ▷ Eq. 12, pdtrsv
            $y_{min} = \min\{y_{p_i} \mid i = 1, 2, \ldots p\}$
        $\mathbf{x} \leftarrow \P_p \left[\mathbf{y}_p, \mathbf{0}_{(N-p)}\right]^T$
until $w_i \leq 0\ \forall i \in \boldsymbol{\mathcal{Z}}$ or $p = p^{max}$ or $\|\mathbf{r}\|_2 \leq \tau\,\|\mathbf{b}\|_2$
if $scale$ **then**
    $\mathbf{x} \leftarrow \Lambda_s \mathbf{x}$
Return $\mathbf{x}$

27: **procedure** UPDATEQR(k)
28: **for** $i = k \rightarrow p$ **do**
29: $j = \widehat{\boldsymbol{\mathcal{P}}}(i)$
30: $\mathbf{q}_i \leftarrow \mathbf{a}_j$
31: **if** $k > 1$ **then**
32: $[\mathbf{q}_k, \mathbf{q}_{k+1}, \ldots \mathbf{q}_p] \leftarrow \mathbf{H}_{k-1}, \mathbf{H}_{k-2}, \ldots \mathbf{H}_1 [\mathbf{q}_k, \mathbf{q}_{k+1}, \ldots \mathbf{q}_p]$ ▷ pdormqr
33: $[\mathbf{q}_k, \mathbf{q}_{k+1}, \ldots \mathbf{q}_p] \leftarrow \mathbf{QR}(k, [\mathbf{q}_k, \mathbf{q}_{k+1}, \ldots \mathbf{q}_p])$ ▷ pdgeqrf
34: **if** $k = p$ **then**
35: $\mathbf{g} \leftarrow \mathbf{H}_p \mathbf{g}$ ▷ pdormqr
36: **else**
37: $\mathbf{g} \leftarrow \mathbf{Q}_p \mathbf{b}$ ▷ pdormqr

3.2 Projected Quasi-Newton

The PQN algorithm is an iterative method, not an active set method, and as such does not lend itself to a threshold based termination criterion that encourages a sparse solution as required by the ECSW hyper-reduction procedure. A simple modification that controls the size of the active set makes it usable for ECSW and results in an algorithm that is referred to here as the Limited PQN (LPQN) method. The parallel implementation of the original method is described first followed by Section 3.2.1, which describes the LPQN modifications. The reader is referred to Kim et al.[5] for details about the algorithm and its convergence properties.

The PQN algorithm is an iterative procedure that produces a sequence of feasible vectors $\{\mathbf{x}^k\}$, $k = 0, 1, \cdots$, $\mathbf{x}^k \in \mathbb{R}^N$, that, under certain conditions, converge to the solution of Eq. 1. It is initialized with any feasible vector, usually $\mathbf{x}^0 = \mathbf{0}$. At each iteration the components of $\mathbf{x}^k$ are partitioned into sets of free and fixed variables where the fixed variables are constrained to zero. A NNLS problem is constructed for the free variables and the current iteration operates on it. The updated free variables combined with the fixed variables, produces the next iterate $\mathbf{x}^{k+1}$. The projection step guarantees feasibility. Once the optimal active set is reached, the method reduces to solving an unconstrained least squares problem for the free variables.

The implementation is described from the point-of-view of the k^{th} iteration where

$\mathbf{x}^k$ and $\mathbf{r}^k$ are known and iterate $\mathbf{x}^{k+1}$ and residual $\mathbf{r}^{k+1}$ are computed. The iteration starts by partitioning $\mathbf{x}^k$ into sets of fixed and free variables as,

$$\begin{aligned} \boldsymbol{\mathcal{Z}}_{pqn}^k &= \left\{ i \mid x_i^k = 0,\ \left[\nabla f(\mathbf{x}^k)\right]_i > 0 \right\} \\ \boldsymbol{\mathcal{P}}_{pqn}^k &= \{1, 2, \ldots, N\} \setminus \boldsymbol{\mathcal{Z}}_{pqn}, \end{aligned} \tag{16}$$

where subscript i is the component index. Because of the positivity constraint on the gradient, the fixed set $\boldsymbol{\mathcal{Z}}_{pqn}^k$ is a subset of the active set. The free set $\boldsymbol{\mathcal{P}}_{pqn}^k$ is the complement of $\boldsymbol{\mathcal{Z}}_{pqn}^k$. Only the free variables are updated during the iteration. The fixed variables, appropriately named, are not.

The partition induces a NNLS problem on the free variables. For convenience, the matrix is permuted so that the free variables are associated with a contiguous block of column vectors starting at column 1, so the NNLS problem description is given in terms of the permuted matrix $\mathbf{A}^{k+1}$ used to compute $\mathbf{x}^{k+1}$. If $\mathbf{P}^k \in \mathbb{R}^{N \times N}$ is a permutation matrix that permutes the $p_k = \mid \boldsymbol{\mathcal{P}}_{pqn}^k \mid$ columns of a matrix whose indices are in set $\boldsymbol{\mathcal{P}}_{pqn}^k$ to the first p_k columns, then the partitioned matrix can be written as

$$\mathbf{A}^{k+1} = \mathbf{A}\,\mathbf{P}^0\,\mathbf{P}^1 \ldots \mathbf{P}^k = \mathbf{A}^k\,\mathbf{P}^k = \left[\, \mathbf{A}_{p_k}^{k+1},\, \mathbf{A}_{z_k}^{k+1} \right], \tag{17}$$

where the subscripts p_k and z_k denote the partitioning based on $\boldsymbol{\mathcal{P}}_{pqn}^k$ and $\boldsymbol{\mathcal{Z}}_{pqn}^k$, respectively, and $\mathbf{A}^0 = \mathbf{A}$. The resulting problem in the free variables is

$$\underset{\mathbf{z} \in \mathbb{R}^{p_k},\, \mathbf{z} \geq \mathbf{0}}{\arg\min}\ \ g^{k+1}(\mathbf{z}) = \tfrac{1}{2} \left\| \mathbf{A}_{p_k}^{k+1} \mathbf{z} - \mathbf{b} \right\|_2^2. \tag{18}$$

The feasible vector $\mathbf{x}^k$ and the gradient $\nabla f\left(\mathbf{x}^k\right)$, which are aligned with the matrix $\mathbf{A}^k$ from the previous iteration, are permuted to align with the matrix $\mathbf{A}^{k+1}$ and are given by

$$\begin{aligned} \mathbf{y}^k = \mathbf{P}^{k^T} \mathbf{x}^k &= \begin{bmatrix} \mathbf{y}_{p_k}^k \\ \mathbf{y}_{z_k}^k \end{bmatrix}, \\ \nabla f\left(\mathbf{y}^k\right) = \mathbf{P}^{k^T} \nabla f\left(\mathbf{x}^k\right) &= \begin{bmatrix} \nabla f\left(\mathbf{y}^k\right)_{p_k} \\ \nabla f\left(\mathbf{y}^k\right)_{z_k} \end{bmatrix}. \end{aligned} \tag{19}$$

The notation is such that the partition of the gradient follows that of the argument

to f; therefore, $\nabla f\left(\mathbf{y}^k\right)$ is aligned with $\mathbf{y}^k$. The feasible vector at the end of the $k+1^{th}$ iteration is written

$$\mathbf{x}^{k+1} = \begin{bmatrix} \mathbf{x}_{p_k}^{k+1} \\ \mathbf{y}_{z_k}^k = \mathbf{0} \end{bmatrix}, \tag{20}$$

where $\mathbf{x}_{p_k}^{k+1}$ is an approximate solution to Eq. 18 given by the current PQN iteration.

The PQN algorithm, as presented by Kim et al.,[5] uses a line search method and a projection that guarantees feasibility. The iteration update is computed as

$$\mathbf{x}_{p_k}^{k+1} = \mathbf{y}_{p_k}^k + \alpha^k \, \mathbf{d}^k, \tag{21}$$

where the search direction, $\mathbf{d}^k$, is given by

$$\mathbf{d}^k = \gamma^k \left(\beta; \mathbf{y}_{p_k}^k\right) - \mathbf{y}_{p_k}^k \, . \tag{22}$$

Parameters α and β define the line search method. Projection γ^k is

$$\gamma^k \left(\beta; \mathbf{y}_{p_k}^k\right) = \mathbb{P}\left[\mathbf{y}_{p_k}^k - \beta \, \mathbf{S}_{p_k}^k \, \nabla g\left(\mathbf{y}_{p_k}^k\right)\right], \tag{23}$$

where $\mathbb{P}$ is the projection onto $\mathbb{R}_+^{p_k}$ and the matrix $\mathbf{S}_{p_k}^k$ is the principal submatrix of an approximation to the inverse of the Hessian, $\nabla^2 f(\mathbf{x}^k)^{-1}$ suitably permuted.

Two line search methods were discussed in Kim et al.[5] and both were implemented. The first is the limited minimization method that fixes β and computes α as the minimum of $g(\mathbf{y}_{p_k}^k + \alpha \, \mathbf{d}^k)$ over the variable α. The resulting line search parameters are

$$\alpha^k = \frac{{\mathbf{d}^k}^T {\mathbf{A}_{p_k}^{k+1}}^T \left[\mathbf{b} - \mathbf{A}_{p_k}^{k+1} \mathbf{y}_{p_k}^k\right]}{\left\| \mathbf{A}_{p_k}^{k+1} \mathbf{d}^k \right\|_2^2}, \qquad \beta = \text{ fixed constant.} \tag{24}$$

The convergence proof in Kim et al.[5] requires β to be sufficiently small. The implementation fixes $\beta = 1$ but reduces it if $\mathbf{d}^k$ is not a descent direction.

The second line search method used is Armijo along projection arc. This method fixes $\alpha = 1$ and computes β as

$$\beta = s^m \sigma, \tag{25}$$

where parameters $s \in (0,1)$ and $\sigma > 0$ are fixed and m is the smallest nonnegative integer for which

$$g^k\left(\mathbf{y}^k_{p_k}\right) - g\left(\gamma^k\left(s^m\sigma;\mathbf{y}^k_{p_k}\right)\right) \geq \eta\nabla g^{k+1}\left(\mathbf{y}^k_{p_k}\right)^T\left(\mathbf{y}^k_{p_k} - \gamma^k\left(s^m\sigma;\mathbf{y}^k_{p_k}\right)\right). \tag{26}$$

Parameter $\eta \in \left(0, \frac{1}{2}\right)$ is also fixed.

The Broyden-Fletcher-Goldfarb-Shanno (BFGS) method of approximating the inverse Hessian $\mathbf{S}^k_{p_k}$ requires storing a dense matrix of size N^2, which is prohibitively large so the limited memory BFGS, or L-BFGS, as given in Nocedal and Wright's[2] work, is used instead. L-BFGS is a recursion algorithm that approximates the product $\mathbf{S}^k_{p_k}\ \nabla g\left(\mathbf{y}^k_{p_k}\right)$ and is given in Algorithm 3. It computes the approximation to the inverse Hessian using a fixed number, K, of difference quantities for both the feasible vector and the gradient of the objective function. Since, in general, a permutation is applied at each iteration, these difference quantities are also be permuted and are defined as

$$\begin{aligned}\mathbf{u}^j &= \mathbf{P}^{k^T}\mathbf{P}^{k^T}\ldots\mathbf{P}^{j+1^T}\left[\mathbf{x}^{j+1} - \mathbf{y}^j\right]\\ \mathbf{w}^j &= \mathbf{P}^{k^T}\mathbf{P}^{k^T}\ldots\mathbf{P}^{j+1^T}\left[\nabla f(\mathbf{x}^{j+1}) - \nabla f(\mathbf{y}^j)\right],\end{aligned} \tag{27}$$

where k is the current iteration counter and $j \in [0,1,\ldots,K-1]$ is the index of the saved difference quantities. These vectors are saved in a round-robin fashion with index $j = l$ representing the latest vector stored.

The PQN implementation with the limited minimization line search method is given in Algorithm 2. Inputs are the matrix $\mathbf{A}$, right-hand side $\mathbf{b}$, a flag to scale matrix by the diagonal matrix Λ_s, which scales $\mathbf{A}$ so that the $\mathcal{L}^2$ norm of each column is 1, and stopping criteria τ, which stops the calculation when $\left\|\mathbf{r}^{k+1}\right\|_2 \leq \tau\left\|\mathbf{r}^0\right\|_2$, and z_{tol}, which stops the calculation when the active set has been found and norm of the gradient associated with the free variables drops below z_{tol}.

Steps 5–7 compute the gradient, and if at least one iteration is performed stores $\Delta\nabla f$. Steps 8–13 construct the set $\boldsymbol{\mathcal{P}}^k_{pqn}$ and apply the associated permutation matrix. Step 14 computes the approximation to the inverse Hessian times the gradient using the L-BFGS method. Steps 15–22 are the PQN iteration using limited minimization with $\beta = 1$. This choice of β does not always produce a descent direction, so steps 17–19 are executed until it does. The round-robin counter l is updated in

step 23 and $\Delta \mathbf{x}$ is saved step 24. The iteration is complete after computing the residual at step 25. When the algorithm terminates, the solution is permuted back to its original order at step 28. Algorithm 3 was taken directly from Nocedal and Wright's[2] book. It is included for completeness but is not discussed.

Both algorithms require primarily matrix-vector multiplications, dot products and permutations, making it ideally suited for parallelizing with ScaLAPACK. Routines pdgemv, pddot, pdgeadd, and pdlapiv were used for this purpose.

Algorithm 2 PQN with Limited Minimization

Input:

$\mathbf{A} \in \mathbb{R}^{M \times N}$, $\mathbf{b} \in \mathbb{R}^{M}$, $scale$, τ, z_{tol}.

Output:

$\mathbf{x}$

Initialize: $k = 0$, $l = -1$, $\mathbf{x}^0 = \mathbf{0}$, $\mathbf{r}^0 = -\mathbf{b}$, $\mathbf{A}^0 = \mathbf{A}$, $\boldsymbol{\mathcal{P}}^0_{pqn} = \emptyset$.
if $scale$ **then**
  $\mathbf{A} \leftarrow \mathbf{A}\,\Lambda_s$
repeat
  $\nabla f\left(\mathbf{x}^k\right) \leftarrow {\mathbf{A}^k}^T \mathbf{r}^k$;
  if $k > 0$ **then**
    $\mathbf{w}^l \leftarrow \nabla f(\mathbf{x}^k) - \nabla f(\mathbf{y}^{k-1})$
  Construct set $\boldsymbol{\mathcal{P}}^k_{pqn}$ and associated permutation matrix $\mathbf{P}^k$
  $\mathbf{A}^{k+1} \leftarrow \mathbf{A}^k \mathbf{P}^k$
  $\mathbf{y}^k \leftarrow {\mathbf{P}^k}^T \mathbf{x}^k$
  $\nabla f\left(\mathbf{y}^k\right) \leftarrow {\mathbf{P}^k}^T \nabla f\left(\mathbf{x}^k\right)$
  if $k > 0$ **then**
    $\mathbf{u}^j \leftarrow {\mathbf{P}^k}^T \mathbf{u}^j$; $\mathbf{w}^j \leftarrow {\mathbf{P}^k}^T \mathbf{w}^j$; $j = 0, 1, \ldots \min(K, k) - 1$
  $\mathbf{S}^k_{p_k} \nabla f\left(\mathbf{y}^k\right)_{p_k} = \text{LBFGS}(l, \min(K, k))$
  $\beta \leftarrow 1$
  $\mathbf{d} \leftarrow \mathbb{P}\left[\mathbf{y}^k_{p_k} - \beta\, \mathbf{S}^k_{p_k} \nabla f\left(\mathbf{y}^k\right)_{p_k}\right] - \mathbf{y}^k_{p_k}$
  while $\mathbf{d}^T \nabla f(\mathbf{y}^k)_{p_k} \geq 0$ **do**
    $\beta \leftarrow \frac{1}{2}\beta$
    $\mathbf{d} \leftarrow \mathbb{P}\left[\mathbf{y}^k_{p_k} - \beta\, \mathbf{S}^k_{p_k} \nabla f\left(\mathbf{y}^k\right)_{p_k}\right] - \mathbf{y}^k_{p_k}$
  $\alpha \leftarrow -\dfrac{\mathbf{d}^T \nabla f(\mathbf{y}^k)_{p_k}}{\|\mathbf{A}^k \mathbf{d}\|_2^2}$
  $\alpha^k \leftarrow \text{mid}(0, \alpha, 1)$
  $\mathbf{x}^{k+1} \leftarrow \begin{bmatrix} \mathbf{y}^k_{p_k} + \alpha^k\, \mathbf{d} \\ \mathbf{0}_{z_k} \end{bmatrix}$
  $l = (l + 1) \bmod K$
  $\mathbf{u}^l \leftarrow \mathbf{x}^{k+1} - \mathbf{y}^k$
  $\mathbf{r}^{k+1} \leftarrow \mathbf{A}^{k+1}\mathbf{x}^{k+1} - \mathbf{b}$
  $k \leftarrow k + 1$
until $\left\|\mathbf{r}^k\right\|_2 \leq \tau \left\|r^0\right\|_2$ or $\left\|\nabla f(\mathbf{x}^{k+1}_{p_k})\right\|_2 \leq z_{tol}$
Return $\left[\P^1 \P^2 \ldots \P^{k+1}\right] \mathbf{x}^{k+1}$

Algorithm 3 L-BFGS 2-loop recursion

procedure $\mathrm{LBFGS}(l, m)$

  if $l < 0$ **then**

    Return $\nabla f(\mathbf{y}^k)_{p_{k+1}}$

  $\mathbf{q} \leftarrow \nabla f(\mathbf{y}^k)_{p_{k+1}}$

  for $i = 0,\, 1,\, \ldots m-1$ **do**

    $j = (l - i) \bmod m$

    $\rho_j = \dfrac{1}{\left(\mathbf{w}^j_{p_{k+1}}\right)^T \mathbf{u}^j_{p_{k+1}}}$

    $\nu_j = \rho_j \left(\mathbf{u}^j_{p_{k+1}}\right)^T \mathbf{q}$

    $\mathbf{q} \leftarrow \mathbf{q} - \nu_j \, \mathbf{w}^j_{p_{k+1}}$

  if $m = 0$ **then**

    $\mathbf{s} \leftarrow \mathbf{q}$

  else

    $\mathbf{s} \leftarrow \dfrac{\left(\mathbf{u}^l_{p_{k+1}}\right)^T \mathbf{w}^l_{p_{k+1}}}{\left(\mathbf{w}^l_{p_{k+1}}\right)^T \mathbf{w}^l_{p_{k+1}}} \, \mathbf{q}$

  for $i = 0,\, 1,\, \ldots m-1$ **do**

    $j = (l + 1 + i) \bmod m$

    $\eta_j \leftarrow \rho_j \left(\mathbf{w}^j_{p_{k+1}}\right)^T \mathbf{s}$

    $\mathbf{s} \leftarrow \mathbf{s} + \mathbf{u}^j_{p_{k+1}} (\nu_j - \eta_j)$

  return $\mathbf{s}$

3.2.1 Limited PQN (LPQN) Method

The LPQN algorithm is identical to the original algorithm with the exception that a limit is imposed on the number of free variables. This limit can be global, variable with each time step or both. Assuming the imposed limit at time step $k+1$ is ${n_f}^{k+1}_{max}$ the set of free variables is given by

$$\boldsymbol{\mathcal{P}}^k_{lpqn} = \left[\boldsymbol{\mathcal{P}}^{k-1}_{lpqn} \setminus \boldsymbol{\mathcal{Z}}^k_{lpqn}\right] \cup \boldsymbol{\mathcal{P}}^k_{+}, \tag{28}$$

where

$$
\begin{aligned}
&\boldsymbol{\mathcal{P}}_{+}^{k} \subset \boldsymbol{\mathcal{P}}_{select}^{k} \\
&\boldsymbol{\mathcal{P}}_{select}^{k} = \left\{ i \mid i \in \overline{\boldsymbol{\mathcal{P}}_{lpqn}^{k-1} \setminus \boldsymbol{\mathcal{Z}}_{lpqn}^{k}}, \quad \left[\nabla f(\mathbf{x}^k)\right]_i < 0 \right\} \\
&\left[\nabla f(\mathbf{x}^k)\right]_i \leq \left[\nabla f(\mathbf{x}^k)\right]_j \quad \forall i \in \boldsymbol{\mathcal{P}}_{+}^{k} \;\&\; \forall j \in \boldsymbol{\mathcal{P}}_{select}^{k} \setminus \boldsymbol{\mathcal{P}}_{+}^{k} \\
&\left|\boldsymbol{\mathcal{P}}_{+}^{k}\right| \leq {n_f}_{max}^{k} - \left|\boldsymbol{\mathcal{P}}_{lpqn}^{k-1} \setminus \boldsymbol{\mathcal{Z}}_{lpqn}^{k}\right|.
\end{aligned}
\tag{29}
$$

The free variables at time step $k+1$ are then the free variables at the previous step provided they have not moved to set $\boldsymbol{\mathcal{Z}}_{lpqn}^{k}$, plus any additional variables whose gradients are the most negative up to a total limit of ${n_f}_{max}^{k}$. This is implemented by sorting in increasing order the components of $\nabla f(\mathbf{x}^k)$ that are in $\boldsymbol{\mathcal{P}}_{select}^{k}$ and adding the appropriate number of them to $\boldsymbol{\mathcal{P}}_{lpqn}^{k}$ so as not to exceed the limit.

The convergence proofs in Kim et al.'s[5] report are valid for the LPQN method provided the limit imposed is greater than or equal to the number of free variables in the optimal solution.

4. Results

This section presents results using the PLH and PQN methods implemented as discussed in the previous section. The codes are first validated on a number of small, random NNLS problems by comparing the optimal solutions generated by each to the original LH code.[6] Scalability results are then presented on a larger random matrix, again solving for the optimal solution. Finally, the codes are applied to NNLS problems associated with the ECSW hyper-reduction step for a reduced-order model associated with the dynamic response of a generic V-hull vehicle to an under-body blast.

All calculations were run on haise.navo.hpc.mil or kilrain.navo.hpc.mil, which are nearly identical 435 TFLOPS IBM iDataPlex HPC systems located at the Navy Distributed Computing Research Center in Mississippi. Each has 1,220 standard memory nodes with 27 GB of accessible memory per node and 2 Intel Xeon Sandy Bridge E5-2670 processors per node with 8 cores/processor. Wall clock times are measured with the "gettimeofday()" function and only include the times to solve the NNLS problems. Times for constructing or reading the matrices are not included. See Navy DSRC[7] for more details on these systems. All calculations were done in double precision.

The solutions are validated on 3 small random NNLS problems with matrices $\in \mathbb{R}^{M\times N}$ where $M < N$, $M = N$, and $M > N$ by considering the solutions from the original LH code as exact. The first set of problems are randomly generated so that the off-diagonal elements of the matricies and the components of the right-hand side are in $[0,\ 1]$, and the random diagonal elements of the matrices are in $[1,\ 10]$. These positive matrices produce large active sets, equivalently a small number p of free variables, and the PLH algorithm requires very few entries into the downdate procedure. The PLH code and 3 variations of the PQN code are tested: the original algorithm with no active set limiting, the LPQN and a limited projected Newton (LPN) method, which is the LPQN but with the exact inverse Hessian. For large active sets, it can be advantageous to compute the exact inverse Hessian instead of approximating it as done in Algorithm 3. For method LPN, the call to L-BFGS at line 14 of Algorithm 2 is replaced with a call to a similar subroutine that uses the exact inverse hessian.

For this first set of NNLS problems, the relative errors e_* between the solution $\mathbf{x}_*$ from each code and the LH solution $\mathbf{x}_{lh}$ is computed by

$$e_* = \frac{\|\mathbf{x}_* - \mathbf{x}_{lh}\|_2}{\|\mathbf{x}_{lh}\|_2} \tag{30}$$

and given in Table 1. The PLH solutions are exact to machine accuracy as expected. The errors in the PQN and LPQN solutions are $O(10^{-7})$ - $O(10^{-8})$. When the solution reaches this level of accuracy the PQN method slows down and additional iterations are not cost effective so the calculation is stopped. Continuing to iterate or solving a least squares problem based on the free variables, which at this point is the complement of the optimal active set, produces a relative error within machine accuracy, but neither is necessary or cost effective. (The code has options to do both.) LPN solutions are exact to machine accuracy. This is because the objective function is quadratic, and once the correct active set is found, the LPN method is exact. Parallel results on more than one processor, not shown in tables, have similar relative errors.

The single processor wall clock times in Table 2 correspond to the calculations reported in Table 1. Compared to LH, PLH code is about 4 times faster, PQN is on

average 6.3 times faster, and LPQN is on average 14.3 times faster. LPQN and LPN used a global limit of 1,000 free variables. The column labeled CN is the matrix condition number.

Table 1 Relative error of PLH, PQN, LPQN and LPN solutions when compared to the original LH code. All calculations are in double precision. LPQN and LPN calculations used a global limit of 1,000 free variables.

Matrix	e_{plh}	e_{pqn}	e_{lpqn}	e_{lpn}
7,000 × 10,000	4.0×10^{-14}	5.2×10^{-8}	6.0×10^{-8}	1.2×10^{-14}
10,000 × 7,000	2.2×10^{-14}	2.3×10^{-8}	6.5×10^{-7}	1.6×10^{-14}
20,000 × 20,000	3.4×10^{-14}	2.2×10^{-7}	7.0×10^{-8}	1.3×10^{-14}

Table 2 Single processor wall clock time in seconds corresponding to the calculations reported in Table 1. The column labeled CN is the matrix condition number.

Matrix	CN	Wall clock time (s)				
		LH	PLH	PQN	LPQN	LPN
7,000 × 10,000	8.65×10^2	45.9	11.4	8.6	3.9	3.1
10,000 × 7,000	8.72×10^2	50.1	13.7	8.5	5.7	4.9
20,000 × 20,000	1.90×10^7	412.6	92.6	53.0	23.0	11.9

Table 3 shows the number of iterations needed by each method to reach the optimal solution. The column labeled p is the number of free variables. For the PLH method, the number of iterations that exceeds p is an indication of the number of times the downdate loop was executed. For these matrices, execution of the downdate procedure was minimal.

Table 3 Number of iterations required for each method corresponding to the calculations reported in Table 1. The column labeled p is the number of free variables in the optimal solution.

Matrix	p	Iteration count			
		PLH	PQN	LPQN	LPN
7,000 × 10,000	250	252	235	152	44
10,000 × 7,000	275	277	222	178	43
20,000 × 20,000	379	379	312	197	47

Additional validations are done using 3 matricies of the same size as before but this time the off-diagonal elements and the components of the right-hand side are in

$[-1, 1]$. The diagonals elements are as before. These matrices are reasonably well-conditioned, produce small active sets (equivalently large p), and the PLH method in one case requires significantly more executions of the downdate loop than required by the positive matrices. The single processor wall clock time in seconds and the iteration counts are in Tables 4 and 5, respectively. Here the benefit of PQN is clear. It is better than 2 orders of magnitude faster than LH. For these problems PLH is significantly slower than LH, but this is as expected since it violates the design assumption that $p \ll N$. The downdate costs are high in first of the 3 problems. As before, the computed active sets are identical for each method, and the relative errors similar to the positive matrices. When p becomes large, computing the exact inverse Hessian is expensive, as expected. Also, when p is large, limiting the size of the free set is not appropriate, so no such calculations were made for this case.

Table 4 Single processor wall clock times for the LH, PLH and PQN methods for 3 random matrices where the off-diagonal entries and the right-hand sides are between –1.0 and 1.0 and the diagonal entries are between 1 and 10. The column labeled CN is the matrix condition number.

Matrix	CN	Wall clock time (s)			
		LH	PLH	PQN	PN
$7{,}000 \times 10{,}000$	1.13×10^1	539.7	2,154.0	4.9	747.1
$10{,}000 \times 7{,}000$	1.13×10^1	571.3	632.6	2.0	78.1
$20{,}000 \times 20{,}000$	2.84×10^4	6,607.0	15,884.4	12.3	2,327.4

Table 5 Number of iterations required for each method corresponding to the calculations reported in Table 4. The column labeled p is the number of free variables in the optimal solution.

Matrix	p	Iteration count		
		PLH	PQN	PN
$7{,}000 \times 10{,}000$	5,060	5,468	100	68
$10{,}000 \times 7{,}000$	3,512	3,518	35	25
$20{,}000 \times 20{,}000$	10,185	10,321	52	35

The wall clock times in Table 2 show that the PLH code on a single processor performs about 4 times faster than the original LH code when $p \ll N$; however, Table 4 shows the opposite when this assumption is violated. Both codes implement the same algorithm; however, the implementation differs in a significant way. The original code applies each elementary reflector to the entire matrix but the PLH

code applies them only to $\mathbf{A}_p$. When the downdate procedure is invoked, the LH code applies Givens rotations to move the offending column vector back to the active set, whereas the PLH code rebuilds the QR factorization from the point of removal. This can be costly or cheap depending on the location of the column vector that needs to be removed.

There are also storage implications in each of these designs. The storage requirement for the PLH code is greater because it keeps a copy of the original matrix and operates on the matrix $\mathbf{A}_p$. This is mitigated by the assumption that $p \ll N$ and so the storage requirement for $\mathbf{A}_p \in \mathbb{R}^{M \times p}$ is significantly reduced. When this assumption is not true, that is when p is large, the storage requirements for $\mathbf{A}_p$ significantly increases, and the code becomes less efficient. Table 2 clearly shows the PLH approach is much better when the assumption that $p \ll N$ is true.

A scalability study was performed using a random, positive matrix of size $65,536 \times 131{,}072$ and a random, positive right-hand side. The problem was chosen so that $p \ll N$ at the optimal solution. Figures 1a and 1b show the wall clock times and speedup, respectively, versus processor count. Here speedup, S_p, is defined as

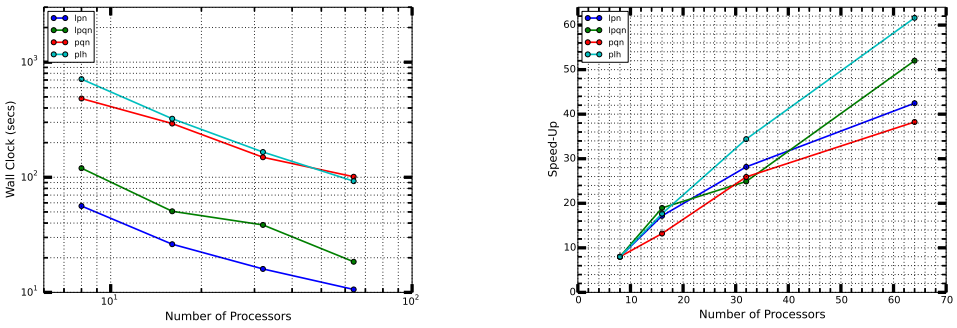
$$S_p = \frac{T_1}{T_n}, \tag{31}$$

where T_1 is the sequential time and T_n is the time using n processors. These problems were too large to fit on a single node, so T_1 was estimated by $T_1 = m\,T_m$, where m is the smallest number of processors used, in this case 8. This puts the first point on the ideal line.

The results show the PLH code achieves near linear speedup and the PQN codes achieve reasonable speedup but not ideal. Despite lower scalability, for this problem and at these processor counts the PQN codes are still faster than the PLH code, and significantly so for the limited versions.

The PLH and PQN codes were integrated into the ECSW hyper-reduction procedure of the Farhat Research Group[8] "aeros" code and used in constructing reduced-order models of a generic V-hull vehicle, shown in Fig. 2, subjected to an under-body blast. The high-dimensional, finite element model has 236,995 flexible shell and rigid beam elements with 6 degrees of freedom per node. The aeros code is also used to obtain the high-fidelity solution. The response of the vehicle to an under-

body blast is computed out to time 1.0×10^{-3}. For the purposes of building the reduced-order model, 200 snapshots were collected at uniform time intervals over this period and are used to construct a ROM with 100 basis vectors using a proper orthogonal decomposition. With 200 snapshots and 100 basis vectors the size of the ECWS matrix is $20,000 \times 236,995$. Using tolerances of $\tau = .1$ and $\tau = .01$, reduced meshes and associated reduced-order models are constructed. Scalability results and a comparison of solutions from the resulting reduce order models to the high-fidelity results are presented.



(a) Wall Clock Time **(b) Speedup**

Fig. 1 Scalability study using a 65,536 $\times$ 131,072 random matrix. Each calculation uses a single MPI process per node. $n_f^{max} = 1{,}000$ for each LPQN and LPN calculation. The number of free variables in the solution is 821. The relative error between the PLH and PQN solutions is $O(10^{-7})$ or better.

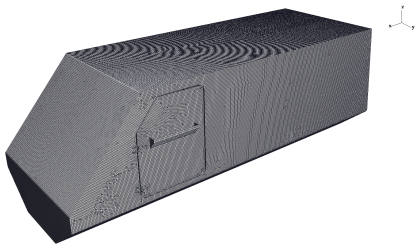
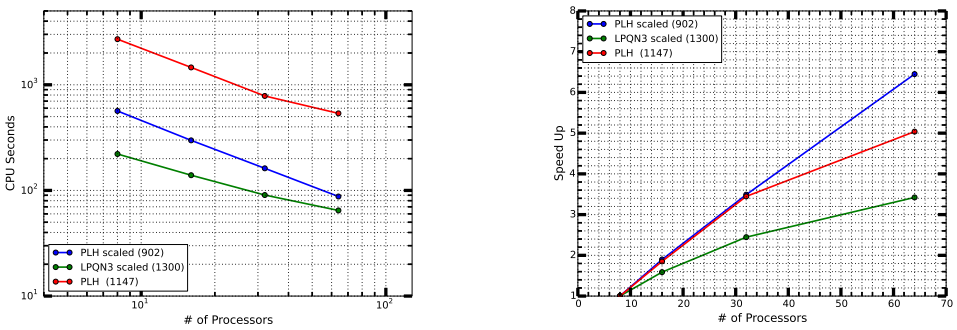


Fig. 2 V-hull finite element model.

Figure 3 shows the results of a scalability study using the generic V-hull vehicle ECSW matrix and $\tau = .1$. The PLH method was applied to both the scaled and unscaled matrix to highlight the performance differences. To understand the character of this matrix, its singular values when scaled and unscaled are plotted in Fig. 4. The PLH data show a significant increase in compute time and a decrease in scalability

for the unscaled case. The primary reason for this increase is the downdate procedure is executed many more times when the matrix is unscaled then scaled. The data therefore indicates the scalability of the downdate procedure is not as good as the main iteration loop. The number in parenthesis beside the figure labels is the size of the reduced mesh generated, or equivalently, the nonzero entries in the NNLS solution.



(a) Wall Clock Time **(b) Speedup**

Fig. 3 Scalability study using ECSW matrix from ROM of the generic vehicle.

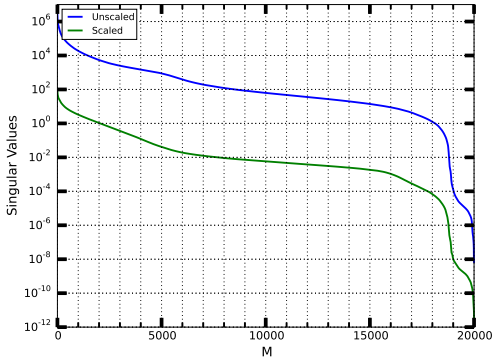


Fig. 4 Singular values for the ECSW matrix for the scaled and unscaled generic V-hull problem.

Scaling, or preconditioning, is generally a good and sometimes necessary step. There is, however, a question as to whether or not it is appropriate for ECSW hyper-reduction. A finite element model might consist of materials with widely varying properties and scaling might bias the construction of the reduced mesh in a way

not intended or wanted. The answer to this is not clear and not addressed here. It is pointed out because if scaling is not desired, the current implementation of the PQN method would not be usable on this particular ECSW matrix because it performs poorly when not scaled. Consequently, no results are presented for the PQN method for the unscaled matrix.

For the scaled matrix, the results in Fig. 3 show the PQN method is faster than the PLH method, but the implementation is not as scalable. The degradation in scalability is more pronounced here then in the larger, random matrix presented before. This could be due in part to $M = 20,000$ being smaller.

Because the PQN method is not appropriate for hyper-reduction without limiting the active set, a global limit and a per iteration limit are imposed. The global limit was set to 3,200 free variables, and the per iteration limit was set to 3 plus the limit at the previous iteration. It was found that if just a global limit is imposed, that limit, as expected, is reached immediately, but the iteration stalls trying to reach the specified tolerance, that is, the residual rate of decrease drops significantly. Imposing a per iteration limit improved convergence. The global limit of 3,200 was reached in the $\tau = .01$ case. In both cases, PQN produced larger reduced meshes than PLH.

Figures 5 and 6 show the results of the displacements and velocities computed with reduced-order models constructed with results form PLH and PQN methods and also shows the corresponding results from the high-fidelity model. The results are from an element on the bottom surface of the vehicle near the explosion. Table 6 shows the reduced mesh sizes associated with the the difference tolerances and methods presented it these figures. Two tolerances are used in the ECSW step: the figures on the left are the results with $\tau = .1$ and the ones on the right are with $\tau = .01$. In this case, both tolerances are acceptable. With $\tau = .01$ the error over the entire domain was computed to be less than 2%.

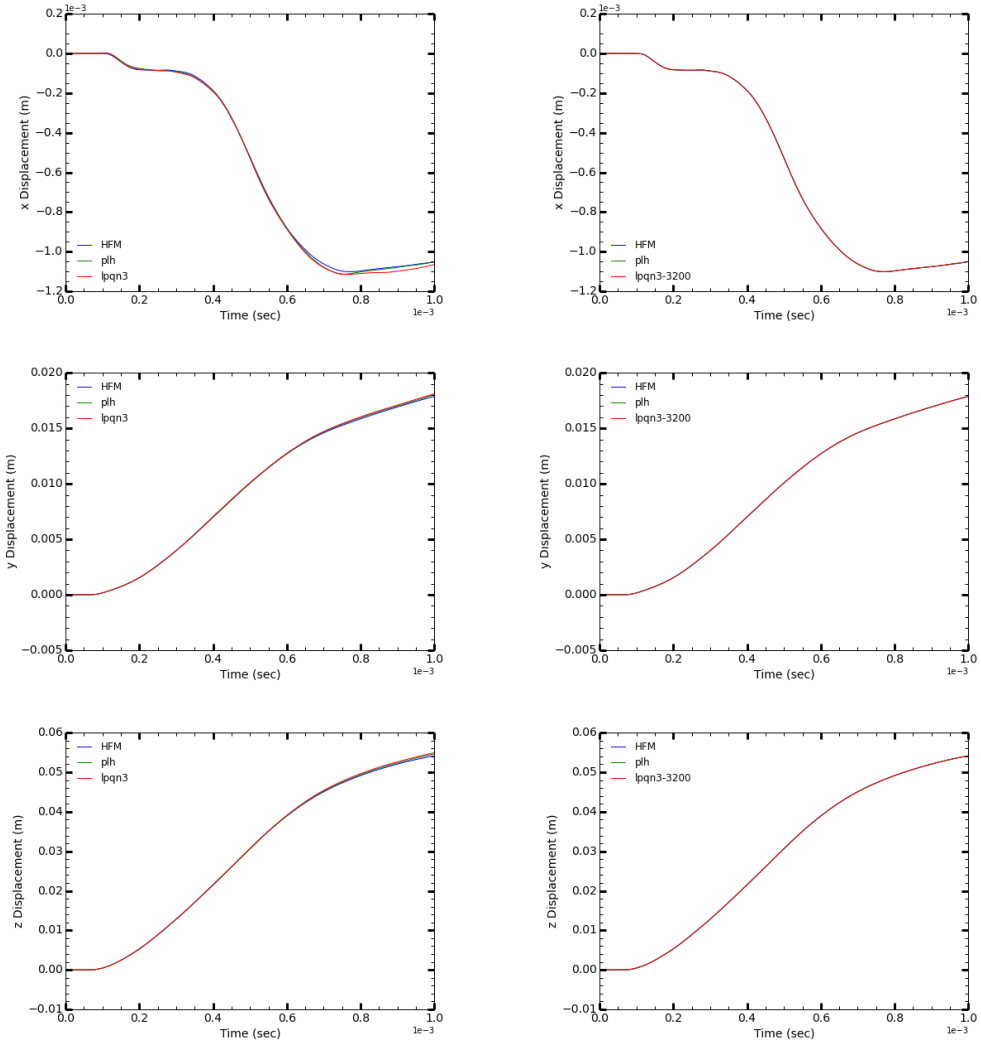


Fig. 5 Displacements at an element on the vehicle near the explosion computed by a reduced-order model constructed using PLH and LPQN compared to the high-fidelity model. Left plots used $\tau = 0.1$ and right plots used $\tau = .01$.

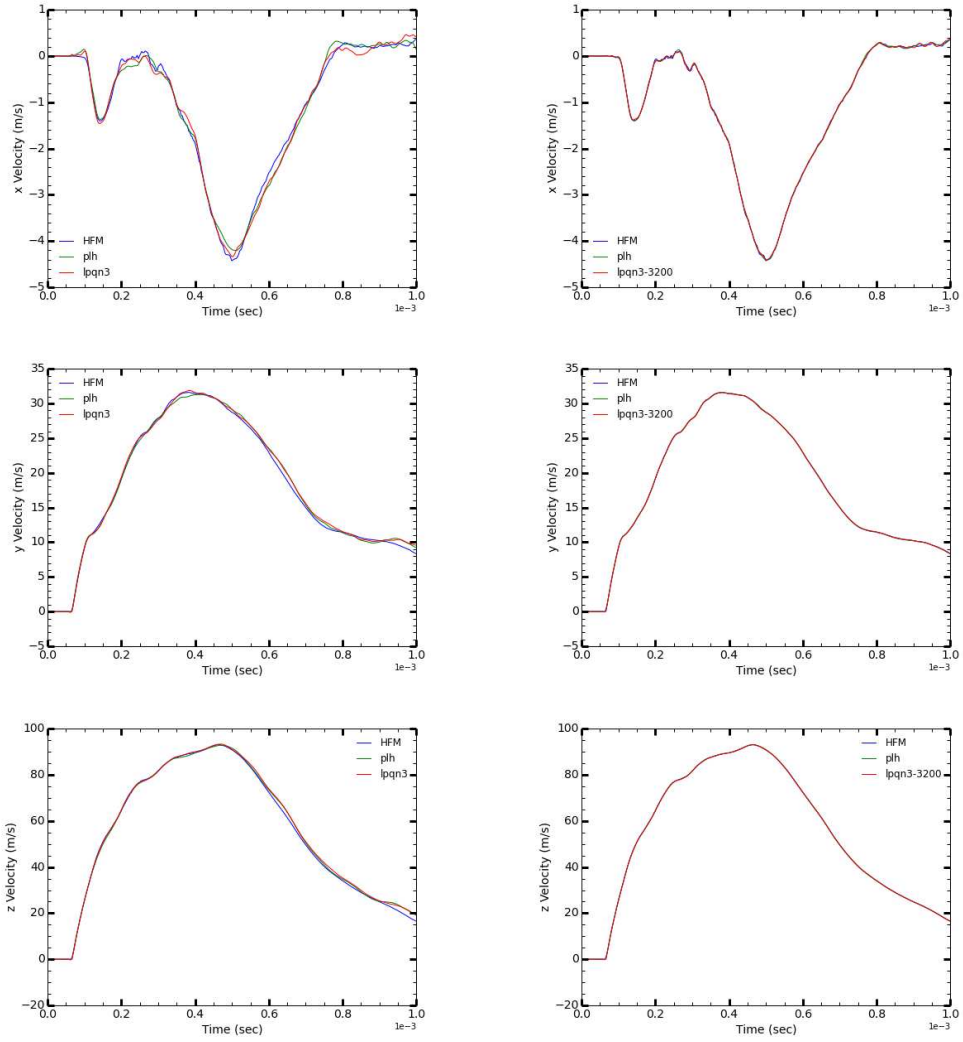


Fig. 6 Velocities at an element on the vehicle near the explosion computed by a reduced-order model constructed using PLH and LPQN methods compared to the high-fidelity model. Left plots used $\tau = 0.1$ and right plots used $\tau = .01$.

Table 6 Reduced mesh sizes produced for each solver in the ECSW hyper-reduction step.

Method	$\tau = .1$	$\tau = .01$
plh	902	2,826
lpqn3	1,017	3,200

5. Conclusions

When solving for optimal solutions, the PLH code displays the best scalability and performs very well when the design assumption that the number of free variables in the solution is significantly less than the column dimension of the matrix is met. It also outperforms the PQN code when the matrix is ill-conditioned and scaling, or preconditioning, is not done. However, in most situations the PQN code outperforms the PLH code, and in many cases significantly so, particularly if the design assumption is violated. PQN is therefore the method of choice for well-behaved NNLS problems.

When applied to ECSW hyper-reduction for the under body blast problem, the results from the 2 codes are mixed. If the matrix is scaled, the PQN method with certain free variable limiting parameters produced solutions significantly faster than the PLH code; however, in all cases the PLH code produced the sparsest solution. When the matrix is not scaled, convergence of the PQN method is either too slow or nonexistent and PLH is the only option. Scalability of the PLH code on this problem is again better than the PQN code. It is even more pronounced here but a contributing factor is most likely due in part to the relatively small row dimension (20,000) of matrix.

Limiting the number of free variables in the PQN method for the ECSW problem is absolutely necessary, but determining the limit per iteration and the global limit is not straightforward. The generic vehicle results presented used a growth factor of 3 free variables per iteration with a maximum of 3,200. This produced solutions significantly faster than PLH, but some trial and error is required to do this. On the other hand, solving for the optimal solution of well-behaved NNLS problems benefited greatly from limiting and only a global limit needed to be defined. Some prior knowledge or some trial and error may be needed to be sure the global limit does not interfere with finding the optimal solution, but it can be large and still beneficial. This is not as restrictive as in the hyper-reduction case.

The PLH code is the best choice for use with ECSW hyper-reduction when the system is large and ill-conditioned and the PQN is best for finding the solutions for well-behaved NNLS problems. More work clearly needs to be done to improve the scalability of the PQN code to make it competitive on very large problems. Even

so, with or without limiting it is the best method for finding optimal solutions on many size problems. Additional work needs to be done to make it competitive for hyper-reduction.

6. References

1. Chen D, Plemmons RJ. Nonnegativity constraints in numerical analysis. In: Bultheel A, Cools R, editors. Proceedings of the Symposium on the Birth of Numerical Analysis; 2007 Oct 29–30; Katholieke Universiteit Leuven, Belgium. Singapore: World Scientific; 2010. p. 109–140.

2. Nocedal J, Wright SJ. Numerical optimization. 2nd ed. New York (NY): Springer; 2006.

3. Farhat C, Avery P, Chapman T, Cortial J. Dimensional reduction of nonlinear finite element dynamic models with finite rotations and energy-based mesh sampling and weighting for computational efficiency. Int J Numer Meth Engng. 2014;98(9):625–662.

4. Lawson CL, Hanson RJ. Solving least squares problems. Philadelphia (PA): SIAM; 1995.

5. Kim D, Sra S, Dhillon IS. A new projected quasi-newton approach for the nonnegative least squares problem. Austin (TX): Department of Computer Sciences, University of Texas at Austin; 2006. Report No.: TR-06-54.

6. Lawson CL, Hanson RJ. Solving least squares problems. [date unknown; accessed 2015 Oct 6]. http://www.netlib.org/lawson-hanson.

7. Navy dod supercomputing resource center. 2012 Apr [accessed 2015 Oct 6]. http://www.navydsrc.hpc.mil/hardware/index.html.

8. Farhat research group. Stanford (CA): Stanford University; [date unknown; accessed 2015 Oct 6]. http://web.stanford.edu/group/frg/.

List of Symbols, Abbreviations, and Acronyms

AHPCRC	Army High Performance Computing Research Center
BFGS	Broyden-Fletcher-Goldfarb-Shanno
ECSW	energy conserving sampling and weighting
KKT	Karush-Kuhn-Tucker
L-BFGS	Limited memory Broyden-Fletcher-Goldfarb-Shanno
LH	Lawson and Hanson
LPN	limited projected Newton
LPQN	limited projected quasi-Newton
NNLS	nonnegative least squares
PLH	parallel Lawson and Hanson
PN	projected Newton
PQN	projected quasi-Newton
ROM	reduced-order model

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
2 (PDF)	DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL IMAL HRA MAIL & RECORDS MGMT
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
6 (PDF)	US ARMY RESEARCH LAB RDRL CIH R NAMBURU D THOMPSON RDRL CIH-C J CLARKE D GROVE J COLLINS RDRL CIH-M S THOMPSON

INTENTIONALLY LEFT BLANK.