2020

Sub-Sampled Matrix Approximations

Joy Azzam
Michigan Technological University, atazzam@mtu.edu

SUB-SAMPLED MATRIX APPROXIMATIONS

By

Joy Azzam

A DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

In Mathematical Sciences

MICHIGAN TECHNOLOGICAL UNIVERSITY

2020

This dissertation has been approved in partial fulfillment of the requirements for the Degree of DOCTOR OF PHILOSOPHY in Mathematical Sciences.

Department of Mathematical Sciences

Dissertation Co-advisor:	*Dr. Allan A. Struthers*
Dissertation Co-advisor:	*Dr. Benjamin W. Ong*
Committee Member:	*Dr. Cécile M. Piret*
Committee Member:	*Dr. Daniel R. Fuhrmann*
Department Chair:	*Dr. Mark S. Gockenbach*

Contents

List of Figures

List of Tables

Definitions

A	Fixed Matrix
$\tilde{A}$	Fixed Matrix Modified to have Effective Rank l
A_l	Best Rank l Approximation to A
B	Approximation to A
$\nabla^2 f(x)$	Hessian Matrix of Scalar Function $f(x)$
$\mathbf{E}[\hat{z}]$	Expectation of $\hat{z}$
$\hat{\gamma}$	Low Rank Tolerance Parameter
H	Approximation to A^{-1}
I_n	Identity Matrix of size $n \times n$
$\langle X, Y \rangle$	Frobenius Inner Product of Matrices X and Y
$\lambda_{min}(X)$	Smallest Eigenvalue of Matrix X
$\lambda_{max}(X)$	Largest Eigenvalue of Matrix X
l	Low Rank Target Dimension
m, n	Problem Dimensions
$\|X\|_F$	Frobenius Norm of Matrix X
$\|X\|_{F(W_1,W_2)}$	Weighted Frobenius Norm of Matrix X
$\mathcal{N}(0,1)$	Gaussian Normal Distribution with mean 0 and standard deviation 1
p	Number of Power Iterations

$\mathcal{P}$	Projection Matrix
$\mathcal{P}_{W^{-1},U}$	Weighted Projection Matrix
Q	Orthogonal Matrix where $Q^T Q = I_n$
$\mathbb{R}^{m \times n}$	Real Vector Space of m by n Matrices
R_k	Residual $R_k = A - B_k$ at Iteration k
ρ	Convergence Rate
s	Number of Sampling Directions
$\mathrm{Tr}[X]$	Trace of Matrix X
U,V	Random Dimension Reduction Matrices
W	Weight Matrix
$\hat{x}$	Random Matrix

List of Abbreviations

AD	Algorithmic Differentiation
AdaRBFGS$_{cols}$	Adaptive Randomized BFGS method (Column Sampling)
AdaRBFGS$_{gauss}$	Adaptive Randomized BFGS method (Gaussian Sampling)
BFGS	Broyden-Fletcher-Goldfarb-Shanno (Optimization Method)
BFGSH	BFGS Inverse Matrix Approximation Update
BFGSA	Accelerated BFGS Matrix Approximation
DFP	Davidon-Fletcher-Powell (Optimization Method)
DFPH	DFP Inverse Matrix Approximation Update
LibSVM	Library of Support Vector Machine Problems
NS	Non-Symmetric Sub-Sampled Update
RRF	Randomized Range Finder
RSSI	Randomized Sub-Space Iteration
S1	Symmetric Sampled Update
SMW	Sherman-Morrison-Woodbury (Inverse Formula)
SS1	Symmetric Sub-Sampled Update
SS1A	Accelerated Sub-Sampled Update
SS2	Two-Step Symmetric Sub-Sampled Update
SSInv	Inverse Sub-Sampled Update (Matrix Acceleration)

SSInvP	Inverse Sub-Sampled Update (Inverse Matrix Acceleration)
SSLR	Sub-Sampled Low Rank Algorithm
SS1SMW	Sub-Sampled Inverse Matrix Approximation Algorithm
SVD	Singular Value Decomposition
SPD	Symmetric Positive Definite

Abstract

Matrix approximations are widely used to accelerate many numerical algorithms. Current methods sample row (or column) spaces to reduce their computational footprint and approximate a matrix A with an appropriate embedding of the data sampled. This work introduces a novel family of randomized iterative algorithms which use significantly less data per iteration than current methods by sampling input and output spaces simultaneously. The data footprint of the algorithms can be tuned (independent of the underlying matrix dimension) to available hardware. Proof is given for the convergence of the algorithms, which are referred to as sub-sampled, in terms of numerically tested error bounds. A heuristic accelerated scheme is developed and compared to current algorithms on a substantial test-suite of matrices.

The sub-sampled algorithms provide a lightweight framework to construct more useful inverse and low rank matrix approximations. Modifying the sub-sampled algorithms gives families of methods which iteratively approximate the inverse of a matrix whose accelerated variant is comparable to current state of the art methods. Inserting a compression step in the algorithms gives low rank approximations having accelerated variants which have fixed computational as well as storage footprints.

Chapter 1

Introduction

Randomly sampled matrix approximations are used to accelerate many numerical algorithms. Randomized sampling of Hessian matrices along blocks of directions was used to accelerate Quasi-Newton minimization algorithms [4, 27] in the numerical optimization setting. More recently, iterative solvers for linear systems have been constructed using random samples of the rows of matrices [15].

Randomized algorithms to approximate the inverse of an $n \times n$ matrix A are developed explicitly in [14] and implicitly when used as pre-conditioners [1, 3] for linear systems. Typical algorithms randomly sample $V \in \mathbb{R}^{n \times s_2}$ with $s_2 < n$ and update (using Quasi-Newton updates based on minimum-change formulations [4, 12, 27]) the current approximation to A^{-1} with the random sample

$$AV \in \mathbb{R}^{n\times s_2}.$$

The sampling matrix V is used to partially reduce the data footprint of the update and can be tuned by selecting $s_2 < n$.

In the optimization context [25] Quasi-Newton schemes generate either a Symmetric Positive Definite (SPD) sequence $B_k \rightarrow A$ or the SPD sequence $H_k \rightarrow A^{-1}$ generated by applying the Sherman-Morrison-Woodbury (SMW) formula to B_k. Both the matrix inverse algorithms [14] and the Block BFGS method [12] apply SMW to approximate A^{-1}. Omitting the SMW computation from the algorithms in [12, 14] gives algorithms which approximate A. Here and in what follows, $\hat{x} \sim p^{i\times j}$ indicates that the matrix $\hat{x} \in \mathbb{R}^{i\times j}$ has entries drawn from the distribution p and $\bar{x} = \mathbf{E}[\hat{x}]$ is the expectation of $\hat{x}$.

The primary motivation for sampled algorithms is to reduce the data footprint of the algorithms and provide (partially) tunable algorithms for modern hardware architectures. The sub-sampled algorithms evaluate

$$U^T AV \in \mathbb{R}^{s_1\times s_2}. \tag{1.1}$$

where $U \sim \mathcal{N}(0,1)^{m\times s_1}$ and $V \sim \mathcal{N}(0,1)^{n\times s_2}$ to sample row and column spaces simultaneously. The data footprint ($s_1 \times s_2$) of these sub-sampled algorithms can be fully tuned (by selecting both $s_1, s_2 \ll m, n$) to available hardware and is significantly smaller than the footprint ($s_1 \times n$ and/or $m \times s_2$) of algorithms which update with samples $U^T A$ and/or $A\,V$.

There are related algorithms in the literature, many of which fall in the general framework described in [16]. Although these algorithms indeed construct *one* (expensive) sub-sample of the form $Q^* A\,Q$ to approximate the action of A associated with its dominant eigenspace, the matrix Q is expensive to compute, requiring *samples* $A\,\Omega$ ($\Omega \sim \mathcal{N}(0,1)^{n\times s_2}$) to construct.

In this work, U and V are cheaply generated by sampling $\mathcal{N}(0,1)$, and self-correcting updates are described which embed the sub-samples $U^T A\,V$ and iteratively sum them to generate various sub-sampled matrix approximations. Chapter 2 describes three methods using sub-sampled data, eq. (1.1), to approximate a matrix A and gives proof of their convergence. Inserting an additional algebraic update into the matrix approximation algorithms generates methods for approximating the inverse matrix A^{-1}. Self-correcting updates constructed in this way are described and numerically tested in Chapter 3. Performing a compression step to reduce the data footprint of the iterate generates sub-sampled low rank methods. Chapter 4 explores this idea and the effects of different compression schemes, some of which have fixed computational footprints as well as overall storage footprints.

1.1 Work Estimates

Throughout the thesis it will be assumed that users are only given 'black-box' access to the target matrix A which efficiently computes products $A\,V$, $U^T A$ and/or $U^T A\,V$ for $U^T \in \mathbb{R}^{s_1 \times m}$ and $V \in \mathbb{R}^{n \times s_2}$. Typically, the cost of accessing this data is proportional to the number of output entries; this will be the primary cost metric to compare efficiency of the algorithms. To understand this metric, consider the approximation of $U^T \nabla^2 f(x)\, V$ for smooth $f : \mathbb{R}^m \to \mathbb{R}$ using standard central differences. For a small δ, each output entry, $\left[U^T \nabla^2 f(x)\, V\right]_{i,j} = u_i^T \nabla^2 f(x)\, v_j$, is approximately

$$\frac{f(x + \delta(v_j + u_i)) - f(x + \delta(v_j - u_i)) - f(x - \delta(v_j - u_i)) + f(x - \delta(v_j + u_i))}{4\delta^2 ||v_j|| ||u_i||}.$$

The cost of evaluating $U^T \nabla^2 f(x)\, V$ is the number of output entries multiplied by the four function evaluations. In fact, the full Hessian action $\nabla^2 f(x) v_j = \left[\nabla^2 f(x)\, V\right]_j$ used in the block BFGS scheme [12] also has a 'black-box' approximation: simply take U to be the identity. Forward-forward mode AD [23] can provide the same functionality while avoiding the difficulty of choosing δ appropriately and the unavoidable precision loss due to cancellation.

When comparing computational cost of the algorithms, the product AV is referred to as an $m \times s_2$ sample of the matrix A, $U^T A$ is an $s_1 \times n$ sample of A, while $U^T AV$ is an $s_1 \times s_2$ *sub-sample* of the matrix A. The sample AV samples the input space of A, the sample $U^T A$ samples the output space of A, and the sub-sample $U^T AV$ samples the input and output spaces of A simultaneously. One should observe that the sub-sample contains only a small portion of each sample $U^T A$ and AV. Because of this, sub-sampled algorithms will require more iteration than sampled algorithms. In spite of this, sub-samples are efficient at capturing information and provide an excellent foundation for compact, efficient algorithms.

1.2 Relationship to Quasi-Newton Algorithms

The goal of this thesis is to develop and analyze iterative approximations to A which use sub-samples $U^T AV$. The resulting algorithms are strongly connected to and motivated by Quasi-Newton algorithms from nonlinear optimization and sampled Quasi-Newton algorithms [14].

Various sampled Quasi-Newton methods [12, 14] have been developed based on block updates [4]. The block optimization algorithm [12] takes several Quasi-Newton steps with a fixed Hessian approximation (to reduce linear algebra) before performing a block update, and accelerates terminal convergence with an ingenious heuristic. Several variant block updates (based on traditional minimum change justifications for DFP and BFGS [25]) are developed and used in iterative algorithms for approximating inverses [14]: additional theory and heuristic acceleration techniques have also been explored [13].

The sub-sampled algorithms presented use minimum-change motivated arguments to determine a family of updates which iteratively incorporate $s_1 \times s_2$ pieces of information from the sub-sample $U^TAV \in \mathbb{R}^{s_1 \times s_2}$ to generate a sequence of approximations to A. The data footprint of each iteration is $s_1 \times s_2$ which is substantially smaller than that of the sampled algorithms [12, 13, 14] and can be fully tuned to the available hardware. Sampled methods using $U^T A$ or $A V$ have additional storage and computational requirements proportional to the problem size n, while sub-sampled methods can have fixed storage and computational requirements proportional to $s_1, s_2 \ll n$. Convergence rates are derived which are comparable with existing Quasi-Newton algorithms [12, 13, 14].

1.3 Notation

Throughout the thesis: SPD is an acronym for symmetric positive definite and W denotes SPD weight matrices; superscript $^+$ denotes the Moore-Penrose pseudo-inverse;

$\langle X, Y\rangle_F = \mathrm{Tr}\left[X^T Y\right]$ and $\|X\|_F^2 = \langle X, X\rangle_F$ denote the Frobenius inner product and norm; residuals are measured using weighted norms,

$$\|X\|^2_{F(W_1^{-1}, W_2^{-1})} = \|W_1^{-1/2} X\, W_2^{-1/2}\|_F^2 \quad \text{and} \quad \|X\|^2_{F(W^{-1})} = \|W^{-1/2} X\, W^{-1/2}\|_F^2,$$

with conforming SPD weights W_1, W_2 and W; algorithms are developed using the W-weighted projector, which projects onto the column space of $W\,U$,

$$\mathcal{P} = P_{W^{-1},U} = W\,U(U^T\,W\,U)^{-1}U^T. \tag{1.2}$$

The weighted projector satisfies

$$\mathcal{P}\,W = W\,\mathcal{P}^T = \mathcal{P}\,W\,\mathcal{P}^T \quad \text{and} \quad W^{-1}\mathcal{P} = \mathcal{P}^T W^{-1} = \mathcal{P}^T\,W^{-1}\,\mathcal{P}.$$

Chapter 2

Matrix Approximation

2.1 Randomized Algorithms for Matrix Approximation

Numerical optimization texts (e.g. [25]) motivate and derive Quasi-Newton update schemes for SPD matrices A using constrained minimum change criteria (for $B \approx A$ and $H \approx A^{-1}$) in weighted Frobenius norms. Traditional algorithms are derived by selecting different weights. Block update algorithms (sampled algorithms in our terminology) which update multiple directions simultaneously are derived [12, 14] similarly. The KKT equations [25] for the quadratic programs,

$$B_{k+1} = \arg\min_{B} \left\{ \frac{1}{2} \|B - B_k\|^2_{F(W^{-1})} \mid B\,U = A\,U \text{ and } B = B^T \right\} \tag{2.1}$$

$$H_{k+1} = \arg\min_{H} \left\{ \frac{1}{2} \|H - H_k\|^2_{F(W^{-1})} \mid U = H\,A\,U \text{ and } H = H^T \right\} \tag{2.2}$$

gives two different updates using the same sample $A\,U_k$: the update to B_k produces B_{k+1}, an improved approximation to A; the update to H_k produces H_{k+1}, an improved approximation to A^{-1}. Solutions to eqs. (2.1) and (2.2) can be obtained by taking the derivative of the norm with respect to the matrix B and enforcing the constraint equations (see section A.2 for the sub-sampled version). The linear algebraic updates that result are

$$\begin{aligned} B_{k+1} &= B_k + \mathcal{P}_B(A - B_k) + (A - B_k)\mathcal{P}_B^T - \mathcal{P}_B(A - B_k)\mathcal{P}_B^T, \\ H_{k+1} &= H_k + \mathcal{P}_H(A^{-1} - H_k) + (A^{-1} - H_k)\mathcal{P}_H^T - \mathcal{P}_H(A^{-1} - H_k)\mathcal{P}_H^T, \end{aligned} \tag{2.3}$$

where the weighted projectors $\mathcal{P}_B$ and $\mathcal{P}_H$ defined by eq. (1.2) are

$$\begin{aligned} \mathcal{P}_B &= P_{W^{-1},U} = W\,U(U^T\,W\,U)^{-1}U^T, \\ \mathcal{P}_H &= P_{W^{-1},A\,U} = W\,A\,U(U^T\,A\,W\,A\,U)^{-1}U^T A. \end{aligned}$$

Block DFP [27] is the B formulation, eq. (2.1), with $W = A$,

$$B_{k+1} = (I_n - \mathcal{P}_{\text{DFP}})\; B_k(I_n - \mathcal{P}_{\text{DFP}}^T) + \mathcal{P}_{\text{DFP}}\,A. \tag{2.4}$$

where

$$\mathcal{P}_{\text{DFP}} = P_{A^{-1},U} = A\,U(U^T A\,U)^{-1}U^T.$$

Block BFGS [12, 14] is the H formulation, eq. (2.2), (inverted using the Shermann-Morisson-Woodbury formula) with $W = A^{-1}$,

$$B_{k+1} = B_k - B_k U \left(U^T B_k U\right)^{-1} U^T B_k + AU \left(U^T AU\right)^{-1} U^T A. \tag{2.5}$$

Setting $W = A^{-1}$ in the B update, eq. (2.1), produces an update containing the term A^{-1}, which is not useful.

Lastly, algorithms that fall in the general framework described in [16] embed the sub-sample $Q^* A\,Q$ into a low-rank approximation $\mathcal{P}_{I_n,Q}\,A\,\mathcal{P}_{I_n,Q}^T$ when A is SPD. For symmetric positive semi-definite A, they apply the Nyström method which constructs $(AQ)(Q^*AQ)^{-1}(AQ)^* = \mathcal{P}_{A^{-1},Q}\,A\,\mathcal{P}_{A^{-1},Q}^T$ using the *sample* $A\,Q$.

The goal of this thesis is to use sub-samples $U^T A\,V$ to iteratively construct approximations to A. The matrix updates $B \to A$ will all satisfy

$$U^T B\,V = U^T A\,V.$$

There are many such potential updates, for instance, $U\,U^+ A\,V\,V^+$ minimizes the unweighted Frobenius norm.

2.1.1 Sub-Sampled Update (NS)

We define updates using the minimal change criterion;

$$B_{k+1} = \arg\min_{B} \left\{ \frac{1}{2}\|B - B_k\|^2_{F(W_1^{-1},W_2^{-1})} \mid U^T B\, V = U^T A\, V \right\}, \tag{2.6}$$

which defines the self-correcting update (for details see section A.2)

$$B_{k+1} = B_k + P_{W_1^{-1},U^k}(A - B_k)P^T_{W_2^{-1},V^k}. \tag{2.7}$$

By construction, eq. (2.7) simply corrects the sub-sampled mismatch $U^T(A - B_k)\, V$. It cannot increase the weighted Frobenius norm $\|A - B_k\|^2_{F(W_1^{-1},W_2^{-1})}$ and, provided the sub-space sequences U_k and V_k eventually exhaust the underlying spaces, the weighted residual must decrease monotonically to zero.

Given $A \in \mathbb{R}^{m\times n}$, an initial estimate $B_0 \in \mathbb{R}^{m\times n}$, sub-sample sizes $\{s_1, s_2\}$, and SPD weights $\{W_1, W_2\}$, eq. (2.7) generates a sequence $\{B_k\}$ that converges to A monotonically in the appropriate weighted Frobenius norm. The resulting algorithm is summarized in algorithm 1: boxed values show the number of output entries computed for the sub-sample $U^T\, A\, V$; the return-line double boxed value is the total number of output entries used.

Algorithm 1 NS: Non-Symmetric Sub-Sampled Approximation

Require: $B_0 \in \mathbb{R}^{m \times n}$, SPD $W_1 \in \mathbb{R}^{m \times m}$, $W_2 \in \mathbb{R}^{n \times n}$, $\{s_1, s_2\} \in \mathbb{N}$.

```
repeat $\{k = 0, 1, \ldots\}$
    Sample $U_k \sim N(0,1)^{m \times s_1}$ and $V_k \sim N(0,1)^{n \times s_2}$
    Compute residual $\Lambda_k = U_k^T A\, V_k - U_k^T B_k V_k \in \mathbb{R}^{s_1 \times s_2}$ .................. $\boxed{s_1 s_2}$
    Update $B_{k+1} = B_k + W_1 U_k (U_k^T W_1 U_k)^{-1} \Lambda_k (V_k^T W_2 V_k)^{-1} V_k^T W_2$
until convergence
return $B_{k+1}$ .............................................. $\boxed{\boxed{(k+1)\,(s_1 s_2)}}$
```

Algorithm 1 does not generate symmetric approximations for symmetric A. The next two sections modify the basic algorithm to preserve symmetry. When discussing symmetric updates, the iterations will always use symmetric initializations $B_0 = B_0^T$ and symmetric weights $W = W_1 = W_2$.

2.1.2 Symmetric Sub-Sampled Update (SS1)

Setting $V_k = U_k$ gives a symmetric sub-sample $U^T A\, U$. Using weights $W = W_1 = W_2$ in algorithm 1 with symmetric initialization $B_0 = B_0^T$ gives a sequence of symmetric approximations, B_k, to a symmetric $n \times n$ matrix A. The resulting algorithm is summarized in algorithm 2 with the number of output entries computed shown in boxes as before.

Algorithm 2 SS1: Symmetric Sub-Sampled Approximation

```
Require: $B_0 \in \mathbb{R}^{n\times n}$ satisfying $B_0^T = B_0$, SPD $W \in \mathbb{R}^{n\times n}$, $s_1 \in \mathbb{N}$.
repeat $\{k = 0, 1, \dots\}$
    Sample $U_k \sim \mathcal{N}(0,1)^{n\times s_1}$
    Compute residual $\Lambda_k = U_k^T A\, U_k - U_k^T B_k U_k \in \mathbb{R}^{s_1\times s_1}$ .................... $\boxed{s_1^2}$
    Compute $\tilde{P}_k = W\, U_k (U_k^T W\, U_k)^{-1}$
    Update $B_{k+1} = B_k + \tilde{P}_k \Lambda_k \tilde{P}_k^T$
until convergence
return $B_{k+1}$ .................................................... $\boxed{\boxed{(k+1)\,(s_1^2)}}$
```

Remark 1. *Algorithm 2 (with $W = I_n$) can be viewed as a sub-sampled BFGS update: apply the orthogonal projection $\mathcal{P}_{I_n,U} = UU^T$ to both sides of eq. (2.5) to get algorithm 2 with $W = I_n$. Algorithm 2 can be viewed as a sub-sampled DFP update.*

Remark 2. *Algorithm 2 does not preserve positivity. A non-SPD result can be observed when*

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 \\ 0 & 9 \end{bmatrix}, \quad \textit{and} \quad U = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

2.1.3 Multi-Step Symmetric Updates (SS2)

An alternative approach to generate symmetric approximations is to symmetrize eq. (2.7) as follows

$$\begin{aligned} B_{k+1/2} &= B_k + P_{W^{-1},U^k}(A - B_k)P^T_{W^{-1},V^k} \\ B_{k+1} &= \frac{1}{2}\left(B_{k+1/2} + B^T_{k+1/2}\right). \end{aligned} \tag{2.8}$$

For symmetric A and B_0, it can be shown that the convergence rate for eq. (2.8) is comparable to Algorithm 2. However, for symmetric A the additional sample,

$$P_{W_2^{-1},V^k} A\, P^T_{W_1^{-1},U^k} = \left(P_{W_1^{-1},U^k} A\, P^T_{W_2^{-1},V^k}\right)^T,$$

can be directly incorporated to give

$$\begin{aligned} B_{k+1/3} &= B_k + P_{W_1^{-1},U^k}(A - B_k)P^T_{W_2^{-1},V^k} \\ B_{k+2/3} &= B_{k+1/3} + P_{W_2^{-1},V^k}(A - B^T_{k+1/3})P^T_{W_1^{-1},U^k} \\ B_{k+1} &= \frac{1}{2}\left(B_{k+2/3} + B^T_{k+2/3}\right), \end{aligned} \tag{2.9}$$

where the last line again enforces symmetry. The two-step symmetric algorithm is summarized in algorithm 3 with sample counts boxed as before. The two-step algorithm has superior convergence properties.

Algorithm 3 SS2: Two-Step Symmetric Sub-Sampled Approximation

Require: $B_0 \in \mathbb{R}^{n\times n}$ satisfying $B_0 = B_0^T$, SPD $W \in \mathbb{R}^{m\times m}$, $\{s_1, s_2\} \in \mathbb{N}$.

```
repeat $\{k = 0, 1, \ldots\}$
    Sample $U_k \sim N(0,1)^{n\times s_1}$ and $V_k \sim N(0,1)^{n\times s_2}$
    Compute residual $\Lambda_k = U_k^T A\, V_k - U_k^T B_k V_k \in \mathbb{R}^{s_1\times s_2}$ .................. $\boxed{s_1 s_2}$
    Compute $B_{k+1/3} = B_k + W\, U_k (U_k^T W\, U_k)^{-1} \Lambda_k (V_k^T W\, V_k)^{-1} V_k^T W$
    Compute residual $\Lambda_{k+1/3} = (U_k^T A\, V_k)^T - V_k^T B_{k+1/3} U_k \in \mathbb{R}^{s_2\times s_1}$
    Compute $B_{k+2/3} = B_{k+1/3} + W\, V_k (V_k^T W\, V_k)^{-1} \Lambda_{k+1/3} (U_k^T W\, U_k)^{-1} U_k^T W$
    Update $B_{k+1} = \frac{1}{2}(B_{k+2/3} + B_{k+2/3}^T)$
until convergence
return $B_{k+1}$ .............................................. $\boxed{\boxed{(k+1)\,(s_1 s_2)}}$
```

2.2 Convergence Analysis

The convergence results rely extensively on properties of randomly generated projectors. In the computational tests, projections are generated by orthogonalizing matrices with individual entries drawn from $N(0,1)$. For square matrices, this process gives rotations drawn from a distribution which is invariant under rotations [28]. The sub-sampled algorithms use symmetric weighted rank s projectors,

$$\hat{z} = W^{1/2} U (U^T\, W\, U)^{-1} U^T\, W^{1/2}, \tag{2.10}$$

where W is an SPD weight matrix and U is simply the first s columns of such a random rotation. The expectation of random symmetric $n \times n$ projections $\hat{z}$, $\mathbf{E}[\hat{z}] \in \mathbb{R}^{n\times n}$, is crucial in the analysis to come. Write z_i for the eigenvalues of $\mathbf{E}[\hat{z}]$ with the standard ordering $z_1 \leq z_2 \leq \cdots \leq z_n$. The extreme eigenvalues z_1 and z_n determine the algorithms convergence with the best results when $z_1 = z_n$.

For clarity the next section collects a number of useful definitions and lemmas.

2.2.1 Mathematical Preliminaries

Definition 3. *A random matrix $\hat{X} \in \mathbb{R}^{m\times n}$ is rotationally invariant if the distribution of $Q_m \, \hat{X} \, Q_n$ is the same for all rotations $Q_i \in \mathcal{O}(i)$.*

Lemma 4 (Random Projections)**.** *For any distribution $\hat{z}$ of real, symmetric rank s projectors in $\mathbb{R}^n$,*

$$0 \leq \lambda_{\min}(\boldsymbol{E}[\hat{z}]) \leq \frac{s}{n} \leq \lambda_{\max}(\boldsymbol{E}[\hat{z}]) \leq 1. \tag{2.11}$$

Further, if $\hat{z}$ is rotationally invariant, then $\boldsymbol{E}[\hat{z}] = \frac{s}{n} I_n$.

Proof. Let $x \in \mathbb{R}^n$ with $x^T x = 1$. Since $\hat{z}$ is a projector,

$$0 = \lambda_{\min}(\hat{z}) \leq x^T \hat{z} \, x \leq \lambda_{\max}(\hat{z}) = 1.$$

Since $\mathbf{E}[x^T \hat{z}\, x] = x^T \mathbf{E}[\hat{z}]\, x$, taking the expectation gives

$$0 \leq x^T \mathbf{E}[\hat{z}]\, x \leq 1,$$

for all unit vectors x. Since the trace is linear, the sum of the eigenvalues of $\mathbf{E}[\hat{z}]$ equals $\mathrm{Tr}(\mathbf{E}[\hat{z}]) = \mathbf{E}[\mathrm{Tr}(\hat{z})] = E(s) = s$, which establishes eq. (2.11). Rotationally invariant $\hat{z}$ satisfy $\mathbf{E}[\hat{z}] = \alpha I_n$ since for all $Q_1, Q_2 \in \mathcal{O}(n)$,

$$\mathbf{E}[\hat{z}] = \mathbf{E}[Q_1\, \hat{z}\, Q_2] = Q_1\, \mathbf{E}[\hat{z}]\, Q_2,$$

Using a similar argument, linearity of the trace gives $\alpha = \frac{s}{n}$. □

Lemma 5 (Projection Cancellation)**.** *For $R \in \mathbb{R}^{m \times n}$ and conforming symmetric projections $\hat{y}, \hat{z}$,*

$$\langle R\, \hat{z}, R\, \hat{z}\rangle_F = \langle R, R\, \hat{z}\rangle_F \tag{2.12}$$

$$\langle \hat{y}\, R\, \hat{z}, \hat{y}\, R\, \hat{z}\rangle_F = \langle \hat{y}\, R\, \hat{z}, R\, \hat{z}\rangle_F = \langle \hat{y}\, R\, \hat{z}, R\rangle_F \tag{2.13}$$

Proof. Expanding the definition of eq. (2.12),

$$\langle R\hat{z}, R\hat{z}\rangle_F = \mathrm{Tr}[\hat{z}^T R^T R\, \hat{z}] = \mathrm{Tr}[R^T R\, \hat{z}\, \hat{z}^T] = \mathrm{Tr}[R^T R\, \hat{z}] = \langle R, R\, \hat{z}\rangle_F,$$

since $\mathrm{Tr}[A\,B] = \mathrm{Tr}[B\,A]$ and $\hat{z}$ is a projector. Similarly for eq. (2.13),

$$\langle \hat{y}\,R\,\hat{z}, \hat{y}\,R\,\hat{z}\rangle_F = \mathrm{Tr}[\hat{z}^T R^T \hat{y}^T \hat{y}\,R\,\hat{z}] = \mathrm{Tr}[\hat{z}^T R^T \hat{y}^T R\,\hat{z}] = \langle \hat{y}\,R\,\hat{z}, R\,\hat{z}\rangle_F,$$

$$\langle \hat{y}\,R\,\hat{z}, R\,\hat{z}\rangle_F = \mathrm{Tr}[\hat{z}^T R^T \hat{y}^T R\,\hat{z}] = \mathrm{Tr}[\hat{z}\,\hat{z}^T R^T \hat{y}^T R] = \mathrm{Tr}[\hat{z}^T R^T \hat{y}^T R] = \langle \hat{y}\,R\,\hat{z}, R\rangle_F.$$

□

Lemma 6 (Spectral Bounds)**.** *For any $R \in \mathbb{R}^{m\times n}$ and conforming symmetric positive semi-definite matrices S_1, S_2, and (in the special case $m = n$) S the following bounds hold:*

$$\lambda_{\min}(S_1)\langle R, R\rangle_F \le \langle S_1\,R, R\rangle_F \le \lambda_{\max}(S_1)\langle R, R\rangle_F, \tag{2.14}$$

$$\lambda_{\min}(S_2)\langle R, R\rangle_F \le \langle R, R\,S_2\rangle_F \le \lambda_{\max}(S_2)\langle R, R\rangle_F, \tag{2.15}$$

$$\lambda_{\min}(S)^2\langle R, R\rangle_F \le \langle S\,R, R\,S\rangle_F \le \lambda_{\max}(S)^2\langle R, R\rangle_F. \tag{2.16}$$

Proof. To establish eq. (2.14) write $R = [r_1|r_2|\cdots|r_n]$ and note that the results follows immediately from $\langle S_1\,R, R\rangle_F = \sum_{i=1}^n r_i^T S_1 r_i$ and $\langle R, R\rangle_F = \sum_{i=1}^n r_i^T r_i$ since

$$\sum_{i=1}^n \lambda_{\min}(S_1)\,r_i^T r_i \le \sum_{i=1}^n r_i^T S_1 r_i \le \sum_{i=1}^n \lambda_{\max}(S_1)\,r_i^T r_i. \tag{2.17}$$

Equation (2.15) follows directly from eq. (2.14) applied to S_2 and R^T since

$$\langle R, R\,S_2\rangle_F = \langle R^T, S_2^T R^T\rangle_F = \langle S_2^T\,R^T, R^T\rangle_F = \langle S_2\,R^T, R^T\rangle_F.$$

To establish eq. (2.16) note that for symmetric positive semi-definite T

$$\langle T^2 R, R T^2 \rangle_F = \langle T\, R, T^2\, T\, R \rangle_F \quad \text{and} \quad \langle T\, R, T\, R \rangle_F = \sum_{i=1}^{n} r_i^T T^2 r_i.$$

Equation (2.16) then follows immediately with $T = S^{1/2}$ from eq. (2.14) applied to $S_1 = T^2$ and the standard bound eq. (2.17) with $S_1 = T^2$. □

2.2.2 Convergence Theorems

Convergence results for algorithms 1 to 3. are for $\mathbf{E}[\|B - A\|_F^2]$. Such results dominate similar results for $\|\mathbf{E}[B - A]\|_F^2$ since

$$\|\mathbf{E}\left[B - A\right]\|_F^2 = \mathbf{E}\left[\|B - A\|_F^2\right] - \mathbf{E}\left[\|B - \mathbf{E}\left[B\right]\|_F^2\right],$$

as shown in [14].

Theorem 7 (Convergence of NS algorithm 1)**.** *Let $A \in \mathbb{R}^{m\times n}$ and $W_1 \in \mathbb{R}^{m\times m}$ and $W_2 \in \mathbb{R}^{n\times n}$ be fixed SPD weight matrices. If $U_k \in \mathbb{R}^{m\times s_1}$ and $V_k \in \mathbb{R}^{n\times s_2}$ are random, independently selected matrices with full column rank (with probability one), then eq. (2.7) generates a sequence, B_k, from an initial guess $B_0 \in \mathbb{R}^{m\times n}$ satisfying*

$$\boldsymbol{E}\left[\|B_{k+1} - A\|^2_{F(W_1^{-1}, W_2^{-1})}\right] \leq (\rho_{NS})^{k+1} \boldsymbol{E}\left[\|B_0 - A\|^2_{F(W_1^{-1}, W_2^{-1})}\right],$$

where $\rho_{NS} = 1 - \lambda_{\min}(\boldsymbol{E}[\hat{y}])\, \lambda_{\min}(\boldsymbol{E}[\hat{z}])$, *with*

$$\hat{y}_k = W_1^{1/2} U_k (U_k^T W_1 U_k)^{-1} U_k^T W_1^{1/2}, \quad \hat{z}_k = W_2^{1/2} V_k (V_k^T W_2 V_k)^{-1} V_k^T W_2^{1/2}. \tag{2.18}$$

Proof. Define the kth residual as $R_k := W_1^{-1/2}(B_k - A)W_2^{-1/2}$. With some algebraic manipulation, eq. (2.7) can be re-written as

$$R_{k+1} = R_k - \hat{y}_k R_k \hat{z}_k. \tag{2.19}$$

Computing the squared Frobenius norm of eq. (2.19),

$$\begin{aligned}
\langle R_{k+1}, R_{k+1} \rangle_F &= \langle R_k - \hat{y}_k R_k \hat{z}_k, R_k - \hat{y}_k R_k \hat{z}_k \rangle_F \\
&= \langle R_k, R_k \rangle_F - \langle R_k, \hat{y}_k R_k \hat{z}_k \rangle_F - \langle \hat{y}_k R_k \hat{z}_k, R_k \rangle_F + \langle \hat{y}_k R_k \hat{z}_k, \hat{y}_k R_k \hat{z}_k \rangle_F \\
&= \langle R_k, R_k \rangle_F - \langle \hat{y}_k R_k \hat{z}_k, R_k \hat{z}_k \rangle_F,
\end{aligned}$$

where we have made use of theorem 5. Taking the expected value with respect to independent samples U_k (leaving V_k and R_k fixed) gives

$$\begin{aligned}
\mathbf{E}\left[\|R_{k+1}\|_F^2 \mid V_k, R_k \right] &= \langle R_k, R_k \rangle_F - \langle \mathbf{E}[\hat{y}_k] R_k \hat{z}_k, R_k \hat{z}_k \rangle_F \\
&\leq \langle R_k, R_k \rangle_F - \lambda_{\min}(\mathbf{E}[\hat{y}_k]) \langle R_k \hat{z}_k, R_k \hat{z}_k \rangle_F \\
&\leq \langle R_k, R_k \rangle_F - \lambda_{\min}(\mathbf{E}[\hat{y}_k]) \langle R_k, R_k \hat{z}_k \rangle_F,
\end{aligned} \tag{2.20}$$

where we applied theorem 6 to the symmetric positive semi-definite matrix $\mathbf{E}[\hat{y}_k]$, and utilized eq. (2.12). Taking the expected value with respect to independent samples V_k and leaving R_k fixed gives

$$\begin{aligned}\mathbf{E}[\|R_{k+1}\|_F^2 \mid R_k] &\leq \langle R_k, R_k\rangle_F - \lambda_{\min}(\mathbf{E}[\hat{y}_k])\langle R_k, R_k\mathbf{E}[\hat{z}_k]\rangle_F \\ &\leq \langle R_k, R_k\rangle_F - \lambda_{\min}(\mathbf{E}[\hat{y}_k])\lambda_{\min}(\mathbf{E}[\hat{z}_k])\langle R_k, R_k\rangle_F.\end{aligned}$$

Taking the full expectation gives

$$\begin{aligned}\mathbf{E}[\|R_{k+1}\|_F^2] &\leq \mathbf{E}\left[\langle R_k, R_k\rangle_F\right] - \lambda_{\min}(\mathbf{E}[\hat{y}_k])\lambda_{\min}(\mathbf{E}[\hat{z}_k])\mathbf{E}\left[\langle R_k, R_k\rangle_F\right] \\ &= (1 - \lambda_{\min}(\mathbf{E}[\hat{y}_k])\lambda_{\min}(\mathbf{E}[\hat{z}_k]))\mathbf{E}[\langle R_k, R_k\rangle_F].\end{aligned}$$

Since

$$\mathbf{E}[\|R_{k+1}\|_F^2] = \mathbf{E}[\|B_k - A\|^2_{F(W_1^{-1}, W_2^{-1})},$$

un-rolling the recurrence for $k+1$ iterations yields the desired result. □

Remark 8. *The condition that U_k and V_k are chosen independently of each other is required to justify $\boldsymbol{E}[\langle \hat{y}_k R_k \hat{z}_k, R_k\hat{z}_k\rangle_F] = \langle \boldsymbol{E}[\hat{y}_k] R_k\hat{z}_k, R_k\hat{z}_k\rangle_F$.*

Theorem 9 (Convergence of SS1 algorithm 2)**.** *Let $A, W \in \mathbb{R}^{n\times n}$ be fixed SPD matrices and $U_k \in \mathbb{R}^{n\times s}$ be a randomly selected matrix having full column rank with probability 1. If $B_0 \in \mathbb{R}^{n\times n}$ is an initial guess for A with $B_0 = B_0^T$, then after applying*

$k+1$ iterations of the update in algorithm 2, the iterates B_{k+1} satisfy

$$\boldsymbol{E}[\|B_{k+1}-A\|^2_{F(W^{-1})}] \leq (\rho_{SS1})^{k+1}\boldsymbol{E}[\|B_0-A\|^2_{F(W^{-1})}], \tag{2.21}$$

where $\rho_{SS1} = 1-\lambda_{\min}(\boldsymbol{E}[\hat{z}])^2$ and

$$\hat{z}_k = W^{1/2}U_k(U_k^T W\, U_k)^{-1}U_k^T W^{1/2}.$$

Proof. Following similar steps outlined in the proof in theorem 7, we arrive at

$$\langle R_{k+1}, R_{k+1}\rangle_F = \langle R_k, R_k\rangle_F - \langle R_k, \hat{z}_k R_k \hat{z}_k\rangle_F.$$

Taking the expected value with respect to U_k leaving R_k fixed we have

$$\begin{aligned}
\mathbf{E}\left[\|R_{k+1}\|_F^2 \mid R_k\right] &= \langle R_k, R_k\rangle_F - \mathbf{E}\left[\langle R_k, \hat{z}_k R_k \hat{z}_k\rangle_F\right] \\
&= \langle R_k, R_k\rangle_F - \mathbf{E}\left[\mathrm{Tr}[R_k^T \hat{z}_k R_k \hat{z}_k]\right] \\
&= \langle R_k, R_k\rangle_F - \mathrm{Tr}\left[\mathbf{E}\left[R_k \hat{z}_k R_k \hat{z}_k\right]\right] \\
&= \langle R_k, R_k\rangle_F - \mathrm{Tr}\left[\mathbf{E}\left[(R_k \hat{z}_k)^2\right]\right] \\
&\leq \langle R_k, R_k\rangle_F - \mathrm{Tr}\left[\mathbf{E}\left[R_k \hat{z}_k\right]^2\right] \\
&= \langle R_k, R_k\rangle_F - \mathrm{Tr}\left[R_k \mathbf{E}\left[\hat{z}_k\right] R_k \mathbf{E}\left[\hat{z}_k\right]\right],
\end{aligned}$$

where the inequality arises from application of Jensen's Inequality. Simplifying and applying eq. (2.16),

$$\begin{aligned}\mathbf{E}[\|R_{k+1}\|^2_{F(W^{-1})} \mid R_k] &= \langle R_k, R_k\rangle_F - \langle \mathbf{E}[\hat{z}_k]R_k, R_k\mathbf{E}[\hat{z}_k]\rangle_F \\ &\leq \langle R_k, R_k\rangle_F - \lambda_{\min}(\mathbf{E}[\hat{z}_k])^2\langle R_k, R_k\rangle_F.\end{aligned}$$

Taking the full expectation and un-rolling the recurrence yields the desired result. □

Theorem 10 (Convergence of SS2 algorithm 3)**.** *Let A, U_k, V_k and B_0 be defined as in theorem 7, and let W be a fixed SPD matrix. After applying $k+1$ iterations of algorithm 3 with $W = W_1 = W_2$, the iterates B_k satisfy*

$$\boldsymbol{E}\left[\|B_k - A\|^2_{F(W^{-1})}\right] \leq (\rho_{SS2})^{k+1}\boldsymbol{E}\left[\|B_0 - A\|^2_{F(W^{-1})}\right],$$

where

$$\rho_{SS2} = 1 - 2\lambda_{\min}(\boldsymbol{E}[\hat{y}])\lambda_{\min}(\boldsymbol{E}[\hat{z}]) + \lambda_{\min}(\boldsymbol{E}[\hat{y}])^2\lambda_{\min}(\boldsymbol{E}[\hat{z}])^2.$$

Proof. Define kth residual R_k and projectors $\hat{y}_k$ and $\hat{z}_k$ as in theorem 7 with $W = W_1 = W_2$. The iteration given in eq. (2.9) can be re-written in terms of R_k as follows.

$$\begin{aligned}R_{k+1/3} &= R_k - \hat{y}_k R_k \hat{z}_k \\ R^T_{k+2/3} &= R^T_{k+1/3} - \hat{z}_k R^T_{k+1/3}\hat{y}_k \\ R_{k+1} &= \frac{1}{2}\left(R_{k+2/3} + R^T_{k+2/3}\right)\end{aligned}$$

Theorem 7 gives

$$\mathbf{E}\left[\left\|R_{k+1/3}\right\|_F^2\right] \leq (\rho_{\mathrm{NS}})\mathbf{E}\left[\|R_k\|_F^2\right],$$

and a repeated application of theorem 7 gives

$$\mathbf{E}\left[\left\|R_{k+2/3}\right\|_F^2\right] \leq (\rho_{\mathrm{NS}})\mathbf{E}\left[\|R_{k+1/3}\|_F^2\right] \leq (\rho_{\mathrm{NS}})^2\mathbf{E}\left[\|R_k\|_F^2\right].$$

Lastly, we observe via the triangle inequality that

$$\begin{aligned}\mathbf{E}\left[\|R_{k+1}\|_F^2\right] &= \mathbf{E}\left[\left\|\frac{1}{2}\left(R_{k+2/3} + R_{k+2/3}^T\right)\right\|_F^2\right] \\ &\leq \frac{1}{2}\mathbf{E}\left[\left\|R_{k+2/3}\right\|_F^2\right] + \frac{1}{2}\mathbf{E}\left[\left\|R_{k+2/3}^T\right\|_F^2\right] \\ &= (\rho_{\mathrm{NS}})^2\mathbf{E}\left[\|R_k\|_F^2\right],\end{aligned}$$

Un-rolling the loop for $k+1$ iterations gives the desired result. □

2.2.3 Optimal Fixed Weight Convergence Rates

To discuss convergence rates, define

$$\begin{aligned}\rho_{\mathrm{NS}}(y_1, z_1) &= 1 - y_1 z_1, \\ \rho_{\mathrm{SS1}}(z_1) &= 1 - z_1^2, \\ \rho_{\mathrm{SS2}}(y_1, z_1) &= (1 - y_1 z_1)^2,\end{aligned} \tag{2.22}$$

and note that the convergence rates for algorithms 1 to 3 can be expressed as

$$\|R_{k+1}\|^2_{F(W_1^{-1},W_2^{-1})} \leq \rho \|R_k\|^2_{F(W_1^{-1},W_2^{-1})} \tag{2.23}$$

with the appropriate ρ, eq. (2.22), evaluated at $y_1 = \lambda_{\min}(\mathbf{E}[\hat{y}])$ and $z_1 = \lambda_{\min}(\mathbf{E}[\hat{z}])$. Since any symmetric rank s random projection $\hat{z}$ on $\mathbb{R}^n$ satisfies $0 \leq z_1 \leq \frac{s}{n} \leq z_n \leq 1$ and rotationally invariant distributions, e.g. UU^+ with $U \sim N(0,1)^{n\times s}$, further satisfy $\mathbf{E}[\hat{z}] = \frac{s}{n}$, minimizing the various convergence rates ρ over the appropriate domains gives the following optimal rates.

Corollary 11. *(Optimal Convergence Rate) The optimal convergence rates for algorithms 1 to 3 are obtained attained for U_k and V_k sampled from rotationally invariant distributions,*

$$\begin{aligned}
\rho_{NS}^{opt} &= 1 - \frac{s_1}{m}\frac{s_2}{n}, \\
\rho_{SS1}^{opt} &= 1 - \left(\frac{s_2}{n}\right)^2, \\
\rho_{SS2}^{opt} &= \left(1 - \frac{s_1}{m}\frac{s_2}{n}\right)^2.
\end{aligned} \tag{2.24}$$

Proof. Each part is simply the result of an explicit optimization,

$$\rho_{\text{NS}}^{\text{opt}} = \min_{\substack{0\leq y\leq s_1/m \\ 0\leq z\leq s_2/n}} (1-yz) = 1 - \left(\frac{s_1}{m}\right)\left(\frac{s_2}{n}\right)$$

$$\rho_{\text{SS1}}^{\text{opt}} = \min_{0\leq z\leq s_2/n} (1-z^2) = 1 - \left(\frac{s_2}{n}\right)^2$$

$$\rho_{\text{SS2}}^{\text{opt}} = \min_{\substack{0\leq y\leq s_1/m \\ 0\leq z\leq s_2/n}} (1-yz)^2 = \left(1 - \frac{s_1}{m}\frac{s_2}{n}\right)^2$$

□

Remark 12. *Theorems 7, 9 and 10 all assume the weight matrix W and distributions are fixed. All of the non-accelerated numerical experiments use fixed weights and sample from fixed rotationally invariant distributions.*

2.2.4 Theoretical Lower Bound for Convergence Rates

Lower bounds (entirely analogous to the upper bounds in theorems 7, 9 and 10 but using the upper bounds in theorem 6) are easily derived. For example, the two-sided error bound for algorithm 1 is

$$\rho_{\text{NS}}(y_m, z_n)\mathbf{E}[\|R_k\|_F^2] \leq \mathbf{E}[\|R_{k+1}\|_F^2] \leq \rho_{\text{NS}}(y_1, z_1)\mathbf{E}[\|R_k\|_F^2],$$

where as before $y_1 \leq y_2 \leq \cdots \leq y_m$ is the spectrum of $\mathbf{E}[\hat{y}]$, $z_1 \leq z_2 \leq \cdots \leq z_n$ is the spectrum of $\mathbf{E}[\hat{z}]$ and the explicit form for ρ_{NS} is in eq. (2.22). We collect the similar results for algorithms 1 to 3 in theorem 13.

Corollary 13 (Two-Sided Convergence Rates)**.** *Given the assumptions of theorems 7, 9 and 10 the explicit formulas eq.* (2.22) *for ρ give two-sided bounds,*

$$\begin{aligned}
\rho_{NS}(y_m, z_n)^{k+1} &\leq \frac{\boldsymbol{E}\left[\|B_{k+1} - A\|^2_{F(W_1^{-1},W_2^{-1})}\right]}{\|B_0 - A\|^2_{F(W_1^{-1},W_2^{-1})}} \leq \rho_{NS}(y_1, z_1)^{k+1} \\
\rho_{SS1}(z_n)^{k+1} &\leq \frac{\boldsymbol{E}\left[\|B_{k+1} - A\|^2_{F(W^{-1})}\right]}{\|B_0 - A\|^2_{F(W^{-1})}} \leq \rho_{SS1}(z_1)^{k+1} \\
\rho_{SS2}(y_n, z_n)^{k+1} &\leq \frac{\boldsymbol{E}\left[\|B_{k+1} - A\|^2_{F(W^{-1})}\right]}{\|B_0 - A\|^2_{F(W^{-1})}} \leq \rho_{SS2}(y_1, z_1)^{k+1}
\end{aligned}$$

where y_1, y_m, z_1, z_n are the extreme eigenvalues of $\boldsymbol{E}[\hat{y}]$ and $\boldsymbol{E}[\hat{z}]$.

Proof. We prove the NS result; the proofs for SS1 and SS2 are analogous. Equation (2.20) of theorem 7 and theorem 6 gives

$$\begin{aligned}
\mathbf{E}\left[\|R_{k+1}\|_F^2 \mid V_k, R_k\right] &= \langle R_k, R_k\rangle_F - \langle \mathbf{E}[\hat{y}_k] R_k \hat{z}_k, R_k \hat{z}_k\rangle_F \\
&\geq \langle R_k, R_k\rangle_F - \lambda_{\max}(\mathbf{E}[\hat{y}_k])\, \langle R_k, R_k \hat{z}_k\rangle_F.
\end{aligned}$$

Following theorem 7 (expectation in V_k and repeating the inequality) gives

$$\begin{aligned}
\mathbf{E}[\|R_{k+1}\|_F^2 \mid R_k] &\geq \langle R_k, R_k\rangle_F - \lambda_{\max}(\mathbf{E}[\hat{y}_k])\, \langle R_k, R_k \mathbf{E}[\hat{z}_k]\rangle_F \\
&\geq \langle R_k, R_k\rangle_F - \lambda_{\max}(\mathbf{E}[\hat{y}_k])\, \lambda_{\max}(\mathbf{E}[\hat{z}_k])\, \langle R_k, R_k\rangle_F.
\end{aligned}$$

Then taking the full expectation gives the inequality

$$\mathbf{E}[\|R_{k+1}\|_F^2] \geq \mathbf{E}\left[\langle R_k, R_k\rangle_F\right] - \lambda_{\max}(\mathbf{E}[\hat{y}_k])\,\lambda_{\max}(\mathbf{E}[\hat{z}_k])\,\mathbf{E}\left[\langle R_k, R_k\rangle_F\right]$$

$$= (1 - \lambda_{\max}(\mathbf{E}[\hat{y}_k])\,\lambda_{\max}(\mathbf{E}[\hat{z}_k]))\,\mathbf{E}[\langle R_k, R_k\rangle_F].$$

Combine this with theorem 7 and unroll the iteration to obtain the NS result. □

Remark 14. *If $\hat{y}$ and $\hat{z}$ are rotationally invariant, the upper and lower probabilistic bounds in theorem 13 coincide since $z_1 = z_n = \frac{s_1}{n}$ and $y_1 = y_m = \frac{s_2}{m}$. Algorithms 1 to 3 all use rotationally invariant distributions and converge predictably at the expected rate. The algorithms still converge with other distributions provided the smallest eigenvalue of the expectation is positive.*

2.3 Numerical Results

The sub-sampled methods given by algorithms 1 to 3 are tested on a variety of SPD matrices: $A = XX^T$, $X \sim \mathcal{N}(0,1)^{n\times n}$; ridge regression matrices chosen from [5]; and matrices chosen from the Sparse Suite Library [7]. Algorithms 1 to 3 were implemented within the MATLAB code framework in [14] and tested on the same collection of problems from [5, 7]. All computational tests were performed on **Superior**, a high-performance computing infrastructure at Michigan Technological University.

The experiments are organized as follows: Section 2.3.1 compares the sub-sampled algorithms with $s = s_1 = s_2 = \lceil\sqrt{n}\rceil$ (the sample size used in [14]) on one moderate sized $n \approx 5000$ matrix from each of the three classes tested in [14]; Section 2.3.2 demonstrates the independence of the convergence on the sample size $s \ll n$ for the same three matrices; the convergence of the sub-sampled algorithms on the remaining matrices from [14] are available in section A.3.

2.3.1 Convergence Test

The convergence,

$$\frac{\|A - B_k\|_F}{\|A - B_0\|_F},$$

of sampled algorithms [14] with sample size $s = \lceil\sqrt{n}\rceil$ are compared to the sub-sampled algorithms with $s_1 = s_2 = s$ on three matrices: ($n = 5000$) XX^T with $X \sim \mathcal{N}(0,1)^{n\times n}$ Figure 2.1; ($n = 5000$) Gisette-Scale [5] Figure 2.2; and ($n = 4704$) NASA [7] Figure 2.3. These figures show: BFGS(◇) as specified by eq. (2.5); DFP (◇) as specified by eq. (2.4); NS (⊗) as specified by Algorithm 1; SS1 (●) as specified by Algorithm 2; SS2 (■) as specified by Algorithm 3. Theoretical convergence rates from eq. (2.24) are shown in dotted lines. Runs were terminated after $5n^2$ iterations or when the relative residual norm fell below 10^{-2}. Algorithms 1 to 3 converge predictably:

linear in the semilog plots matching the theoretical convergence rates (dotted lines). DFP and BFGS have target dependent weight matrices which may initially improve convergence. For the Gisette-Scale matrix fig. 2.2 DFP and BFGS show a dramatic improvement. However, fig. 2.3 and various examples from [14] in the supplementary materials show that BFGS can fail to converge.

Sampling U_k and V_k from rotationally invariant distributions, all the experiments show the predictable optimal convergence rates from eq. (2.24) (dotted lines). With these choices the expected convergence rate of both NS and SS1 is $1 - \left(\frac{s}{n}\right)^2$ while the expected convergence rate of SS2 is $\left(1 - \left(\frac{s}{n}\right)^2\right)^2 = 1 - 2\left(\frac{s}{n}\right)^2 + \left(\frac{s}{n}\right)^4$.

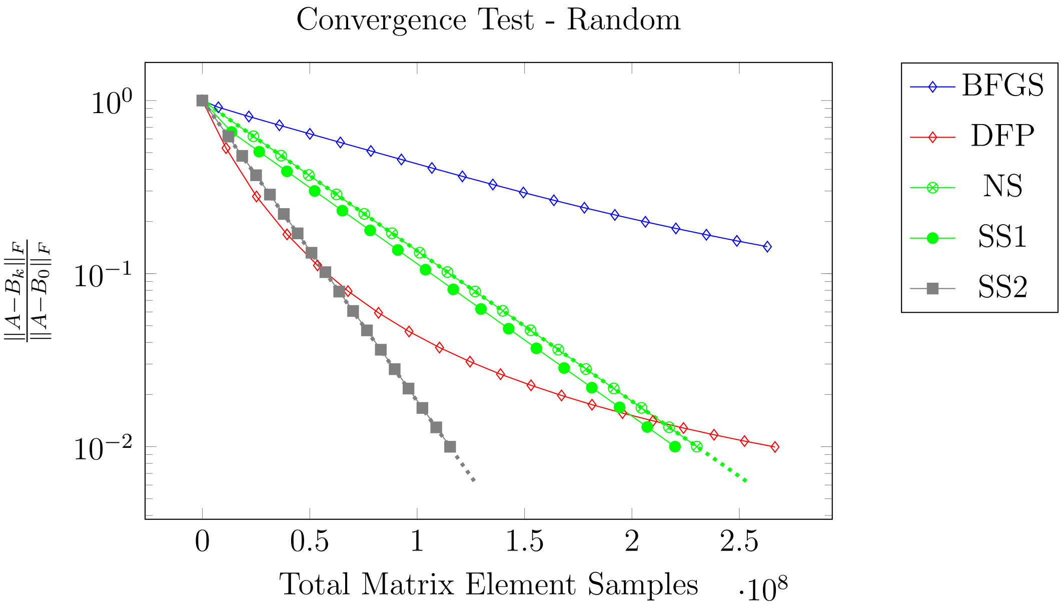


Figure 2.1: ($n = 5000$) Approximation of XX^T where $X \sim \mathcal{N}(0,1)^{n\times n}$ with $s = 71 = \lceil\sqrt{5000}\rceil$. The DFP and BFGS updates converge more slowly while the sub-sampled methods match their theoretical convergence rates (shown in dotted lines).

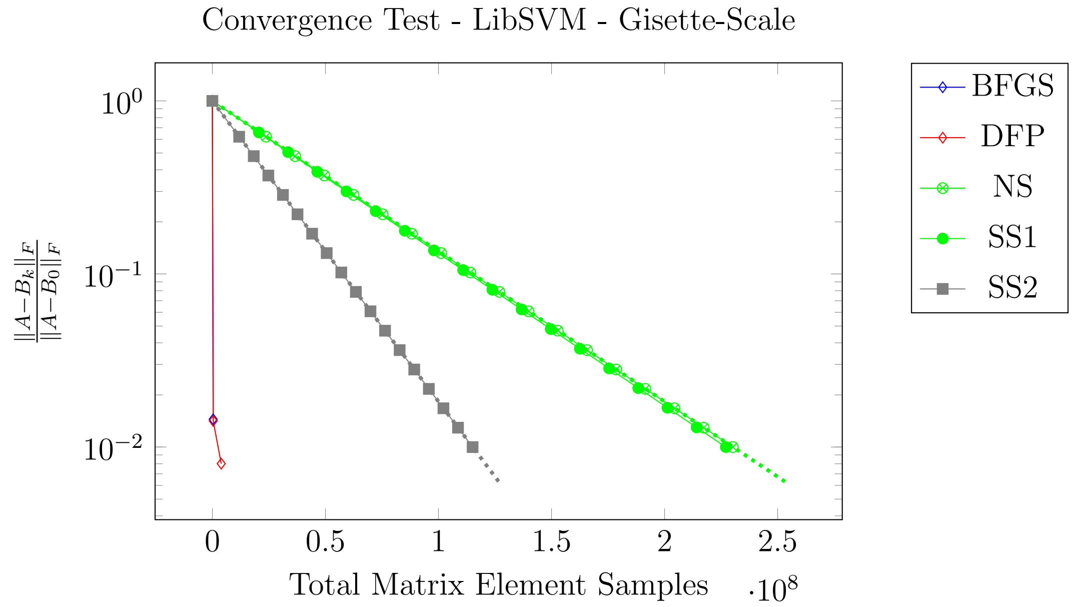


Figure 2.2: ($n = 5000$) Approximation of Hessian from **Gisette Scale** [5] with $s = 71 = \lceil\sqrt{5000}\rceil$. The DFP and BFGS method show incredible acceleration for this matrix. The sub-sampled methods work consistently as predicted by their theoretical convergence rates (shown in dotted lines).

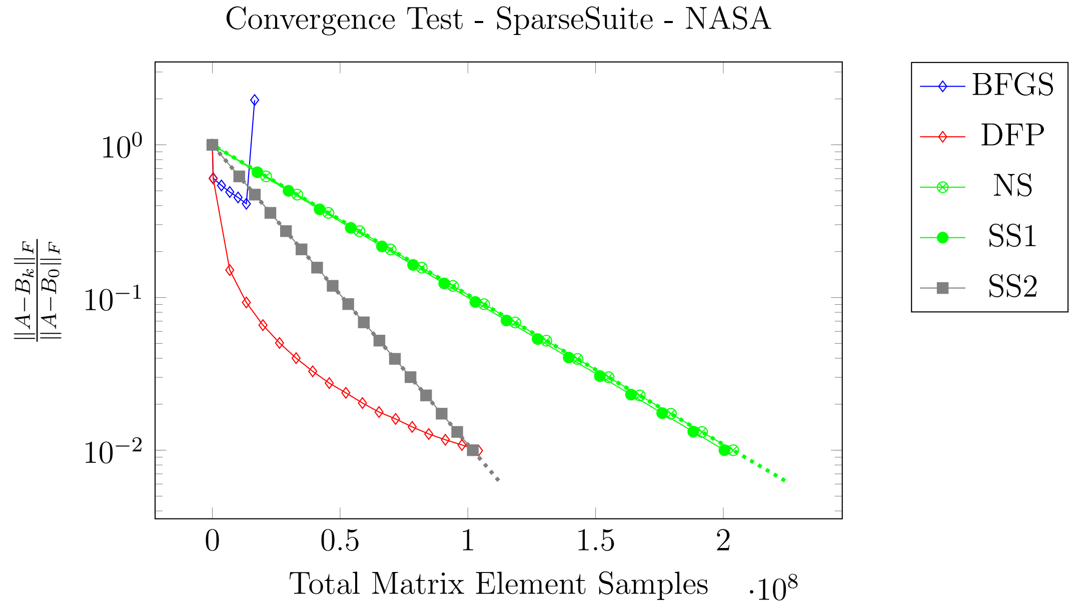


Figure 2.3: ($n = 4700$) Approximation of **NASA4704** from [7]. $s = 69 = \lceil\sqrt{4704}\rceil$. The DFP method performs similar to the random matrix $A = XX^T$ while the BFGS method fails. Sub-sampled methods match their theoretical convergence rates (shown in dotted lines).

2.3.2 Sample Size Tests

Equation (2.24) gives the expected convergence rate, ρ, of the various algorithms as a function of the ratio of sample size s and matrix dimension n. Consider two experiments running SS1 with rotationally invariant sampling on the same $A \in \mathbb{R}^{n\times n}$ with sample size s and $2s$: the first experiment involves s^2 matrix samples at each step, and one expects the residual to be reduced by a factor of $1 - \left(\frac{s}{n}\right)^2$ after each step; the second experiment involves $(2s)^2$ matrix samples each step, and one expects the residual to be reduced by a factor of of $1 - \left(\frac{2s}{n}\right)^2$ after each step. Since the primary cost metric for algorithms is the number of matrix samples, four steps of size s^2 is the same amount of work as one step of size $(2s)^2$. Taking four steps of size s^2 gives approximately the same reduction as one step of size $(2s)^2$ since

$$\left(1 - \left(\frac{s}{n}\right)^2\right)^4 = \left(1 - \left(\frac{2s}{n}\right)^2\right)^1 + O\left(\left(\frac{s}{n}\right)^4\right).$$

All formulas in eq. (2.24) have the same scaling behavior and as a result the expected convergence of all the sub-sampled algorithms should be essentially independent of s for $1 \ll s \ll n$. In practice, one has the freedom to choose s to suit the available computational hardware.

This behavior is verified for the sub-sampled algorithms algorithms 1 to 3 on the three test problems from section 2.3.1. Table 2.1 reports the total computational effort for each matrix, normalized by the corresponding number of matrix samples for $s = 512$. All of the entries are very close to one, indicating that the computational effort is independent of s.

Matrix	s	NS	SS1	SS2
Rand	128	0.997	0.992	0.999
	256	0.996	0.996	1.004
	512	1.000	1.000	1.000
Gisette Scale	128	0.996	0.990	0.995
	256	0.996	0.993	0.998
	512	1.000	1.000	1.000
NASA4704	128	0.996	0.998	0.994
	256	0.996	1.002	0.998
	512	1.000	1.000	1.000

Table 2.1
Computational effort of various sub-sampled algorithms for $s = \{512, 256, 128\}$ relative to $s = 512$. Three matrices are testes: **Rand** XX^T ($n = 5000$), with $X \sim \mathcal{N}(0,1)^{n\times n}$; **Gisette Scale** ($n = 5000$) Hessian [5]; and **NASA4704** ($n = 4704$) [7]. Values of 1 indicate that the method is converging at the same rate as having sampled with $s = 512$ (with respect to the total number of matrix samples cost metric).

2.4 Heurestic Accelerated Schemes

Motivated by the sub-sampled analysis, a heuristic accelerated scheme is presented. Numerical convergence and acceleration is verified in section 2.4.2. Lastly, observations are made comparing the heuristic scheme to other accelerated sampled algorithms in section 2.4.3, and how algorithm 4 can be interpreted as a modified block Krylov method section 2.4.4.

2.4.1 Eigenvector Acceleration

The update underlying algorithm 2 samples and then corrects the sample mismatch in the residual $R_k = A - B_k$. Larger corrections (and consequently more significant improvements in the approximation B_{k+1}) occur if $U^T R_k U$ is large. Block-power iteration on R_k is a simple heuristic to enhance subspaces associated with the larger eigenvalues of R_k. Algorithm 4 summarizes an extension to algorithm 2 by incorporating a fixed number, p, of inner block-power iterations. As before, work estimates are boxed on the right (p steps of a block power iteration involving $p\,n\,s$ matrix samples and a square symmetric sample involving s^2 matrix samples) at each step with the total double boxed on the result line. This is not a sub-sampled algorithm (each internal power iteration involves a sample) and involves significantly more matrix samples per iteration. Despite this, algorithm 4 is competitive for small values of p.

Algorithm 4 SS1A: Accelerated Symmetric Approximation

```
Require: $B_0 \in \mathbb{R}^{n\times n}$ satisfying $B_0^T = B_0$, SPD $W \in \mathbb{R}^{n\times n}$, $s \in \mathbb{N}$.
repeat $\{k = 0, 1, \ldots\}$
    Sample $U_{0,k} \sim \mathcal{N}(0,1)^{n\times s}$
    $B_{0,k} = B_k$
    loop $\{i = 1, 2, \ldots, p\}$
        $\Lambda = AU_{i-1,k} - B_{i-1,k}U_{i-1,k}$
        $\Sigma = \Lambda(U_{i-1,k}^T W U_{i-1,k})^{-1} U_{i-1,k}^T W$
        $B_{i,k} = B_{i-1,k} + \Sigma + \Sigma^T - WU_{i-1,k}(U_{i-1,k}^T W U_{i-1,k})^{-1} U_{i-1,k}^T \Sigma$
        $U_{i,k} = \Lambda$
    end loop .............................................. $\boxed{p\,n\,s}$
    Compute residual $\Lambda_k = U_{p,k}^T A U_{p,k} - U_{p,k}^T B_{p,k} U_{m,k} \in \mathbb{R}^{s\times s}$ ................ $\boxed{s^2}$
    Compute $\tilde{P}_k = W\,U_{p,k}(U_{p,k}^T W\,U_{p,k})^{-1}$
    Update $B_{k+1} = B_k + \tilde{P}_k \Lambda_k \tilde{P}_k^T$
until convergence
return $B_{k+1}$ ........................................ $\boxed{\boxed{(k+1)(p\,n\,s + s^2)}}$
```

Remark 15. *Implementing similar acceleration for algorithm 3 would target the input/output spaces of the interior non-symmetric updates. For symmetric target matrices A, the residual R_k is symmetric and little acceleration is realized unless the input and output spaces match as in algorithm 4.*

2.4.2 Acceleration Convergence Results

We now compare the performance of algorithm 4 SS1A, (with rotationally invariant sampling and $p = 2$) to various algorithms: S1, BFGS, DFP, and a re-interpretation of the heuristic accelerated BFGS algorithm from [14] which is termed BFGSA. Specifically, BFGSA is obtained by applying the Sherman-Morrison-Woodbury formula to the the adaptively sampled algorithm AdaRBFGS in [14], which approximates A^{-1}. The sampled algorithm, S1, is the B formulation in eq. (2.3) with rotationally invariant weight $W = I_n$.

The convergence (relative Frobenius residual $\|A - B_k\|_F / \|A - B_0\|_F$ against matrix samples) of accelerated algorithms with sample size $s = \lceil \sqrt{n} \rceil$ from [14] are compared to the heuristically accelerated algorithm 4 with $s_1 = s_2 = s$ on the three matrices from section 2.3: ($n = 5000$) XX^T with $X \sim \mathcal{N}(0, 1)^{n \times n}$ Figure 2.4; ($n = 5000$) Gisette-Scale [5] Figure 2.5; and ($n = 4704$) NASA [7] Figure 2.6. These figures show: BFGSA ($\ast$) as specified by eq. (2.5) with adaptive sampling described in [14]; S1 ($\circ$) as specified by eq. (2.3); SS1A (⬠) as specified by Algorithm 4; BFGS ($\diamond$) as specified by eq. (2.5); DFP ($\diamond$) as specified by eq. (2.4). Runs were terminated after $5n^2$ iterations or when the relative residual norm fell below 10^{-2}. The results show SS1A matching or outperforming the other algorithms for the three matrices from section 2.3.1. Further accelerated experiments are discussed in section A.4.

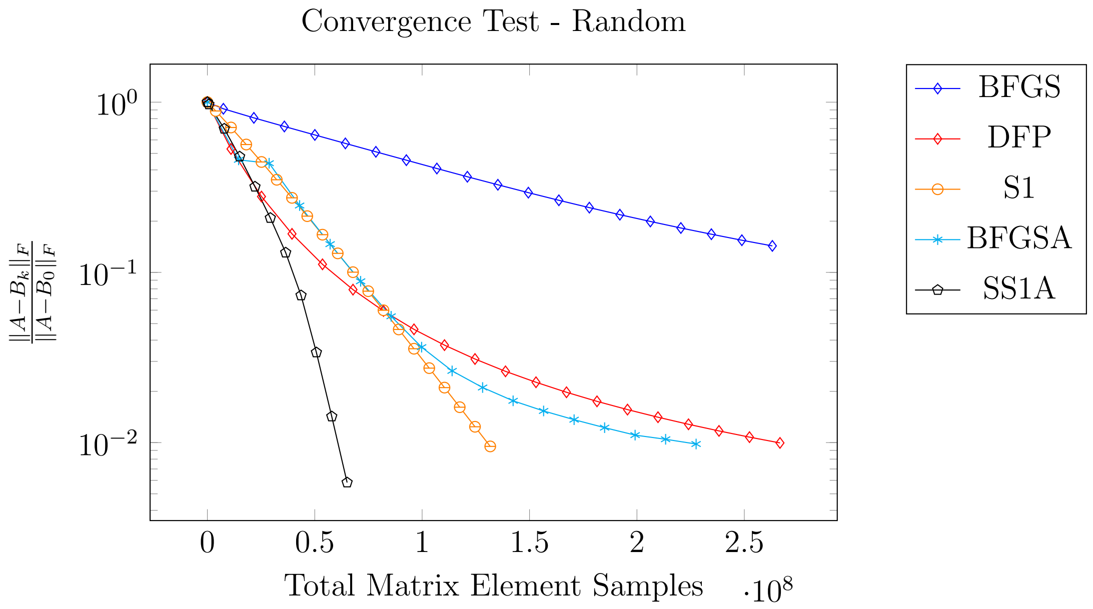


Figure 2.4: ($n = 5000$) Approximation of XX^T where $X \sim \mathcal{N}(0,1)^{n\times n}$ with $s = 71$. Acceleration of the BFGSA method ($*$, adapted from [14]) can be seen in comparison to the BFGS method ($\diamond$). Acceleration of the algorithm 4 (⬠) can be seen in comparison to S1 ($\circ$). The acceleration of algorithm 4 continues as the method is targeting the dominant space of the residual at each step.

2.4.3 Relationship to Algorithms in Literature

We revisit the algorithms that fall in the general framework described in [16]. Recall that such algorithms construct a single (expensive) sub-sample, $Q^* A\, Q$, to approximate the action of A associated with its dominant eigenspace. This matrix Q can be computed using a modified block power method, as described in algorithm 5.

Further, recall that for SPD A, the sub-sampled data is embedded using the low-rank approximation $\mathcal{P}_{I_n,Q}\, A\, \mathcal{P}_{I_n,Q}^T$. Hence, algorithm 5 can be viewed as a single outer loop

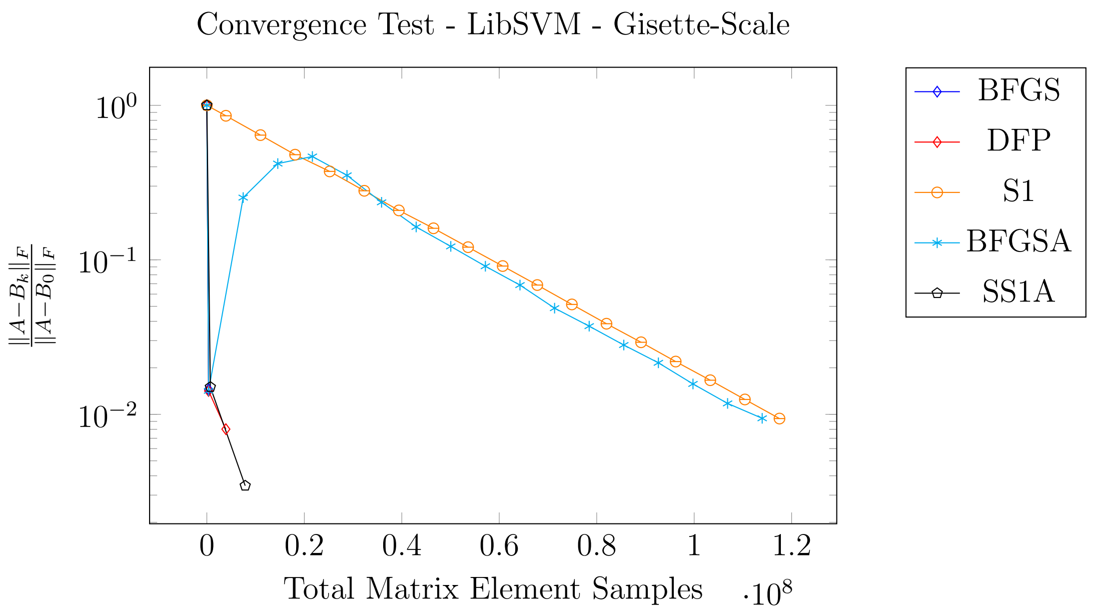


Figure 2.5: ($n = 5000$) Approximation of Hessian from **Gisette Scale** [5] with $s = 71$. The DFP and BFGS updates perform well as does algorithm 4. BFGSA ($*$) performs well initially with slow terminal convergence.

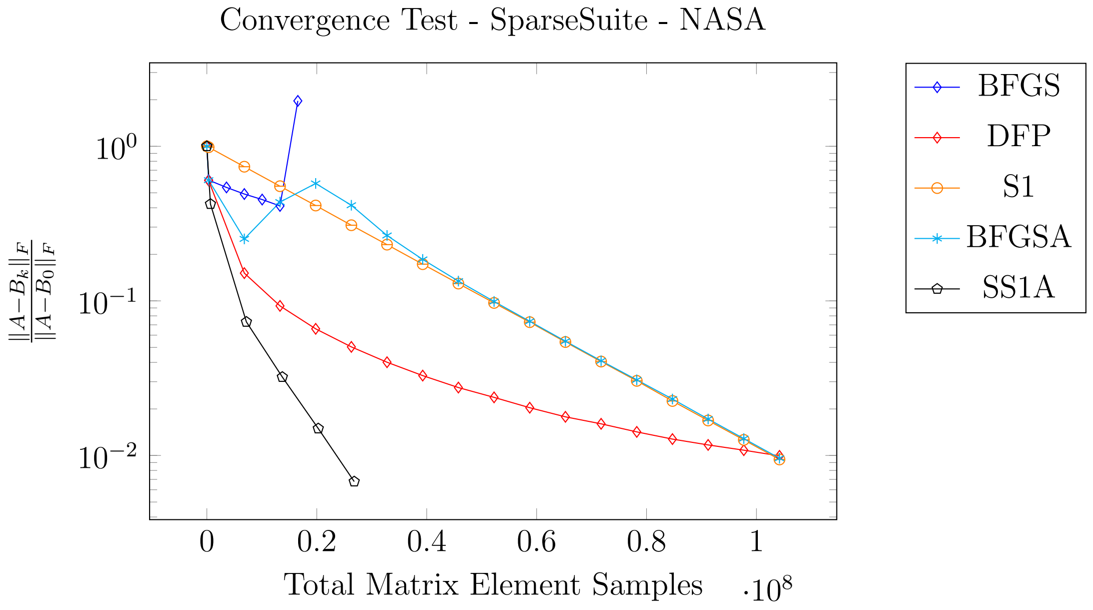


Figure 2.6: ($n = 4700$) Approximation of **NASA4704** from [7] with $s = 69$. The DFP method performs well while the BFGS method fails. The accelerated BFGSA method shows inconsistent convergence while algorithm 4 shows consistent acceleration in comparison to the consistent sampled method S1.

Algorithm 5 RSSI: Randomized Subspace Iteration (Stage A) [16]

```
Sample $U \sim \mathcal{N}(0,1)^{n \times s}$
Compute $Y_0 = A\,U$ ........ $\boxed{n\,s}$
Compute QR-decomposition $Y_0 = Q_0\,R_0$
loop $\{i = 1, 2, \ldots, p\}$
    Compute $\tilde{Y}_i = A^*\,Q_{i-1}$
    Compute QR-decomposition $\tilde{Y}_i = \tilde{Q}_i\,\tilde{R}_i$
    Compute $Y_i = A\,\tilde{Q}_i$
    Compute QR-decomposition $Y_i = Q_i\,R_i$
end loop ........ $\boxed{2\,p\,n\,s}$
return $Q_p$ ........ $\boxed{\boxed{(2\,p+1)(n\,s)}}$
```

of algorithm 4 with the modification that intermediate data $\Lambda = AU_{i-1,k} - B_{i-1,k}U_{i-1,k}$ is not used.

2.4.4 Krylov Spaces

Block Krylov method [20] computes a low-rank matrix approximation by searching the Krylov space $\mathcal{V}_p(U_{0,k})$ of A, where

$$\mathcal{V}_p(U_{0,k}) = \text{span}\{A\,U_{0,k}, (A\,A^T)A\,U_{0,k}, \ldots, (A\,A^T)^{p-1}\,A\,U_{0,k}\}.$$

Algorithm 6 is adapted from the description given in algorithm 2 of [20].

Algorithm 6 Block Krylov Iteration (Stage A) [20]

```
Sample $U \sim \mathcal{N}(0,1)^{n \times s}$
Compute $K = \left[A\,U_{0,k} | (A\,A^T) A\,U_{0,k} | \ldots | (A\,A^T)^{p-1}\,A\,U_{0,k}\right]_{m \times p\,s}$
Compute QR-decomposition $K = Q\,R$
return $Q$ ............................................ $\boxed{\boxed{(2\,p-1)(n\,s)}}$
```

Algorithm 4 can also be viewed as a modified block Krylov method. Each inner iteration builds approximations in the space

$$\operatorname{span}\{U_{0,k}, (A - B_{0,k})U_{0,k}, (A - B_{1,k})(A - B_{0,k})U_{0,k}, \ldots, \left(\prod_{i=0}^{p-1}(A - B_{i,k})\right) U_{0,k}\},$$

which approximates the Krylov space $\mathcal{V}_p(U_{0,k})$ of the residual $A - B_k$. In Algorithm 4 each intermediate space $U_{i-1,k} \approx (A - B)^i\,U_{0,k}$ is only stored during one inner iteration and the Krylov matrix K is never formed.

Chapter 3

Inverse Approximation

3.1 Introduction

Approximations of the inverse of a matrix are used to accelerate many algorithms [21, 22]. For example, inverse matrix approximations can be used as pre-conditioners [1, 19, 21, 22] using the limited memory BFGS update [24]. More recently, accelerated inverse matrix approximations [14] were developed using classical block Quasi-Newton matrix updates [4].

Current methods for approximating the inverse of a matrix $A \in \mathbb{R}^{n \times n}$ sample the matrix A by computing $U^T A$ and/or $A V$, as described in chapter 1. Inverse approximations are obtained by applying the Sherman-Morrison-Woodbury (SMW) formula

[29] at each step, generating an iterative method for inverse approximation. The SMW formula converts a rank s update for a matrix to the corresponding update for the inverse matrix. This approach was used in the classical BFGS and DFP quasi-newton optimization methods.

3.2 Relationship to Quasi-Newton Algorithms

The block BFGS update is derived in the H formulation (see eq. (2.2)) yielding an inverse matrix approximation which is then heuristically accelerated [14]. The block DFP update is derived in the B formulation (matrix approximation, see eq. (2.1)) but an application of the SMW yields an inverse matrix approximation which is highly competitive and seems to have seen little to no usage or mention in literature. The linear algebraic update that results in applying the SMW formula to eq. (2.4) is

$$H_{k+1} = H_k - H_k\, A\, U \left(U^T\, A^T\, H_k\, A\, U\right)^{-1} U^T\, A^T\, H_k^T + U\, U^T\, A\, U\, U^T. \tag{3.1}$$

Remark 16. *The application of the SMW formula to a minimum change formulation does not produce a minimum change inverse update. That is, the block BFGS B formula is not a minimum change in the context of eq.* (2.1)*. Similarly the block DFP H formula is not a minimum change in the sense of eq.* (2.2)*. They are the inverse of a minimum change update, that is $B_{k+1}\, H_{k+1} = H_{k+1}\, B_{k+1} = I_n$.*

3.3 Sub-Sampled Inverse Approximation

The sub-sampled methods in algorithms 1 to 4 produce matrix approximations as described in chapter 2 using the generic sub-sampled data $U^T A V$. For inverse approximation, symmetric sub-sample $U^T A U$ will be used. For random $U \sim \mathcal{N}(0,1)^{n \times s}$, this data contains a projection of each invariant space of A. The dominant spaces with larger eigenvalues are represented proportionately more than those with smaller eigenvalues. This allows the sub-sampled methods to better approximate the dominant eigenspace of A. Application of the SMW formula to `SS1` yields a method `SS1SMW` which at each iteration constructs the inverse of the matrix approximate iterate.

The sub-samples favors the dominant eigenspace of the original matrix, in the sense that they contain larger amounts of the dominant spaces per update. Any error in the least dominant eigenspace will be amplified in the corresponding inverse matrix approximation when one applies the SMW formula to such an update. Nevertheless inverse approximations which minimize $\|A^{-1} - H\|$ are possible. With the addition of appropriate filtering at each iteration, approximations which minimize $\|I - H A\|$ will be shown to be quite effective and competitive to current methods. The norm $\|I - H A\|$ is common metric from the sparse inverse approximation literature (see [2, 6] for example).

Filtering of the data sub-sampled from the residual $\Lambda_k = U_k^T A U_k - U_k^T B_k U_k$ is required for numerical stability of the initial iterates. For matrices with very small singular values (eigenvalues for $A = A^T$) or ill-conditioned matrices A the un-filtered updates with have oscillatory errors which eventually correct themselves. For the unknowing user, such errors make the approximations less helpful as one would need to detect or know when to terminate the iteration. Algorithms 1 to 3 are all self-correcting and application of the SMW will yield convergent inverse approximations in the sense of $\|A^{-1} - H\|$, as the iterates will eventually give the inverse of the matrix approximations which converge. Application of a filter prevents the convergence of $H_k \to A^{-1}$ but expediently computes a matrix H which is the inverse of a low-rank approximation of A. The inverse matrix approximation can reduce the norm $\|I - H\,A\|$ used in [14]. The resulting algorithm is described in algorithm 7 and the filtering methods are given in section 3.3.1.

Algorithm 7 SS1SMW: Sub-Sampled Matrix Inverse Approximation

```
Require: $B_0 \in \mathbb{R}^{n \times n}$ and $H_0 = B_0^{-1}$ SPD, $s \in \mathbb{N}$.
repeat $\{k = 0, 1, \ldots\}$
    Sample $U_k \sim \mathcal{N}(0,1)^{n \times s}$
    Compute residual $\Lambda_k = U_k^T A\, U_k - U_k^T B_k U_k \in \mathbb{R}^{s \times s}$
    $\Lambda_k = \text{Filter}(\Lambda_k)$
    Update $B_{k+1} = B_k + U_k \Lambda_k U_k^T$
    Update $H_{k+1} = H_k - H_k\, U_k (\Lambda_k^{-1} + U_k^T H_k\, U_k)^{-1} U_k^T H_k$
until convergence
return $B_{k+1}$ and $H_{k+1}$
```

3.3.1 Filtering

Filtering matrix approximations to maintain the SPD property is commonly used in the implementation of various quasi-newton optimization procedures. The data $\Lambda_k \in \mathbb{R}^{s \times s}$ has a user controlled fixed storage requirement and computation of the singular value (eigenvalue) decomposition, or any other matrix factorization, are attainable at a (cheap) fixed computational cost as the user is able to choose the sampling dimension $s \ll n$. Further, in practice one can use whatever factorization is done in the filter step as an inexpensive ”on-the-fly” check to determine when the algorithms have reached their limit in terms of minimizing $\|I - H\,A\|$. The minimum relative norm $\frac{\|I-H_k\,A\|}{\|I-H_0\,A\|}$ will depend in practice on the spectrum of A, but detection of when the algorithm nears this minimum can be done cheaply.

As is noted in chapter 2, the sub-sampled update `SS1` does not maintain the SPD property for B_k at each iteration. Application of the SMW formula to the unweighted `SS1` update ($W = I_n$) produces an update having the following projector

$$\mathcal{P}_{SMW} = P_{B_k,U} = H_k\,U(U^T\,H_k\,U)^{-1}U^T. \tag{3.2}$$

This projection is only properly defined when B_k is SPD so a filtering method which maintains this at each iteration will produce sub-sampled algorithms which minimize the sub-sampling error as described above. The following is a description of some common filtering techniques used to maintain the SPD property in optimization literature based on the eigenvalue decomposition $\Lambda_k = X\,\Omega\,X^T$.

$$\text{Filter}(\Lambda_k) = X\,|\Omega|\,X^T,$$
$$\text{Filter}(\Lambda_k) = \tilde{X}\,\tilde{\Omega}\,\tilde{X}^T,$$

where $|\Omega|$ replaces the eigenvalues with their absolute value and $\tilde{\Omega}$ drops search directions which had negative eigenvalues and updates on the reduced set of directions $\tilde{X}$.

3.4 Sub-Sampled Accelerated Inverse Approximations

The general framework of the hybrid accelerated method algorithm 4 applies p steps of the block power iteration updating with the intermediate data from each inner step using a sampled algorithm. The last piece of data is then used as a proxy for the dominant eigenspace of the residual at that iteration and an accelerated sub-sample is used. For inverse matrix approximation, one can apply an inverse matrix approximation such as BFGS-H or DFP-H at each intermediate step. Applying the SMW formula to the sub-sampled update in the accelerated step gives an accelerated inverse matrix approximation. Doing so in this setting where the sampled sub-space has been enriched produces an algorithm which avoids the sub-sampling issues described.

Algorithm 8 produces an approximation H to its inverse A^{-1} and is competitive with current state of the art methods.

Algorithm 8 SSInv: Accelerated Inverse Symmetric Approximation

```
Require: $H_0, B_0 \in \mathbb{R}^{n\times n}$ satisfying $B_0^T = B_0, H_0^T = H_0$, SPD $W \in \mathbb{R}^{n\times n}$, $s \in \mathbb{N}$.
repeat $\{k = 0, 1, \ldots\}$
    Sample $U_{0,k} \sim \mathcal{N}(0,1)^{n\times s}$
    $B_{0,k} = B_k$
    loop $\{i = 1, 2, \ldots, p\}$
        $AU = AU_{i-1,k}$
        $\mathcal{P}_{\mathrm{DFP}} = AU(U_{i-1,k}^T AU)^{-1} U_{i-1,k}^T$
        $B_{i,k} = (I_n - \mathcal{P}_{\mathrm{DFP}})\, B_{i-1,k}(I_n - \mathcal{P}_{\mathrm{DFP}}^T) + \mathcal{P}_{\mathrm{DFP}}\, A$
        $H_{i,k} = H_{i-1,k} - H_{i-1,k}\, AU \left(U^T A\, H_{i-1,k}\, AU\right)^{-1} U^T A\, H_{i-1,k} + U\, U^T\, AU\, U^T$
        $U_{i,k} = AU - B_{i-1,k} U_{i-1,k}$
    end loop ........................................................ $\boxed{p\,n\,s}$
    Compute residual $\Lambda_k = U_{p,k}^T AU_{p,k} - U_{p,k}^T B_{p,k} U_{m,k} \in \mathbb{R}^{s\times s}$ ................ $\boxed{s^2}$
    Compute $\tilde{P}_k = W\, U_{p,k}(U_{p,k}^T W\, U_{p,k})^{-1}$
    Update $B_{k+1} = B_{p,k} + \tilde{P}_k \Lambda_k \tilde{P}_k^T$
    Update $H_{k+1} = H_{p,k} - H_{p,k} U_{p,k} \left(\Lambda_k^{-1} + U_{p,k}^T H_{p,k} U_{p,k}\right)^{-1} U_{p,k}^T H_{p,k}$
until convergence
return $H_{k+1}$ .............................................. $\boxed{\boxed{(k+1)(p\,n\,s + s^2)}}$
```

Remark 17. *The sub-sampled matrix update eq.* (2.7) *works on the residual* $A - B_k$*. Applying the SMW formula yields the update to* H_{k+1} *in algorithm 8 seen in the last line. The update requires the residual* $A - B_k$ *and inverting two* $s \times s$ *matrices. Due to* *these requirements the SSInv algorithm simultaneously builds a matrix and inverse matrix approximation from the samples and accelerated sub-samples of A.*

3.4.1 Inverse Power method

The inverse matrix approximation algorithm 8 constructs the inverse of the matrix approximation from algorithm 4. The acceleration used in that setting targets the dominant eigenspace of the residual $A-B_k$. The heuristic acceleration can be adapted to approximate the dominant space of the matrix residual $B-A$ or the inverse residual $I - H\,A$. For inverse matrix approximation targeting the dominant eigenspace of the inverse residual $A^{-1} - H_k$ may yield an improved convergence rate.

The above can be understood in terms of the projectors used in the B and H formulations eq. (2.3). In the inverse (H) formulation the projector acts on the inverse residual $A^{-1} - H_k$. When constructing an approximation of A^{-1} one cannot directly target this space, but the following can be done.

$$\begin{aligned}(A^{-1} - H_k)A\,U &= (A^{-1}A - H_k\,A)U \\ &= (I_n - H_k\,A)U \\ &= U - H_k\,A\,U\end{aligned}$$

The above gives a weighted sample of the inverse residual using only the original samples AU. The following modified algorithm targets the inverse residual in this way and is also competitive against current state of the art methods.

Algorithm 9 SSInvP: Accelerated Inverse Symmetric Approximation

Require: $H_0, B_0 \in \mathbb{R}^{n\times n}$ satisfying $B_0^T = B_0, H_0^T = H_0$, SPD $W \in \mathbb{R}^{n\times n}$, $s \in \mathbb{N}$.

```
repeat $\{k = 0, 1, \ldots\}$
    Sample $U_{0,k} \sim \mathcal{N}(0,1)^{n\times s}$
    $B_{0,k} = B_k$
    loop $\{i = 1, 2, \ldots, p\}$
        $AU = AU_{i-1,k}$
        $\mathcal{P}_{\text{DFP}} = AU(U_{i-1,k}^T\, AU)^{-1} U_{i-1,k}^T$
        $B_{i,k} = (I_n - \mathcal{P}_{\text{DFP}})\; B_{i-1,k}(I_n - \mathcal{P}_{\text{DFP}}^T) + \mathcal{P}_{\text{DFP}}\, A$
        $H_{i,k} = H_{i-1,k} - H_{i-1,k}\, AU\left(U^T\, A\, H_{i-1,k}\, AU\right)^{-1} U^T\, A\, H_{i-1,k} + U\, U^T\, AU\, U^T$
        $U_{i,k} = U_{i-1,k} - H_{i-1,k} AU$
    end loop ........................................................ $\boxed{p\,n\,s}$
    Compute residual $\Lambda_k = U_{p,k}^T AU_{p,k} - U_{p,k}^T B_{p,k} U_{m,k} \in \mathbb{R}^{s\times s}$ ................ $\boxed{s^2}$
    Compute $\tilde{P}_k = W\, U_{p,k}(U_{p,k}^T W\, U_{p,k})^{-1}$
    Update $B_{k+1} = B_{p,k} + \tilde{P}_k \Lambda_k \tilde{P}_k^T$
    Update $H_{k+1} = H_{p,k} - H_{p,k} U_{p,k}\left(\Lambda_k^{-1} + U_{p,k}^T H_{p,k} U_{p,k}\right)^{-1} U_{p,k}^T H_{p,k}$
until convergence
return $H_{k+1}$ .......................................... $\boxed{\boxed{(k+1)(p\,n\,s + s^2)}}$
```

3.5 Numerical Results

Our inverse hybrid algorithms algorithms 8 and 9 are tested on a variety of SPD matrices: $A = XX^T$, $X \sim \mathcal{N}(0,1)^{n\times n}$; ridge regression matrices chosen from [5]; and matrices chosen from the Sparse Suite Library [7]. Algorithms 8 and 9 were implemented within the MATLAB [18] code framework in [14] and tested on the same problems from [5, 7]. All computational tests were performed on **Superior**, a high-performance computing infrastructure at Michigan Technological University.

The convergence,

$$\frac{\|I_n - H_k A\|_F}{\|I_n - H_0 A\|_F},$$

of sampled algorithms [14] with sample size $s = \lceil\sqrt{n}\rceil$ are compared to the hybrid algorithms with $s_1 = s_2 = s$ on three matrices: $(n = 5000)$ XX^T with $X \sim \mathcal{N}(0,1)^{n\times n}$ Figure 3.1; $(n = 5000)$ Gisette-Scale [5] Figure 3.2; and $(n = 4704)$ NASA [7] Figure 3.3. These figures show: AdaRBFGS$_{cols}$($\otimes$) as specified by [14]; AdaRBFGS$_{gauss}$ (●) as specified by [14]; DFP (◇) as specified by eq. (2.4); SSInv (○) as specified by Algorithm 8; SSInvP (⬠) as specified by Algorithm 9.

Runs were terminated after $5n^2$ iterations or when the relative residual norm fell below 10^{-2}.

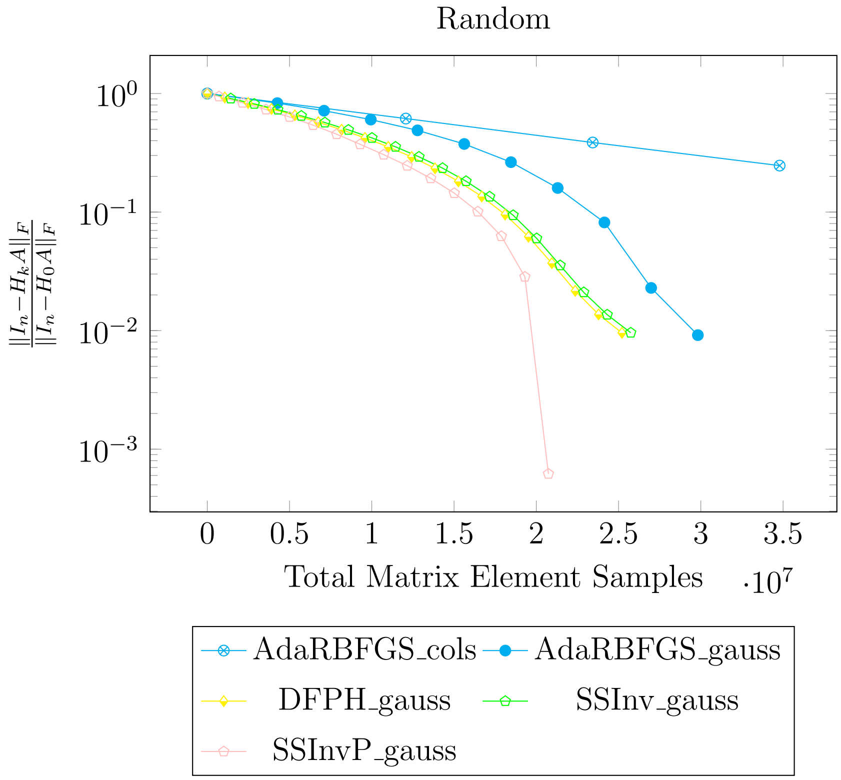


Figure 3.1: ($n = 5000$) Approximation of the inverse of XX^T where $X \sim \mathcal{N}(0,1)^{n\times n}$ with $s = 71 = \lceil\sqrt{5000}\rceil$. AdaRBFGS$_{cols}$ performs slowly compared to AdaRBFGS$_{gauss}$. The DFPH update and algorithm 8 show improvement compared to AdaRBFGS$_{gauss}$ while algorithm 9 shows an increased terminal convergence rate.

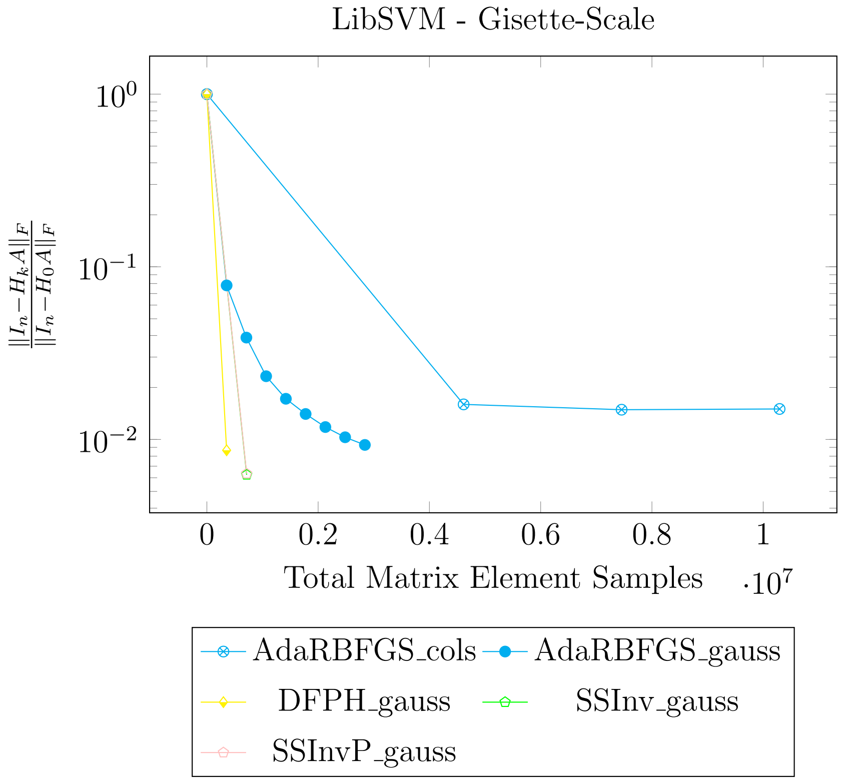


Figure 3.2: ($n = 5000$) Approximation of the inverse of the Hessian from **Gisette Scale** [5] with $s = 71 = \lceil\sqrt{5000}\rceil$. AdaRBFGS$_{gauss}$ and AdaRBFGS$_{cols}$ initially performs well and then slow down. The DFPH update and algorithms 8 and 9 perform well.

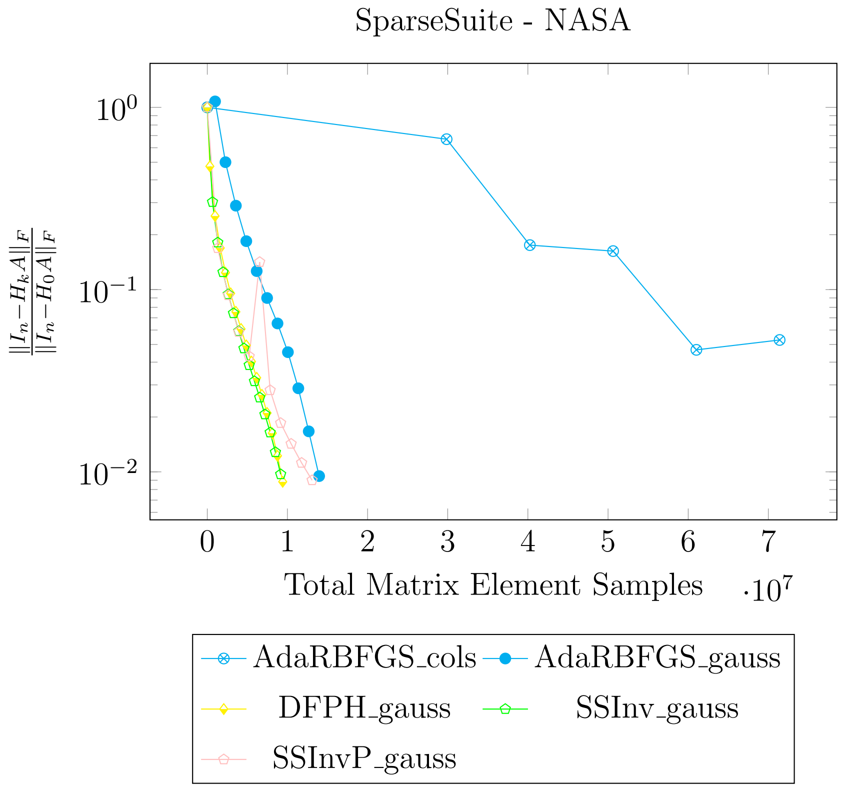


Figure 3.3: ($n = 4700$) Approximation of the inverse of **NASA4704** matrix from [7] with $s = 69 = \lceil\sqrt{4704}\rceil$. AdaRBFGS$_{cols}$ shows slow convergence. The DFPH method, AdaRBFGS$_{gauss}$, and algorithm 8 performs consistently while algorithm 9 shows an approximation error which quickly corrects itself.

Chapter 4

Low Rank Approximation

4.1 Introduction

Typical data sets arising from a wide range of applications in scientific computing and data analysis are very large but have an underlying (effectively) low rank structure (see [16] for a list of applications). This structure can be explored to compare and analyse data (see [11] for examples) or provide useful starting conditions for iterative methods (see [1, 19]). Specifically, low rank approximations of the data set can be used to compress the information and indeed provide best approximations (see [16] for general analysis).

The optimal low rank approximation is the truncated SVD. Given a target matrix A

and desired rank l:

$$A_l = \arg\min_B \{\|A - B\| \mid \text{rank}(B) \leq l\}, \tag{4.1}$$

is the best rank l matrix approximating A. A_l can be computed directly from the reduced SVD $A_l = U_l \, \Sigma_{ll} \, V_l^T$ where $\text{rank}(U_l) = \text{rank}(V_l) = l$ and Σ_{ll} is a diagonal matrix. The column space of U_l spans the most dominant column space for the matrix A. The column space of V_l spans the most dominant row space. In this way the low-rank approximation A_l captures most of the action of the matrix A and will provide approximate solutions to $Ax = b$ with minimal error.

Randomized methods [16] approximate this dominant space by sampling the column space $U^T A$ or row space $A V$. Using an oversampling parameter to accelerate the convergence, authors of [16] construct rank r approximations using samples of the matrix via $A\,Q$ where $Q \in \mathbb{R}^{n \times r}$ where $r = l + q$. The additional q sampling directions will give more information for the best l directions and as shown in [16] is crucial to the convergence of their methods.

In this thesis, approximations are constructed sampling both row and column spaces simultaneously at each iteration by evaluating the sub-sample $U^T A\, V \in \mathbb{R}^{s_1 \times s_2}$, where $U \in \mathbb{R}^{m \times s_1}$ and $V \in \mathbb{R}^{n \times s_2}$. In many settings, sub-samples can be computed with less memory overhead and computation. Constructing approximations this way requires iterating until the necessary amount of data has been accumulated. Each iteration requires computation of $s_1 s_2$ output elements and the data is accumulated until the desired convergence is reached, a maximum amount of computation is done, or a maximum amount of matrix access are done.

4.2 Low Rank Sub-Sampled Approximation

Initializing algorithms 1 to 3 with $B_0 = 0 \cdot I_n$ will give iterates whose ranks increase by s each iteration. In this setup, the iterates quickly become full-rank well before the convergence is realized. As a result, producing low-rank approximations require compression of the update $\hat{B}_{k+1} \to B_{k+1}$ at each iteration.

4.2.1 Generic Low Rank Update

Inserting a subroutine for the compression at each iteration of algorithms 1 to 3 will yield low rank approximations. A pseudo-code description is below.

Algorithm 10 SSLR: Sub-Sampled Low-Rank Approximation

```
Require: $B_0 \in \mathbb{R}^{n\times n}$, SPD $W \in \mathbb{R}^{n\times n}$, $s_1 \in \mathbb{N}$, $\hat{\gamma} \in \mathbb{R}$.
1: repeat $\{k = 0, 1, \dots\}$
2:     Update $\hat{B}_{k+1} = \text{SSUpdate}[B_k]$
3:     $B_{k+1} = \text{Compress}[\hat{B}_{k+1}]$
4: until convergence
5: return $B_{k+1}$ .............................................. $\boxed{(k+1)\,(s_1^2)}$
```

4.2.2 Compression

The sub-sampled update step (line 2 of algorithm 10) increases the rank of B_k by s (with probability 1) for the updates given in algorithms 1 to 3 or $(p+1)s$ when using the accelerated update from algorithm 4. To generate a low rank approximation, one needs to reduce the ranks periodically throughout the iteration. Constructing a factorization of the iterate and truncating to a fixed rank r, a max rank r_k (where $r_k \leq 2s$ for example), or an adaptive rank r_k such that

$$\|\hat{B}_{k+1} - B_{k+1}\| \leq \hat{\gamma}\|A\|, \tag{4.2}$$

compresses the approximation at each iteration. This allows a user to have control over the storage per iterate as well as the amount of computation. The numerical experiments will show that the parameter $\hat{\gamma}$ will control the rank of iterates and can be used to target the best approximation of a matrix.

Remark 18. *It should be noted with a tolerance based scheme, such as eq.* (4.2)*, the iterates may still increase in rank initially. Different values of $\hat{\gamma}$ will allow the rank of the iterates to reduce as $B_{k+1} \to A_l$ with weak dependence on the singular value distribution of A.*

4.3 Convergence Analysis

The convergence of algorithm 10 with the tolerance based compression given in eq. (4.2) depends on the parameter $\hat{\gamma}$ and the underlying convergence of algorithms 1 to 3. Recall that theorems 7, 9 and 10 give bounds of the form

$$\mathbf{E}\left[\|B_{k+1} - A\|^2_{F(W_1^{-1},W_2^{-1})}\right] \leq \rho^{k+1}\mathbf{E}\left[\|B_0 - A\|^2_{F(W_1^{-1},W_2^{-1})}\right].$$

The following corollary gives related bound for $\mathbf{E}\left[\|B_{k+1} - A\|_{F(W_1^{-1},W_2^{-1})}\right]$.

Corollary 19. *The expected value of the norm of the residuals for algorithms 1 to 3 satisfies,*

$$\boldsymbol{E}\left[\|B_{k+1} - A\|_{F(W_1^{-1},W_2^{-1})}\right] \le (\rho_{NS})^{(k+1)/2}\boldsymbol{E}\left[\|B_0 - A\|_{F(W_1^{-1},W_2^{-1})}\right] \tag{4.3}$$

$$\boldsymbol{E}\left[\|B_{k+1} - A\|_{F(W^{-1})}\right] \le (\rho_{SS1})^{k+1)/2}\boldsymbol{E}\left[\|B_0 - A\|_{F(W^{-1})}\right] \tag{4.4}$$

$$\boldsymbol{E}\left[\|B_{k+1} - A\|_{F(W_1^{-1},W_2^{-1})}\right] \le (\rho_{SS2})^{k+1)/2}\boldsymbol{E}\left[\|B_0 - A\|_{F(W_1^{-1},W_2^{-1})}\right]. \tag{4.5}$$

Proof. Let $R_{k+1} = W_1^{1/2}(B_{k+1} - A)W_2^{1/2}$ ($W_1 = W_2$ for algorithm 2) and let $\|\cdot\|_F$ be the appropriate norm. For each sub-sampled algorithm,

$$\mathbf{E}\left[\|R_{k+1}\|_F\right] = \mathbf{E}\left[\sqrt{\|R_{k+1}\|_F^2}\right] \le \sqrt{\mathbf{E}\left[\|R_{k+1}\|_F^2\right]},$$

where the last inequality arises from applying Jensen's inequality to the square root function. Applying the results of theorems 7, 9 and 10 and taking the square root gives

$$\mathbf{E}\left[\|R_{k+1}\|_F\right] \le \sqrt{\rho^{k+1}\mathbf{E}\left[\|R_0\|_F^2\right]} = \rho^{(k+1)/2}\sqrt{\|R_0\|_F^2}$$

after noting $\mathbf{E}\left[\|R_0\|_F^2\right] = \|B_0 - A\|_F^2$ as B_0 and A are constant. □

Remark 20. *Taking a square root and applying Jensen's inequality as above in the proofs of theorems 7, 9 and 10 produces the same results as theorem 19.*

4.3.1 Mathematical Preliminaries

The compression of each iterate introduces an error term. The following lemma will be of use to bound the accumulation of these errors.

Lemma 21. *If* $a_{k+1} \leq \mu\, a_k + \gamma$ *with* $0 \leq \mu < 1$, $\gamma \geq 0$ *and* $a_0 > 0$ *then*

$$a_n \leq \mu^n a_0 + \frac{\gamma}{1-\mu}. \tag{4.6}$$

Proof. Writing out the summation and observing a telescoping sum gives the equality. Applying the assumption ($a_{k+1} \leq \mu\, a_k + \gamma$) gives the finite geometric series with initial term γ and rate μ which is bounded by the infinite geometric sum.

$$a_n - \mu^n a_0 = \sum_{k=0}^{n-1} \mu^k \left(a_{n-k} - \mu a_{n-k-1}\right) \leq \sum_{k=0}^{n-1} \mu^k \gamma \leq \frac{\gamma}{1-\mu} \tag{4.7}$$

□

4.3.2 Convergence Theorem

Theorem 22 (Convergence of Tolerance Low Rank algorithm 10)**.** *Let $A \in \mathbb{R}^{m\times n}$ and fix SPD weight matrices $W_1 \in \mathbb{R}^{m\times m}$ and $W_2 \in \mathbb{R}^{n\times n}$. If $U_k \in \mathbb{R}^{m\times s_1}$ and $V_k \in \mathbb{R}^{n\times s_2}$ are random, independently selected matrices with full column rank (with probability one), then algorithm 10, with SSUpdate given by algorithms 1 to 3, generates a sequence, B_k, from an initial guess $B_0 \in \mathbb{R}^{m\times n}$ satisfying*

$$\boldsymbol{E}\left[\|B_{k+1} - A\|_F\right] \leq (\rho)^{(k+1)/2}\boldsymbol{E}\left[\|B_0 - A\|_F\right] + \frac{\hat{\gamma}\|A\|_F}{1-\rho^{1/2}},$$

where ρ and $\|\cdot\|_F$ are the appropriate convergence rates and norms from theorems 7, 9 and 10 and $\hat{\gamma} > 0$.

Proof. Let $\hat{B}_{k+1}$ be the un-compressed update in algorithm 10 with compression given by eq. (4.2), then

$$\begin{aligned}\mathbf{E}\left[\|B_{k+1} - A\|_F\right] &= \mathbf{E}\left[\left\|B_{k+1} - B\hat{}_{k+1} + B\hat{}_{k+1} - A\right\|_F\right]\\ &\leq \mathbf{E}\left[\left\|B_{k+1} - B\hat{}_{k+1}\right\|_F\right] + \mathbf{E}\left[\left\|B\hat{}_{k+1} - A\right\|_F\right],\end{aligned}$$

where the last steps are given by the triangle inequality and linearity of the expected value. Applying the compression inequality eq. (4.2) we have

$$\mathbf{E}\left[\left\|B_{k+1} - \hat{B_{k+1}}\right\|_F\right] + \mathbf{E}\left[\left\|\hat{B_{k+1}} - A\right\|_F\right] \leq (\rho)^{(k+1)/2}\mathbf{E}\left[\|B_0 - A\|_F\right] + \gamma\|A\|_F.$$

To obtain the result, apply theorem 21 with $a_k = \mathbf{E}\left[\|\hat{B}_k - A\|_F\right]$, $\mu = \rho^{1/2}$, and $\gamma = \hat{\gamma}\|A\|_F$. □

Theorem 22 shows that as the sub-sampled method converges the error term from the compression of each iterate can at worst accumulate to the bound given, in practice such large accumulations are not typically seen.

4.4 Numerical Results

The sub-sampled low rank method algorithm 10 was implemented and tested using matrix updates via algorithms 1 to 3 on two synthetic matrices. The first matrix $A = X\,X^T$, where $X \sim \mathcal{N}(0,1)^{n\times n}$ has the typical slowly decaying spectrum associated with random SPD matrices of that form. The second matrix $\tilde{A}$ is constructed to have a steep drop in the spectrum at a specific target rank. Specifically, given

$$A = U_l \Sigma_{ll} V_l^T + (I - U_l U_l^T) A (I - V_l V_l^T),$$

where $A_l = U_l \Sigma_{ll} V_l^T$ is the best rank l approximation, define

$$\tilde{A} = U_l \Sigma_{ll} V_l^T + 10^{-3}(I - U_l U_l^T) A (I - V_l V_l^T). \tag{4.8}$$

The modified matrix $\tilde{A}$ is said to have an effective rank of l with a drop of 10^{-3}.

As in section 2.3.1, the size of the aggregated pieces of information will be the primary cost metric for the algorithms. Algorithm 10 was implemented in Mathematica 12.0 [17] and computational tests were performed on a Windows 10 laptop.

The experiments are organized as follows: the first tests the convergence of algorithm 10 with update from algorithms 1 to 3 on the two synthetic matrices A and $\tilde{A}$ of size $n = 256$, the second experiment tests the effective compression in terms of storage of the iterates in response to the compression parameter $\hat{\gamma}$, the last tests the storage requirements using different compression schemes.

4.4.1 Convergence Test

The convergence,

$$\frac{\|A - B_k\|_F}{\|A - B_0\|_F},$$

of the sub-sampled algorithms with $s_1 = s_2 = 16$ is tested on the matrices A and $\tilde{A}$ with size $n = 256$. $\tilde{A}$ was modified such that the singular values drop off by a factor of 10^{-3} after $l = 16$. The codes were run with $B_0 = 0 \cdot I_n$ using $\hat{\gamma} = 10^{-3}$. Note the choice of $\hat{\gamma}$ is the same as the drop factor in eq. (4.8) which a user may know from some other knowledge of their application. The figures show: RRF ($\diamond$(p=1)) as discussed in section 2.4.3, RSSI ($*$(p=1), $*$(p=2), $*$(p=3)) as specified by algorithm 5, and algorithm 10 labeled by matrix updates: NS ($\otimes$) as specified by Algorithm 1; SS1 ($\bullet$) as specified by Algorithm 2; SS2 (🟨) as specified by Algorithm 3. Runs were terminated when the relative residual norm fell below 10^{-3} or a maximum amount of n^2 output entries were sampled.

The RRF and RSSI methods from [16] are not iterative. To give fair comparison, methods of [16] were run with an increasing over-sampling parameter q. Recall, methods from [16] construct rank $r = l + q$ approximations. The total number of output entries computed is determined by the number of power iterations p and sampling size r. To indicate this, non-iterative methods from [16] are shown with distinct points indicating individual approximations being constructed. Additionally, rank $r = l + q$ approximations were reduced to rank l using the SVD to give fair comparison in norm to the sub-sampled methods which reduce the rank every iteration.

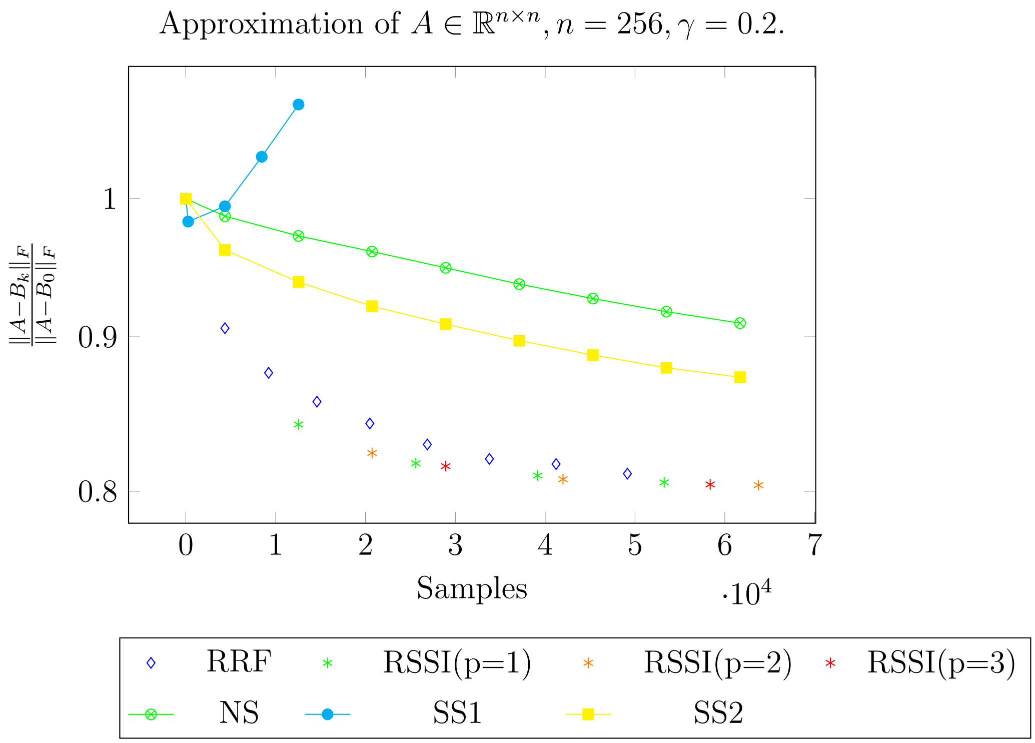


Figure 4.1: ($n = 256$) Approximation of $A = XX^T$ where $X \sim \mathcal{N}(0, 1)^{n\times n}$ with $s = 16$, $\gamma = 0.2$. The low rank algorithm 2 performs poorly while the low rank versions of algorithms 1 and 3 perform well. Low rank accelerated algorithm 4 shows acceleration compared to algorithms 1 and 3 and is competitive with the algorithms from [16]. All methods approach the best rank 16 approximation of A.

4.4.2 Parameter Test

The tolerance based compression described in section 4.2.2 depends on the parameter $\hat{\gamma}$. If $\hat{\gamma}$ is too small, the iterates may increase in rank and use considerable storage during early iterations. If $\hat{\gamma}$ is too high, the algorithms may converge more slowly as they will not be making much progress improving the current best low rank search directions. In addition, if the user misjudges $\|A - A_l\|_F$ when choosing $\hat{\gamma}$, the low

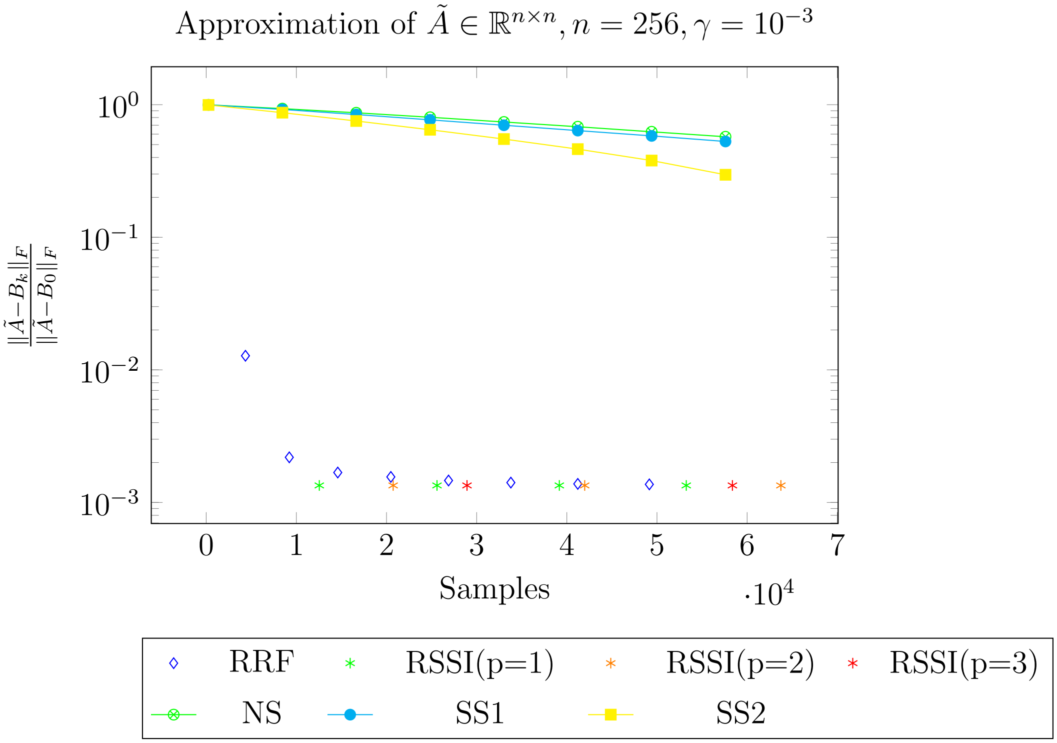
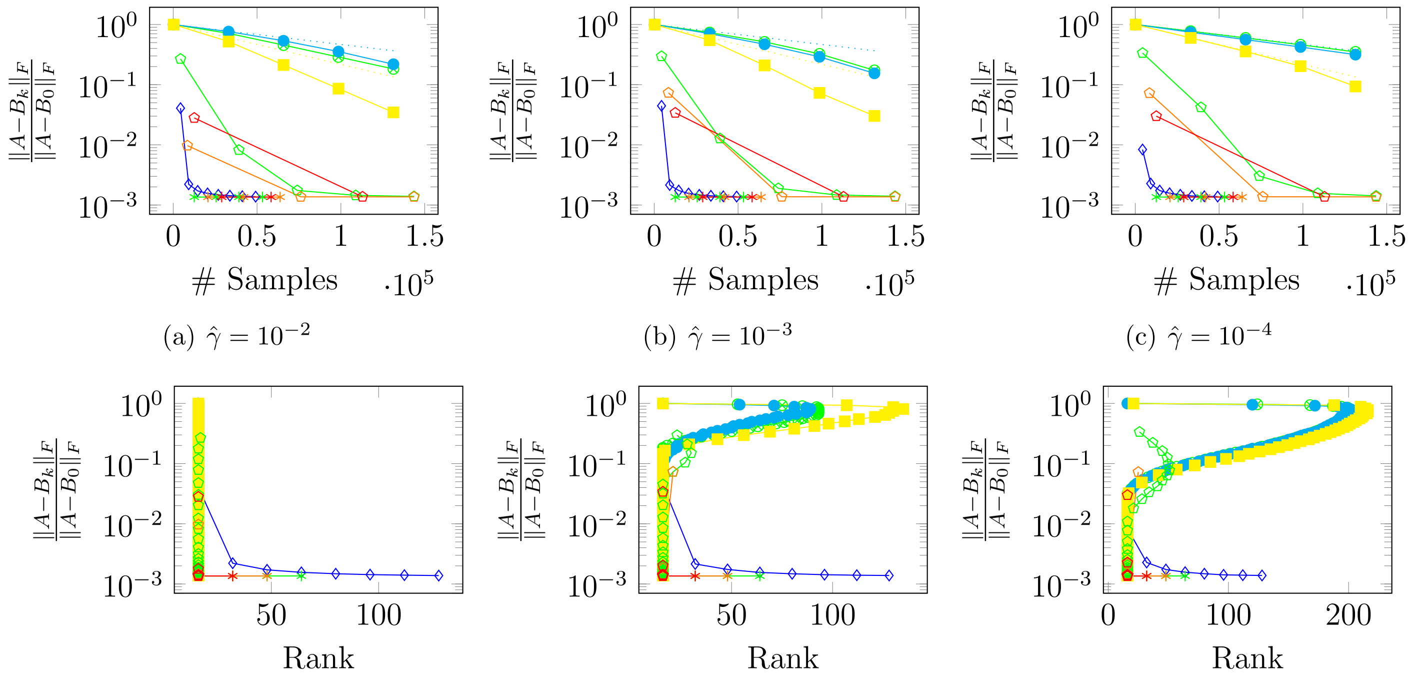


Figure 4.2: ($n = 256$) Approximation of modified matrix $\tilde{A}$ having effective rank 16 with $s = 16$ and $\gamma = 10^{-3}$. Low rank variants of algorithms 1 to 3 show comparable convergence rates to the optimal theoretical rates of the un-compressed versions shown in dotted lines. Accelerated variants converge quickly to the best rank 16 approximation.

rank approximation may miss some of the most dominant space and only converge on A_{l-k} for some $k > 0$.

To demonstrate the above discussion algorithm 10 was implemented with updates from algorithms 1 to 3 and tested on the modified matrix $\tilde{A}$. The methods were run with values of $\hat{\gamma} = 10^{-2}, 10^{-3}, 10^{-4}$. As predicted the lower values of $\hat{\gamma}$ gave early iterates which have a higher rank and the higher value of $\hat{\gamma} = 10^{-2}$ gave a slightly slower convergence but only stored a minimum of 16 directions at each iterate.



(a) $\hat{\gamma} = 10^{-2}$

(b) $\hat{\gamma} = 10^{-3}$

(c) $\hat{\gamma} = 10^{-4}$

(d) $\hat{\gamma} = 10^{-2}$

(e) $\hat{\gamma} = 10^{-3}$

(f) $\hat{\gamma} = 10^{-4}$

Figure 4.3: Comparison of storage while approximating the modified matrix $\tilde{A}$ using $\hat{\gamma} = 10^{-2}, 10^{-3}, 10^{-4}$. Choosing tolerance parameter $\hat{\gamma}$ according to the known drop in spectrum of $\tilde{A}$ compresses the storage after enough iterations. Higher values of $\hat{\gamma}$ (left) show minimal storage requirement for the sub-sampled methods. Larger values of $\hat{\gamma}$ (right) show large storage requirements initially which compress faster for the choice closer to the drop of 10^{-3} in the spectrum of $\tilde{A}$.

To further demonstrate the relationship between storage requirement and choosing the low rank tolerance $\hat{\gamma}$ based on the spectral drop, an experiment was run on the matrix A above whose singular values decay slowly. When $\hat{\gamma}$ is chosen systematically (according to the known distribution) the tolerance based scheme still converges on a compressed low rank approximation according to $\hat{\gamma}$ (see fig. 4.4).

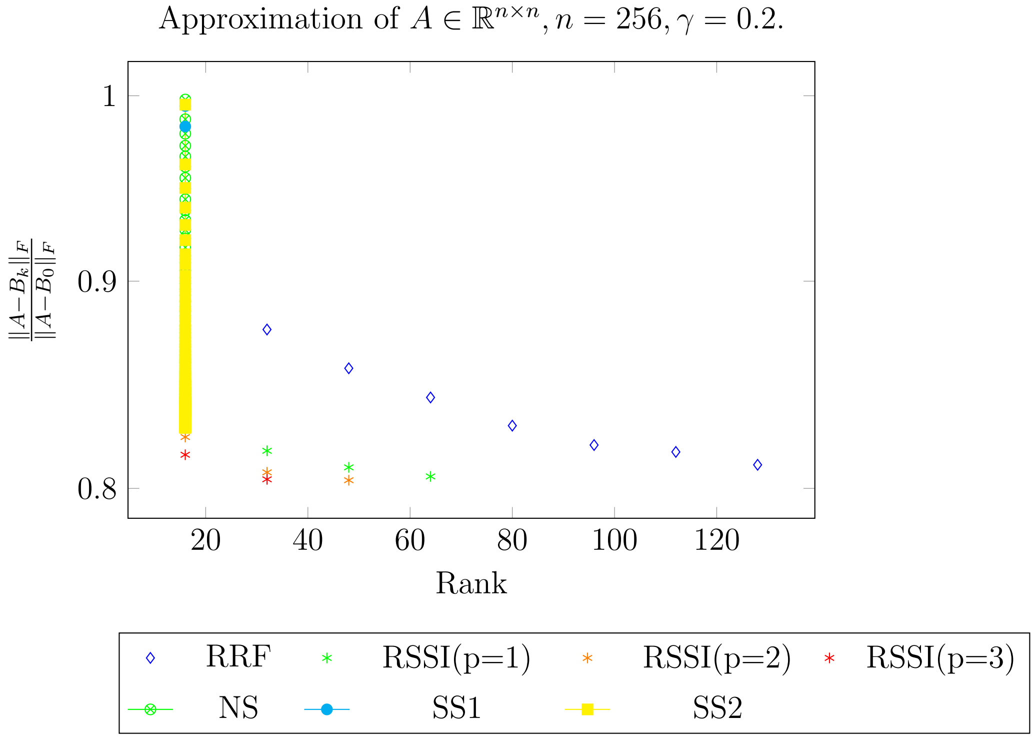


Figure 4.4: ($n = 256$) Approximation of $A = XX^T$ where $X \sim \mathcal{N}(0,1)^{n \times n}$ with $s = 16$ and $\gamma = 0.2$. Graphs show the rank of each iterate being controlled by the tolerance parameter $\gamma = 0.2 \approx \|A - A_s\|_F$, matching the known distribution of A. The methods from [16] require an increasing rank to converge to a best approximation while the sub-sampled methods only ever store the s directions desired and s more while the iteration is taking place giving a fixed (user controlled) computational and storage requirement per iteration.

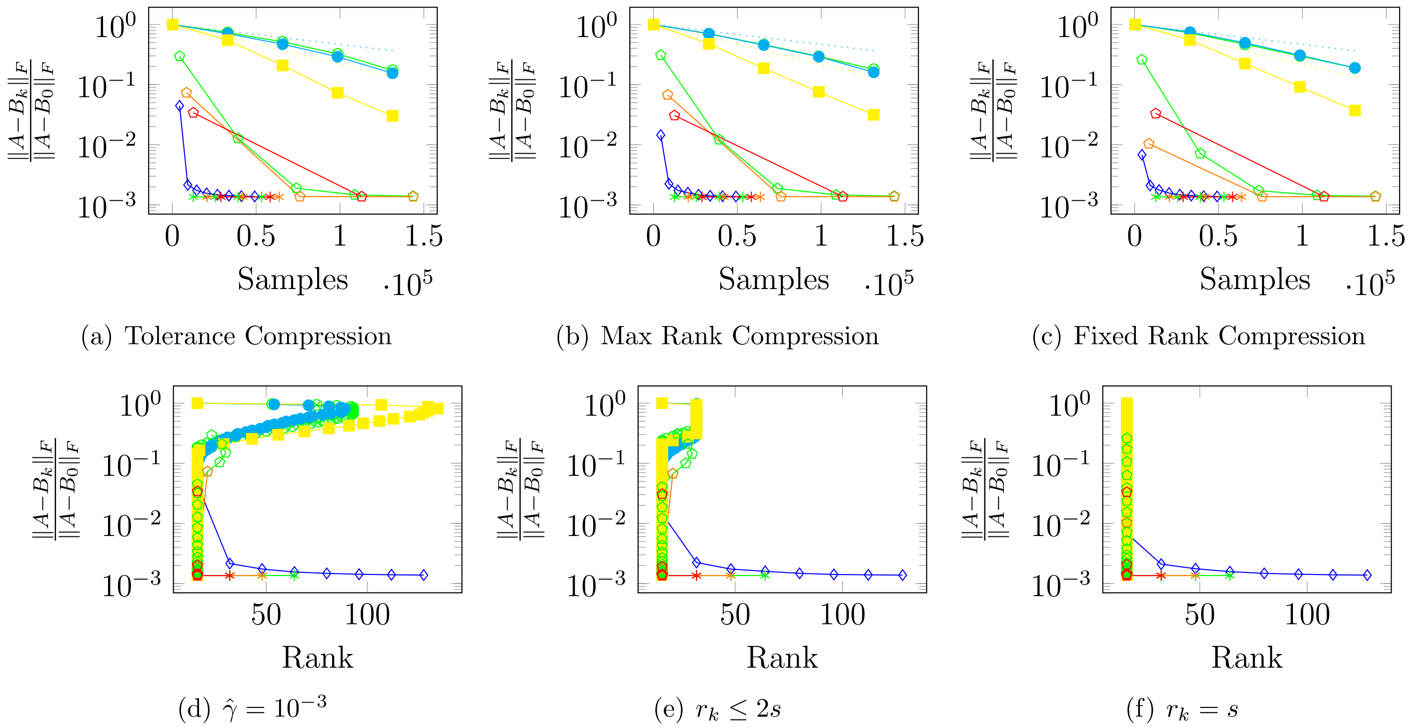
4.4.3 Compression Test

To demonstrate the performance of the tolerance based low rank compression in terms of storage, algorithm 10 was implemented: with tolerance compression having $\hat{\gamma}$ chosen according to the known spectrum of the matrices A and $\tilde{A}$ described above, with a tolerance based compression with the addition of a maximum rank $r_k = 2s$ per iteration, and compressed to a fixed rank $r_k = s$ at each iteration.

4.5 Heuristic Accelerated Schemes

The heuristically accelerated method SS1A described in algorithm 4 relies on samples computed in a power method scheme to enrich a representation of the dominant eigenspace. The final sub-space representation is then used for an accelerated sub-sampled update.

This approach is similar to that of the randomized power iteration and randomized sub-space iteration of [16]. The approach taken in this work differs in that the sub-sampled methods and accelerated variants work on the residual matrix and are able to iterate freely to a user specified computational limit. Adding a compression step to this iteration gives low-rank approximations which are competitive with those in



(a) Tolerance Compression

(b) Max Rank Compression

(c) Fixed Rank Compression

(d) $\hat{\gamma} = 10^{-3}$

(e) $r_k \leq 2s$

(f) $r_k = s$

Figure 4.5: Comparison of storage requirements while approximating the modified matrix $\tilde{A}$ using different compression schemes. Compressing $r_k = s$ at each iteration (right) yields a fixed storage method. Choosing tolerance parameter $\hat{\gamma}$ according to the known drop in spectrum of $\tilde{A}$ also compresses the storage after enough iterations (left). The performance (center) of the tolerance algorithm with a cap on the rank at $r_k \leq 2s$ performs well and has a fixed, user controllable overall storage requirement.

[16] and enjoy a fixed computational and storage footprint.

To implement this scheme one inserts the accelerated update from algorithm 4 for the sub-sampled update step in algorithm 10.

4.5.1 Acceleration Convergence Results

The performance of the low rank version of SS1A (with rotationally invariant sampling and $p = 1, 2, 3$) is compared to the Randomized Range Finder (RRF) and Randomized Sub-Space Iteration (RSSI) algorithms from [16]. For the methods of [16], the Hermitian post-processing described in section 2.1 is used to construct the approximation B. Specifically, the convergence (relative Frobenius residual $\|A - B_k\|_F / \|A - B_0\|_F$ against the total number of matrix elements sampled) of the accelerated algorithms from [16] with increasing over-sample parameters is compared to algorithm 10, the low rank variant of the accelerated algorithm 4, with $s_1 = s_2 = s$ on two matrices, $A = XX^T$ with $X \sim \mathcal{N}(0,1)^{n \times n}$ and $\tilde{A}$ with modified singular values having a drop factor of 10^{-3} after $l = 32$. The figures show: RRF (◇(p=1)) as discussed in section 2.4.3, RSSI (∗(p=1), ∗(p=2), ∗(p=3)) as specified by algorithm 5, SS1A (⬠(p=1), ⬠(p=2), ⬠(p=3)) as specified by algorithm 10. Runs were terminated when the relative residual norm fell below 10^{-3}. or a maximum number of n^2 output elements were sampled.

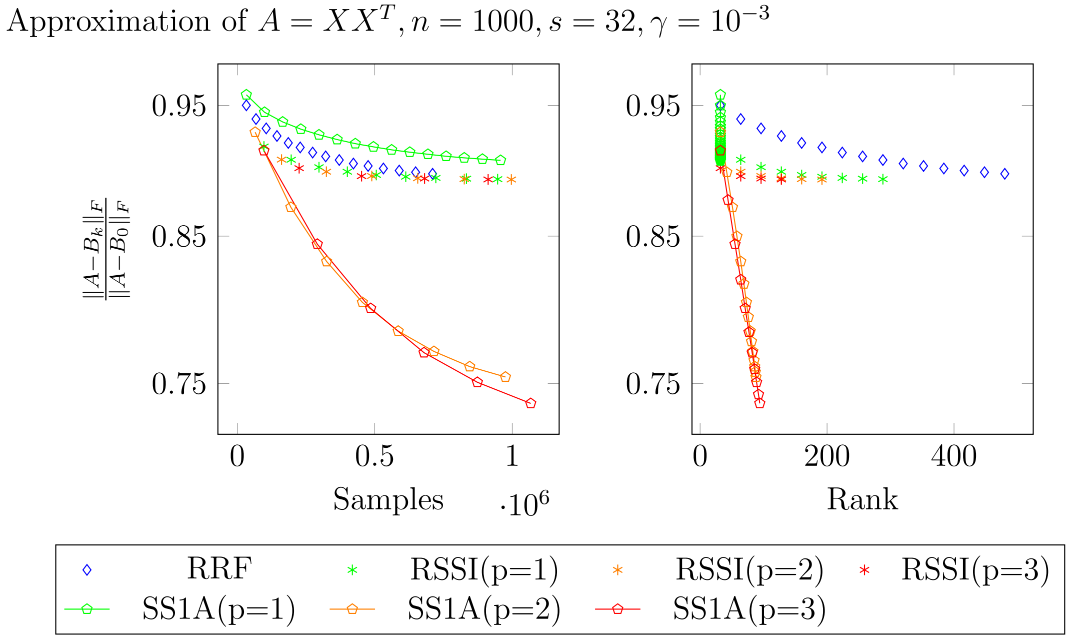


Figure 4.6: ($n = 1000$) Approximation of $A = XX^T$ where $X \sim \mathcal{N}(0,1)^{n\times n}$ with $s = 32$ and $\gamma = 0.001$. Tolerance based sub-sampled low rank methods outperform the methods of [14] for matrices with slowly decaying spectrum (left). RRF and RSSI methods require increasing rank to converge to the best rank s approximation while the tolerance based low rank only ever stores an amount proportional to the parameter $\hat{\gamma}$.

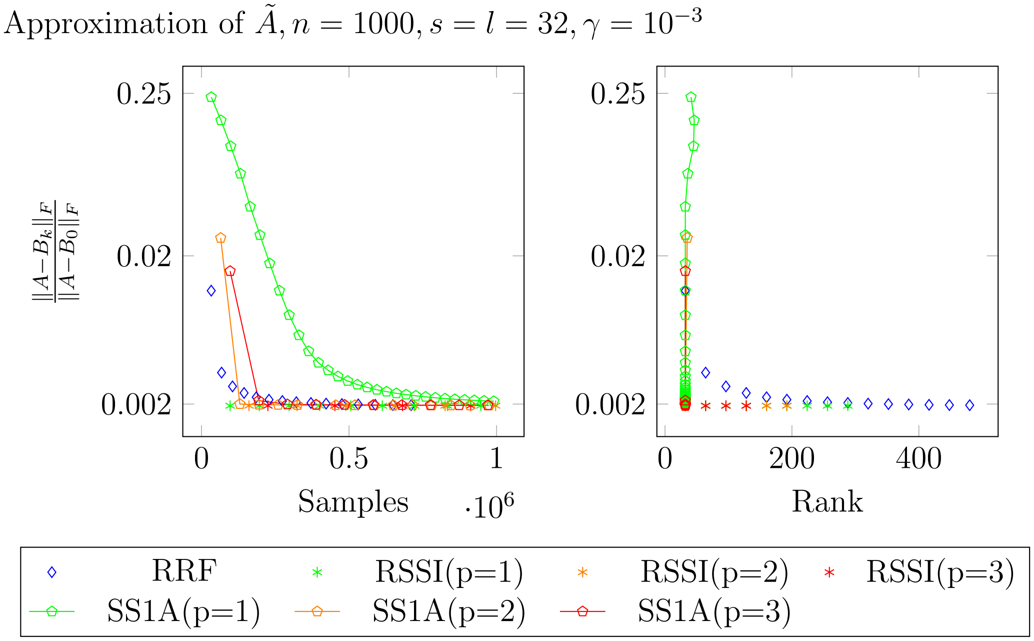


Figure 4.7: ($n = 1000$) Approximation of modified matrix $\tilde{A}$ having effective rank 32 with $s = 32$ and $\gamma = 0.001$. All the methods converge to the best rank s approximation in similar amounts of samples(left). Low rank, sub-sampled accelerated methods only require fixed storage indicated by the nearly vertical line (right) while the methods of [16] require increasing storage.

Chapter 5

Conclusions and Future Works

5.1 Conclusions

In this thesis novel sub-sampled methods to iteratively approximate a matrix by simultaneously sampling input and output spaces were developed and analyzed. These sub-sampled methods have a significantly smaller data-footprint than sampled algorithms which can be tuned by selecting sample sizes s_1 and s_2. The iterative methods are self-correcting with computable convergence rates under reasonable assumptions since they systematically reduce a weighted Frobenius norm of the residual $A - B_k$. The analysis demonstrates that rotationally symmetric sampling is desirable.

Experimentally the sub-sampled algorithms match their provable convergence rates (with only weak dependence on the sampling parameters s_1 and s_2) and have rates comparable to those of sampled algorithms in the literature.

An accelerated hybrid method (algorithm 4) is developed by combining simultaneous iteration (to enrich a subspace) with the sub-sampled update algorithm 2. This accelerated method is shown experimentally to be competitive (in terms of matrix samples) with current accelerated schemes.

The sub-sampled framework can be used to approximate the inverse of a matrix. Applying the Sherman-Morrison-Woodbury matrix identity to the sub-sampled updates give rise to inverse approximations. The methods can be used to provide iteratively approximate solutions to linear systems or with appropriate filtering quickly construct a pre-conditioner. Their convergence was compared to current methods in literature. Accelerated variants specific to inverse approximation were described and tested numerically.

Inserting a rank compression sub-routine into the sub-sampled algorithms allows for low rank approximations which have fixed computational and storage footprints at each iteration. Different compression schemes were presented with provable convergence. The methods were compared with those found in the literature and have various advantages. Specifically, current algorithms improve their rank l approximations and give bounds to the quality in terms of an over-sampling parameter p. Implementing this requires storage of more than the desired rank $r = l + p$ and gives an approximation which is of rank r. Using an iterative approach to over-sample at each iteration allows the user to control the storage and computation of each step giving a smaller data and computational footprint at each iteration and over all.

5.2 Future Works

The methods described, analyzed, and tested in this thesis are built from the usage of sub-sampled data, eq. (1.1). The iterations given are constructed in a minimum Frobenius norm setting, eq. (2.6). We posit that one can derive the solution or a computationally feasible approximate solution to minimum change problems using different norms. For instance, similar rank s updates can be derived by enforcing $U^T A V = U^T B V$ and $U^T A U = U^T B U$, instead of the two-step approach as in algorithm 3. The solution of the minimum change in that case requires solving certain $s \times s$ Sylvester type equations, but is an effective update in terms of the metric used in section 2.3.

The convergence of the sub-sampled matrix approximations was analyzed theoretically and verified with a wide range of numerical experiments. To adapt this theoretical basis to other settings required inserting sub-routines into the iteration for inverse approximation and low rank approximations. Other special types of matrix properties could be inserted such as a mechanism of enforcing a particular sparsity pattern or compressing to other matrix factorizations.

Another interesting research direction is to explore computational efficiency of the algorithms. Specifically if the computation of the sub-samples is not feasible for the way the matrix A is represented, one could incorporate ideas from [8, 9, 10, 11]. In that work, the authors (adaptively) sample the rows, columns, or both to reduce the computation. Sub-samples of the reduced set of rows and columns could be used after they are selected.

The sub-sampled matrix approximation algorithms give a solid theoretical foundation for many future works. The insertion of the inverse and low rank subroutines are an example of how the iterations can be modified and tuned for various applications and due to the nature of the random sampling have great approximation power. The inverse approximations are useful as iterative linear system solvers, pre-conditioners, and could be adopted for pseudo-inverses. The low rank approximations could be used for non-square data analysis, sparse approximations, or compressed storage formats for internal computations in algorithms like newtons method for solving non-linear systems or quasi-newton methods in the numerical optimization setting for approximating Hessian or inverse Hessian matrices.

References

[1] Caliciotti Andrea, Fasano Giovanni, and Roma Massimo. Novel preconditioners based on quasi–newton updates for nonlinear conjugate gradient methods. *Optimization Letters*, 11(4):835–853, Apr 2017.

[2] Michele Benzi and Miroslav Tůma. A comparative study of sparse approximate inverse preconditioners. In *Proceedings on the on Iterative Methods and Preconditioners*, IMACS'97, page 305–340, USA, 1999. Elsevier North-Holland, Inc.

[3] Luca Bergamaschi, Rafael Bru, Angeles Martinez, and Mario Putti. Quasi-newton preconditioners for the inexact newton method. *Electronic transactions on numerical analysis ETNA*, 23:76–87, 01 2006.

[4] R.H. Bryd, R.B. Schnabel, and G.A. Schulz. Parallel quasi-newton methods for unconstrained optimization. *Mathematical Programming*, 42:273–306, 04 1988.

[5] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27,

2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[6] Edmond Chow and Yousef Saad. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM Journal on Scientific Computing*, 19(3):995–1023, 1998.

[7] Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, December 2011.

[8] Petros Drineas, Ravi Kannan, and Michael W. Mahoney. Fast monte carlo algorithms for matrices ii: Computing a low-rank approximation to a matrix. *SIAM Journal on Computing*, 36(1):158–183, 2006.

[9] Petros Drineas, Ravi Kannan, and Michael W. Mahoney. Fast Monte Carlo algorithms for matrices. III. Computing a compressed approximate matrix decomposition. *SIAM J. Comput.*, 36(1):184–206, 2006.

[10] Petros Drineas, Ravindran Kannan, and Michael Mahoney. Fast monte carlo algorithms for matrices i: Approximating matrix multiplication. *SIAM J. Comput.*, 36:132–157, 01 2006.

[11] Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast monte-carlo algorithms for finding low-rank approximations. *J. ACM*, 51(6):1025–1041, November 2004.

[12] Wenbo Gao and Donald Goldfarb. Block BFGS Methods. *SIAM J. Optim.*, 28(2):1205–1231, 2018.

[13] Robert Gower, Filip Hanzely, Peter Richtarik, and Sebastian U Stich. Accelerated stochastic matrix inversion: General theory and speeding up bfgs rules for faster second-order optimization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 1619–1629. Curran Associates, Inc., 2018.

[14] Robert M. Gower and Peter Richtárik. Randomized quasi-Newton updates are linearly convergent matrix inversion algorithms. *SIAM J. Matrix Anal. Appl.*, 38(4):1380–1409, 2017.

[15] Robert M. Gower and Peter Richtárik. Randomized iterative methods for linear systems. *SIAM Journal on Matrix Analysis and Applications*, 36(4):1660–1690, 2015.

[16] N. Halko, P. Martinsson, and J. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

[17] Wolfram Research, Inc. Mathematica, Version 12.1, 2020.

[18] The Mathworks, Inc., Natick, Massachusetts. *MATLAB version 9.3.0.713579 (R2017b)*, 2017.

[19] José Morales and Jorge Nocedal. Automatic preconditioning by limited memory quasi-newton updating. *SIAM Journal on Optimization*, 10:1079–1096, 06 2000.

[20] Cameron Musco and Christopher Musco. Randomized block krylov methods for stronger and faster approximate singular value decomposition. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 1396–1404, Cambridge, MA, USA, 2015. MIT Press.

[21] S. Nash. Newton-type minimization via the lanczos method. *SIAM Journal on Numerical Analysis*, 21(4):770–788, 1984.

[22] S. Nash. Preconditioning of truncated-newton methods. *SIAM Journal on Scientific and Statistical Computing*, 6(3):599–616, 1985.

[23] U. Naumann. *The Art of Differentiating Computer Programs.* Society for Industrial and Applied Mathematics, 2011.

[24] Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980.

[25] Jorge Nocedal and Stephen J. Wright. *Numerical optimization.* Springer Series in Operations Research and Financial Engineering. Springer, New York, second edition, 2006.

[26] Albert S. Berahas, Raghu Bollapragada, and Jorge Nocedal. An investigation of Newton-sketch and subsampled Newton methods. *PrePrint*, 05 2017.

[27] R.B. Schnabel. Quasi-newton methods using multiple secant equations. *Computer Science Technical Reports*, 244:41, 06 1983.

[28] G. Stewart. The efficient generation of random orthogonal matrices with an application to condition estimators. *SIAM Journal on Numerical Analysis*, 17(3):403–409, 1980.

[29] Max A. Woodbury. *Inverting Modified Matrices*. SRG Memorandum report; 42. Princeton University, first edition, 1950.

Appendix A

Sub-Sample Appendices

A.1 Weight Matrix Interpretation

The fixed non-rotationally symmetric weight matrices on which classical sampled methods are based (BFGS $W = A$ and DFP $= W^{-1}$) produce an enhanced initial drop in the appropriate residuals. Implementing algorithms algorithms 1 to 3 with $W = A$ would produce the same temporary effect but as noted before algorithms with $W = A$ are automatically sampled algorithms which require significantly more access to A than sub-sampling. Moreover, the enhancement is transitory and despite the additional cost of each step such weighted algorithms ultimately converge at the rates in theorems 7, 9 and 10 as B_k resolves A. This is to be expected since the algorithms sample and correct the residual $A - B_k$. Weights tuned to A become irrelevant as $B_k \to A$. The heuristic underlying the accelerated algorithm, algorithm 4, is that non-constant weighting based on the residual $W_k = A - B_k$ should sample directions that are not yet well resolved: as noted in the discussion of algorithm 4 such dynamic weighting requires samples $A\,U$.

A.2 Minimum Change Solutions

The KKT equations [25] for constrained minimum change formulations eqs. (2.1) and (2.2) are solved analytically using a change of variables. Substitute

$$\hat{A} = W_1^{-1/2}\,A\,W_2^{-1/2}, \quad \hat{B} = W_1^{-1/2}\,B\,W_2^{-1/2}, \quad \hat{B}_k = W_1^{-1/2}\,B_k\,W_2^{-1/2},$$

and

$$\hat{U} = W_1^{1/2}\,U, \quad \hat{V} = W_2^{1/2}\,V,$$

into eq. (2.1) to get the unweighted problem,

$$\hat{B}_{k+1} = \arg\min_{\hat{B}} \left\{ \frac{1}{2}\|\hat{B} - \hat{B}_k\|_F^2 \mid \hat{U}^T\hat{B}\,\hat{V} = \hat{U}^T\hat{A}\,\hat{V} \right\}.$$

This reduces to

$$\arg\min_{E}\left\{\frac{1}{2}\|E\|_F^2 \mid \hat{U}^T E\hat{V} - Z = 0\right\}, \tag{A.1}$$

where $E = \hat{B} - \hat{B}_k$ and $Z = \hat{U}^T(\hat{A} - \hat{B}_k)\hat{V}$. Writing Λ for the matrix of Lagrange multipliers, the Lagrange condition is obtained by setting the derivative of $\mathcal{L}(E, \Lambda)$ with respect to the matrix argument E to 0, giving

$$0 = E + \hat{U}\Lambda\hat{V}^T.$$

Substituting into eq. (A.1) gives

$$\hat{U}^T(\hat{U}\Lambda\hat{V}^T)\hat{V} + Z = 0,$$

which gives the multiplier matrix

$$\Lambda = -(\hat{U}^T\hat{U})^{-1}\hat{U}^T(\hat{A} - \hat{B}_k)\hat{V}(\hat{V}^T\hat{V})^{-1}.$$

Substituting and converting back to the original variables gives eq. (2.7). Equation (2.2) can be solved in a similar fashion.

A.3 Additional Non-Accelerated Computational Results

The convergence test from section 2.3.1 was performed on the remaining matrices tested in [14]. As before, these figures show: BFGS(◇) as specified by eq. (2.5); DFP (◇) as specified by eq. (2.4); NS (⊗) as specified by Algorithm 1; SS1 (●) as specified by Algorithm 2; SS2 (■) as specified by Algorithm 3. All numerical experiments indicate that the non-accelerated sub-sampled algorithms converge predictably and consistently. Tests were carried out on the following matrices:

† Figure A.1 the LibSVM matrix Aloi of size $n = 128$;

† Figure A.2 the LibSVM matrix Protein of size $n = 357$;

† Figure A.3 the LibSVM matrix Real-Sim of size $n = 20958$;

† Figure A.4 the Sparse Suite matrix ND6K of size $n = 18000$;

† Figure A.5 the Sparse Suite matrix ex9 of size $n = 3363$;

† Figure A.6 the Sparse Suite matrix Chem97ZtZ of size $n = 2541$.

† Figure A.7 the Sparse Suite matrix Body of size $n = 17556$.

† Figure A.8 the Sparse Suite matrix bcsstk of size $n = 11948$.

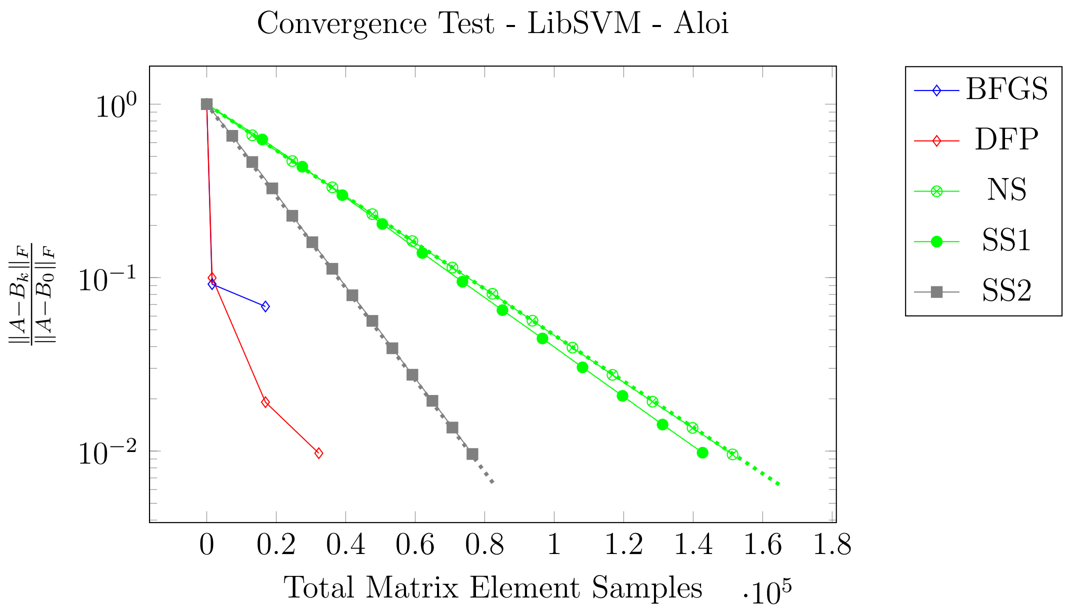


Figure A.1: Approximation of the Hessian matrix from the LibSVM problem, **Aloi** ($n = 128$) [5] with $s = 12 = \lceil\sqrt{128}\rceil$. The DFP and BFGS updates perform well. Algorithms 1 to 3 match their theoretical convergence rates (shown in dotted lines).

† Figure A.9 the Sparse Suite matrix wathen of size $n = 30401$.

A.4 Additional Accelerated Computational Results

The convergence test from section 2.4.1 was performed on the remaining matrices tested in [14]. The experiments illustrate the relative performance of the following algorithms: ($\diamond$) BFGS, eq. (2.5), ($\diamond$) DFP, eq. (2.4) ($\circ$) S1, eq. (2.3) with $W = I_n$, ($*$) BFGSA, eq. (2.5) with adaptive sampling described in [14], (⬠) SS1A+, Algorithm 4

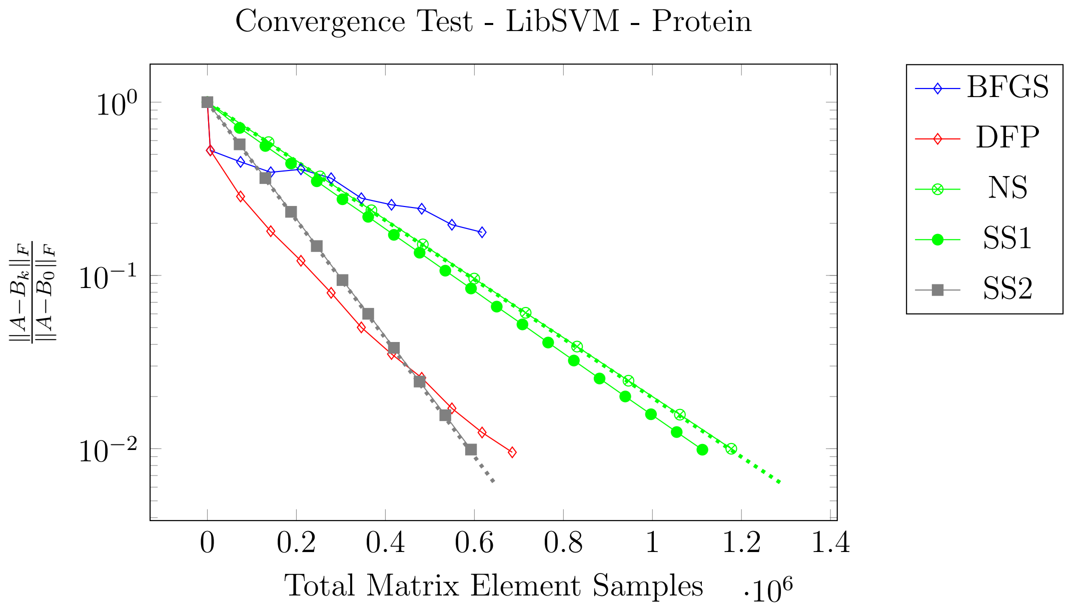


Figure A.2: Approximation of the Hessian matrix from the LibSVM problem, **Protein** ($n = 357$) [5] with $s = 19 = \lceil\sqrt{357}\rceil$. The DFP method performs well while the BFGS method shows poor performance. Algorithms 1 to 3 match their theoretical convergence rates (shown in dotted lines).

on the following matrices:

† Figure A.10 the LibSVM matrix Aloi of size $n = 128$;

† Figure A.11 the LibSVM matrix Protein of size $n = 357$;

† Figure A.12 the LibSVM matrix Real-Sim of size $n = 20958$;

† Figure A.13 the Sparse Suite matrix ND6K of size $n = 18000$;

† Figure A.14 the Sparse Suite matrix ex9 of size $n = 3363$;

† Figure A.15 the Sparse Suite matrix Chem97ZtZ of size $n = 2541$.

† Figure A.16 the Sparse Suite matrix Body of size $n = 17556$.

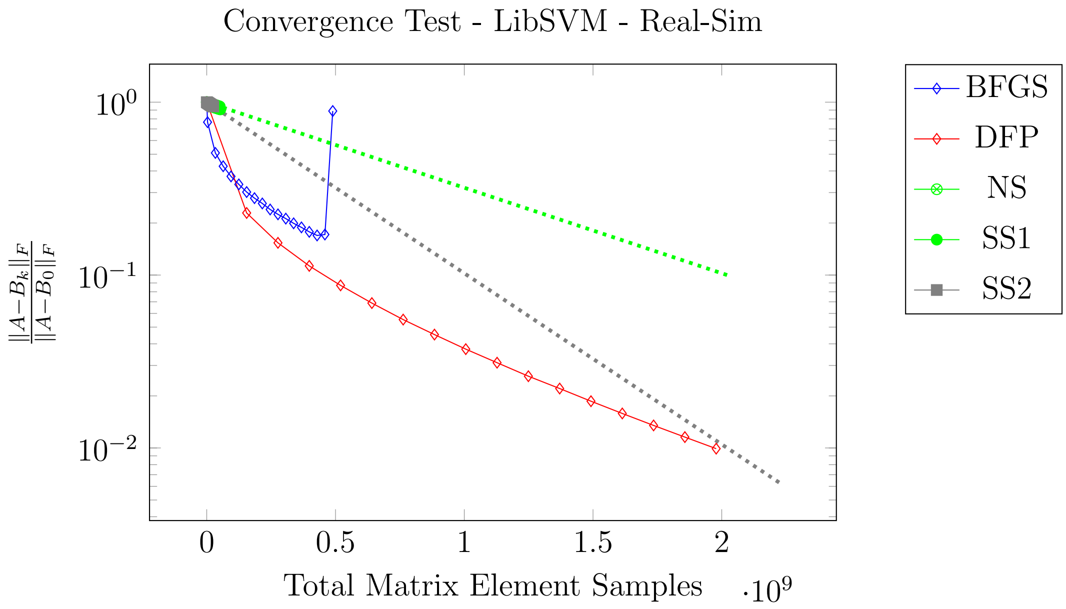


Figure A.3: Approximation of the Hessian matrix from the LibSVM problem, **Real-Sim** ($n = 20,958$) [5] with $s = 145 = \lceil\sqrt{20,958}\rceil$. The DFP method performs well eventually matching the theoretical convergence rates for algorithms 1 to 3. The BFGS method fails. Algorithms 1 to 3 methods match their theoretical rates shown in dotted lines but are terminated after a maximum run-time.

- † Figure A.17 the Sparse Suite matrix bcsstk of size $n = 11948$.

- † Figure A.18 the Sparse Suite matrix wathen of size $n = 30401$.

The accelerated method algorithm 4 performs well on all matrices including those with large $n \approx 10^4$ (see figs. A.12, A.13 and A.16 to A.18).

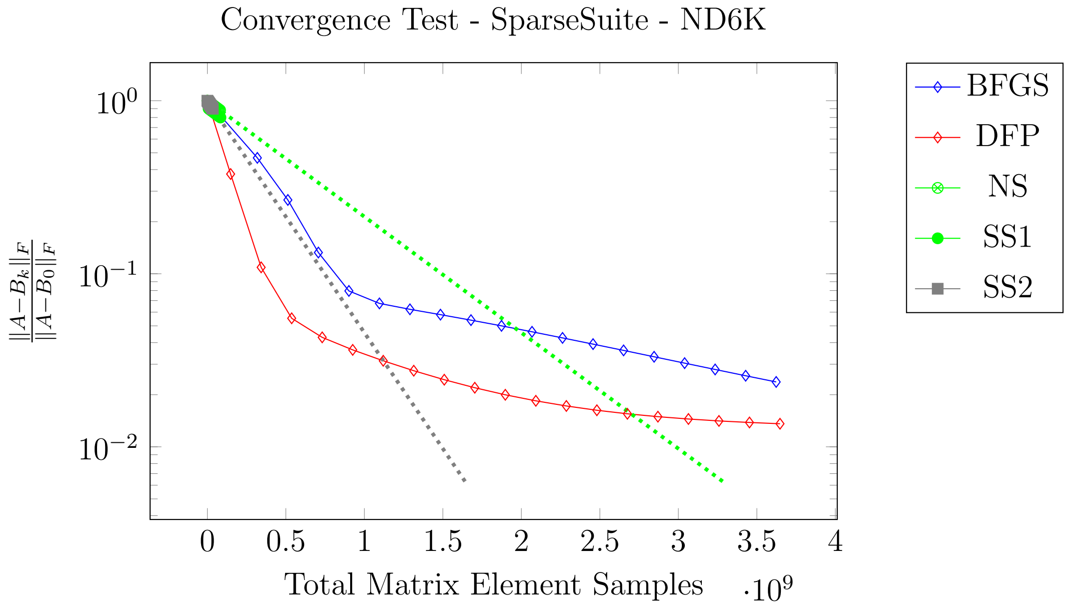


Figure A.4: Approximation of **ND6K** matrix ($n = 18,000$) from [7] with $s = 135 = \lceil\sqrt{18,000}\rceil$. The DFP and BFGS updates show fast initial convergence which slows over time. Algorithms 1 to 3 methods match their theoretical rates shown in dotted lines but are terminated after a maximum run-time.

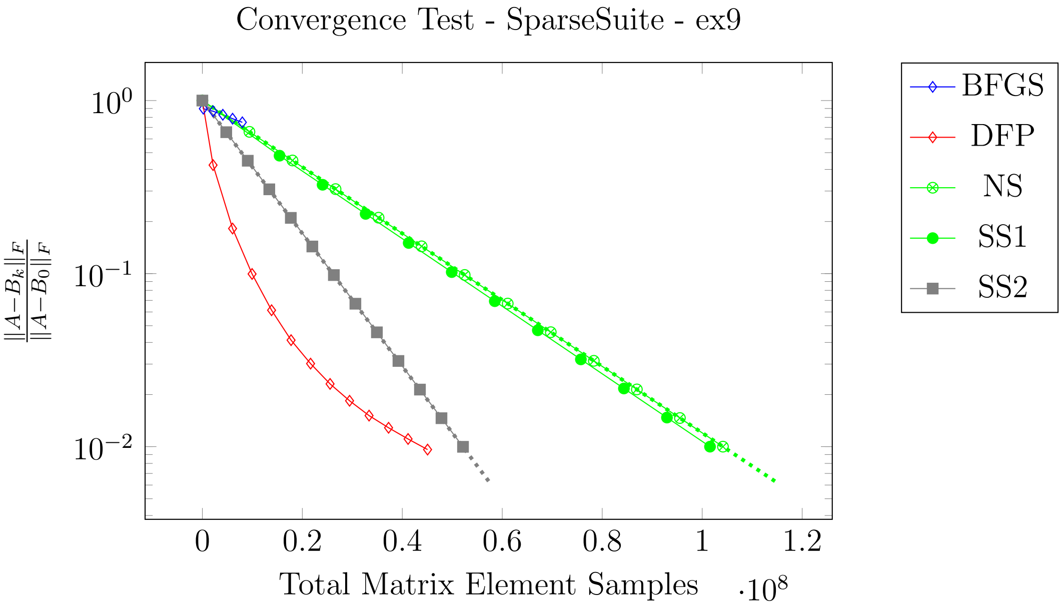


Figure A.5: Approximation of **ex9** matrix ($n = 3363$) from [7] with $s = 58 = \lceil\sqrt{3363}\rceil$. The DFP update performs consistently while the BFGS update fails early in the iteration. Algorithms 1 to 3 methods match their theoretical rates shown in dotted lines.

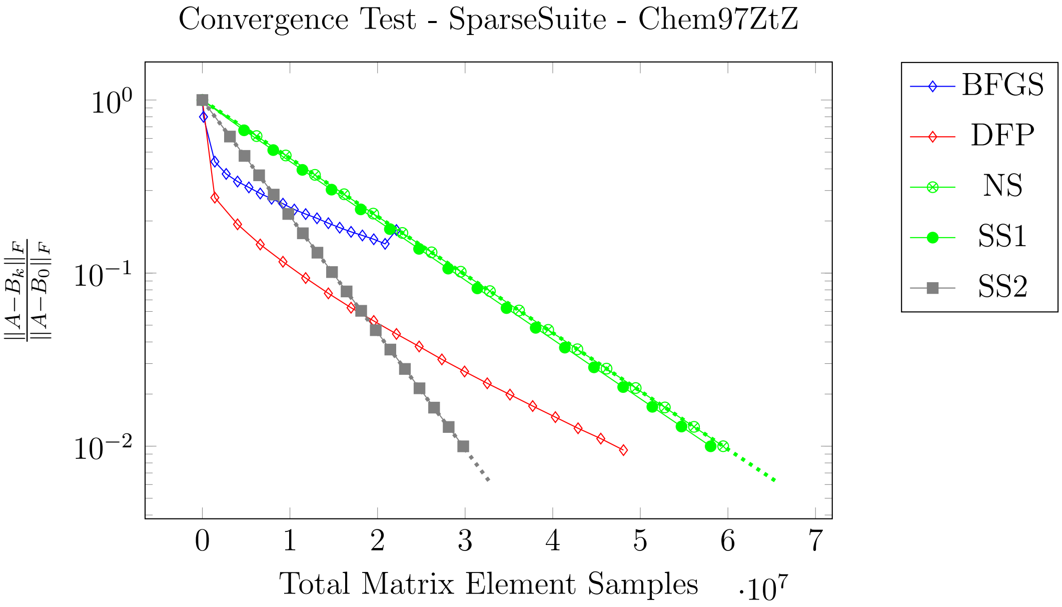


Figure A.6: Approximation of **Chem97ZtZ** matrix ($n = 2541$) from [7] with $s = 51 = \lceil\sqrt{2541}\rceil$.The DFP update performs consistently while the BFGS update fails early in the iteration. Algorithms 1 to 3 methods match their theoretical rates shown in dotted lines.

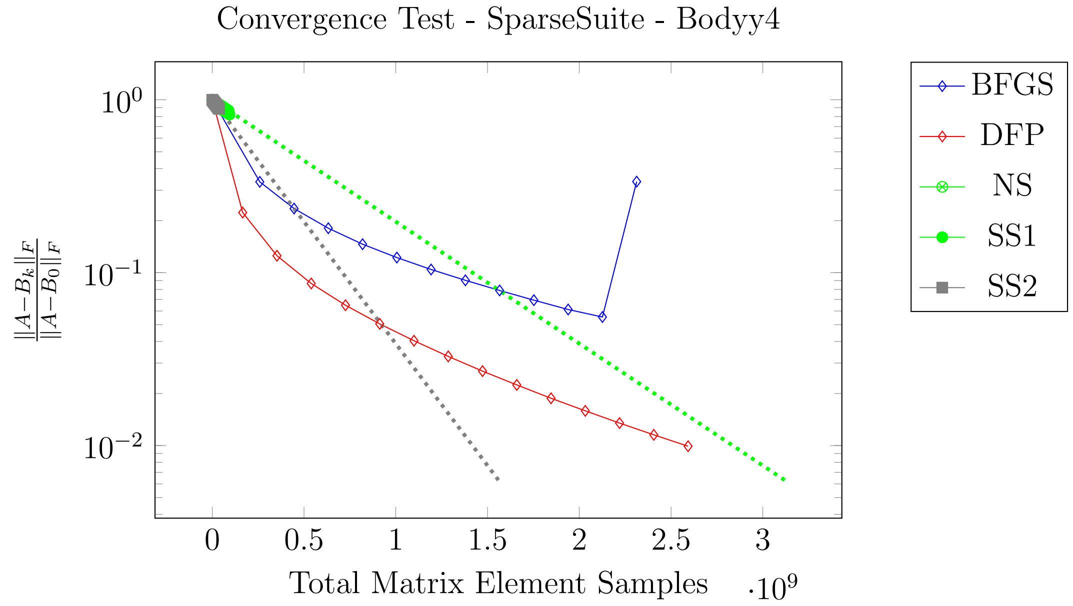


Figure A.7: Approximation of **Body** matrix ($n = 17,546$) from [7] with $s = 133 = \lceil\sqrt{17,546}\rceil$. The DFP update performs well while the BFGS update fails. Algorithms 1 to 3 methods match their theoretical rates shown in dotted lines but are terminated after a maximum run-time.

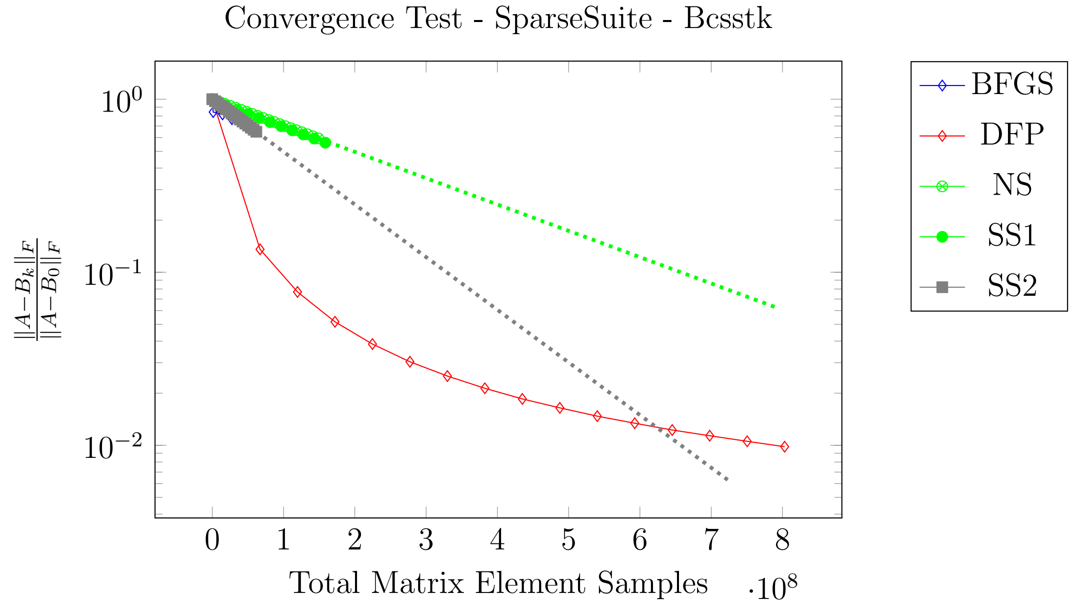


Figure A.8: Approximation of **bcsstk** matrix ($n = 11,948$) from [7] with $s = 110 = \lceil\sqrt{11,948}\rceil$. Algorithms 1 to 3 methods match their theoretical rates shown in dotted lines but are terminated after a maximum run-time.

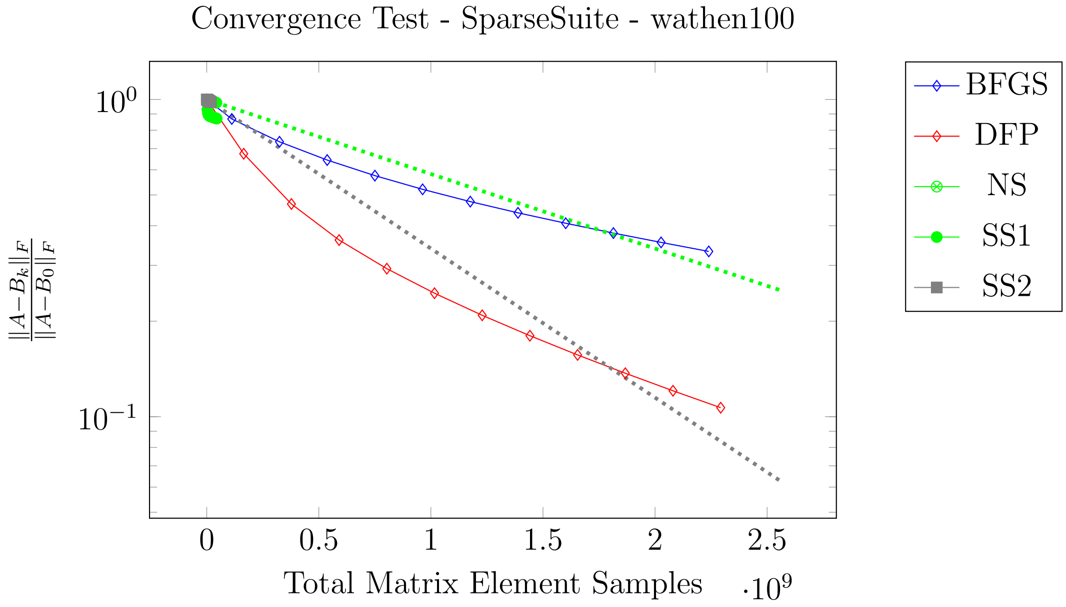


Figure A.9: Approximation of **wathen** matrix ($n = 30,401$) from [7] with $s = 175 = \lceil \sqrt{30,401} \rceil$. The DFP and BFGS methods perform consistently. Algorithms 1 to 3 methods match their theoretical rates shown in dotted lines but are terminated after a maximum run-time.

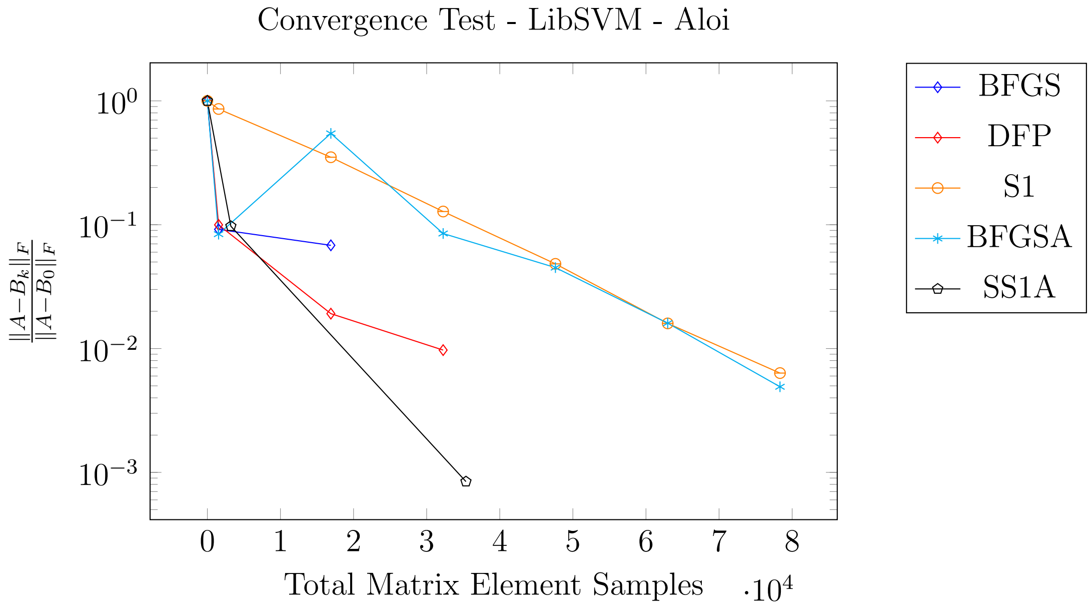


Figure A.10: Approximation of the Hessian matrix from the LibSVM problem, **Aloi** ($n = 128$) [5] with $s = 12 = \lceil\sqrt{128}\rceil$. All methods show quick initial convergence. The accelerated BFGSA method adapted from [14] shows slower performance while the DFP method and algorithm 4 show performance consistent with other tests.

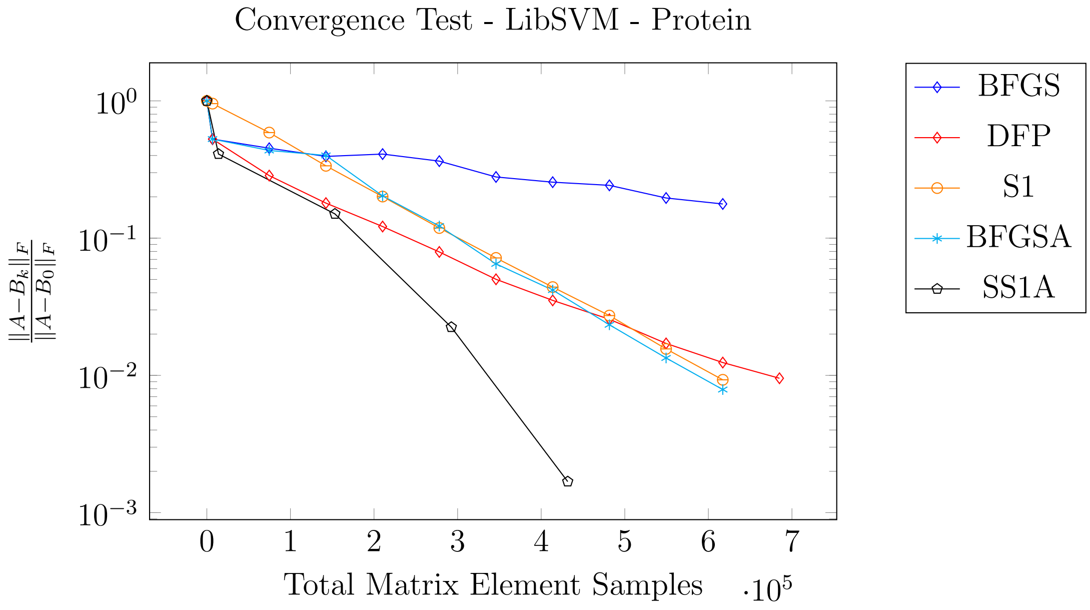


Figure A.11: Approximation of the Hessian matrix from the LibSVM problem, **Protein** ($n = 357$) [5] with $s = 19 = \lceil\sqrt{357}\rceil$. The un-accelerated BFGS method performs poorly and the accelerated BFGSA method shows minimal acceleration in comparison to other tests. The DFP method also shows weak performance compared to algorithm 4.

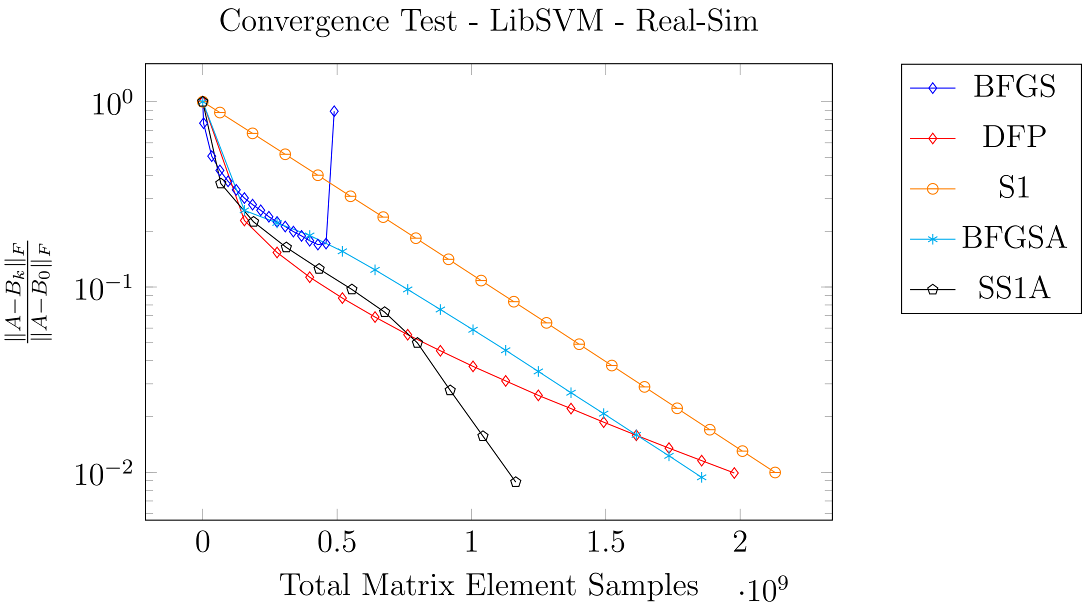


Figure A.12: Approximation of the Hessian matrix from the LibSVM problem, **Real-Sim** ($n = 20,958$) [5] with $s = 145 = \lceil\sqrt{20,958}\rceil$. The BFGS method fails after initially performing the same as the accelerated BFGSA method adapted from [14]. The DFP method and algorithm 4 initially perform similarly but algorithm 4 shows acceleration consistent with other tests.

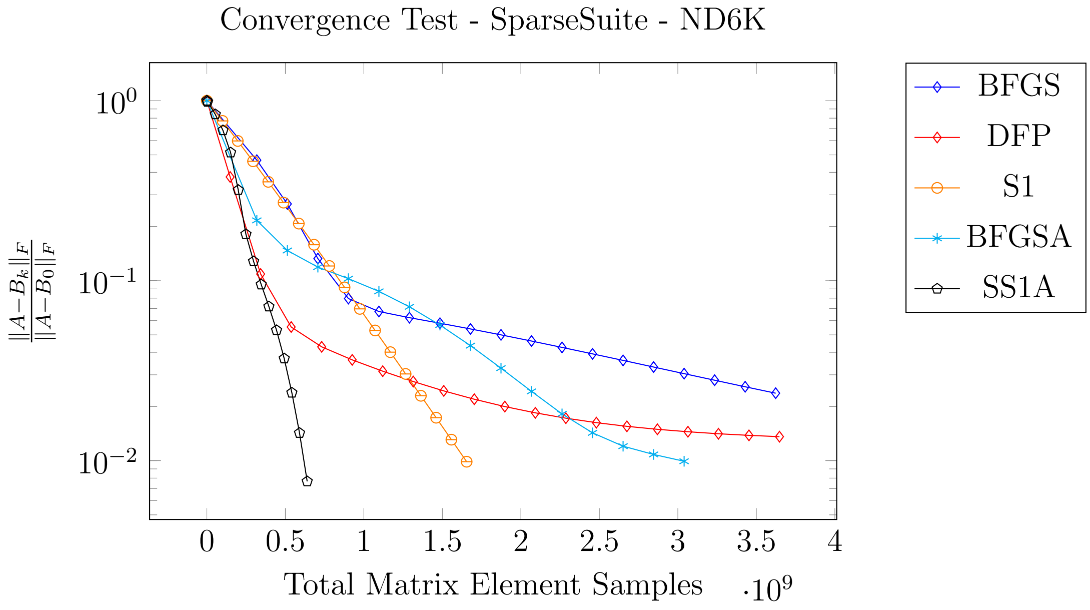


Figure A.13: Approximation of **ND6K** matrix ($n = 18,000$) from [7] with $s = 135 = \lceil\sqrt{18,000}\rceil$. All the methods perform well. BFGSA shows varied acceleration over the BFGS update while DFP shows performance consisten with other tests. Algorithm 4 outperforms other methods with consistent acceleration seen against the un-accelerated S1 method.

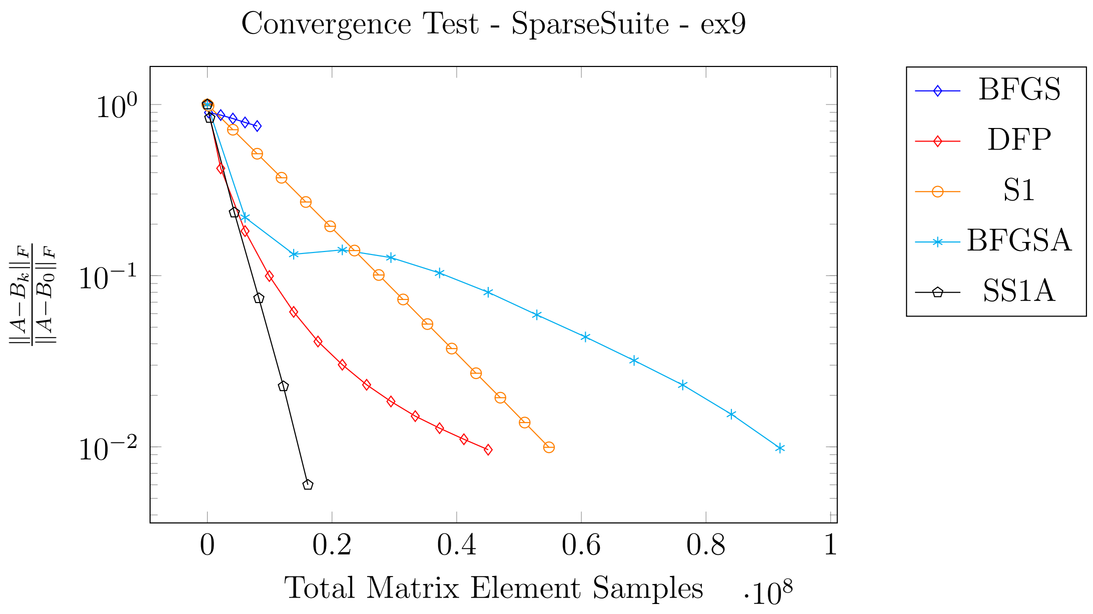


Figure A.14: Approximation of **ex9** matrix ($n = 3363$) from [7] with $s = 58 = \lceil\sqrt{3363}\rceil$. The BFGS update fails on this matrix but the accelerated BFGSA converges. The DFP method and algorithm 4 show consistent performance.

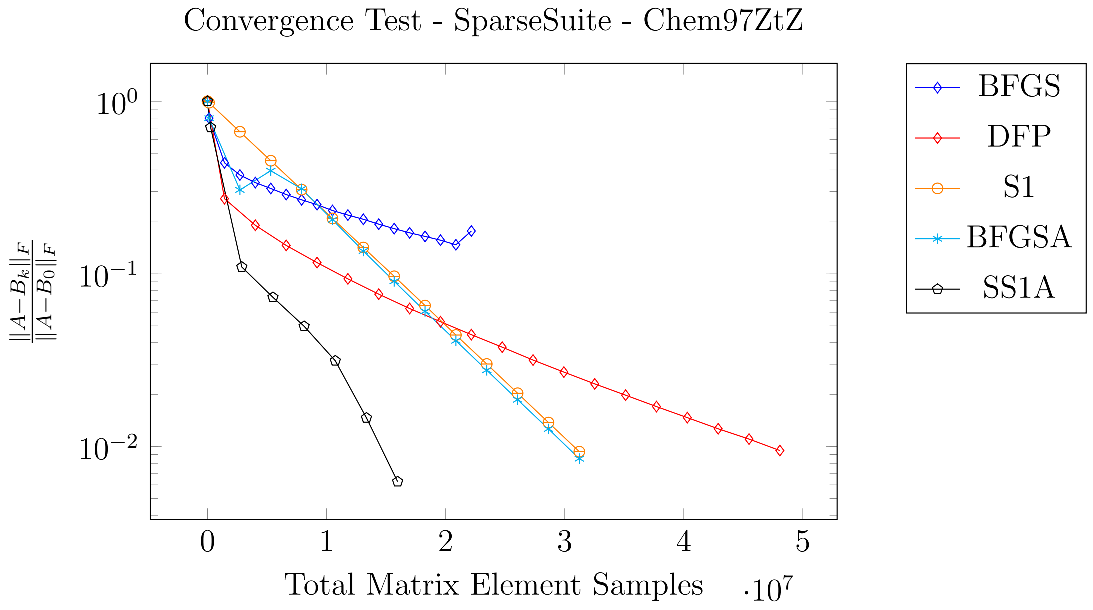


Figure A.15: Approximation of **Chem97ZtZ** matrix ($n = 2541$) from [7] with $s = 51 = \lceil\sqrt{2541}\rceil$. BFGS method fails while the other methods converge with similar performance to other tests.

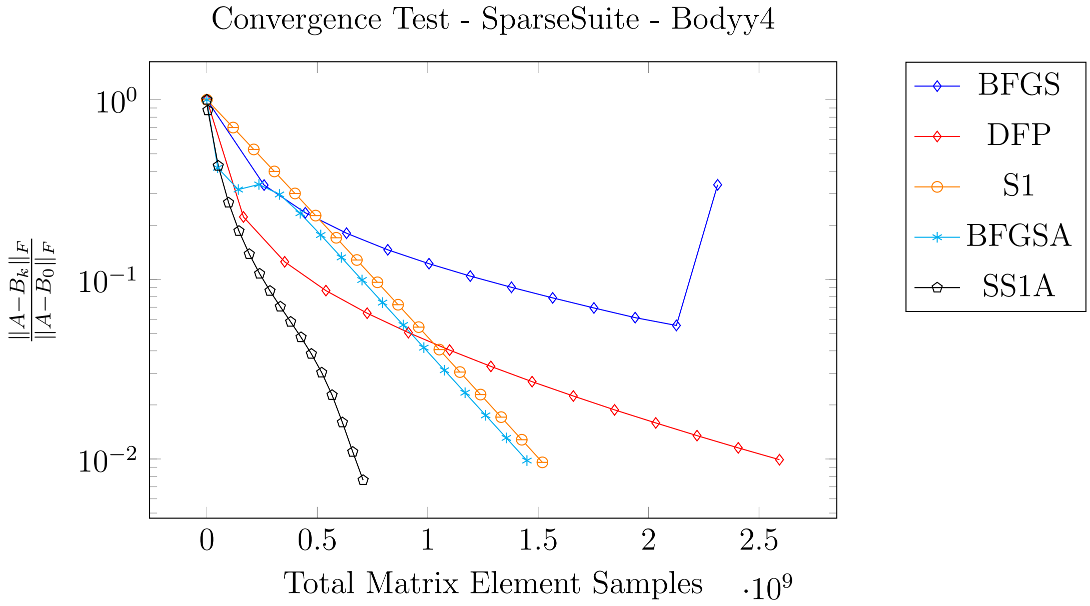


Figure A.16: Approximation of **Body** matrix ($n = 17,546$) from [7] with $s = 133 = \lceil\sqrt{17,546}\rceil$. The BFGS method fails after good initial convergence. Algorithm 4 shows consistent acceleration as in other experiments.

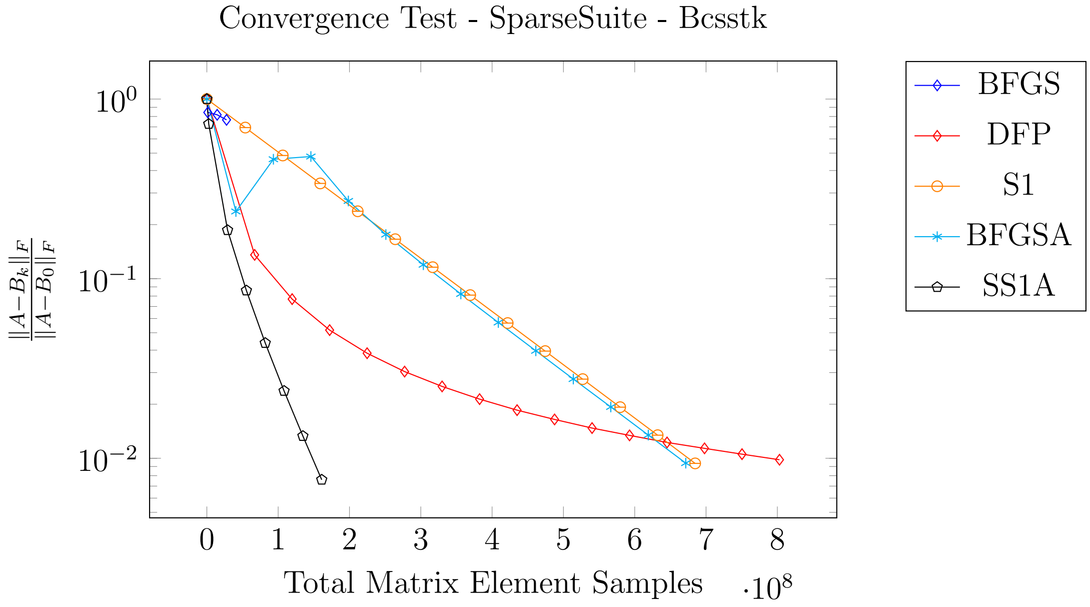


Figure A.17: Approximation of **bcsstk** matrix ($n = 11,948$) from [7] with $s = 110 = \lceil\sqrt{11,948}\rceil$. The BFGS update fails early in the iteration and was terminated. Algorithm 4 shows consistent acceleration compared to BFGSA and S1.

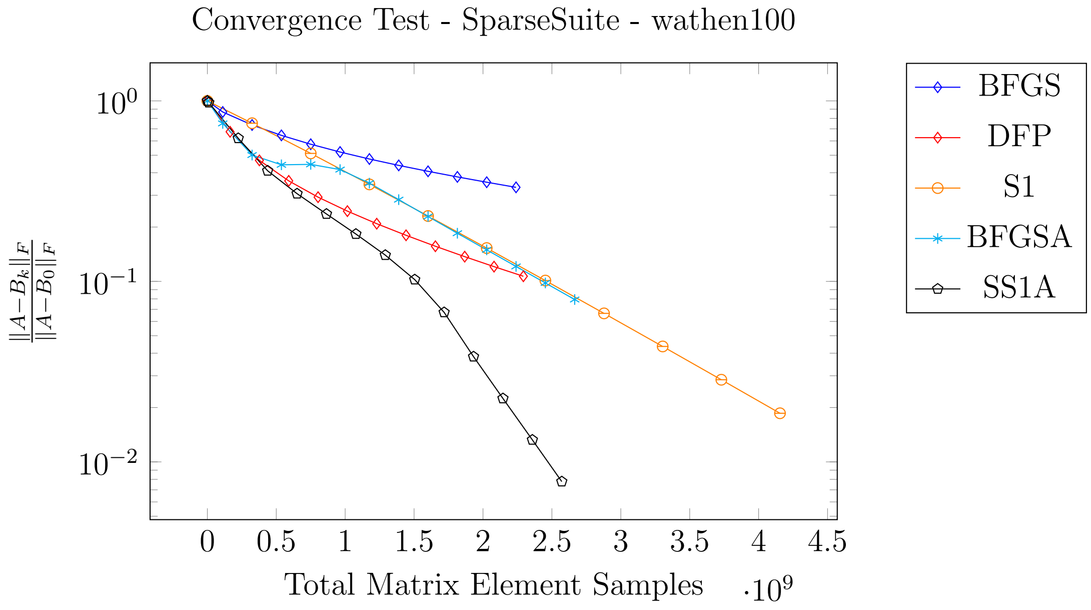


Figure A.18: Approximation of **wathen** matrix ($n = 30,401$) from [7] with $s = 175 = \lceil\sqrt{30,401}\rceil$. All methods perform consistently regardless of the matrix being very large. The DFP and BFGS methods were terminated due to maximum run-time while algorithm 4 shows consistent acceleration and faster run-times.

Appendix B

Inverse Matrix Approximation

B.1 Additional Computational Results

The convergence test from section 2.3.1 was performed on the remaining matrices tested in [14]. As before, these figures show: BFGS($\diamond$) as specified by eq. (2.5); DFP ($\diamond$) as specified by eq. (2.4); SS1 ($\bullet$) as specified by Algorithm 2. All numerical experiments indicate that the non-accelerated sub-sampled algorithms converge predictably and consistently,

† Figure B.1 the LibSVM matrix Aloi of size $n = 128$;

† Figure B.2 the LibSVM matrix Protein of size $n = 357$;

† Figure B.3 the LibSVM matrix Real-Sim of size $n = 20958$;

† Figure B.4 the Sparse Suite matrix ND6K of size $n = 18000$;

† Figure B.5 the Sparse Suite matrix ex9 of size $n = 3363$;

† Figure B.6 the Sparse Suite matrix Chem97ZtZ of size $n = 2541$.

† Figure B.7 the Sparse Suite matrix Body of size $n = 17556$.

† Figure B.8 the Sparse Suite matrix bcsstk of size $n = 11948$.

† Figure B.9 the Sparse Suite matrix wathen of size $n = 30401$.

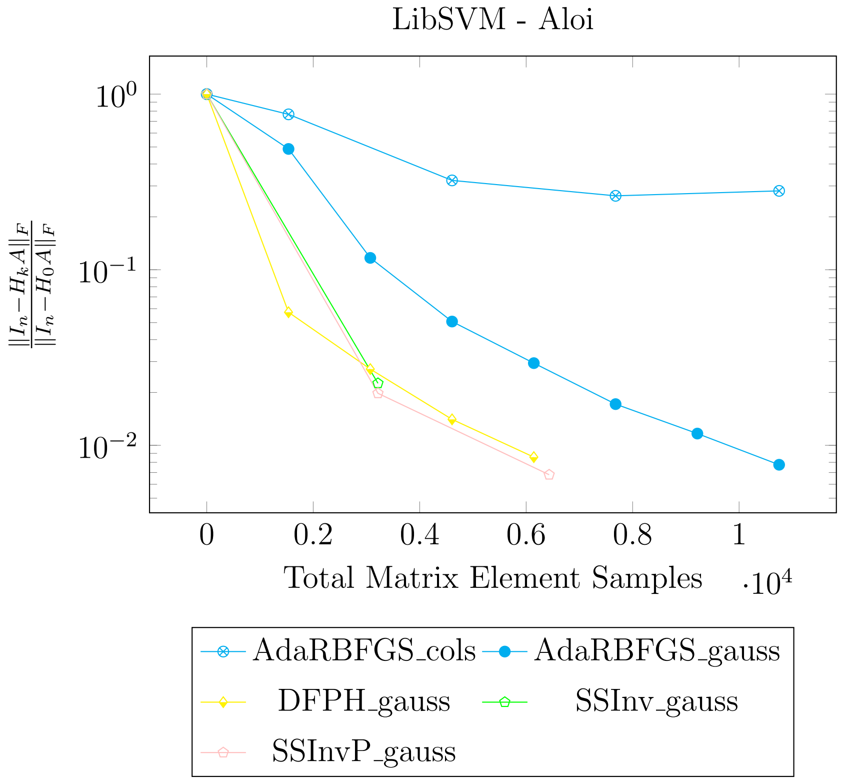


Figure B.1: Computing an approximate inverse of the Hessian matrix from the LibSVM problem, **Aloi** ($n = 128$) [5] with $s = 12 = \lceil\sqrt{128}\rceil$. AdaRBFGS$_{cols}$ method shows slow convergence while AdaRBFGS$_{gauss}$ has improved performance.

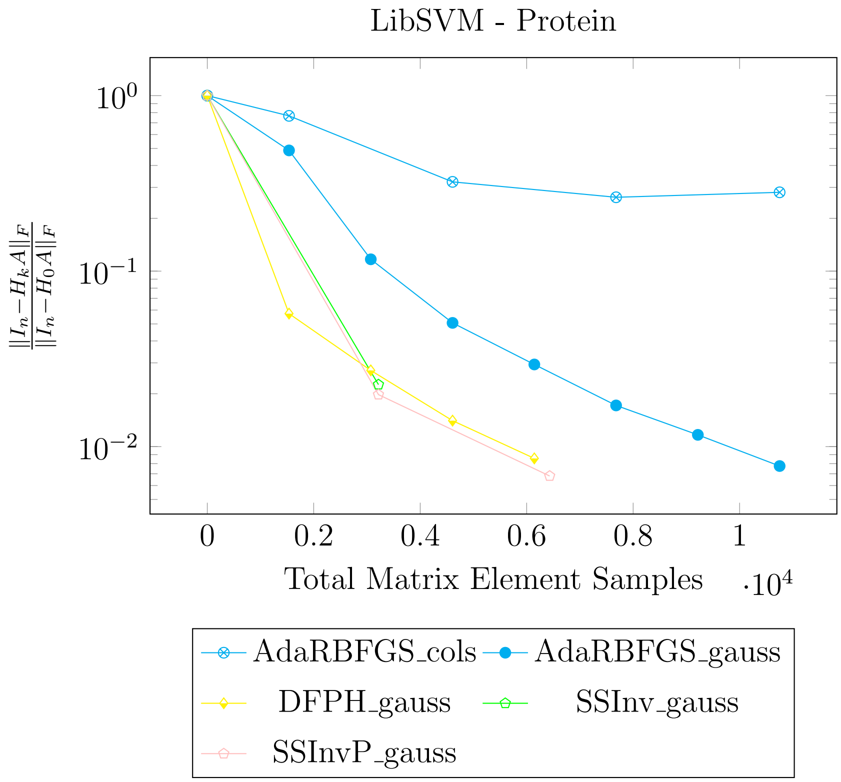


Figure B.2: Computing an approximate inverse of the Hessian matrix from the LibSVM problem, **Protein** ($n = 357$) [5] with $s = 19 = \lceil\sqrt{357}\rceil$. AdaRBFGS$_{cols}$ method shows slower convergence.

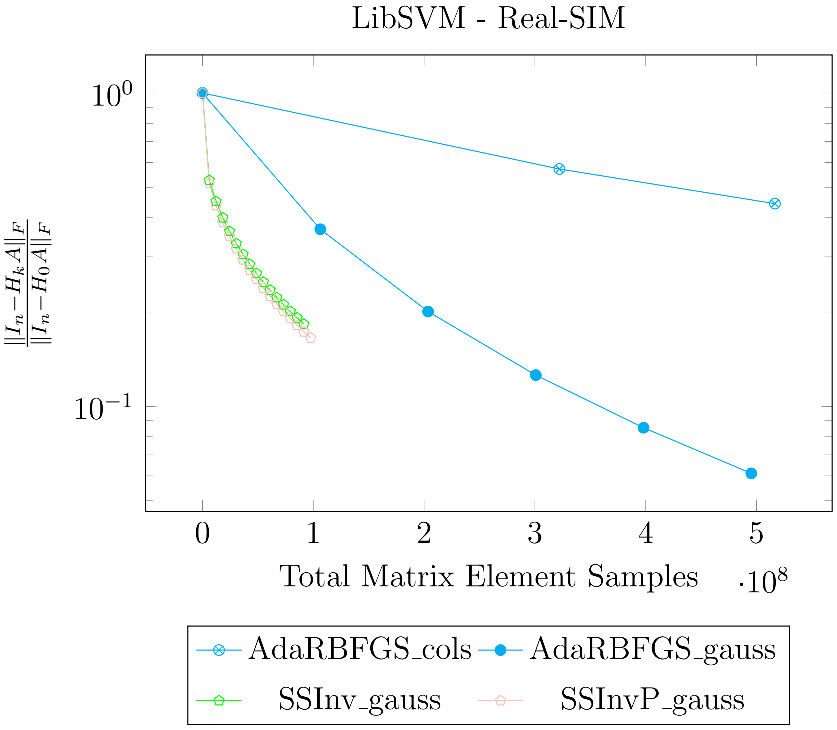


Figure B.3: Computing an approximate inverse of the Hessian matrix from the LibSVM problem, **Real-Sim** ($n = 20,958$) [5] with $s = 145 = \lceil\sqrt{20,958}\rceil$. Plot indicates a maximum run-time was reached for the subsampled methods.

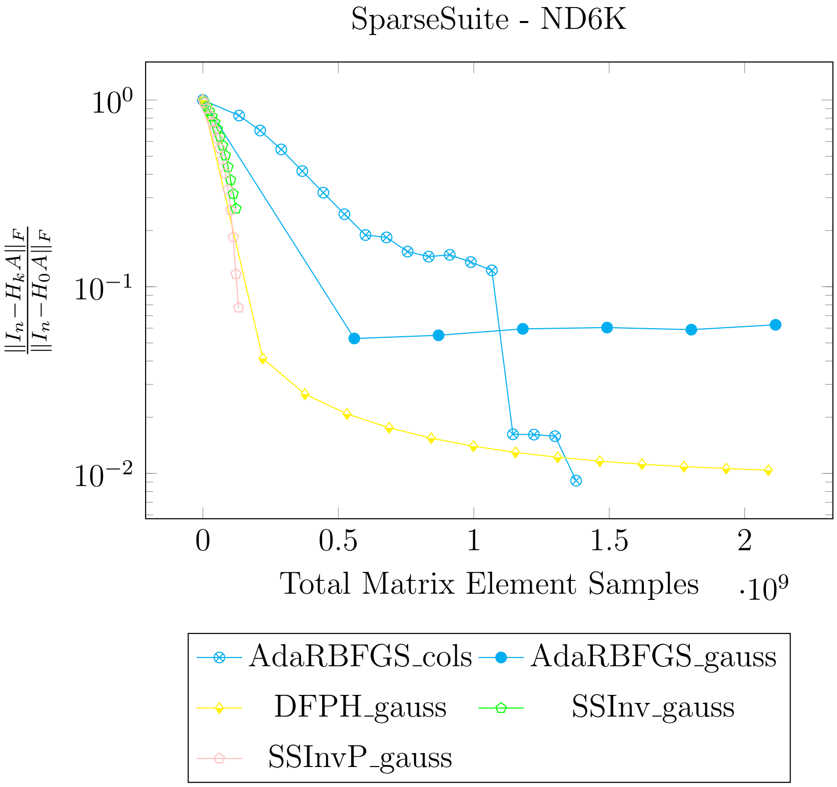


Figure B.4: Computing an approximate inverse of **ND6K** matrix ($n = 18,000$) from [7] with $s = 135 = \lceil\sqrt{18,000}\rceil$. AdaRBFGS$_{gauss}$ shows faster initial convergence which is outperformed by AdaRBFGS$_{cols}$. Algorithms 8 and 9 were terminated due to maximum run-time but show fast convergence.

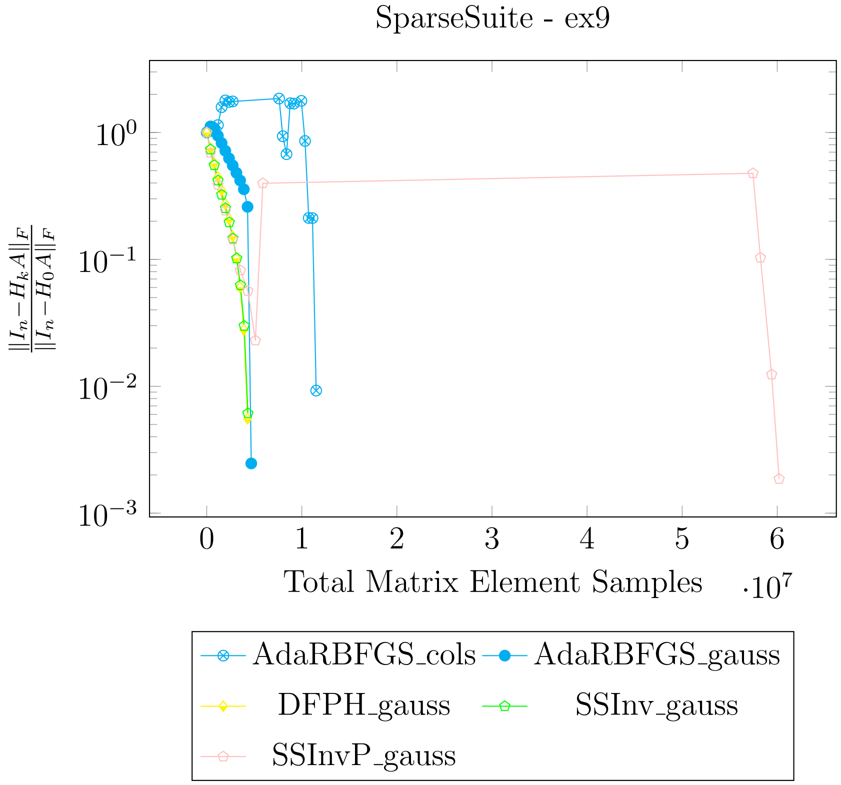


Figure B.5: Computing an approximate inverse of **ex9** matrix ($n = 3363$) from [7] with $s = 58 = \lceil\sqrt{3363}\rceil$. Plots indicate slow convergence for AdaRBFGS$_{cols}$ and SSInvP initially which is later corrected.

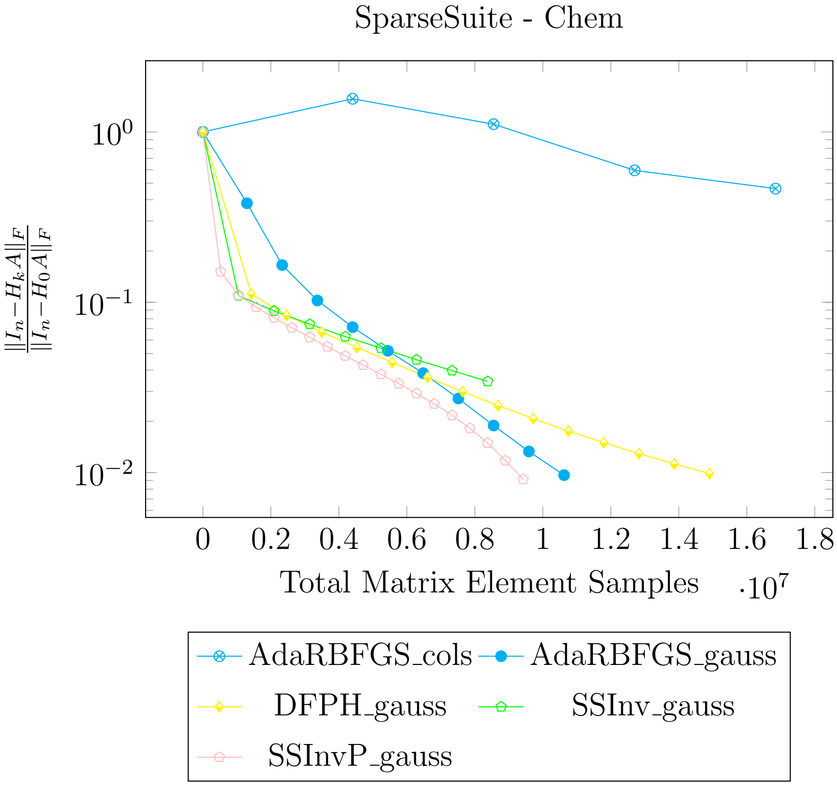


Figure B.6: Computing an approximate inverse of **Chem97ZtZ** matrix ($n = 2541$) from [7] with $s = 51 = \lceil\sqrt{2541}\rceil$. AdaRBFGS$_{cols}$ shows poor convergence. Algorithm 9 outperforms other algorithms.

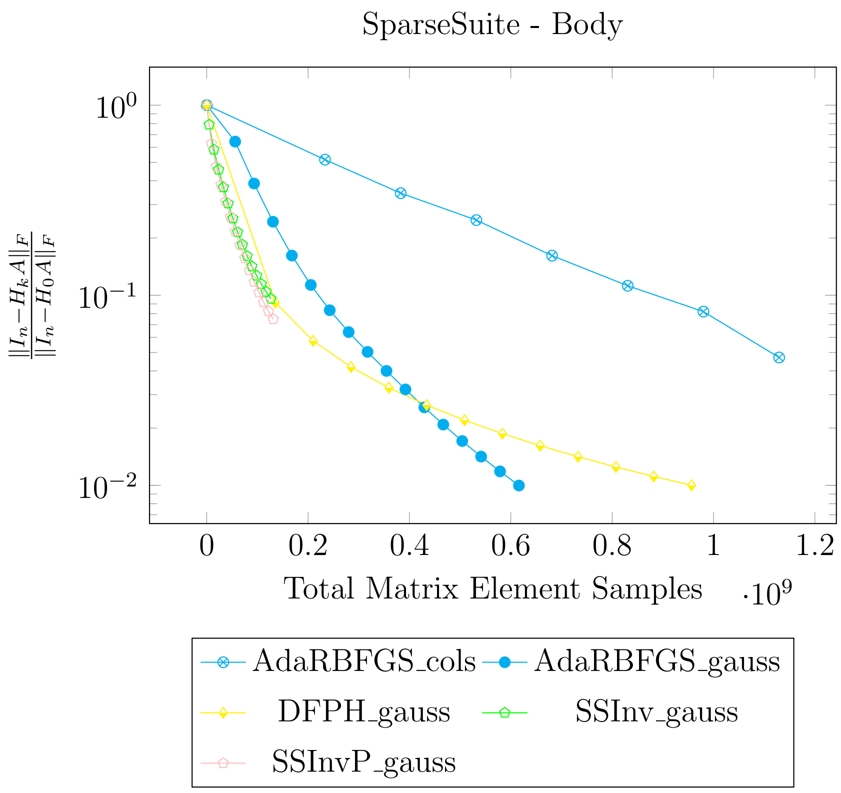


Figure B.7: Computing an approximate inverse of **Body** matrix ($n = 17,546$) from [7] with $s = 133 = \lceil\sqrt{17,546}\rceil$. AdaRBFGS$_{cols}$ performs slowly while AdaRBFGS$_{gauss}$ and the DFPH update converge. Algorithms 8 and 9 were terminated due to maximum run-time.

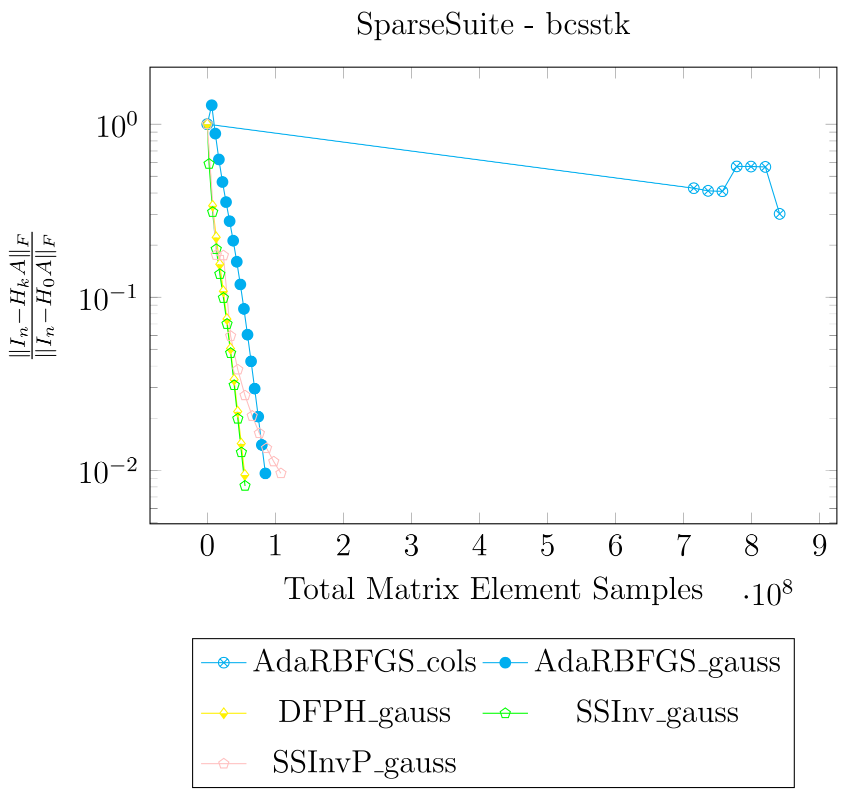


Figure B.8: Computing an approximate inverse of **bcsstk** matrix ($n = 11,948$) from [7] with $s = 110 = \lceil\sqrt{11,948}\rceil$. Note the slow convergence of the AdaRBFGS$_{cols}$ method. For this test matrix algorithm 8 outperforms algorithm 9.

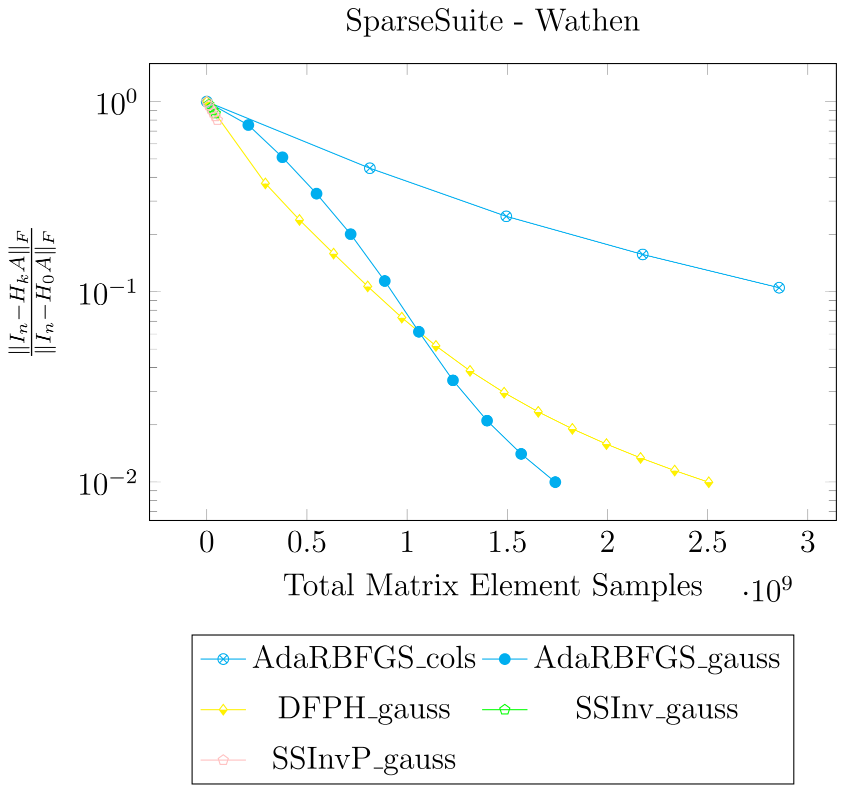


Figure B.9: Computing an approximate inverse of **wathen** matrix ($n = 30,401$) from [7] with $s = 175 = \lceil \sqrt{30,401} \rceil$. AdaRBFGS$_{cols}$ shows slow performance while AdaRBFGS$_{gauss}$ and the DFPH update perform well. Algorithms 8 and 9 show consistent initial performance but were terminated due to exceeding the maximum run-time.